

Domain Models • A Compendium

Dines Bjørner

early draft: incomplete texts, incomplete formalizations ...



The Domain Models are according to the revision, [63, 2023], of this book!
March 12, 2024: 10:48 am

Dines Bjørner
Technical University of Denmark
Fredsevej 11. DK-2840 Holte, Denmark

- This is a vastly incomplete version.
- Chapter 1 lacks several small sections.
- Chapter 2 is relatively “complete”.
- Chapter 3 is relatively “complete”.
- Chapter 4 is yet to be written.
- The models of the appendix have all been edited.
Some text should be either removed or edited.

- **This document is not intended for book publication.**
- **It will, instead, be “published” on the Internet:**
- <https://www.imm.dtu.dk/~dibj/2024/models/domain-models.pdf>

Preface

We present a collection of more-or-less “complete” domain models.
These were worked out in the period 1993–2023.

The Triptych Dogma

In order to *specify* **S**oftware, we must understand its **R**equirements.
In order to *prescribe* **R**equirements we must understand the **D**omain.
So we must study, analyze and describe **D**omains.
 $\mathbb{D}, \mathbb{S} \models \mathbb{R}$

In proofs of **S**oftware correctness,
with respect to **R**equirements,
assumptions are made with respect to the **D**omain.

Contents

I	Foundations	1
1	Introduction	3
1.1	A Completely New Approach to Software Development	3
1.2	Aims & Objectives	4
1.3	The Method	4
1.4	Caveats	5
1.5	Have a Good Read!	7
2	Domains	9
2.1	Domains: What are They?	10
2.2	A Domain Analysis & Description Ontology	12
2.3	The Name, Type and Value Concepts	14
2.4	Phenomena and Entities	14
2.5	Endurants and Perdurants	15
2.6	Phases, Stages and Steps of Domain Study	16
2.7	External and Internal Endurant Qualities	17
	– Tangibles and Intangibles	17
2.8	Perdurant Concepts	30
2.9	Perspectives	36
3	The AMoL Language	37
3.1	A Resumé of Domains	39
3.2	Values, Types and Sorts, Axioms	39
3.3	Expressions, Statements, Clauses	40
3.4	Specification Units	40
3.5	Types and Values	45
3.6	Expressions	46
3.7	Statements	52
3.8	Concurrency	55
3.9	Summary	61
II	Conclusion	63
4	Conclusion	65
5	Bibliography	67

III	APPENDIX	79
	APPENDIX	79
IV	Conceptual Domain Models	81
A	A Graph Domain	83
	A.1 Introduction	85
	A.2 Examples of Networks	86
	A.3 Classical Mathematical Models	90
	A.4 Our General Graph Model	94
	A.5 The Nets Domain	112
B	Rivers	113
	B.1 Introduction	113
	B.2 External Qualities – The Endurants	115
	B.3 Internal Qualities	116
	B.4 Conclusion	120
C	Canals	121
	C.1 Introduction	122
	C.2 Visualisation of Canals	122
	C.3 The Endurants	123
	C.4 Conclusion	153
D	The 7 Seas	155
	D.1 Introduction	156
	D.2 Endurants	156
	D.3 Perdurants	167
	D.4 Conclusion	167
V	Concrete Domain Models	169
E	Road Transport	171
	E.1 The Road Transport Domain	172
	E.2 External Qualities	172
	E.3 Internal Qualities	174
	E.4 Perdurants	182
	E.5 System Initialisation	188
F	Rail Systems	191
	F.1 Endurants – Rail Nets and Trains	192
	F.2 Transcendental Deduction	204
	F.3 Perdurants	206
	F.4 Closing	209
G	Simple Credit Card Systems	211
	G.1 Introduction	211
	G.2 Endurants	212
	G.3 Perdurants	215

H	A Simple Retailer System	223
H.1	Two Approaches to Modeling	224
H.2	The Retailer Market Case Study	225
H.3	Endurants: External Qualities	229
H.4	Endurants: Internal Qualities	232
H.5	Merchandise	242
H.6	Perdurants	243
H.7	Conclusion	258
I	Pipelines	261
I.1	Endurants: External Qualities	262
I.2	Endurants: Internal Qualities	263
I.3	Perdurants	272
I.4	Review	276
J	Shipping	277
J.1	Informal Sketches of the Shipping Domain	279
J.2	Endurants: External Qualities	283
J.3	Endurants: Internal Qualities	286
J.4	Perdurants	294
J.5	Review	303
K	Container Terminals	305
K.1	Introduction	308
K.2	Some Pictures	309
K.3	SECT	313
K.4	Main Behaviours	315
K.5	Endurants	317
K.6	Perdurants	335
K.7	Conclusion	360
L	The Blue Skies	361
L.1	Introduction	361
L.2	Endurants	362
L.3	Perdurants	362
L.4	Conclusion	362
M	Document Systems	363
M.1	Introduction	364
M.2	Managing, Archiving and Handling Documents	365
M.3	Principal Endurants	365
M.4	Unique Identifiers	365
M.5	Mereology	366
M.6	Documents: A First View	366
M.7	Behaviours: An Informal, First View	367
M.8	Channels, A First View	368
M.9	An Informal Graphical System Rendition	369
M.10	Behaviour Signatures	369
M.11	Time	369
M.12	Behaviour "States"	370
M.13	Inter-Behaviour Messages	371
M.14	A General Discussion of Handler and Document Interactions	373
M.15	Channels: A Final View	373
M.16	An Informal Summary of Behaviours	373

M.17	The Behaviour Actions	376
M.18	Documents in Public Government	383
M.19	Documents in Urban Planning	383
N	Swarms of Drones	385
N.1	An Informal Introduction	387
N.2	Entities, Endurants	388
N.3	Operations on Universe of Discourse States	401
N.4	Perdurants	402
N.5	Conclusion	415
O	Automobile Assembly Lines	417
O.1	Introduction	419
O.2	A Domain Analysis & Description	419
O.3	Discussion	451
O.4	Conclusion	451
P	Nuclear Power Plants	455
P.1	Introduction	457
P.2	Informal Characterisation	457
P.3	Sketch of a Conceptual Domain Model	459
P.4	System Domains	489
P.5	Varieties of Generation III-IV Reactors	495
P.6	Closing	495
P.7	Bibliography	497
VI	System Models	499
Q	Urban Planning	501
Q.1	Structures and Parts	504
Q.2	Unique Identifiers	508
Q.3	Mereologies	512
Q.4	Attributes	515
Q.5	The Structure Translators	525
Q.6	Channel Analysis and Channel Declarations	526
Q.7	The Atomic Part Translators	530
Q.8	Initialisation of The Urban Space Analysis & Planning System	545
Q.9	Further Work	546
R	Weather Systems	551
R.1	On Weather Information Systems	552
R.2	Major Parts of a Weather Information System	553
R.3	Endurants	554
R.4	Perdurants	559
R.5	Conclusion	565
S	The Tokyo Stock Exchange, 2009	567
S.1	Introduction	568
S.2	The Problem	568
S.3	A Domain Description	568
S.4	Conclusion	573
S.5	Tetsuo Tamai's Paper	575
S.6	Tokyo Stock Exchange arrowhead Announcements	583

T	XVSM: An Extensible Virtual Shared Memory	591
T.1	Introduction	592
T.2	XVSM Trees	595
T.3	XTree Operations	598
T.4	Indexing	603
T.5	Queries	604
T.6	Conclusion	606

Part I

Foundations

Chapter 1

Introduction

The Triptych Dogma

In order to *specify Software*, we must understand its *Requirements*.
In order to *prescribe Requirements* we must understand the *Domain*.
So we must study, analyze and describe *Domains*.
 $\mathbb{D}, \mathbb{S} \models \mathbb{R}$

Contents

1.1	A Completely New Approach to Software Development	3
1.2	Aims & Objectives	4
1.2.1	Aims	4
1.2.2	Objectives	4
1.3	The Method	4
1.4	Caveats	5
1.4.1	Definitions versus Characterizations	5
1.4.2	The Bases	5
1.4.2.1	Type and Values · Identifiers and Names	5
1.4.2.1.1	Type and Values.	5
1.4.2.1.2	Identifiers and Names.	5
1.4.2.2	Two Languages	6
1.4.2.2.1	Presentation Language	6
1.4.2.2.2	Specification Language	6
1.4.3	Unfolding an Ontology, Unfolding a Method, Unfolding Domain Models	7
1.5	Have a Good Read!	7

This chapter sets the stage for the compendium.

1.1 A Completely New Approach to Software Development

I feel obliged to inform the reader that this compendium and its predecessor publications [51, 55–58, 61–63], represent a rather different approach to software development than the reader may be acquainted with.

Traditionally software development “grew” out in the “shadow” of the first von Neumann computers – around 1946. The focus was on “taming” the behaviour of the hardware computer: exploiting the intricacies of its instruction set. From that grew **Fortran** [136] from around 1954. Then **Algol 60** [121]. And so forth.

For many years computer scientists were concerned, first with programming styles, then with correctness of programs. Around late 1970s there then emerged the concept of *requirements engineering*. But nobody took the full step – as expressed in the **Triptych Dogma**, cf. top og Page 3.

With domain science & engineering we are, in a sense, turning the software development matter, “*the right side up*”!

We start where, as we obviously think that everybody should be thinking, with the domains from which applications of computing arise.¹

1.2 Aims & Objectives

1.2.1 Aims

The aim of this compendium is to present a number ($4+11+4^2$) domain models. That is: informal, narrated, and formal descriptions of segments of “*the world ‘out’ there*”!

1.2.2 Objectives

The objective of this compendium is to help the reader get started on their own domain modelling. By seeing how the author of the method himself would model a domain, the stage- and stepwise approach, it is hoped that the reader will have courage to start!

1.3 The Method

By a **method** we shall understand a set of **principles**³ and **procedures**⁴ for selecting and applying a set of **techniques**⁵ and **tools**⁶ to a *problem*⁷ in order to achieve an orderly construction of a **solution**⁸, i.e., an **artefact**.

By **methodology** we shall understand the *study & application* of one or more methods.

By a **formal method** we shall understand a method whose *principles* include that of (i) considering models of its artefacts as *mathematical* quantities, of (ii) *abstraction*, etc.; whose decisive *procedures* include that of the sequential analysis & description of first endurants, then perdurants, and, within the analysis & description of endurants, the sequential analysis & description of first their external qualities and then their internal qualities, whose *techniques* include those of specific ways of specifying properties; and whose *tools* include those of one or more **formal languages**.

By a **language** we shall here understand a set of strings of characters, i.e., sentences, sentences which are structured according to some **syntax**, i.e., **grammar**, are given meaning by some **semantics**, and are used according to some **pragmatics**.

¹So did John von Neumann and his colleagues some 70 years ago: Their applications were, domain description-wise rather well described and sufficiently well understood. It was the mathematics domain of *PDEs: partial differential equations*.

²Four models are of “conceptual” domains; 11 of reasonably realistic [“application”] domains; and four of “systems”, i.e., domains with a high degree of computing.

³By a **principle** we mean: *a principle is a proposition or value that is a guide for behavior or evaluation [Wikipedia], i.e., code of conduct*

⁴By a **procedure** we mean: *instructions or recipes, a set of commands that show how to achieve some result, such as to prepare or make something [Wikipedia], i.e., an established way of doing something*

⁵By a **technique** we mean: *a technique, or skill, is the learned ability to perform an action with determined results with good execution often within a given amount of time, energy, or both [Wikipedia], i.e., a way of carrying out a particular task*

⁶By a **tool** we mean: *a tool is an object that can extend an individual's ability to modify features of the surrounding environment [Wikipedia]*

⁷By a **problem** we shall here understand such as wishing to understand a domain, or such as obtaining a precise description of a domain from which to develop software.

⁸By a **solution** – to a problem of the kind “footnoted” kind – we shall here understand *a domain description*.

By a **formal language** we shall here understand a language whose *syntax* and *semantics* can both be expressed **mathematically** and for whose sentences one can **rationally reason** (*argue, prove*) **properties**.

The method for analyzing and describing domains has been researched for many year. Several “generations” of domain models have been worked out and their experimental development has led to refinements and simplifications of the method. This compendium records some of these examples in Chapters A–P. The specific method deployed in this compendium took its initial steps around 2009. Since then many versions have been documented [38, 41, 51, 55–58, 61, 62]. The current status of the method is documented in [63].

Chapter 2 (pages 9–36) summarizes the domain modelling method.

1.4 Caveats

1.4.1 Definitions versus Characterizations

Readers with a background in [theoretical] computer science need, perhaps, be warned. We shall, in the next chapter, Chapter 2, present a number of “*characterizations*”. They are characterizations of an informal world. A world that has not [yet] been formalized. Our domain analysis and description may aim at formalizing, in **AMoL**, in a domain model, segments of that world. But still, before such a model exists, the world we shall analyze and describe, is informal. Therefore one cannot in the conventional style of [theoretical] computer science give a so-called precise definition. Therefore we use the term ‘characterization’.

1.4.2 The Bases

The author, i.e., me (!), embarks, with You, the reader/student, on this enterprise: the writing, presentation, respectively, the study, reading, of this compendium on two bases: mine and Yours. I shall in this section, ‘*The Bases*’, very briefly outline the [two] bases on which I started researching and developing the *domain analysis and description* approach, ‘*the method*’, outlined in [38, 41, 51, 56, 58]. In outlining these two bases You, the student/reader, should be, somehow, “brought-in-line”, that is, understand and “synchronized” with the author’s intentions. They underlie the rest of this compendium, that is, is a foundation for what is presented.

1.4.2.1 Type and Values · Identifiers and Names

Hand-in-hand with types and values go identifiers naming these.

1.4.2.1.1 Type and Values. Very briefly.

Types are [special] classes, “like” sets, of values.

Values are, in this compendium, mathematical quantities.

Examples of values are: Boolean truth values, numbers, ..., sets, Cartesians, lists, maps and functions. Elements of sets, Cartesians, lists and maps are such values. Arguments and results of functions, are in this compendium, are truth values, numbers, ..., sets, Cartesians, lists and maps.

More on this in Chapter 3.

1.4.2.1.2 Identifiers and Names. Very briefly: Identifiers are sequences of alphabetic lower- and uppercase letters sometimes with infix ‘-’ and/or a suffix digit: ‘0, 1, ..., 9’, Names are identifiers naming

- (i) *values*: **value** $\text{id:T} = \dots$,
such as Boolean truth values, (natural, integer, real) numbers, ..., sets, Cartesians, lists, maps and functions;

- (ii) *types*: **type** T or **type** $T = \dots$,
where a defining right-hand side (...) may be **Bool**, **Nat**, **Int**, **Real**, ..., **T-set** (sets), (Cartesians) $T \times T \times \dots \times T$, T^* (lists), $T \xrightarrow{m} T$ (maps) and $T \rightarrow T$ (functions);
- (iii) *variables*: **variable** $v:T$; and
- (iv) *channels*: **channel** $ch[...]:T$.

where id , T , v , ch are identifiers; **Bool**, **Nat**, **Int**, **Real** are ground names of atomic types; **-set**, \times , $*$, \xrightarrow{m} and \rightarrow are type constructors (i.e., operators); and **value**, **type**, **variable**, **channel** are literals.

More on this in Chapter 3.

1.4.2.2 Two Languages

Two completely separate languages are “at play” here. (α) The language in which we formally describe domains, *the specification* or *description language*.⁹ (β) And the language in which we explain (*the presentation language*) **AMoL** as well as the language in which we “*narrate*”, i.e., informally, describe domains.

1.4.2.2.1 Presentation Language The general presentation language of this compendium is English. We present a domain modelling method: its principles, techniques and tools in English. We also “*narrate*”, “tell-the-story”, of each and every specific domain in English.

The narration text denotes, refers to, designates, the domain being told about.

We also use English to explain the specification language, here **AMoL**. Finally we “adorn” our English presentation language with a number of domain analysis “predicates”: **is_entity**, **is_endurant**, **is_perdurant**, **is_solid**, **is_fluid**, **is_part**, **is_atomic**, **is_compound**, **is_Cartesian**, **is_part_set**, etc., and analysis “functions”: **record_Cartesian_part_type_names**, **record_part_set_part_type_names**, etc. These predicates and functions are informal. They represent ideas by which the domain analyzer cum describers proceed in their work. Consider them prompts issued by the brain of the domain analyzer cum describer. Prompts, similar to *cues* issued by a theater play prompter to the actors who may have forgotten “their next line”.¹⁰

1.4.2.2.2 Specification Language The specification language of this compendium is **AMoL**.

AMoL is basically an applicative, i.e., a functional language. It is a simple mathematics-based language. It is formal, that is: it has a well-defined syntax, semantics, and proof system.¹¹ It derives from RSL, the RAISE Specification Language, [100], where RAISE stands for Rigorous Approach to Industrial Software Engineering [101]. RSL “derives from”, i.e., was inspired by applicative languages LISP [137], AE [124], VDM [5–7, 15], and ML [105, 106]. **AMoL** is not the only possible, formal language for describing domains. We suggest that others could be used. For example VDM, Z [138, 148, 173, 174, 182], RSL and Alloy [115]. Also algebraic specification languages like **cafeOBJ** [86, 99] or **CASL** [141].

The formal text denotes a mathematical object.

The relation between the domain and the mathematical object of “its” description is that of a *transcendental deduction*.¹²

The mathematics of **AMoL** specifications allows the rigorous “derivation” of domain descriptions into requirements prescriptions [58, *Chapter 8*], and these into “executable” software [26–28].

The **AMoL** language is outlined in Chapter 3.

⁹We shall, in this compendium, call (refer to this) this language (by) **AMoL**: for ‘a modelling language’.

¹⁰French: *signal*, German: *stichwort*.

¹¹We shall not present the proof system in this compendium.

¹²See Sect. 2.8.2 on page 30

1.4.3 Unfolding an Ontology, Unfolding a Method, Unfolding Domain Models

We are “*riding three horses*” in this compendium.

- First and foremost we are presenting a *new way of looking at the phenomenon of domains*.
- Secondly we are presenting a *method, with its principles, procedures, techniques and tools, for analyzing & describing [such] domains*.
- Thirdly we are *presenting* a sizable number (19!) of *domain models*, several of these developed according to ‘the method’.

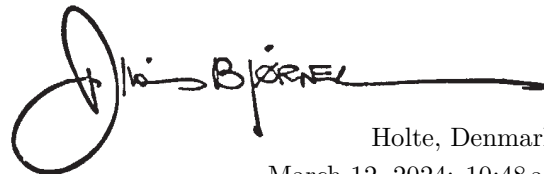
The *new way of looking at the phenomenon of domains* is manifested in two ways: (i) in our focus on domains, not on requirements, not on software; and (ii) in our novel way at analyzing & describing domains – as outlined in Chapter 2.

The *method for analyzing & describing domains* is manifested as follows: (i) in taking the term *method* serious; (ii) in endowing that term with *principles, procedures, techniques and tools*; and (iii) in justifying a domain analysis & description *ontology* as inevitable – with reference to Kai Sørlander’s philosophy [167–172]

The *presentation of domain models* is manifested as follows: (i) as the method is “unravelling”, in Chapter 2, we likewise “unravel” a [specific] domain analysis & description, i.e., a model; and (ii) we present, as the main purpose of this compendium, in Appendix Chapters A–T, 18 more-or-less detailed domain models – pages 83–606 – some 520 pages!

So we beg the reader’s patience and ability to distinguish which compendium texts represent one or another of these three facets.

1.5 Have a Good Read!

A handwritten signature in black ink, appearing to read "John Bjørnel". The signature is stylized with a large, looping initial "J" and a long horizontal stroke extending to the right.

Holte, Denmark

March 12, 2024: 10:48 am

Chapter 2

Domains

Contents

2.1	Domains: What are They?	10
2.1.1	A Characterization	11
2.1.2	Endurants and Perdurants	11
2.1.3	A Discussion of Our Characterization of a Concept of Domain	12
2.2	A Domain Analysis & Description Ontology	12
2.2.1	The Chosen Ontology	12
2.2.2	Discussion of The Chosen Ontology	13
2.3	The Name, Type and Value Concepts	14
2.3.1	Names	14
2.3.2	Types	14
2.3.3	Values	14
2.4	Phenomena and Entities	14
2.5	Endurants and Perdurants	15
2.5.1	Endurants	15
2.5.2	Perdurants	15
2.5.3	Ontological Choice	16
2.6	Phases, Stages and Steps of Domain Study	16
2.7	External and Internal Endurant Qualities	17
	– Tangibles and Intangibles	17
2.7.1	External Qualities – Tangibles	17
2.7.1.1	The Universe of Discourse	17
2.7.1.2	Solid and Fluid Endurants	18
2.7.1.2.1	Discrete or Solid Endurants.	18
2.7.1.2.2	Fluids.	18
2.7.1.3	Parts and Living Species Endurants	18
2.7.1.3.1	Parts.	19
2.7.1.3.1.1	Atomic Parts.	19
2.7.1.3.1.2	Compound Parts.	19
2.7.1.3.1.3	Cartesians.	19
2.7.1.3.1.4	Part Sets.	20
2.7.1.4	Compound Observers.	21
2.7.1.5	Example Domain Models: External Qualities	22
2.7.1.6	States.	22
2.7.1.7	Validity of Endurant Observations.	22
2.7.1.8	Summary of Analysis Predicates.	22

2.7.2	Internal Qualities – Intangibles	23
2.7.2.1	Unique Identity.	23
2.7.2.1.1	Uniqueness of Parts	24
2.7.2.1.2	Example Domain Model Unique Identifiers:	24
2.7.2.2	Mereology.	25
2.7.2.2.1	Example Domain Model Mereologies:	25
2.7.2.3	Attributes.	26
2.7.2.3.1	General	26
2.7.2.3.2	Michael A. Jackson’s Attribute Categories.	27
2.7.2.3.2.1	A Presentation	27
2.7.2.3.2.2	A Revision	27
2.7.2.3.3	Analytic Attribute Extraction Functions:	28
2.7.2.3.4	Example Domain Model Attributes:	29
2.7.3	Intentional Pull	29
2.7.4	Summary of Endurants	30
2.8	Perdurant Concepts	30
2.8.1	“Morphing” Parts into Behaviours	30
2.8.2	Transcendental Deduction	30
2.8.3	Actors – A Synopsis	31
2.8.3.1	Action.	31
2.8.3.2	Event.	31
2.8.3.3	Behaviour.	31
2.8.4	Channel	31
2.8.5	Behaviours	32
2.8.5.1	Behaviour Signature.	32
2.8.5.2	Inert Arguments: Some Examples.	33
2.8.5.3	Behaviour Invocation.	33
2.8.5.4	Argument References	33
2.8.5.4.1	Evaluation of Monitorable Attributes	34
2.8.5.4.2	Update of Biddable Attributes	34
2.8.5.5	Behaviour Description – Examples	35
2.8.5.5.1	Example Domain Model Behaviours:	35
2.8.5.6	Behaviour Initialization.	36
2.9	Perspectives	36

The Triptych Dogma

In order to *specify* Software, we must understand its *Requirements*.
 In order to *prescribe* *Requirements* we must understand the *Domain*.
 So we must study, analyze and describe *Domains*.

$$\mathbb{D}, \mathbb{S} \models \mathbb{R}^1$$

This chapter presents a *method*, its *principles*, *procedures*, *techniques* and *tools*, for *analyzing* &² *describing* domains [38, 41, 51, 56, 58].

2.1 Domains: What are They ?

But what do we mean by ‘domain’ ?

¹In proofs of Software correctness, with respect to *Requirements*, assumptions are made with respect to the *Domain*.

²We use here the ampersand, ‘&’, as in *A&B*, to emphasize that we are treating *A* and *B* as one concept.

2.1.1 A Characterization

Characterization 1 Domain: By a *domain* we shall understand a *rationally describable* segment of a *discrete dynamics* fragment of a *human assisted* reality: the world that we daily observe – in which we work and act, a reality made significant by human-created entities. The domain embody *endurants* and *perdurants* ■

Example 1 Some Domain Examples: A few, more-or-less self-explanatory examples:

- **Rivers** – with their natural sources, deltas, tributaries, waterfalls, etc., and their man-made dams, harbours, locks, etc. – and their conveyage of materials (ships etc.), cf. Chapter B.
- **Road nets** – with street segments and intersections, traffic lights and automobiles – and the flow of these, cf. Chapter E.
- **Pipelines** – with their wells, pipes, valves, pumps, forks, joins and wells and the flow of fluids, cf. Chapter I. and
- **Container terminals** – with their container vessels, containers, cranes, trucks, etc. – and the movement of all of these, cf. Chapter K. ■

Characterization 1 relies on the understanding of the terms '*rationally describable*', '*discrete dynamics*', '*human assisted*', '*solid*' and '*fluid*'. The last two will be explained later. By **rationally describable** we mean that what is described can be understood, including reasoned about, in a rational, that is, logical manner – in other words **logically tractable**.³ By **discrete dynamics** we imply that we shall basically rule out such domain phenomena which have properties which are continuous with respect to their time-wise, i.e., dynamic, behaviour. By **human-assisted** we mean that the domains – that we are interested in modelling – have, as an important property, that they possess man-made entities.

2.1.2 Endurants and Perdurants

The above characterization hinges on the characterizations of endurants and perdurants.

Characterization 2 Endurants: Endurants are those quantities of domains that we can observe (see and touch), in *space*, as “complete” entities at no matter which point in *time* – “material” entities that persists, endures – capable of enduring adversity, severity, or hardship [Merriam Webster] ■

Endurants are either *natural* [“God-given”] or *artefactual* [“man-made”]. Endurants may be either solid (discrete) or fluid, and solid endurants, called parts, may be considered *atomic* or *compound* parts; or, as in this compendium solid endurants may be further unanalysed *living species*: *plants* and *animals* – including *humans*.

Characterization 3 Perdurants: Perdurants are those quantities of domains for which only a fragment exists, in *space*, if we look at or touch them at any given snapshot in *time* [Merriam Webster] ■

Perdurants are here considered to be *actions*, *events* and *behaviours*.

• • •

We exclude, from our treatment of domains, issues of ethics, biology and psychology.

³Another, “upside-down” – after the fact – [perhaps ‘cheating’] way of defining ‘describable’ is: is it describable in terms of the method of this chapter !

2.1.3 A Discussion of Our Characterization of a Concept of Domain

Characterization 1 on the preceding page is our attempt to delineate the subject area. That is, “our” concept of ‘domain’ is ‘novel’: *new and not resembling something formerly known or used*. As such it may be unfamiliar to most readers. So it takes time to digest that characterization. So the reader may have to return to the page, Page 11, to be reminded of the definition.

2.2 A Domain Analysis & Description Ontology

2.2.1 The Chosen Ontology

Figure 2.1 expresses an ontology⁴ for our analysis of domains. Not a taxonomy⁵ for any one specific domain.

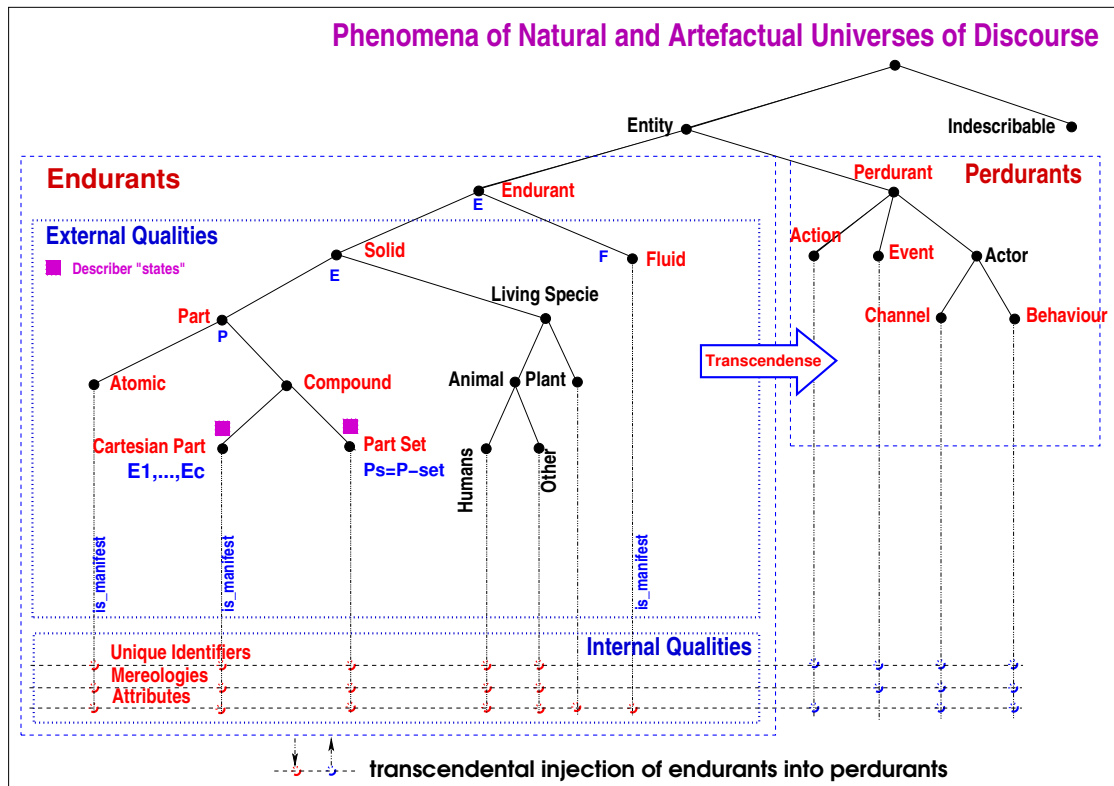


Figure 2.1: A Domain Analysis & Description Ontology

The idea of Fig. 2.1 is the following:

- It presents a recipe for how to **analyze** a domain.
- You, the *domain analyzer cum describer*, are ‘confronted’⁶ with, or by a domain.

⁴An ontology is the philosophical study of being. It investigates what types of entities exist, how they are grouped into categories, and how they are related to one another on the most fundamental level (and whether there even is a fundamental level) [Wikipedia].

⁵A taxonomy (or taxonomic classification) is a scheme of classification, especially a hierarchical classification, in which things are organized into groups or types [Wikipedia].

⁶By ‘confronted’ we mean: You are reading about it, in papers, in books, in postings on the Internet, visiting it, talking with domain stakeholders: professional people working “in” the domain; You may, yourself, “be an entity” of that domain!

- You have Fig. 2.1 on the facing page in front of you, on a piece of paper, or in Your mind, or both.
- You are then asked, by the domain **analysis** & description method of this chapter, to “start” at the uppermost •, just below and between the ‘r’ and the first ‘s’ in the main title, Phenomena of Natural and Artefactual Universes of Discourse.
- The **analysis** & description ontology of Fig.2.1 then *directs* You to inquire as to whether the phenomenon – whichever You are ”looking at/reading about/...” – is either *rationally describable*, i.e., is an *entity* (**is_entity**) or is *indescribable*.
- That is, You are, in general, “positioned” at a bullet, •, labeled α , “below” which there may be two alternative bullets, one, β , to the right and one to the left, γ .
- It is Your decision whether the answer to the “query” that each such situation warrants, is yes, **is- β** , or no, **is- γ** .
- The characterizations of the concepts whose names, α, β, γ etc., are attached to the •s of Fig. 2.1 are given in the following sections.
- Whether they are precise enough to guide You in Your obtaining reasonable answers, “yes” or “no”, to the •ed queries is, of course, a problem. I hope they are.
- If Your answer is “yes”, then Your **analysis** is to proceed “down the tree”, usually indicated by “yes” or “no” answers.
- If one, or the other is a “leaf” of the ontology tree, then You have finished examining the phenomena You set out to **analyze**.
- If it is not a leaf, then further **analysis** is required.
- (We shall, in this compendium, leave out the analysis and hence description of *living species*.)
- If an **analysis** of a phenomenon has reached one of the (only) two ■’s, then the **analysis** at that • results in the domain describer **describing** some of the properties of that phenomenon.
- That **analysis** involves “setting aside”, for subsequent **analysis & description**, one or more [thus **analysis** etc.-pending] phenomena (which are subsequently to be tackled from the “root” of the ontology).

We do not [need to] prescribe in which order You analyze & describe the phenomena that has been “set aside”.

2.2.2 Discussion of The Chosen Ontology

We shall in the following motivate the choice of the *ontological classification* reflected in Fig 2.1 on the preceding page. We shall argue that this classification is not “an accidental choice”. In fact, we shall try justify the classification with reference to the philosophy of Kai Sørlander [167–172]⁷. Kai Sørlander’s aim in these books is to examine *that which is absolutely necessary, inevitable, in any description of the world*. In [58, Chapter2] we present a summary of Sørlander’s philosophy. In paragraphs, in the rest of this chapter, marked **Ontological Choice**, we shall relate Sørlander’s philosophy’s “inevitability” to the ontology for studying domains.

⁷The 2022 book, [171], is presently a latest in Kai Sørlander’s work. It refines and further develops the theme of the earlier, 1994–2016 books. [172] is an English translation of [171]

2.3 The Name, Type and Value Concepts

Domain *modelling*, as well as *programming*, depends, in their *specification*, on *separation of concerns*: which kind of *values* are subjectable to which kinds of *operations*, etc., in order to achieve ease of *understanding* a model or a program, ease of *proving properties* of a model, or *correctness* of a program.

2.3.1 Names

We name things in order to refer to them in our speech, models and programs. Names of types and values in models and programs are usually not so-called “first-citizens”, i.e., values that can be arguments in functions, etc. The “science of names” is interesting.⁸ In botanicalsociety.org.za/the-science-of-names-an-introduction-to-plant-taxonomy the authors actually speak of a “science of names” in connection with plant taxonomy: the “art” of choosing such names that reflect some possible classification of what they name.

2.3.2 Types

The type concept is crucial to programming and modelling.

Characterization 4 *Type*: A *type* is a class of values (“of the same kind”) ■

We name types.

Example 2 *Type Names*: Some examples of type names are:

- RT – the class of all road transport instances: the *Metropolitan London Road Transport*, the *US Federal Freeway System*, etc.
- RN – the class of all road net instances (within a road transport).
- SA – the class of all automobiles (within a road transport) ■

You, the domain describer, choose type names. Choosing type names is a “serious affair”. It must be done carefully. You can choose short (as above) or long names: *Road_Transport*, *Road_Net*, etc. We prefer short, but not cryptic names, like X, Y, Z, Names that are easy to *memorize*, i.e., *mnemonics*.

2.3.3 Values

Values are what programming and modelling, in a sense, is all about”. In programming, values are the *data* “upon” which the program code specifies computations. In modelling values are, for example, what we observe: the entities in front of our eyes.

2.4 Phenomena and Entities

Characterization 5 *Phenomena*: By a *phenomenon* we shall understand a fact that is observed to exist or happen ■

Some phenomena are rationally describable – to some degree⁹ – others are not.

⁸The study of names is called *onomastics* or *onomatology*. *Onomastics* covers the naming of all things, including place names (toponyms) and personal names (*anthroponyms*).

⁹That is: It is up to the domain analyzer cum describer to decide as to how many rationally describable phenomena to select for analysis & description. Also in this sense one practices abstraction by “abstracting away” [the analysis & description of] phenomena that are irrelevant for the “current” (!) domain description.

Characterization 6 Entities: By an entity *By an entity* we shall understand a more-or-less rationally describable phenomenon ■

We introduce the informal presentation language predicate `is_entity` to hold for phenomena ϕ if `is_entity(ϕ)` holds.

Example 3 Phenomena and Entities: Some, but not necessarily all aspects of a river can be rationally described, hence can be still be considered entities. Similarly, many aspects of a road net can be rationally described, hence will be considered entities ■

If You are not happy with this ‘characterization’, then substitute “rationally describable” with: *describable in terms of the endurants and perdurants brought forward in this chapter: their external and internal qualities, unique identifiers, mereologies amd attributes, channels and behaviours!*

Ontological Choice: We choose to “initialize” our ontological “search” to a question of whether a phenomenon is rationally describable – based on the tenet of Kai Sørlander’s philosophy, namely that “whatever” we postulate is either *true* or *false* and that *a principle of contradiction* holds: *whatever we so express can not both hold and not hold* ■

Kai Sørlander then develops his inquiry – *as to what is absolutely necessary in any description of the world* – into the rationality of such descriptions necessarily be based on time and space and, from there, by a series of transcendental deductions, into a base in *Newton’s* physics. We shall, in a sense, stop there. That is, in the domain concept, such as we have delineated it, we shall not need to go into *Einsteinian* physics.

2.5 Endurants and Perdurants

2.5.1 Endurants

We repeat characterization 2 on page 11.

Characterization 2 Endurant: Endurants are those quantities of domains that we can observe (see and touch), in *space*, as “complete” entities at no matter which point in *time* – “material” entities that persists, endures – capable of enduring adversity, severity, or hardship [Merriam Webster] ■

Example 4 Endurants: Examples of endurants are: a street segment [link], a street intersection [hub], an automobile ■

We introduce the informal presentation language predicate `is_endurant` to hold for entity e if `is_endurant(e)` holds.

2.5.2 Perdurants

We repeat characterization 3 on page 11.

Characterization 3 Perdurant: Perdurants are those quantities of domains for which only a fragment exists, in *space*, if we look at or touch them at any given snapshot in *time* [Merriam Webster] ■

Example 5 Perdurant: A moving automobile is an example of a perdurant ■

We introduce the informal presentation language predicate `is_perdurant` to hold for entity e if `is_perdurant(e)` holds.

2.5.3 Ontological Choice

The **ontological choice** of entities being “viewed” as either endurants or perdurants is motivated as follows: The concept of endurants can be justified in terms of Newton’s physics without going into kinematics, i.e., without including time considerations. The concept of perdurants can then, on one hand, be justified in terms of Newton’s physics now taking time into consideration, hence kinematics, and from there causality, etc.; and, on the other hand, and as we shall see, by transcendently deducing perdurants from solid endurants ■

2.6 Phases, Stages and Steps of Domain Study

We shall next outline recommended phases, stages, sub-stages and steps of domain analysis & description. That outline will refer [forward] to a number of [domain ontology] concepts. These will be characterized in the following sections.

- An *initial phase* of domain analysis and description is that of focusing, for a while, on the study of domain endurants.
- That *phase* consists, as will transpire from the next sections, of *stages* of studying
 - first the so-called *external qualities* of domain endurants,
 - then the so-called *internal qualities* of domain endurants.
- The *stage* of studying *external qualities* consists of a number of *steps*.
 - These *steps* analyze and describe the so-called
 - * *atomic* and
 - * *compound*
 endurants of the domain.
 - The number of *steps* of studying, analyzing and describing the external qualities of domains, depends of the *taxonomy* of the endurants of the domain, that is, of how many different kinds, that is: *sorts*, of endurants the studied domain exhibits.¹⁰
- The *stage* of studying *internal qualities* consists of a number of sequentially ordered *sub-stages*.
 - There is first the *sub-stage* of *studying* the so-called *unique identification* of endurant parts.
 - Then there is the *sub-stage* of *studying* the so-called *mereologies* of endurant parts.
 - Following there is the *sub-stage* of *studying* the so-called *attributes* of endurant parts.
 - Finally there is the optional *sub-stage* of *studying* the so-called *intentional pull* of endurant parts.

Each of these sub-stages have one or more *steps*.¹¹

- The second *phase* of domain analysis and description is that of focusing, for a while, on the study of domain perdurants. That phase consists of several, sequentially ordered stages.
 - The *first stage* is that of *declaring* the *channels* by means of which domain behaviours interact.
 - The *second stage* is that of associating with each part the *signature* of the transcendently deduced *part behaviours*.

¹⁰ Section 2.7.1 (pages 17–23) both unveils our ontology of external qualities of endurants and elements of our method for describing these. It is necessary to keep these two separate aspects in mind: The first “broadens Your mind”! The second “hones Your engineering skills”!

¹¹A remark, as that of footnote 10 likewise applies here, to cover Sect. 2.7.2 (pages 23–29)

- The *third stage* is that of *defining* the body of the behaviour definitions – and all their subsidiary and auxiliary functions.
- The *fourth stage*, finally, is that of describing the initialization of the described domain.¹²

2.7 External and Internal Endurant Qualities – Tangibles and Intangibles

The main contribution of this section is that of a calculus of domain analysis and description prompts. Two facets are being presented. Aspects of a domain science: of how we suggest domains can, and should, be viewed – ontologically. And aspects of a domain engineering: of how we suggest domains can, and should, be analyzed and described.

We begin by characterizing the two concepts: external and internal qualities.

Characterization 4 External Qualities: External qualities of endurants of a manifest domain are, in a simplifying sense, those we can see, touch and have spatial extent. They, so to speak, take form.

Characterization 5 Internal Qualities: Internal qualities are those properties [of endurants] that do not occupy *space* but can be measured or spoken about ■

Perhaps we should instead label these two qualities *tangible* and *intangible* qualities.

Ontological Choice: The rational, analytic philosophy issues of the inevitability of these qualities is this: (i) can they be justified as inevitable, and (ii) can they be suitably “separated”, i.e., both disjoint and exhaustive? Or are they merely of empirical nature?

MORE TO COME

The choice here is also that we separate our inquiry into examining *both external and internal qualities* of endurants [not ‘either or’] ■

2.7.1 External Qualities – Tangibles

Example 6 External Qualities: An example of external qualities of a domains is: the Cartesian¹³ of sets of solid atomic street intersections, and of sets of solid atomic street segments, and of sets of solid automobiles of a road transport system where *Cartesian*, *sets*, *atomicity*, and *solidity* reflect external qualities ■

2.7.1.1 The Universe of Discourse

The most immediate external quality of a domain is the “entire” domain – “itself” ! So any domain analysis starts by identifying that “entire” domain! By giving it a name, say UoD, for *universe of discourse*, Then describing it, in *narrative* form, that is, in natural language containing terms of professional/technical nature, the domain. And, finally, *formalizing* just the name: giving the name “status” of being a type name, that is, of the type of a class of domains whose further properties will be described subsequently.

Narration:

The name, and hence the type, of the domain] is UoD
The UoD domain can be briefly characterized by ...

Formalization:

type UoD

¹²A remark, as those of footnotes 10 on the preceding page and 11 on the facing page, likewise applies here, to cover Sect. 2.8 (pages 30–36)

¹³Cartesian after the French philosopher, mathematician, scientist René Descartes (1596–1650)

2.7.1.2 Solid and Fluid Endurants

Given then that there are endurants we now postulate that they are either [mutually exclusive] *solid* (i.e., discrete) or *fluid*.

Ontological Choice: Here we [seem to] make a practical choice, not one based on a philosophical argument, one of logical necessity, but one based on empirical evidence. It is possible for endurants to either be solid or fluid; and here we shall not consider the case where solid [fluid] endurants, due to being heated [cooled], enters a fluid state [or vice versa] ■

2.7.1.2.1 Discrete or Solid Endurants.

Characterization 6 *Discrete or Solid Endurants:* By a *solid* [or *discrete*] endurant we shall understand an endurant which is separate, individual or distinct in form or concept, or, rephrasing: have ‘body’ [or magnitude] of three-dimensions: length, breadth and depth [127] [*OED, Vol. II, pg. 2046*] ■

Example 7 *Solid Endurants:* Examples of solid endurants are *wells, pipes, valves, pumps, forks, joins* and *sinks* of pipelines. [These units may, however, and usually will, contain fluids, e.g., oil, gas or water] ■

We introduce the informal presentation language predicate `is_solid` to hold for endurant *e* if `is_solid(e)` holds.

2.7.1.2.2 Fluids.

Characterization 7 *Fluid Endurants:* By a *fluid endurant* we shall understand an endurant which is prolonged, without interruption, in an unbroken series or pattern; or, rephrasing: a substance (liquid, gas or plasma) having the property of flowing, consisting of particles that move among themselves [127] [*OED, Vol. I, pg. 774*] ■

Example 8 *Fluid Endurants:* Examples of fluid endurants are: *water, oil, gas, compressed air, smoke* ■

Fluids are otherwise liquid, or gaseous, or plasmatic, or granular¹⁴, or plant products, i.e., chopped sugar cane, threshed, or otherwise¹⁵, et cetera. Fluid endurants will be analyzed and described in relation to solid endurants, viz. their “containers”.

We introduce the informal presentation language predicate `is_fluid` to hold for endurant *e* if `is_fluid(e)` holds.

2.7.1.3 Parts and Living Species Endurants

Given then that there are solid endurants we now postulate that they are either [mutually exclusive] *parts* or *living species*.

Ontological Choice: With Sørlander, [172, *Sect. 5.7.1, pages 71–72*] we reason that one can distinguish between parts and living species ■

¹⁴ This is a purely pragmatic decision. “Of course” sand, gravel, soil, etc., are not fluids, but for our modelling purposes it is convenient to “compartmentalise” them as fluids!

¹⁵ See footnote 14.

2.7.1.3.1 Parts.

Characterization 8 Parts: The non-living species solids are what we shall call parts ■

Parts are the “work-horses” of man-made domains. That is, we shall mostly be concerned with the analysis and description of endurants into parts.

Example 9 Parts: Example 7, of solids, is an example of parts ■

We introduce the informal presentation language predicate `is_part` to hold for solid endurants `e` if `is_part(e)` holds.

We distinguish between atomic and compound parts.

Ontological Choice: It is an empirical fact that parts can be composed from parts. That possibility exists. Hence we can [philosophy-wise] reason likewise ■

2.7.1.3.1.1 Atomic Parts.

Characterization 9 Atomic Part: By an *atomic part* we shall understand a part which the domain analyzer considers to be indivisible in the sense of not meaningfully consist of sub-parts ■

Example 10 Atomic Parts: Examples of atomic parts are: hubs, H, i.e., street intersections; links, L, i.e., the stretches of roads between two neighbouring hubs; and automobiles, A:

type H, L, A ■

We introduce the informal presentation language predicate `is_atomic` to hold for parts `p` if `is_atomic(p)` holds.

2.7.1.3.1.2 Compound Parts.

Characterization 10 Compound Part: Compound parts are those which are observed to [potentially] consist of several parts ■

Example 11 Compound Parts: An example of a compound parts is: a road net consisting of a set of hubs, i.e., street intersections or “end-of-streets”, and a set of links, i.e., street segments (with no contained hubs), is a Cartesian compound; and the sets of hubs and the sets of links are part set compounds ■

We introduce the informal presentation language predicate `is_compound` to hold for parts `p` if `is_compound(p)` holds.

We, pragmatically, distinguish between Cartesian product- and set-oriented parts.

Ontological Choice: The Cartesian versus set parts is an empirical choice. It is not justified in terms of philosophy, but in terms of mathematics – of mathematical expediency! ■

2.7.1.3.1.3 Cartesians. Cartesians are product-like types – and are named after the French philosopher, scientist and mathematician René Descartes (1596–1640) [Wikipedia].

Characterization 11 Cartesians: Cartesian parts are those compound parts which are observed to consist of two or more distinctly sort-named endurants (solids or fluids) ■

Example 12 Cartesians: Road Transport: A road transport, `rt:RT`, is observed to consist of an aggregate of a road net, `rn:RN`, and a set of automobiles, `SA`, where the road net is observed, i.e., abstracted, as a Cartesian of a set of hubs, `ah:AH`, i.e., street intersections (or specifically designated points segmenting an otherwise “straight” street into two such), and a set of links, `al:AL`, i.e., street segments between two “neighbouring” hubs.

type

RT, RN, SA, AH = H-set, AL = L-set

value

obs_RN: RT → RN, **obs_SA**: RT → SA,, **obs_AH**: RN → AH, **obs_AL**: RN → AL ■

We introduce the informal presentation language predicate `is_Cartesian` to hold for compound parts `p` if `is_Cartesian(p)` holds.

Once a part, say `p:P`, has been analyzed into a Cartesian, we inquire as to the type names of the endurants¹⁶ of which it consists. The inquiry: `record_Cartesian_part_type_names(p:P)`, we decide, then yields the type of the constituent endurants.

Schema 1 *record-Cartesian-part-type-names***value**

`record_Cartesian_part_type_names`: P → T-set
`record_Cartesian_part_type_names(p)` as { $\eta E_1, \eta E_2, \dots, \eta E_n$ } ■

Here T is the **name** of the type of all type names, and ηE_i is the **name** of type E_i .

Please note the novel introduction of type names as values. Where a type identifier, say T, stands for, denotes, a class of values of that type, ηT denotes the name of type T.

Please also note that `record_Cartesian_part_type_names` is not a description language construct. It is an analysis language, i.e., an informal natural language, here English, construct. As such it is being used by the domain analyzer cum describer who “applies” in to an observed endurant and notes down, in her mind or jots it on a scratch of paper, her decision as to appropriate [new] type names.

Example 13 *Cartesian Parts*: The Cartesian parts of a road transport, `rt:RT`, is thus observed to consists of

- an aggregate of a road net, `rn:RN`, and
- an aggregate set of automobiles, `sa:SA`:

that is:

- `record_Cartesian_part_type_names(rt:RT) = { $\eta RN, \eta SA$ }`

where the type name ηRT was – and the type names ηRN and ηSA are – coined, i.e., more-or-less freely chosen, by the domain analyzer cum describer ■

2.7.1.3.1.4 Part Sets.

Characterization 12 *Part Sets*: Part sets are those compound parts which are observed to consist of an indefinite number of zero, one or more parts ■

We introduce the informal presentation language predicate `is_part_set` to hold for compound parts `e` if `is_part_set(e)` holds.

Once a part, say `e:E`, has been analyzed into a part set we inquire as to the set of parts and their type of which it consists. The inquiry: `record_part_set_part_type_names`, we decide, then yields the (single) type of the constituent parts.

Schema 2 *record-part-set-part-type-names*

¹⁶We emphasize that the observed elements of a Cartesian part may be both solids, at least one, and fluids.

value

```
record_part_set_part_type_names: E →  $\mathbb{TP}_s \times \mathbb{TP}$ 
record_part_set_part_type_names(e:E) as ( $\eta P_s, \eta P$ ) ■
```

Here the name of the value, e , and the type names ηP_s and ηP are coined, i.e., more-or-less freely chosen, by the domain analyzer cum describer ■

Please also note that `record_part_set_part_type_names` is not a description language construct. It is an analysis language, i.e., an informal natural language, here English, construct. As such it is being used by the domain analyzer cum describer who “applies” in to an observed endurant and notes down, in her mind or jots it on a scratch of paper, her decision as to appropriate [new] type names.

Example 14 Part Sets: Road Transport: The road transport contains a set of automobiles. The part set type name has been chosen to be `SA`. It is then determined (i.e., analyzed) that `SA` is a set of Automobile of type `A`

- `record_part_set_part_type_names(sa:SA) = ($\eta A_s, \eta A$)` ■

2.7.1.4 Compound Observers.

Once the domain analyzer cum describer has decided upon the names of atomic and compound parts, `obs_erver` functions can be applied to Cartesian and part set, $e:E$, parts:

Schema 3 Describe-Cartesians-and-Part-Set-Parts

value

```
let { $\eta P_1, \eta P_2, \dots, \eta P_n$ } = record_Cartesian_part_type_names(e:E) in
```

“**type**

```
P1, P2, ..., Pn;
```

value

```
obs_P1: E→P1, obs_P2: E→P2, ...n obs_Pn: E→Pn ”
```

[respectively:]

```
let ( $\eta P_s, \eta P$ ) = record_part_set_part_type_names(e:E) in
```

“**type**

```
P, Ps = P-set,
```

value

```
obs_Ps: E→Ps ”
```

end end ■

The “...” texts are the RSL texts “generated”, i.e., written down, by the domain describer. They are *domain model specification units*. The “surrounding” RSL-like texts are not written down as phrases, elements, of the domain description. They are elements of the development of domain models. We have introduced a core domain modelling tool the `obs_...` observer function, one to be “applied” mentally by the domain describer, and one that appears in (AMoL) domain descriptions The `obs_...` observer function is “applied” by the domain describer, it is not a computable function.

Please also note that `Describe-Cartesians-and-Part-Set-Parts` schema, 3, is not a description language construct. It is an analysis language, i.e., an informal natural language, here English, construct. As such it is being used by the domain analyzer cum describer who “applies” in to an observed endurant and notes down, but now in a final form, elements, that is *domain description units*.

A major step of the development of domain models has now been presented: that of the analysis & description of the external qualities of domains.

Schema ?? on page ?? is the first manifestation of the domain analysis & description method leading to actual domain description elements.

From unveiling a *science of domains* we have “arrived” at an *engineering of domain descriptions*.

2.7.1.5 Example Domain Models: External Qualities

These are now the models of external qualities of domains illustrated in Part V’s Chapters E–P:

• Road Transport	Sect. E.2, pp 172–174	• Container Terminals	Sect. K.5.1, pp 317–321
• Rail Systems	Sect. F.1.1, pp 192–194	• Document Systems	Sect. M.3, pp 365–365
• Credit Cards	Sect. G.2.1, pp 212–212	• Swarms of Drones	Sect. N.2.1.1, pp 389–390
• Market Systems	Sect. H.3.1, pp 229–232	• Assembly Lines	Sect. O.2.2.1.7, pp 428–429
• Pipelines	Sect. I.1, pp 262–263	• Nuclear Power Plants	Sect. P.3.1.1, pp 459–463
• Shipping	Sect. J.2, pp 283–286		

2.7.1.6 States.

Characterization 13 States: By a *state* we shall mean any subset of the parts of a domain ■

Example 15 Road Transport State:

variable

$hs:AH := \mathbf{obs_AH}(\mathbf{obs_RN}(rt)),$
 $ls:AL := \mathbf{obs_AL}(\mathbf{obs_RN}(rt)),$
 $as:SA := \mathbf{obs_SA}(rt),$
 $\sigma:(H|L|A)\text{-set} := hs \cup ls \cup as$ ■

We have chosen to model domain states as **variables** rather than as **values**. The reason for this is that the values of monitorable, including biddable part attributes¹⁷ can change, and that domains are often extended and “shrunk” by the addition, respectively removal of parts:

Example 16 Road Transport Development: adding or removing hubs, links and automobiles ■

We omit coverage of the aspect of bidding changes to monitorable part attributes.

2.7.1.7 Validity of Endurant Observations.

We remind the reader that the **obs_**erver functions, as all later such functions: **uid_**-, **mereo_**- and **attr_**-functions, are applied by humans and that the outcome of these “applications” is the result of human choices, and possibly biased by inexperience, taste, preference, bias, etc. How do we know whether a domain analyzer & describer’s description of domain parts is valid? Whether relevantly identified parts are modeled reasonably wrt. being atomic, Cartesians or part sets Whether all relevant endurants have been identified? Etc. The short answer is: we never know. Our models are conjectures and may be refuted [147]. A social process of peer reviews, by domain stakeholders and other domain modelers is needed – as may a process of verifying¹⁸ properties of the domain description held up against claimed properties of the (real) domain.

2.7.1.8 Summary of Analysis Predicates.

Characterizations 6–12 imply the following analysis predicates (Defn.: δ , page π):

¹⁷The concepts of monitorable, including biddable part attributes is treated in Sect. 2.7.2.3.2.

¹⁸testing, model checking and theorem proving

- **Endurant Ontology:**
 - **is_entity**, $\delta 6 \pi 15$
 - **is_entity**, $\delta 6 \pi 15$
 - **is_entity**, $\delta 6 \pi 15$
 - **is_endurant**, $\delta 2 \pi 15$
 - **is_perdurant**, $\delta 3 \pi 15$
 - **is_solid**, $\delta 6 \pi 18$
 - **is_fluid**, $\delta 7 \pi 18$
 - **is_part**, $\delta 8 \pi 19$
 - **is_atomic**, $\delta 9 \pi 19$
 - **is_compound**, $\delta 10 \pi 19$
 - **is_Cartesian**, $\delta 11 \pi 19$
 - **is_part_set**, $\delta 12 \pi 20$

We remind the reader that the above predicates represent “formulas” in the presentation, **not** the description, language. They are not **AMoL** clauses. They are in the mind of the domain analyzers cum describers. They are “executed” by such persons. Their result, whether **true**, **false** or **chaos**¹⁹, are noted by these persons and determine their next step of domain analysis.

2.7.2 Internal Qualities – Intangibles

The previous section has unveiled an ontology of the external qualities of endurants. The unveiling consisted of two elements: a set of analysis predicates, predicates 6–12, and analysis functions, schemas 1–2, and a pair of description functions, schema ?? on page ??

The application of description functions result in **AMoL** text.

That text conveys certain properties of domains: that they consists of such-and-such endurants, notably parts, and that these endurants “derive” from other endurants. But the **AMoL** description texts do not “give flesh & blood” to these endurants. Questions like: ‘*what are their spatial extents?*’, ‘*how much do they weigh?*’, ‘*what colour do they have?*’, et cetera, are left unanswered. In the present section we shall address such issues. We call them *internal qualities*.

Characterization 14 Internal Qualities: Internal qualities are those properties [of endurants] that do not occupy *space* but can be measured or spoken about ■

Example 17 Internal qualities: Examples of internal qualities are the *unique identity* of a part, the *mereological relation* of parts to other parts, and the *endurant attributes* such as temperature, length, colour, etc. ■

This section therefore introduces a number of domain description tools:

- **uid_**: the unique identifier observer of parts;
- **mereo_**: the mereology observer of parts;
- **attr_**: (zero,) one or more attribute observers of endurants; and
- **attributes_**: the attribute query of endurants.

2.7.2.1 Unique Identity.

Ontological Choice: We postulate that separately discernible parts have unique identify. The issue, really, is a philosophical one. We refer to [58, *Sects. 2.2.2.3–2.2.2.4, pages 14–15*] for a discussion of the existence and uniqueness of entities ■

Characterization 15 Unique Identity: A unique identity is an immaterial property that distinguishes any two *spatially* distinct solids²⁰ ■

¹⁹The outcome of applying an analysis predicate of the prescribed kind may be **chaos** if the prerequisites for its application does not hold.

The unique identity of a part p of type P is obtained by the postulated observer \mathbf{uid}_P :

Schema 4 *Describe-Unique-Identity-Part-Observer*

```

“type
  P,PI
value
   $\mathbf{uid}_P: P \rightarrow PI$ ” ■

```

Here PI is the type of the unique identifiers of parts of type P .

Example 18 *Unique Road Transport Identifiers*: The unique identifiers of a road transport, $rt:RT$, consists of the unique identifiers of the

- road transport – $rti:RTI$,
- (Cartesian) road net – $rni:RNI$,
- (set of) automobiles – $sa:SAI$,
- automobile, $ai:AI$,
- (set of) hubs, $hai:AHl$,
- (set of) links, $lai:LAI$,
- hub, $hi:HI$, and
- link, $li:LI$,

where the type names are all coined, i.e., more-or-less freely chosen, by the domain analyzer cum describer – though, as You can see, these names were here formed by “suffixing” Is to relevant part names ■

We have thus introduced a core domain modelling tool the \mathbf{uid}_{\dots} observer function, one to be “applied” mentally by the domain describer, and one that appears in (AMoL) domain descriptions. The \mathbf{uid}_{\dots} observer function is “applied” by the domain describer, it is not a computable function.

2.7.2.1.1 Uniqueness of Parts No two parts have the same unique identifier.

Example 19 *Road Transport Uniqueness*:

variable

```

 $hs_{uids}:HI\text{-set} := \{ \mathbf{uid}_H(h) \mid h:H \bullet u \in \sigma \}$ 
 $ls_{uids}:LI\text{-set} := \{ \mathbf{uid}_L(l) \mid l:L \bullet u \in \sigma \}$ 
 $as_{uids}:AI\text{-set} := \{ \mathbf{uid}_A(a) \mid a:A \bullet u \in \sigma \}$ 
 $\sigma_{uids}:(HI|LI|AI)\text{-set} := \{ \mathbf{uid}_{-(H|L|A)}(u) \mid u:(H|L|A) \bullet u \in \sigma \}$ 

```

axiom

□ $\mathbf{card} \sigma = \mathbf{card} \sigma_{uids}$ ■ For σ see Sect. 2.7.1.6 on page 22.

We have chosen, for the same reason as given in Sect. 2.7.1.6, to model a unique identifier state. The □ [always] prefix in the **axiom** then expresses that changes of parts or addition of parts to and deletions of parts from the domain shall maintain their uniqueness over time (i.e., always).

2.7.2.1.2 Example Domain Model Unique Identifiers: These are now the models of unique identification of domain parts illustrated in Part V’s Chapters E–P:

• Road Transport	Sect. E.3.1, pp 174–176	• Pipelines	Sect. I.2.1, pp 263–264
• Rail Systems	Sect. F.1.2.1, pp 194–195	• Shipping	Sect. J.3.1, pp 286–288
• Credit Cards	Sect. G.2.2.1, pp 213–213	• Container Terminals	Sect. K.5.3, pp 322–323
• Market Systems	Sect. H.4.1, pp 232–233	• Document Systems	Sect. M.4, pp 365–365

²⁰For pragmatic reasons we do not have to speculate as to whether “bodies” of fluids can be ascribed unique identity. The pragmatics is that we, in our extensive modelling experiments have not found a need for such ascription!

- **Swarms of Drones** Sect. N.2.2, pp 390–392
- **Assembly Lines** Sect. O.2.2.2.1, pp 429–435
- **Nuclear Power Plants** Sect. P.3.1.2.1, pp 463–465

2.7.2.2 Mereology.

The concept of mereology is due to the Polish mathematician, logician and philosopher Stanisław Leśniewski (1886–1939) [47, 54, 68, 135, 179].

Characterization 16 Mereology: Mereology is a theory of [endurant] part-hood relations: of the relations of an [endurant] parts to a whole and the relations of [endurant] parts to [endurant] parts within that whole ■

Ontological Choice: Stanisław Leśniewski was not satisfied with Bertrand Russell’s “repair” of Gottlob Frege’s axiom systems for set theory. Instead he put forward his axiom system for, as he called it, mereology. Both as a mathematical theory and as a philosophical reasoning ■

Example 20 Mereology: Examples of mereologies are that a link is topologically *connected* to exactly one or, usually, two specific hubs, that hubs are *connected* to zero, one or more specific links, and that links and hubs are *open* to the traffic of specific subsets of automobiles ■

Mereologies can be expressed in terms of unique identifiers.

Example 21 Mereology Representation: For our ‘running road transport example’ the mereologies of links, hubs and automobiles can thus be expressed as follows:

- **mereo_L(l)** = {hi',hi''} where hi,hi',hi'' are the unique identifiers of the hubs that the link connects, i.e., are in hs_{uids} ;
- **mereo_H(h)** = {li₁,li₂,...,li_n} where li₁,li₂,...,li_n are the unique identifiers of the links that are imminent upon (i.e., emanates from) the hub, i.e., are in ls_{uids} ; and
- **mereo_A(a)** = {ri₁,ri₂,...,ri_m} where ri₁,ri₂,...,ri_m are unique identifiers of the road (hub and link) elements that make up the road net, i.e., are in $hs_{uids} \cup ls_{uids}$ ■

Once the unique identifiers of all parts of a domain has been described we can analyse and describe their mereologies. The inquiry: **mereo_P(p)** yields a mereology type (name), say PMer, and its description²¹:

Schema 5 Describe-Mereology

```

“type
  PMer =  $\mathcal{M}(PI_1,PI_2,\dots,PI_m)$ 
value
  mereo_P: P → PMer
axiom
   $\mathcal{A}(pm:PMer)$ ” ■

```

where $\mathcal{M}(PI_1,PI_2,\dots,PI_m)$ is a type expression over unique identifier types of the domain; **mereo_P** is the mereology observer function for parts p:P; and $\mathcal{A}(pm:PMer)$ is an axiom that secures that the unique identifiers of any part are indeed of parts of the domain.

2.7.2.2.1 Example Domain Model Mereologies: These are now the models of mereologies of domain parts illustrated in Part V’s Chapters E–P:

²¹Cf. Sect. 2.7.1.4

• Road Transport	Sect. E.3.2, pp 176–178	• Container Terminals	Sect. K.5.5, pp 325–330
• Rail Systems	Sect. F.1.2.2, pp 195–200	• Document Systems	Sect. M.5, pp 366–366
• Credit Cards	Sect. G.2.2.2, pp 213–215	• Swarms of Drones	Sect. N.2.3, pp 392–395
• Market Systems	Sect. H.4.2, pp 233–236	• Assembly Lines	Sect. O.2.2.2.2, pp 435–445
• Pipelines	Sect. I.2.2, pp 264–265	• Nuclear Power Plants	Sect. P.3.1.2.2, pp 465–472
• Shipping	Sect. J.3.2, pp 288–292		

2.7.2.3 Attributes.

Attributes are what finally gives “life” to endurants: The external qualities “only” named and gave structure to their atomic or compound types. The internal qualities of uniqueness and mereology are intangible quantities. The internal quality of attributes gives “flesh & blood” to endurants: they let us express endurant properties that we can more easily, i.e., concretely, relate to.

2.7.2.3.1 General

Characterization 17 Attributes: Attributes are properties of endurants that can be measured either physically (by means of length (ruler) and spatial quantity measuring equipment, electronically, chemically, or otherwise) or can be objectively spoken about ■

Ontological Choice: First some empirical observation: in reasoning about “the world around us” we express its properties in terms of predicates. These predicates, for example: “*that building’s wall is red*”, *building* refers to an endurant part whereas *wall* and *red* refers to attributes. Now the “rub”: endurant attributes is what give “*flesh & blood*” to domains ■²²

Attributes are of types and, accordingly have values.

We postulate an informal domain analysis function, `record_attribute_type_names`: The domain analyzer, in observing a part, $p:P$, analyzes it into the set of attribute names of parts $p:P$

Schema 6 *record-attribute-type-names*

value

`record_attribute_type_names`: $P \rightarrow \eta\mathbb{T}\text{-set}$
`record_attribute_type_names(p:P)` as $\eta\mathbb{T}\text{-set}$ ■

Example 22 Road Net Attributes, I: Examples of attributes are: hubs have states, $h\sigma:H\Sigma$: the set of pairs of link identifiers, $(f|l,li)$, of the links *from* and *to* which automobiles may enter, respectively leave the hub; and hubs have state spaces, $h\omega:H\Omega$: the set of hub states “signaling” which states are open/closed, i.e., **green/red**; links that have lengths, LEN ; and automobiles have road net positions, $APos$, either *at a hub*, atH , or *on a link*, onL , some fraction, $f:Real$, down a link, identified by li , from a hub, identified by fhi , towards a hub, identified by thi . Hubs and links have *histories*: time-stamped, chronologically ordered sequences of automobiles entering and leaving links and hubs, with automobile histories similarly recording hubs and links entered and left.

type

$H\Sigma = (LI \times LI)\text{-set}$
 $H\Omega = H\Sigma\text{-set}$
 $LEN = Nat\ m$
 $APos = atH \mid onL$
 $atH :: HI$
 $onL :: LI \times (fhi:HI \times f:Real \times thi:HI)$

$HHis, LHis = (TIME \times AI)^*$
 $AHis = (TIME \times (HI \mid LI))^*$

value

$attr_H\Sigma: H \rightarrow H\Sigma$
 $attr_H\Omega: H \rightarrow H\Omega$
 $attr_LEN: L \rightarrow LEN$
 $attr_APos: A \rightarrow APos$

²²**Editorial remark:** I am not yet satisfied with this reasoning. The issue is: to force the concept of attributes to be justified philosophically, as an inevitable element of any world description, and not being forced upon us solely from empirical evidence.

<pre> attr_HHis: H → HHis attr_LHis: L → LHis attr_AHis: A → AHis </pre>	<pre> axiom ∀ (li,(fhi,f,thi)):onL • 0<f<1 ∧li∈lsuids∧{fhi,thi}⊆hsuids∧... ■ </pre>
--	--

Schema 7 *Describe-endurant-attributes(e:E)*

```

let {ηA1,ηA2,...,ηAn} = record_attribute_type_names(e:E) in
  “type
    A1, A2, ..., An
  value
    attr_A1: E → A1, attr_A2: E → A2, ..., attr_An: E → An
  axiom
    ∀ a1:A1, a2:A2, ..., an:An:  $\mathcal{A}(a1,a2,...,an)$  ”
end ■

```

2.7.2.3.2 Michael A. Jackson’s Attribute Categories.

2.7.2.3.2.1 A Presentation Michael A. Jackson [116] has suggested a hierarchy of attribute categories: from *static* (`is_static`²³) to *dynamic* (`is_dynamic`²⁴) values – and within the dynamic value category: *inert* values (`is_inert`²⁵), *reactive* values (`is_reactive`²⁶), *active* values (`is_active`²⁷) – and within the dynamic active value category: *autonomous* values (`is_autonomous`²⁸), *biddable* values (`is_biddable`²⁹), and *programmable* values (`is_programmable`³⁰). We postulate informal domain analysis predicates, “performed” by the domain analyzer:

```

value
  is_static,is_autonomous,is_biddable,is_programmable [etc.]: η T → Bool

```

We refer to [116] and [58] [*Chapter 5, Sect. 5.4.2.3*] for details. We summarize Jackson’s attribute categorization in Fig. 2.2.

2.7.2.3.2.2 A Revision We suggest a minor revision of Michael A. Jackson’s attribute categorization, see Fig. 2.3 on the following page.

We single out the inert from the ontology of Fig. 2.2 on the next page

Inert attributes seem to be “set externally” to the endurant. So we now distinguish between `is_external` and `is_internal` dynamic attributes.

This distinction has [pragmatic] consequences for how we treat arguments of the behaviours of parts, cf. Sect. 2.8.5.1 (page 32).

Example 23 *Road Net Attributes, II*: The link length and hub state space attributes are static, hub states and automobile positions programmable. Automobile speed and acceleration attributes, which we do not model, are monitorable ■

The attributes categorization determines, in the next major section on perdurants, the treatment of hub, link and automobile behaviours.

²³`static`: values are constants, cannot change

²⁴`dynamic`: values are variable, can change

²⁵`inert`: values can only change as the result of external stimuli where these stimuli prescribe new values

²⁶`reactive`: values, if they vary, change in response to external stimuli, where these stimuli either come from outside the domain of interest or from other endurants.

²⁷`active`: values can change (also) on their own volition

²⁸`autonomous`: values change only “on their own volition”; the values of an autonomous attributes are a “law onto themselves and their surroundings”.

²⁹`biddable`: values are prescribed but may fail to be observed as such

³⁰`programmable`: values can be prescribed

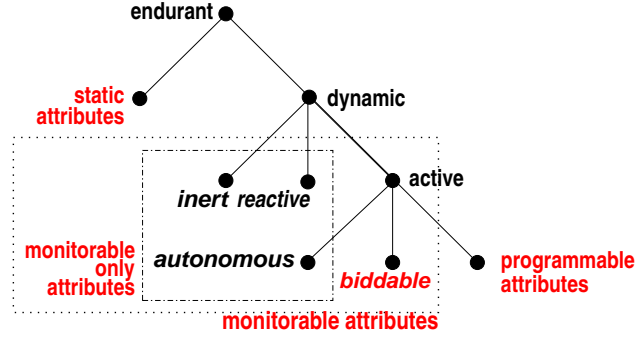


Figure 2.2: Michael Jackson's Attribute Categories

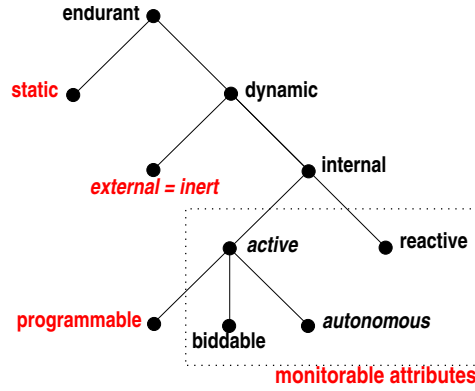


Figure 2.3: Revised Attribute Categories

2.7.2.3.3 Analytic Attribute Extraction Functions: For later purpose we need characterize three specific attribute category extraction functions: `static_attributes`, `monitable_attributes`, and `programmable_attributes`:

value

```
p:P
tns = record_attribute_type_names(p)
```

```
static_attributes:  $\eta T$ -set  $\rightarrow$   $\eta T$ -set
static_attributes(tns)  $\equiv$  {  $\eta tn \mid \eta tn:\eta T \cdot \eta tn \in tns \wedge is\_static(tn)$  }
```

```
inert_attributes:  $\eta T$ -set  $\rightarrow$   $\eta T$ -set
inert_attributes(tns)  $\equiv$  {  $\eta tn \mid \eta tn:\eta T \cdot \eta tn \in tns \wedge is\_inert(tn)$  }
```

```
monitable_attributes  $\eta T$ -set  $\rightarrow$   $\eta T$ -set
monitable_attributes(tns)  $\equiv$  {  $\eta tn \mid \eta tn:\eta T \cdot \eta tn \in tns \wedge is\_monitable(tn)$  }
```

```
programmable_attributes  $\eta T$ -set  $\rightarrow$   $\eta T$ -set
programmable_attributes(tns)  $\equiv$  {  $\eta tn \mid \eta tn:\eta T \cdot \eta tn \in tns \wedge is\_programmable(tn)$  }
```

```
is_monitable: T  $\rightarrow$  Bool
is_monitable(t)  $\equiv$   $\sim is\_static(t) \wedge \sim is\_inert(t) \wedge \sim is\_programmable(t)$ 
```

Please be reminded that these functions are informal. They are part of the presentation language. Do not be confused by their AMoL-like appearance.

2.7.2.3.4 Example Domain Model Attributes: These are now the models of attributes of domain durants illustrated in Part V's Chapters E–P:

• Road Transport	Sect. E.3.3, pp 178–181	• Container Terminals	Sect. K.5.6, pp 330–335
• Rail Systems	Sect. F.1.2.3, pp 200–203	• Document Systems	Sect. M.6.6, pp 366–367
• Credit Cards	Sect. G.2.2.4, pp 215–215	• Swarms of Drones	Sect. N.2.4, pp 395–400
• Market Systems	Sect. H.4.3, pp 236–242	• Assembly Lines	Sect. O.2.2.2.3, pp 445–449
• Pipelines	Sect. I.2.4, pp 267–272	• Nuclear Power Plants	Sect. P.3.1.2.3, pp 472–479
• Shipping	Sect. J.3.3, pp 292–294		

2.7.3 Intentional Pull

Ontological Choice: In [170, pages 167–168] Sørlander argues wrt. “*how can entities be the source of forces ?*” and thus reasons for *gravitational pull*. That same kind of reasoning, with proper substitution of terms, leads us to the concept of *intentional pull* ■

Two or more parts of different sorts, but with overlapping sets of intents³¹ may exert an intentional “pull” on one another. This *intentional “pull”* may take many forms. Let $p_x : X$ and $p_y : Y$ be two parts of *different sorts* (X, Y), and with *common intent*, ι . *Manifestations* of these, their common intent must somehow be *subject to constraints*, and these must be *expressed predicatively*.

When a compound artifact models “itself” as put together with a number of other durants then it does have an intentionality and the components’ individual intentionalities does, i.e., shall relate to that.

Example 24 Road Transport Intentionality: *Automobiles* include the *intent* of ‘transport’, and so do *hubs* and *links*. *Manifestations* of “transport” are reflected in *hubs*, *links* and *automobiles* having the *history* attribute. The *intentional “pull”* of these manifestations is this: For every automobile, if it records being in some hub or on some link at time τ , then the designated hub, respectively link, records exactly that automobile; and vice versa: for every hub [link], if it records the visit of some automobile at time τ , then the designated automobile records exactly that hub [link]. We leave the formalization of the above to the reader ■

Example 25 Double-entry Bookkeeping: Another example of intentional “pull” is that of double-entry bookkeeping. Here the income/expense ledger must balance the actives/passives ledger ■

Example 26 The Henry George Theorem.: The Henry George theorem states that under certain conditions, aggregate spending by government on public goods will increase aggregate rent based on land value (land rent) more than that amount, with the benefit of the last marginal investment equaling its cost ■^{32,33}

³¹Intent: purpose; God-given or human-imposed!

³²Stiglitz, Joseph (1977). “The Theory of Local Public Goods”. In Feldstein, M.S.; Inman, R.P. (eds.). *The Economics of Public Services*. Palgrave Macmillan, London. pp. 274333. doi:10.1007/978-1-349-02917-4_12. ISBN 978-1-349-02919-8.

³³Henry George (September 2, 1839 – October 29, 1897) was an American political economist and journalist. His writing was immensely popular in 19th-century America and sparked several reform movements of the Progressive Era. He inspired the economic philosophy known as Georgism, the belief that people should own the value they produce themselves, but that the economic value of land (including natural resources) should belong equally to all members of society. George famously argued that a single tax on land values would create a more productive and just society.

2.7.4 Summary of Endurants

We have completed our treatment of endurants. That treatment was based on an ontology for the observable phenomena of domains – such as we have delineated the concept of domains. The treatment was crucially based on an ontology for the structure of domain phenomena, and, in a sense, “alternated” between analysis predicates, analysis functions, and description functions. The question of whether the postulated ontology, cf. Fig. 2.1, is the only possible ontology for the analysis & description of domains, we shall presently defer.

2.8 Perdurant Concepts

The main contribution of this section is that of *transcendentally deducing* perdurants from endurant parts, in particular *behaviours* “of” parts.

Major perdurants are those of actions, events and behaviours with behaviours generally being sets of sequences of **actions**, **events** and **behaviours**.

2.8.1 “Morphing” Parts into Behaviours

As already indicated we shall transcendentally deduce (perdurant) behaviours from those (endurant) parts which we, as domain analyzers cum describers, have endowed with all three kinds of internal qualities: unique identifiers, mereologies and attributes. We shall use the CSP [110] constructs of **AMoL** (derived from RSL [100]) to model concurrent behaviours.

2.8.2 Transcendental Deduction

Characterization 18 *Transcendental*: By **transcendental** we shall understand the philosophical notion: **the a priori or intuitive basis of knowledge, independent of experience** ■

A priori knowledge or intuition is central: By *a priori* we mean that it not only precedes, but also determines rational thought.

Characterization 19 *Transcendental Deduction*: By a **transcendental deduction** we shall understand the philosophical notion: **a transcendental “conversion” of one kind of knowledge into a seemingly different kind of knowledge** ■

Example 27 *Transcendental Deductions – Informal Examples*: We give some intuitive examples of transcendental deductions. They are from the “domain” of programming languages. There is the syntax of a programming language, and there are the programs that supposedly adhere to this syntax. Given that, the following are now transcendental deductions.

The software tool, **a syntax checker**, that takes a program and checks whether it satisfies the syntax, including the statically decidable context conditions, i.e., the *statics semantics* – such a tool is one of several forms of transcendental deductions.

The software tools, **an automatic theorem prover** and **a model checker**, for example SPIN [113], that takes a program and some **theorem**, respectively a **Promela** statement, and proves, respectively checks, the program correct with respect the theorem, or the statement.

A **compiler** and an **interpreter** for any programming language.

Yes, indeed, any **abstract interpretation** [82, 83] reflects a transcendental deduction: firstly, these examples show that there are many transcendental deductions; secondly, they show that there is no single-most preferred transcendental deduction ■

Ontological Choice: So this, then, is, in a sense, our “final” ontological choice: that of transcendentally deduce behaviours from, or of, parts ■

2.8.3 Actors – A Synopsis

This section provides a summary overview.

Characterization 20 Actors: An actor is anything that can initiate an **action, event** or **behaviour** ■

2.8.3.1 Action.

Characterization 21 Actions: An action is a function that can purposefully change a state ■

Example 28 Road Net Actions: These are some road transport actions: an automobile leaving a hub, entering a link; leaving a link, entering a hubs; entering the road net; and leaving the road net ■

2.8.3.2 Event.

Characterization 22 Events: An event is a function that surreptitiously changes a state ■

Example 29 Road Net Events: These are some road net events: The blocking of a link due to a mud slide; the failing of a hub traffic signal due to power outage; an automobile failing to drive; and the blocking of a link due to an automobile accident ■

We shall not formalize events.

2.8.3.3 Behaviour.

Characterization 23 Behaviours: Behaviours are sets of sequences of actions, events and behaviours ■

Concurrency is modeled by the *sets* of sequences. Synchronization and communication of behaviours are effected by CSP *output/inputs*: $ch[\{i,j\}]!value/ch[\{i,j\}]?$.

Example 30 Road Net Traffic: Road net traffic can be seen as a behaviour of all the behaviours of automobiles, where each automobile behaviour is seen as sequence of start, stop, turn right, turn left, etc., actions; of all the behaviours of links where each link behaviour is seen as a set of sequences (i.e., behaviours) of “following” the link entering, link leaving, and movement of automobiles on the link; of all the behaviours of hubs (etc.); of the behaviour of the aggregate of roads, viz. *The Department of Roads*, and of the behaviour of the aggregate of automobiles, viz. *The Department of Vehicles*.

2.8.4 Channel

Characterization 24 Channel: A channel is anything that allows synchronization and communication of values between behaviours ■

Schema 8 Channel

We suggest the following schema for describing channels:

“channel { $ch[\{ui,uj\}] \mid ui,ij:UI \bullet \dots$ } M

where ch is the describer-chosen name for an array of channels, ui,uj are channel array indices of the unique identifiers, UI , of the chosen domain ■

Example 31 Road Transport Interaction Channel:

channel { $ch[\{ui,uj\}] \mid \{ui,ij\}:(HI|LI|AI)\text{-set} \bullet ui \neq uj \wedge \{ui,uj\} \subseteq \sigma_{uids}$ } M

Channel array ch is indexed by a “pair” of distinct unique part identifiers of the domain. We shall later outline M , the type of the “messages” communicated between behaviours ■

2.8.5 Behaviours

We single out the perdurants of behaviours – as they relate directly to the parts of Sect. 2.7. The treatment is “divided” into three sections.

2.8.5.1 Behaviour Signature.

Schema 9 Behaviour Signature

By the *behaviour signature*, for a part p , we shall understand a pair: the name of the behaviour, B_p , and a function type expression as indicated:

value

$B_p: \text{Uid}_p \rightarrow^{34} \text{Mereo}_p \rightarrow \text{Sta_Vals}_p \rightarrow \text{Inert_Vals}_p \rightarrow \text{Mon_Refs}_p \rightarrow \text{Prgr_Vals}_p \rightarrow \{ \text{ch}[\{i,j\}] \mid \dots \} \text{Unit}$

We explain:

- Uid_p is the type of unique identifiers of part p , $\text{uid}_P(p) = \text{Uid}_p$;
- Mereo_p is the type of the mereology of part p , $\text{mereo}_P(p) = \text{Mereo}_p$;
- Sta_Vals_p is a Cartesian of the type of inert attributes of part p . Given $\text{record_attribute_type_names}(p)$ $\text{static_attributes}(\text{record_attribute_type_names}(p))$ yields Sta_Vals_p ;
- Inert_Vals_p is a Cartesian of the type of static attributes of part p . Given $\text{record_attribute_type_names}(p)$ $\text{inert_attributes}(\text{record_attribute_type_names}(p))$ yields Inert_Vals_p ;
- Mon_Refs_p is a Cartesian of the **attr**_ibute observer functions of the types of monitorable attributes of part p . Given $\text{record_attribute_type_names}(p)$ analysis function $\text{monitorable_attributes}(\text{record_attribute_type_names}(p))$ yields Mon_Vals_p ;
- Prgr_Vals_p is a Cartesian of the type of programmable attributes of part p . Given $\text{record_attribute_type_names}(p)$ analysis function $\text{programmable_attributes}(\text{record_attribute_type_names}(p))$. yields Prgr_Vals_p ;
- $\{ \text{ch}[\{i,j\}] \mid \dots \}$ specifies the channels over which part p behaviours, B_p , may communicate;

and:

- **Unit** is the type name for the () value³⁵ ■

The Cartesian arguments may “degenerate” to the non-Cartesian of no, or just one type identifier, In none, i.e., (), then () may be skipped. If one, e.g., (a), then (a) is listed.

Example 32 Road Transport Behaviour Signatures:

value

```

hub: Hl → MereoH → (HΩ × ...) → (...) → (HHist × ...)
      → {ch[ {uid_H(p), ai} ] | ai: Al • ai ∈ asuid } Unit
link: Ll → MereoL → (LEN × ...) → (...) → (LHist × ...)
      → {ch[ {uid_L(p), ai} ] | ai: Al • ai ∈ asuid } Unit
automobile: Al → MereoA → (...) → (attr_AVel × attr_HAcc × ...) → (APos × AHist × ...)
      → {ch[ {uid_H(p), ri} ] | ri: (Hl | Ll) • ri ∈ hsuid ∪ lsuid } Unit

```

Here we have suggested additional part attributes: monitorable automobile velocity and acceleration, AVel, AAcc, and omitted other attributes ■

³⁴We have Schönfinckel'ed https://en.wikipedia.org/wiki/Moses_Schönfinkel#Further_reading (Curried <https://en.wikipedia.org/wiki/Currying>) the function type

³⁵– You may “read” () as the value yielded by a statement, including a never-terminating function

2.8.5.2 Inert Arguments: Some Examples.

Let us give some examples of inert attributes of automobiles. (i) Driving uphill, on a level road, or downhill, exert some **inert** “drag” or “pull”. (ii) Velocity can be treated as a reactive attribute – but it can be [approximately] calculated on the basis of, for example, these **inert** attributes: drag/pull and accelerator pedal pressure, and the **static** engine power attribute.

2.8.5.3 Behaviour Invocation.

Schema 10 Behaviour Invocation

Behaviours are invoked as follows:

$$\begin{aligned} & \text{“}B_p(\text{uid}_p(p))^{36} \\ & \quad (\text{mereo}_P(p)) \\ & \quad \quad (\text{attr_sta}A_1(p), \dots, \text{attr_sta}A_s(p)) \\ & \quad \quad \quad (\text{attr_inert}A_1(p), \dots, \text{attr_inert}A_i(p)) \\ & \quad \quad \quad \quad (\text{attr_mon}A_1, \dots, \text{attr_mon}A_m) \\ & \quad \quad \quad \quad \quad (\text{attr_prg}A_1(p), \dots, \text{attr_prg}A_p(p))\text{”} \end{aligned}$$

- All arguments are passed *by value*.
- The *uid* value is never changed.
- The *mereology* value is usually not changed.
- The *static attribute* values are fixed, never changed.
- The *inert attribute* values are fixed, but can be updated by receiving explicit input communications.
- The *monitorable attribute* values are functions, i.e., it is as if the “actual” monitorable values are passed *by name*!
- The *programmable attribute* values are usually changed, “updated”, by actions described in the behaviour definition ■

2.8.5.4 Argument References

Within behaviour descriptions, see next section, references are made to the behaviour arguments. References, *a*, to *unique identifier*, *mereology*, *static* and *programmable attribute* arguments yield their value. References, *a*, to *monitorable attribute* arguments also yield their value. This value is an **attr_A** observer function. To yield, i.e., read, the monitorable attribute value this function is applied to that behaviour’s uniquely identified part, p_{uid} , in the global part state, σ . To update, i.e., write, say, to a value v , for the case of a *biddable*, *monitorable* attribute, that behaviour’s uniquely identified part, p_{uid} , in the global part state, σ , shall have part p_{uid} ’s A attribute changed to v – with all other attribute values of p_{uid} unchanged. Common to both the read and write functions is the *retrieve part* function:

1. Given a unique part identifier, pi , assumed to be that of an existing domain part,
2. *retr_part* reads the global [all parts] variable σ to retrieve that part p whose unique part identifier is pi .

³⁶We show the arguments of the invocation on separate lines only for readability. That is: normally we show the invocation arguments as $B(\dots)(\dots)(\dots)(\dots)(\dots)$.

value

2. $\text{retr_part}: PI \rightarrow P$ **read**
2. $\text{retr_part}(pi) \equiv \text{let } p:P \bullet p \in \mathbf{c} \sigma \wedge \text{uid_P}(p)=pi \text{ in } p \text{ end}$
1. **pre:** $\exists p:P \bullet p \in \mathbf{c} \sigma \wedge \text{uid_P}(p)=pi$

You may think of the functions being illustrated in this section, Sect. 2.8.5.4, retr_part , read_A_from_P and update_P_with_A , as “belonging” to the description language, but here suitably expressed for any domain, that is, with suitable substitutions for A and P .

2.8.5.4.1 Evaluation of Monitorable Attributes

3. Let $pi:PI$ be the unique identifier of any part, p , with monitorable attributes, let A be a monitorable attribute of p , and let ηA be the name of attribute A .
4. Evaluation of the [current] attribute A value of p is defined by function read_A_from_P .

value

3. $pi:PI, a:A, \eta A:\eta\mathbb{T}$
4. $\text{read_A_from_P}: PI \times \mathbb{T} \rightarrow \text{read } \sigma A$
4. $\text{read_A}(pi, \eta A) \equiv \text{attr_A}(\text{retr_part}(pi))$

2.8.5.4.2 Update of Biddable Attributes

5. The update of a monitorable attribute A , with attribute name ηA of part p , identified by pi , to a new value **writes** to the global part state σ .
6. Part p is retrieved from the global state.
7. A new part, p' is formed such that p' is like part p :
 - (a) same unique identifier,
 - (b) same mereology,
 - (c) same attributes values,
 - (d) except for A .
8. That new p' replaces p in σ .

value

3. $\sigma, a:A, pi:PI, \eta A:\eta\mathbb{T}$
5. $\text{update_P_with_A}: PI \times A \times \eta\mathbb{T} \rightarrow \text{write } \sigma$
5. $\text{update_P_with_A}(pi, a, \eta A) \equiv$
6. **let** $p = \text{retr_part}(pi)$ **in**
7. **let** $p':P \bullet$
- 7a. $\text{uid_P}(p')=pi$
- 7b. $\wedge \text{mereo_P}(p)=\text{mereo_P}(p')$
- 7c. $\wedge \forall \eta A' \in \text{record_attribute_type_names}(p) \setminus \{\eta A\} \Rightarrow \text{attr_A}'(p)=\text{attr_A}'(p')$
- 7d. $\wedge \text{attr_A}(p')=a$ **in**
8. $\sigma := \mathbf{c} \sigma \setminus \{p\} \cup \{p'\}$
5. **end end**
6. **pre:** $\exists p:P \bullet p \in \mathbf{c} \sigma \wedge \text{uid_P}(p)=pi$

2.8.5.5 Behaviour Description – Examples

Behaviour descriptions rely strongly on CSPs' [110] expressivity. Leaving out some details (–, '...'), and *without* “further ado”, we exemplify.

Example 33 *Automobile Behaviour at Hub*:

9. We abstract automobile behaviour at a Hub (hi).

- (a) Either the automobile remains in the hub,
- (b) or, internally non-deterministically,
- (c) leaves the hub entering a link,
- (d) or, internally non-deterministically,
- (e) stops.

```

9 automobile(ai)(ris)(...)(atH(hi),ahis,_) ≡
9a  automobile_remains_in_hub(ai)(ris)(...)(atH(hi),ahis,_)
9b  []
9c  automobile_leaving_hub(ai)(ris)(...)(atH(hi),ahis,_)
9d  []
9e  automobile_stop(ai)(ris)(...)(atH(hi),ahis,_)

```

10. [9a] The automobile remains in the hub:

- (a) time is recorded,
- (b) the automobile remains at that hub, “idling”,
- (c) informing (“first”) the hub behaviour.

```

10 automobile_remains_in_hub(ai)(ris)(...)(atH(hi),ahis,_) ≡
10a  let τ = record_TIME in
10c  ch[ {ai,hi} ] ! τ ;
10b  automobile(ai)(ris)(...)(atH(hi),⟨(τ,hi)⟩^ahis,_) end

```

11. [9c] The automobile leaves the hub entering link li:

- (a) time is recorded;
- (b) hub is informed of automobile leaving and link that it is entering;
- (c) “whereupon” the vehicle resumes (i.e., “while at the same time” resuming) the vehicle behaviour positioned at the very beginning (0) of that link.

```

11 automobile_leaving_hub(ai)({li}∪ris)(...)(atH(hi),ahis,_) ≡
11a  let τ = record_TIME in
11b  (ch[ {ai,hi} ] ! τ || ch[ {ai,li} ] ! τ) ;
11c  automobile(ai)(ris)(...)(onL(li,(hi,0,_) ),⟨(τ,li)⟩^ahis,_) end
11  pre: [hub is not isolated]

```

The choice of link entered is here expressed (11) as a non-deterministic choice³⁷. One can model the leave hub/enter link otherwise.

12. [9e] Or the automobile “disappears — off the radar” !

```

12 automobile_stop(ai)(ris),(...)(atH(hi),ahis,_) ≡ stop ■

```

2.8.5.5.1 Example Domain Model Behaviours: These are now the models of domain behaviours illustrated in Part V’s Chapters E–P:

³⁷– as indicated by the **pre**- condition: the hub mereology must specify that it is not isolated. Automobiles can never leave isolated hubs.

• Road Transport	Sect. E.4.2.2, pp 185–188	• Container Terminals	Sect. K.6.9, pp 350–360
• Rail Systems	Sect. F.3.2, pp 207–209	• Document Systems	Sect. M.16, pp 373–382
• Credit Cards	Sect. G.3.4, pp 218–221	• Swarms of Drones	Sect. N.4.4, pp 408–415
• Market Systems	Sect. H.6.2, pp 244–258	• Assembly Lines	Sect. O.2.3.3.2, pp 451–451
• Pipelines	Sect. I.3.4, pp 273–276	• Nuclear Power Plants	Sect. P.3.2.3.2, pp 483–488
• Shipping	Sect. J.4.5.2, pp 296–303		

2.8.5.6 Behaviour Initialization.

For every manifest part it must be described how its behaviour is initialized.

Example 34 Road Transport Initialization: We “wrap up” the main example of this paper: We omit treatment of monitorable attributes.

13. Let us refer to the system initialization as an action.
14. All hubs are initialized,
15. all links are initialized, and
16. all automobiles are initialized.

value

13. $rts_initialisation: \mathbf{Unit} \rightarrow \mathbf{Unit}$

13. $rts_initialisation() \equiv$

14. $\parallel \{ \text{hub}(\mathbf{uid_H}(l))(\mathbf{mereo_H}(l))(\mathbf{attr_H}\Omega(l), \dots)(\mathbf{attr_H}\Sigma(l), \dots) \mid h:H \cdot h \in hs \}$

15. $\parallel \parallel \{ \text{link}(\mathbf{uid_L}(l))(\mathbf{mereo_L}(l))(\mathbf{attr_L}\text{LEN}(l), \dots)(\mathbf{attr_L}\Sigma(l), \dots) \mid l:L \cdot l \in ls \}$

16. $\parallel \parallel \{ \text{automobile}(\mathbf{uid_A}(a))(\mathbf{mereo_A}(a))(\mathbf{attr_A}\text{Pos}(a)\mathbf{attr_A}\text{His}(a), \dots) \mid a:A \cdot a \in as \}$

We have here omitted possible monitorable attributes. For hs, ls, as we refer to Sect. 2.7.1.6 ■

2.9 Perspectives

We have summarized a method to be used by [human] domain analyzers cum describers in studying and modelling domains. The next chapter steps “back” to review the description language **AMoL**. Appendix Chapters A–T present a variety of models of “more-or-less” domains! (We refer to pages 79, 81, 169, and 499, for introductions to their “variety”.) Domain models can be developed for either of a number of reasons:

- (i) in order to *understand* a human-artifact domain;
- (ii) in order to *re-engineer the business processes* of a human-artifact domain; or
- (iii) in order to develop *requirements prescriptions* and, subsequently *software application* “within” that domain.

[(ii)] We refer to [103, 104, 117] and [28, *Chapter 19, pages 404–412*] for the concept of *business process engineering*. [(iii)] We refer to [58, *Chapter 9*] for the concept of *requirements engineering*.

Chapter 3

The AMoL Language

Contents

3.1	A Résumé of Domains	39
3.1.1	Endurants	39
3.1.1.1	External Qualities:	39
3.1.1.2	Internal Qualities:	39
3.1.2	Perdurants	39
3.2	Values, Types and Sorts, Axioms	39
3.2.1	Values	39
3.2.2	Types and Sorts	39
3.2.2.1	Two Kinds of Values and Types/Sorts	40
3.2.3	Axioms	40
3.3	Expressions, Statements, Clauses	40
3.4	Specification Units	40
3.4.1	Two Forms of Type Specification Units	41
3.4.2	Value Specification Units	41
3.4.3	Axiom Specification Units	42
3.4.4	Variable Specification Units	42
3.4.5	Channel Specification Units	42
3.4.6	Intentional Pull	42
3.4.7	Domain Initialization	42
3.4.8	Special Forms of Combined Specification Units	42
3.4.8.1	Type and Value Specification Units	43
3.4.8.1.1	Cartesians:	43
3.4.8.1.2	Set Parts:	43
3.4.8.1.3	Unique Identification:	43
3.4.8.1.4	Type, Value and Axiom Specification Units	44
3.4.8.1.4.1	Mereologies:	44
3.4.8.1.4.2	Attributes:	44
3.5	Types and Values	45
3.5.1	Informal Summary of Type Expressions	45
3.5.2	Semi-Formal Summary of Type Expressions and Definitions	45
3.5.3	Informal Meaning of Type Expressions	45
3.5.4	Some Comments of Type Definitions	46
3.6	Expressions	46
3.6.1	Atomic Expressions	46

3.6.1.1	Base Constants:	46
3.6.1.2	Value Identifiers	47
3.6.1.3	TIME and Time Interval Values	47
3.6.1.4	POINT Values	47
3.6.2	Enumerated Expressions	48
3.6.3	Quantified Expressions	48
3.6.4	Definite Expressions	48
3.6.5	Comprehended Expressions	48
3.6.6	Operator/Operand Expressions	49
3.6.6.1	Prefix Operators, o_p	49
3.6.6.2	Infix Operators, o_i	49
3.6.7	The let ... in ... end Expressions	49
3.6.8	Non-function Bindings	50
3.6.9	Patterns	50
3.6.10	Function Bindings	52
3.6.11	Structured Expressions	52
3.7	Statements	52
3.7.1	Statements – a Motivation	53
3.7.2	Imperative Variable Declarations	53
3.7.3	Imperative Variable Expressions	53
3.7.4	Atomic Statements	54
3.7.4.1	The Assignment Statement	54
3.7.4.2	Statement Interpretation.	54
3.7.4.3	The skip Statement	54
3.7.4.4	The stop Statement	54
3.7.5	Enumerated Statements	55
3.7.6	Conditional Statements	55
3.7.6.1	The “if ... then ... else ... end” Statement	55
3.7.6.2	The “case ... of ... end” Statement	55
3.7.6.3	The McCarthy Statement	55
3.8	Concurrency	55
3.8.1	AMoL Behaviours	56
3.8.2	AMoL Behaviour Specification Units	56
3.8.3	AMoL Behaviour Invocation	58
3.8.4	AMoL Channel Specification Units	58
3.8.5	AMoL Concurrency Initialization	58
3.8.6	AMoL Concurrency Clauses	59
3.8.7	Output	59
3.8.8	Input	59
3.8.9	More on Behaviour Definition Bodies	59
3.8.10	Parallel Composition	61
3.9	Summary	61

Language, in spoken and/or written form, is the primary means for communication between humans. In order to communicate knowledge about a domain, between humans, we use language, both informal, and, since it is possible, also formal, that is, as here, in **AMoL**. In order to communicate request for calculations, between humans and computers, we use formal language, typically some programming language. **AMoL** is an “abstract programming” language. In this section we shall unfold the **AMoL** language.

3.1 A Résumé of Domains

From Chapter. 2 we summarize.

A domain consists of *endurants* and *perdurants*.

3.1.1 Endurants

Endurants are *manifest*: can be seen and touched, that is, have *external qualities*, and have intangible, we shall call them, *internal qualities*.

3.1.1.1 External Qualities:

Endurants are either *solid* (i.e., discrete) or *fluids*. Solid endurants are either *parts* or are *living species*. Solids and fluids can be characterized by their *external* and by their *internal qualities*. The external qualities of solid parts are that they are either *atomic* or *compound*. External qualities of compound parts are that they are either *Cartesians* or *sets* of endurants.

3.1.1.2 Internal Qualities:

The internal qualities of endurants are that parts have *unique identification*, have *mereologies*, and have distinctly *named attributes* and *attribute values*.

3.1.2 Perdurants

Perdurants are related to endurant parts by *transcendental deductions* — considered to reflect the *behaviour* of *parts*. Behaviours are *functions*, i.e., values; syntactically expressed they have *signatures*, i.e., names and types, have behaviour [body] *definitions*. Behaviours, operationally speaking, *communicate* and *synchronize* [over channels] and may share common, i.e., *global variables*.

3.2 Values, Types and Sorts, Axioms

This section serves to remind the reader of some fundamental concepts of specifications. Which are the *values* that we are analyzing and describing? The corresponding *type concept*: that of “compartmentalizing” classes of domain values; and *And axioms*: the expression of constraints on values.

3.2.1 Values

With endurants we can therefore associate *values* of the following kinds: external qualities, unique identifiers, mereologies, and attributes.

3.2.2 Types and Sorts

We then “impose” on these endurant values that they are of decidable *types*. That is, that atomic and compound parts (Cartesians and sets) as well as fluid values have types, and that unique identifiers, mereologies and attributes are of distinct types.

We shall use the term ‘*sort*’ in lieu of the term ‘*type*’ of the external qualities of endurants. The reason is that we shall not define these ‘*sorts*’ in terms of [mathematical] sets, Cartesians, lists, maps, etc.

By a *type* and *sort* we shall understand a class, a collection, of values [“*of the same kind*”] — whether these values are those manifest in the domain or denoted by a formal domain description.

We try to avoid using the term ‘set’, and uses instead ‘class’ or ‘collection’. The reason is that “*types are not sets*” [140, 160, 166].¹

3.2.2.1 Two Kinds of Values and Types/Sorts

When You and the domain analyzer cum describer is facing, i.e., observing the domain, the values of what You see and **informally describe** — or as we shall say: *narrate* — are those of **the manifest endurants and perdurants**, their external and internal qualities, et cetera.

In contrast, when You **formally describe**, in **AMoL**, the values, types and sorts – of what You have first informally described – are now abstract **mathematical quantities**.

In the “transition”, from informal to formal description what has “occurred” is a **transcendental deduction**.

3.2.3 Axioms

Among our [formal] definitions of types and values we may have to “insert” *axioms*, that is, predicates over types or over values that restrict the range of these.

3.3 Expressions, Statements, Clauses

Expressions, statements and clauses are main textual units of **AMoL**.

Expressions denote *values*.

Statements denote [possible] *state changes* – where states are expressed in terms of declared variables.

Clauses are either:

- (i) expressions, or (ii) statements, or (iii) expressions with statements,

with the latter, (iii), also being, hence (iv), a statement sequence ending with an expression including **AMoL**’s rendition of CSP constructs: the output: $\text{ch}[\{i, j\}]! \text{val}$ – which we consider to be a statement – effecting no state change, or the input: $\text{ch}[\{i, j\}]?$ – which we consider to be an expression.

3.4 Specification Units

The above view of domains justifies, we claim, the following.

Description of a domain, in **AMoL**, is in the form of a sequence of *specification units*.² Each specification unit introduces one or more (i) *types* and *sorts*, or (ii) *values*, or (iii) *axioms*, or (iv) *variables*, or (v) *channels*, or expresses an (vi) *intentional pull*, or the (vii) *domain initialization*. The latter two (vi,vii) are optional.

Each specification unit relates to one or more domain entities: endurants or perdurants, or external endurant quality, or internal endurant quality: unique identification, or mereology, or attribute.

Several specification units may relate to the same domain entities. We therefore suggest five kinds of specification units. The main specification units are of the form:

- **type** TypeName and/or • **type** TypeName = TypeExpression
- **value** id:TypeName and/or • **value** id:TypeName = ValueExpression

¹In [166] ‘types’ are referred to as ‘domains’. Scott’s *Domain Theory* thus “conflicts” with our use of that term. Please accept that!

²We saw, in Chapter 2, how the domain modeller can proceed in stages and steps, and, within each step, being able to describe “chunks, bits and pieces”, of a subject domain.

We shall in the following abbreviate `TypeName` and `TypeExpression` into respectively T and \mathcal{TE} . We shall similarly abbreviate `ValueExpression` into \mathcal{E} .

Three further specification units are of the form:

- **axiom** `PredicateExpression` abbreviated \mathcal{A}
- **variable** `v:T := E`
- **channel** `ch[{uii, uij}] : T`

In the following sub-sections we shall briefly elaborate on these five kinds of specification units.

3.4.1 Two Forms of Type Specification Units

There are two kinds of type specification units:

- **type** T
- **type** $T = \mathcal{TE}$

The former introduces a type, named T , without further defining “details” about this type. The latter introduces a type, named T , and further defines “details” about this type in the form of the type expressions \mathcal{TE} , see Sect. 3.5 on page 45.

These kinds of specification units are used, in formal domain descriptions, in the following contexts.

The “**type** T ” form is typically used when introducing enduring sorts and unique identifiers. They are then typically paired with **value** and/or grouped with **value** and **axiom** specification units. See Sect. 3.4.8.1 below.

The “**type** $T = \mathcal{TE}$ ” form is typically used when introducing enduring mereologies and attributes – and are usually paired with **value** and/or grouped with **value** and **axiom** specification units. See Sect. 3.4.8.1.4 below. And see Sect. 3.5 for forms of type expressions \mathcal{TE} .

3.4.2 Value Specification Units

There are two, actually three (!), kinds of value specification units:

- **value** `id:T`
- **value** `id:T=C`
- **value** `f:A→B, f(a)≡C(f,a)`

The third form is a specialization of the second form.

The first form introduces a value named `id` to be of type T . The specific value ascribed ($=$) to `id` is left undefined.

The second form also introduces a value named `id` to be of type T . The specific value ascribed ($=$) to `id` is to be the value of clause \mathcal{C} – whose elaboration “ends” in that of an expression. The value of \mathcal{C} must be of type T . If not, then the specification unit, and the whole specification, is erroneous. All is **chaos!** **AMoL** is designed so that it can be *statically decided* whether specification units are type-incorrect. That is, **AMoL** is a *strongly typed* formal language.

The third form introduces a *function* of type $A \rightarrow B$, i.e., from arguments $a:A$ into results $b:B$ and defines $f(a)$ by its equality, \equiv , to the value of clause $\mathcal{C}(f,a)$. The function may be partial in which case we use $\tilde{\rightarrow}$. Clause $\mathcal{C}(f,a)$ shall be understood to contain [*free*] occurrences of identifiers f and a . They are now, in the third form, *bound* to the left-hand side of $f(a) \equiv \mathcal{C}(f,a)$.³

³We shall restrict the introduction of identifiers in **AMoL** texts such that we need not “deploy” the λ -Calculus notion of *free* and *bound* identifiers.

3.4.3 Axiom Specification Units

Axioms are predicates over domain entities which have been introduced in [other] specification units, whether [concrete, mathematical] types or values. Their purpose is [usually] to constrain, to limit, the range of values (of types).

- **axiom** \mathcal{A}

Predicate \mathcal{A} is thus an expression. Usually such predicates evolve around quantified expressions. These contain identifiers, type names and variables introduced in [other] specification units. See Sect. 3.6.3 on page 48 for more on quantified expressions.

3.4.4 Variable Specification Units

Variables denote “*storage cells*”. That is: capabilities for remembering values that can be updated. **AMoL** variables are “*global*”, that is: only a definite number of them can be defined in any specification and they can all be “*accessed*”, i.e., *referred to*, “*read*” in any expression of any specification units, cf. Sect. 3.7.3 on page 53. Further, the value contained in the [storage cell of the] variable, can be “*updated*” [*written, re-assigned*] in any statement of any specification units, cf. Sect. 3.7.4 on page 54.

- **variable** $v:T := \mathcal{C}$

We say that the *declared* variable, v , is *initialized* to some initial value [the value of clause \mathcal{C}]. The value of \mathcal{C} must be of type T . If not, then the specification unit, and the whole specification, is erroneous. All is **chaos!** **AMoL** is designed so that it can be *statically decided* whether specification units are type-incorrect. That is, **AMoL** is a *strongly typed* formal language.

3.4.5 Channel Specification Units

AMoL behaviours, and individual **AMoL** behaviours consist of an expression or a sequence of two or more clauses — expressions and statements. Elaboration of a set of two or more **AMoL** behaviours may result in the synchronization and communication between two **AMoL** behaviours. Communication between any pair of behaviours is said *to take place in a “medium”* which we shall rephrase, by transcendental deduction, into *to take place over a channel*. This concept of *channel* is purely an abstraction.

- **channel** $\{ \text{ch}[\{ui_i, ui_j\} \mid ui_i, ui_j:UI \wedge ui_i, ui_j \in \text{uiset}] : T$

The above channel *declaration* specifies a name, ch , and considers the “medium” to be “*matrix*”-like: “two-dimensional”, over distinct indices ui_i, ui_j where ui_i and ui_j are the *unique identifiers* of distinct domain *parts*, and where UI is the type name for all such *unique identifiers*. The *communicated messages* are of type T introduced in some type specification unit.

We refer to Sect. 3.8 on page 55 for more on channels and channel *output/input*.

3.4.6 Intentional Pull

TO BE WRITTEN

3.4.7 Domain Initialization

TO BE WRITTEN

3.4.8 Special Forms of Combined Specification Units

We schematize some forms of combined domain specification units.

3.4.8.1 Type and Value Specification Units

We illustrate this kind of [paired] specification units for the description of compound parts ($p:P$) and part ($p:P$) identification.

3.4.8.1.1 Cartesians: [Cf. Sect. 2.7.1.3.1.3 on page 19] Some solid endurants are compound parts, $p:P$, and in the form of “containing” a definite number of two or more distinguishable part types. We do not model them as [real] Cartesians, e.g. $E1 \times E2 \times \dots \times Em$. The reason is that “*the whole, i.e., $p:P$, is more than the “sum”* [here Cartesian, $(e1:E1 \times e2:E2 \times \dots \times em:Em)$], *of its constituents*. That which is “more” is that the Cartesian-like parts, $p : P$, have *unique identity, mereology* and *attributes* separate from those of its constituents.

So this is the *schema* form of a combined type and **obs_erver** function value specification unit for Cartesians.

Narration:

t Cartesian parts, $p:P$, include m endurant sorts, $E1, \dots, Em$. $E1$ stands for ...; $E2$ stands for ...; ...; and Em stands for

o From these parts $p:P$ one can **obs_erve** these.

Formalization:

t **type**
 $E1, \dots, Em$

o **value**
 $\mathbf{obs_E1}: P \rightarrow E1, \dots, \mathbf{obs_Em}: P \rightarrow Em$

Sort names $E1, \dots, Em$ are all distinct. And distinct from any other type and sort names introduced in other type specification units. Usually they are *mnemonics* of more descriptive (composite) names. The $\mathbf{obs_Ei}$ observer functions are not defined. Their type, $\mathbf{obs_Ei}: P \rightarrow Ei$, is given. They cannot be defined. They are observed by the domain analyzer cum describer. That person, or those persons,

3.4.8.1.2 Set Parts: [Cf. Sect. 2.7.1.3.1.4 on page 20] Some solid endurants are compound parts, $p:P$, and in the form of “containing” an indefinite number of zero or more parts of the same part, one, type.

So this is the *schema* form of a combined type and **obs_erver** function value specification unit for part sets.

Narration:

t Solid endurants, $p:P$, consists of an indefinite set of endurants, $e:E$. We name the type of such sets PS. And we define the concrete, mathematical type of PS by $PS = E\text{-set}$.

o From solid endurants, $p:P$, we can observe

the set parts $ps:PS$.

Formalization:

t **type**
 $E, PS = E\text{-set}$

o **value**
 $\mathbf{obs_PS}: E \rightarrow PS$

The usually mnemonic type names E and PS are distinct and distinct from any other type names introduced in any other type specification units.

3.4.8.1.3 Unique Identification: [Cf. Sect. 2.7.2.1 on page 23] Manifest parts have distinct, i.e., unique identification. There is but one simple, atomic sort of unique identifiers, UI. With each part, of any type, P , we can then associate a unique identifier observer function, $\mathbf{uid_UI}: P \rightarrow UI$. That function cannot be defined. It is, as one may colloquially say it, “*built-in*”. The domain analysis & description method mandates it. The functions, $\mathbf{uid_UI}: P_i \rightarrow UI$, $\mathbf{uid_UI}: P_j \rightarrow UI$, ..., $\mathbf{uid_UI}: P_k \rightarrow UI$, one for each part sort P , are, in a sense, pre-defined once a part sort P has been introduced.

Narration:	Formalization:
t Parts $p:P$ can be ascribed a unique identifier type.	t type PI
o From parts $p:P$ one can observe their unique identifiers.	o value uid_P : $P \rightarrow PI$

3.4.8.1.4 Type, Value and Axiom Specification Units We illustrate this kind of [grouped] specification units for the description of enduring mereologies and attributes.

3.4.8.1.4.1 Mereologies: [Cf. Sect. 2.7.2.2 on page 25] With most manifest parts we can associate how such parts relate, or associate, topologically or conceptually, to other parts. These mereological relationships can be expressed in terms of the set(s) of unique identifiers of these other parts – with which a part is related. We can model these mereological relationships in a number of ways: (i) as simple set; (ii) as a pair of sets: one set referring to the parts to which output is offered, the other set referring to the parts from which input is accepted; (iii) or any other such structure, as convenient for subsequent use of mereologies.

Narration:	constrained, etc.
(t) The <i>type</i> , M , of the mereology of parts $p : P$ relates [to] the Unique Identifiers, i.e., UI types, of the [other] parts, $p_1 : P_1, \dots, p_m : P_m$, to which p is topologically or conceptually “connected”.	Formalization: (t) type $M = \mathcal{M}(UI)$
(o) From parts $p : P$ one can <i>observe</i> , mereo_M .	(o) value mereo_M : $P \rightarrow M$
(a) The mereology of parts $p : P$ may be or is	(a) axiom $\mathcal{A}(M)$

3.4.8.1.4.2 Attributes: [Cf. Sect. 2.7.2.3 on page 26] Manifest parts and fluids possess further intangible properties in the form of attributes. These are either those of physically measurable quantities having distinct attribute type names and physical unit dimensions. Attribute type names, usually mnemonics, are freely chosen, distinct identifiers, by the domain analyzer cum describer. Besides the attribute type name one can advisably adorn them with so-called **SI Units**⁴ such as *m* (meters), *kg* (kilograms), *sec* (seconds), *speed* (speed: *m/sec*), etc. Or the attributes are of conceptual concepts typically of “*historical*” nature, can be “spoken about” as events that have or are occurring. Again these attribute type names are freely chosen, distinct identifiers, by the domain analyzer cum describer. Occasionally they

Narration:	(a) The defined attribute types are, or may be, constrained as follows:
(t) Endurants $e : E$ have a number, n , of attributes, A_i . Attribute A_i is of the form, i.e., type,	Formalization: (t) type $A_1=TE_1, \dots, A_n=TE_n$
(o) From endurants $e : E$ one can <i>observe</i> , attr_Ai , attributes A_i .	(o) value attr_A1 : $P \rightarrow A_1, \dots, \mathbf{attr_An}$: $P \rightarrow A_n$

⁴https://en.wikipedia.org/wiki/International_System_of_Units

(a) axiom $\mathcal{A}(A_1, \dots, A_n)$

3.5 Types and Values

Most of the specification units show the specification of types. Hence, let us take a brief look at types.

So, really, the main concepts of a formal domain description, as in **AMoL**, are those of *types* and *values*. To put Your understanding of this at ease: Values in **AMoL** are either “like” that of data of a computer program, or of program procedures. Types are then, with reference to computer programming in some reasonably “high-level” language, like **Java**, “like” the **integer**, **float**, **record**, **array**, etc. types of that computer programming language.

Whereas types in “ordinary” computer programming are “geared” to the data types and storage structure of the hardware computers on which compiled programs shall be “executed”, **AMoL** types are “geared” to discrete mathematics, incl. mathematical logic, and recursive function theory.

3.5.1 Informal Summary of Type Expressions

We now summarize the type concept of **AMoL**. There are atomic types. They are named **Bool**, **Nat**, **Int**, **Real**, **Char**, **TIME**, **POINT** for Booleans (truth values), numbers (natural, integers and reals), characters, times and spatial points. There are abstract, algebraic types – which we refer to as sorts. They are given names, for example **T**. Their further “structure” is not defined but transpires from their being subject to various observer functions (**obs_T1, ..., uid_T, mereo_T, attr_T, ...**). And there are composite types, that is concrete mathematical structures. They are types which denote finite or potentially infinite sets, Cartesians, finite or potentially infinite length lists, maps and functions. These are defined using **AMoL**’s type operators: **-set**, **-infset**, \times , $*$, $^\omega$, \overrightarrow{m} , \rightarrow , and $\tilde{\rightarrow}$. There are “alternative” types, defined using **AMoL**’s type operator $|$. And there are types which are (proper) sub-types of elsewhere defined types.

3.5.2 Semi-Formal Summary of Type Expressions and Definitions

We now, informally, illustrate how these type expressions or type definitions (=) may occur in **AMoL** type specification units:

type

Bool, **Nat**, **Int**, **Real**, **Char**, **TIME**, **POINT**

T, **T_i**, **T_j**

T = **TE**, **T** = **T_i** | **T_j** | ... | **T_k**

where: **TE** = **T-set** | **T-infset** | (**T_i** × ... × **T_n**) | **T_j*** | **T_i^ω** | **T_i** \overrightarrow{m} **T_j** | **T_i** → **T_j** | **T_i** $\tilde{\rightarrow}$ **T_j**

T = { | **t_i** | **t_i**:**T_i** • **B(t_i)** | }

– where $n \geq 2$, **T**, **T_i**, **T_j**, ..., **T_k**, **T_n** are type names, and **B(t_i)** is a predicate.

3.5.3 Informal Meaning of Type Expressions

We explain, again informally, these type clauses:

- (i) **Bool** denotes the three valued type of truth values: **true**, **false**, **chaos**.
- (ii) **Nat** denotes the infinite number of natural number values: 0, 1, 2, ...
- (iii) **Int** denotes the infinite number of integer number values: ..., -2, -1, 0, 1, ...
- (iv) **Real** denotes the infinite number of real number values.
- (v) **Char** denotes the finite number of character values: ‘a’, ‘b’, ..., ‘z’, ‘A’, ‘B’, ..., ‘Z’.
- (vi) **TIME** denotes the infinite number of time values, e.g.: *March 12, 2024: 10:48 am*.
- (vii) **POINT** denotes the infinite number of spatial point values, for example in some global co-

ordinate system: longitude, latitude and altitude.

- (viii) **Ti-set** denotes the indefinite set of finite sets of T_i values.
- (ix) **T-infset** denotes the potentially infinite set of finite and possibly infinite sets of T_i values.
- (x) $(T_i \times \dots \times T_m)$ denotes the potentially infinite set of Cartesian values $(t_{i_a}, t_{j_b}, \dots, t_{k_z})$ where individual t_{i_c} s range over type T_i values.
- (xi) T_j^* denotes the potentially infinite set of finite length lists of T_j values.
- (xii) T_j^ω denotes the potentially infinite set of finite and infinite length lists of T_j values.
- (xiii) $T_i \xrightarrow{\overline{m}} T_j$ denotes the set of maps, i.e., finite definition set discrete functions, from T_i to T_j values.
- (xiv) $T_i \rightarrow T_j$ denotes the set of total functions from (subset of) T_i to (subset of) T_j values.
- (xv) $T_i \xrightarrow{\sim} T_j$ denotes the set of partial functions from (subset of) T_i to (subset of) T_j values.
- (xvi) $\{ | t_i | t_i : T_i \bullet \mathcal{B}(t_i) | \}$ denotes the sub-type of T_i values that satisfy predicate $\mathcal{B}(stv)$.

3.5.4 Some Comments of Type Definitions

Please note that we do not suggest using type expressions, other than identifiers of atomic types and [general] type identifiers, in type expressions involving the type constructors **-set**, **-infset**, \times , $*$, $^\omega$, $\xrightarrow{\overline{m}}$, \rightarrow , $\xrightarrow{\sim}$, $|$ and $\{ | \dots | \}$.

Please observe that *domain endurants are not recursive!* That is, a set of type specification units over domain endurants do not refer recursively to one another, that is, they can be sequentially ordered — most general endurants first, then immediately “contained” next, and so forth.

• • •

Most of the specification units show the specification of expressions. Some of axioms. Really, the expressions should be “generalized” to *clauses*. Clauses are either expressions, or “final” expressions of statements, or are “final” expressions of concurrency clauses. The next three sections cover the **AMoL** concepts of expressions, statements and concurrency.

3.6 Expressions

Applicative clauses, i.e., expressions, describe values. Functions are [also] values.

The applicative, or functional, programming version of **AMoL**, is the base **AMoL**.

3.6.1 Atomic Expressions

There are four kinds of atomic expressions. They denote base constants, values in general, **TIME** (and **Time** intervals), respectively **POINTS**.

3.6.1.1 Base Constants:

There are constants:

- **c-** constants

Constants come in four forms, all name values,

These are examples of **AMoL** constants:

- * **true, false:** **Booleans**;
- * **0, 1, 2, ..., -1, -2, ...:** **Integers**; and
- * **'a', 'b', ..., 'z', 'A', 'B', ..., 'Z', ...:** **Characters**.
- * **“ab...c”, “AB...C”, ...:** **Text**

3.6.1.2 Value Identifiers

There are [‘function’] variables:

- `id` – ‘variables’

These [function] variables denote values, The use of variables assume that there is a context, we shall call such contexts for *envipnments*, ρ for ‘rho’. *Envipnments*, ρ , map variable identifiers to values:

$$* \rho:ENV: [id_1 \mapsto v_1, id_2 \mapsto v_2, \dots, id_n \mapsto v_n]: (ID \xrightarrow{\rho} VAL)$$

If an function variable `id` is expressed in a context, an environment, where `id` is not [defined] in ρ , then the value is **chaos** – and the whole specification “blows up”, i.e., is **chaos**.

The function variable `id` may denote any value as unveiled below: Booleans, numbers, characters, texts, sets, Cartesians, lists and map – and also functions (over these).

3.6.1.3 TIME and Time Interval Values

Domains exist is **TIME**. Time is not an attribute. It is a universal, unique property of any domain. There is but one kind, i.e., type, of **TIME** – so named!

The (“functional”) atomic expression:

- `record_TIME`

[with or without argument “(s)”] evaluates to whichever **TIME** it is at the time this expression is evaluated.

TIME Interval values, **TI**, arise as the result of subtracting a smaller **TIME** from a larger time:

- `-`: $TIME \times TIME \rightarrow TI$

Other operations on **TIMES** and **TIME** Intervals are postulated:

- `+, -`: $TIME \times TI \rightarrow TIME$
- `+, -`: $TI \times TI \rightarrow TI$
- `*, /`: $TI \times Real \rightarrow TI$
- `=, \neq, <, \leq, >, \geq`: $(TIME \times TIME | TI \times TI) \rightarrow Bool$

The zero **TIME** Interval is expressed by τ_0 .

3.6.1.4 POINT Values

Domains exist in **SPACE**. **SPACE** consist of a dense set of **POINTS**. **SPACE** and **POINTS**, also, are not attributes of entities. It and they are all-pervading.

The (“functional”) atomic expression:

- `record_POINT(e)`

evaluates to some **POINT** in **SPACE** of the entity `e`. The following operations on **POINTS** are postulated:

- `\delta`: $POINT \times POINT \rightarrow DISTANCE$
- `=, \neq`: $POINT \times POINT \rightarrow Bool$

where we shall presently not go into the spatial concepts of **DISTANCE**, **CURVE**, **PLANE**, **LENGTH**, **VOLUME**, **AREA**, etc.

We shall rarely model spatial properties of domain entities.

3.6.2 Enumerated Expressions

Let e_i , dei , rei stand for expressions. These are some enumerated expressions:

$\{e_1, e_2, \dots, e_n\}$, for $n=0$: $\{\}$ set enumerations
 (e_1, e_2, \dots, e_n) , for $n>1$: Cartesian enumerations
 $\langle e_1, e_2, \dots, e_n \rangle$, for $n=0$: $\langle \rangle$ list enumerations
 $[de_1 \mapsto re_1, de_2 \mapsto re_2, \dots, den \mapsto ren]$, for $n=0$: $[]$ set enumerations

In explaining their semantics let us [somewhat loosely] introduce, in *text*, the following semantics types:

type
 $VAL = Bool | Int | Char | Text | SET | CART | LIST | MAP$
 $SET = VAL\text{-}set$, $CART = VAL \times VAL \times \dots \times VAL$, $LIST = VAL^*$, $MAP = VAL \xrightarrow{m} VAL$

Let us refer to the expression semantics evaluation function as \mathbb{E} . The “single quoted” texts below, to the left of the \rightarrow , stand for syntactic phrases. Informally \mathbb{E} can be expressed as follows:

$\mathbb{E}(e)(\rho) \equiv$
case e **of**
 $\text{'}\{e_1, e_2, \dots, e_n\}\text{'}$ $\rightarrow \{\mathbb{E}(e_1)(\rho), \mathbb{E}(e_2)(\rho), \dots, \mathbb{E}(e_n)(\rho)\}$,
 $\text{'}(e_1, e_2, \dots, e_n)\text{'}$ $\rightarrow (\mathbb{E}(e_1)(\rho), \mathbb{E}(e_2)(\rho), \dots, \mathbb{E}(e_n)(\rho))$,
 $\text{'}\langle e_1, e_2, \dots, e_n \rangle\text{'}$ $\rightarrow \langle \mathbb{E}(e_1)(\rho), \mathbb{E}(e_2)(\rho), \dots, \mathbb{E}(e_n)(\rho) \rangle$,
 $\text{'}[de_1 \mapsto re_1, de_2 \mapsto re_2, \dots, den \mapsto ren]\text{'}$ \rightarrow
 $[\mathbb{E}(de_1)(\rho) \mapsto \mathbb{E}(re_1)(\rho), \mathbb{E}(de_2)(\rho) \mapsto \mathbb{E}(re_2)(\rho), \dots, \mathbb{E}(den)(\rho) \mapsto \mathbb{E}(ren)(\rho)]$,
 \dots
end

3.6.3 Quantified Expressions

There are three forms of quantified expressions:

- $\forall a:A \bullet \mathcal{B}(a)$, $\exists a:A \bullet \mathcal{B}(a)$, $\exists ! a:A \bullet \mathcal{B}(a)$

These are the *universally*, *existentially* and *unique existentially quantified* expressions. The former holds, i.e., evaluates to **true**, if it holds for all elements a in A . The existential quantification holds if $\mathcal{B}(a)$ holds for at least one a in A . The unique existential quantification holds if $\mathcal{B}(a)$ holds for at exactly one a in A .

3.6.4 Definite Expressions

- $\iota v:V \bullet \mathcal{P}(v)$

This expression definitively describe the unique value v , of type V , for which a property $\mathcal{P}(v)$ holds, that is, for which no other v' , of type V , has $\mathcal{P}(v')$ hold. If no value – or if more than one value – v can be found for which $\mathcal{P}(v)$ holds, then the value is **chaos**.

3.6.5 Comprehended Expressions

There are *comprehended* set, list and map expressions:

- $\{ f(a) \mid a:A \bullet a \in as \}$
- $\langle g(i,a) \mid i:\mathbf{Nat}, a:A \bullet i \in is \wedge a \in as \rangle$
- $[p(a) \mapsto q(a) \mid a:A \bullet a \in as]$

The set comprehension expresses the set of all elements $f(\mathbf{a})$ where \mathbf{a} , of type A , is in some set \mathbf{as} , and f is some function. The list comprehension expresses the list of all elements $f(i, \mathbf{a})$ where i is in an index set \mathbf{is} , of type \mathbf{Int} , \mathbf{a} is of type A , is in some set \mathbf{as} , and g is some function – and where the ordering of the list elements is the ordering of the integers. The map comprehension expresses a map, i.e., a discrete, definite definition set function, from definition set elements $p(\mathbf{a})$ to range set elements $q(\mathbf{a})$, etc.

3.6.6 Operator/Operand Expressions

Operator/operand Expressions are either prefix or infix operator⁵ expressions:

- o_p expr, expr o_i expr

3.6.6.1 Prefix Operators, o_p

The prefix operators are:

- **Boolean:** \sim negation;
- **Number:** $-$ (minus);
- **Set:** cardinality;
- **List:** length, elements, indices;
- **Map:** domain (defn. set), rng (range).

3.6.6.2 Infix Operators, o_i

The infix operators are:

- **Boolean:** $\wedge, \vee, =, \neq, \equiv$;
- **Number:** $+, -, *, /, =, \neq, >, \geq, <, \leq$,
($+, -, *, /$ [addition, subtraction, multiplication, division] yield numbers –
 $=, \neq, >, \geq, <, \leq$ [equality, non-equality, greater than, greater than or equal, less than, less than or equal] yield Booleans);
- **Character:** $=, \neq$;
- **TIME:** $=, \neq, >, \geq, <, \leq$;
- **POINT:** $=, \neq$;
- **Set:** $\cup, \cap, \subset, \subseteq, \supset, \supseteq, =, \neq$,
(\cup, \cap , [union, intersection] yield sets –
 $\subset, \subseteq, \supset, \supseteq, =$ [proper subset, subset, proper superset, superset, equality, non-equality] yield Booleans);
- **List:** $\hat{\ } , =, \neq$,
($\hat{\ }$ concatenation yields lists, $=, \neq$ yield Booleans);
- **Map:** $\cup, \dagger, \backslash, =, \neq$,
(\cup [union], \dagger [map override] and \backslash [map restriction] yields maps; $=$ [equality] and \neq [non-equality] yields Booleans).

3.6.7 The let ... in ... end Expressions

There are several forms of this expression.

⁵We omit treatment [and use] of suffix operators, viz.: ! factorial, etc.

3.6.8 Non-function Bindings

- **let** $id:T = \mathcal{E}_d$ **in** \mathcal{E}_b **end**

The above is the simplest form.

A more general form of the above is:

- **let** $\mathcal{P}:T = \mathcal{E}_d$ **in** \mathcal{E}_b **end**,

The repeated form is:

- **let** $\mathcal{P}_1:T_1 = \mathcal{E}_{d_1}, \mathcal{P}_2:T_2 = \mathcal{E}_{d_2}, \dots, \mathcal{P}_n:T_n = \mathcal{E}_{d_n}$, **in** \mathcal{E}_b **end**,

3.6.9 Patterns

\mathcal{P} is a **Pattern**. The identifiers of patterns are free identifiers, i.e., identifiers not bound otherwise. Patterns are either:

- | | | |
|---------------------------|--|--------------------------------------|
| 18. " id " | 20b. " (in_1, \dots, in_m) ", $m \geq 2$ | 22. " $[did \mapsto rid] \cup map$ " |
| 19. " $\{id\} \cup set$ " | 21. " $\langle id \rangle \wedge list$ " | |

To explain the single occurrence of general form pattern-binding form let us formalize the syntax of left-hand side Patterns:

- | | |
|--|--|
| 17. A pattern is either | (c) such that at least one is an identifier, |
| 18. an identifier, id , or | or |
| 19. an arbitrary set element identifier, $\{id\}$, and the identifier. set , of the "remaining" set, or | 21. an identifier for the first element of a list, $\langle id \rangle$, and the identifier, $list$, of the "remaining" list, or |
| 20. an n-tuple grouping of | 22. the identifiers for an arbitrary mapping, $[did \mapsto rid]$, and the identifier, map , of the "remaining" map. |
| (a) distinct identifiers | |
| (b) or "nils", in_j s, for example concretely designated by "–" ⁶ – | 23. Identifiers are further undefined. |

type

- 17. $P = ID \mid SET1 \mid CARn \mid LIST1 \mid MAP1$
- 18. $ID :: Id$
- 19. $SET1 :: Id \times Id$
- 20. $CARn :: IN \times IN \times \dots \times IN$
- 20b. $IN = Id \mid "nil"$
- 21. $LIST1 :: Id \times Id$
- 22. $MAP1 :: (Id \times Id) \times Id$
- 23. Id [left further undefined]

axiom

- 20c. $\forall mkCARn(in_1, in_2, \dots, in_m): CARn \bullet \{in_1, in_2, \dots, in_m\} \setminus \{"nil"\} \neq \{\}$

Similarly the abstract syntax of values: yielded by evaluation of the right-hand side \mathcal{E}_d :

- | | |
|---|------------------------------|
| 24. Expression values are either | 26. sets of values, or |
| 25. Base values, like Booleans, Reals, Characters, Texts, TIMES, TIs, POINTs or are | 27. Cartesians of values, or |

⁶– but here abstracted as "nil"

28. lists of values, or

29. maps, generally, from values to values.

type

24. VAL = BASE | VSET | VCAR | VLIST | VMAP
 25. BASE :: (Bool|Real|Char|Text|TIME|TI|POINT)
 26. VSET :: VAL-set
 27. VCAR :: VAL×VAL×...×VAL
 28. VLIST :: VAL*
 29. VMAP :: VAL $\overrightarrow{\mapsto}$ VAL

We wish to illustrate the binding of values to pattern *identifiers*. We do so by, somewhat informally, explaining the semantics, Eval, of a “patterned” let clause. The general idea is that set-oriented patterns require non-empty set values, Cartesian-oriented patterns require Cartesian values, and so forth.

30. A let clause consists of two texts: the let definition: a pattern and an expression, and a body expression.
 31. A let clause evaluation function takes a let clause and an environment and yields a value.
 32. Evaluation of the
 33. let clause body is to take place in an extended environment, ρ' which is obtained as follows (34.–34e).
 34. We “pair” the let clause pattern with the evaluated let definition value, p, v .

In successive “tests” we inquire as to whether pattern p may “match” value v :

- (a) If the pattern is [just] an identifier, *id*, then a match is made, and that identifier bound to that value extends the environment.
 (b) If the pattern is “ $\{id\} \cup set$ ” and if the value is a non-empty set, then identifier *id* is bound to an arbitrary set element and *set* to the “remaining” set.

- (c) If the pattern is “ $(in_1, in_2, \dots, in_m)$ ” and if the value is an *m*-tuple Cartesian, then identifiers in “ $(in_1, in_2, \dots, in_m)$ ” are bound to [“matching”] Cartesian element values – except for “nil” (or underscore) identifiers.
 (d) If the pattern is “ $\langle id \rangle \hat{\ } list$ ” and if the value is a non-empty list, then identifier *id* is bound to an the first list element (the head) and *list* to the “remaining” (the tail) list.
 (e) If the pattern is “ $[did \mapsto rid] \cup map$ ” and if the value is a non-empty map, then an arbitrary map pair: $[dv \mapsto rv]$ is selected, and *did* is bound to *dv*, *rid* to *rv*, and *map* to the remaining map.
 (f) If no match can be found then “the whole thing blows up”, the AMoL specification is erroneous, **chaos** ensues!

35.

type

30. Let :: (P × Expr) × Expr

value

31. Eval: Let → ENV → VAL
 31. Eval(mkLet((p,d),e)) $\rho \equiv$
 32. **let** $v = \text{Eval}(d)\rho$ **in**
 33. **let** $\rho' = \rho \dagger$
 34. **case** (p,v) **of**
 34a. (mkId(id),_) → [id \mapsto v],
 34b. (mkSET1(id1,id2),mkVSET(vs)) → (**axiom** vs \neq {})
 34b. **let** se:VAL•se ∈ vs **in** [id1 \mapsto se,id2 \mapsto vs\{se}] **end**,

- 34c. $(\text{mkCARn}(\text{id1}, \text{id2}, \dots, \text{idn}), \text{mkVCAR}(v1, v2, \dots, vm)) \rightarrow (\text{axiom } n, m \geq 2 \wedge n = m)$
 34c. $[\text{id1} \mapsto v1, \text{id2} \mapsto v2, \dots, \text{idn} \mapsto vm] \setminus \{\text{"nil"}\},$
 34d. $(\text{mkLIST1}(\text{id1}, \text{id2}), \text{mkLIST}(vl)) \rightarrow (\text{axiom len } vl \geq 1)$
 34d. $[\text{id1} \mapsto \mathbf{hd } vl, \text{id2} \mapsto \mathbf{tl } vl],$
 34e. $(\text{mkMAP1}((\text{id1}, \text{id2}), \text{id3}), \text{mkMAP}(vm)) \rightarrow (\text{axiom card dom } vm \geq 1)$
 34e. $\mathbf{let } d: \text{VAL} \cdot d \in \mathbf{dom } vm \mathbf{ in } [\text{id1} \mapsto d, \text{id2} \mapsto vm(d), \text{id3} \mapsto vm \setminus \{d\}] \mathbf{ end},$
 34f. $_ \rightarrow \mathbf{chaos}$
 33. $\text{Eval}(e)\rho'$
 31. $\mathbf{end end end}$

Note that patterns are not recursively defined, i.e., patterns may contain proper sub-patterns (other than identifiers). The explicit binding of patterns to \mathbb{T} can be omitted.

3.6.10 Function Bindings

Functions can be defined in let clauses:

- $\mathbf{let } f(a) = \mathcal{E}_d(f, a) \mathbf{ in } \mathcal{E}_b(f) \mathbf{ end}$ same as: $\mathbf{let } f = \lambda a. \mathcal{E}_d(f, a) \mathbf{ in } \mathcal{E}_b(f) \mathbf{ end}$

Function f is possibly recursively defined. Occurrences of f (and its argument a) are in $\mathcal{E}_d(f, a)$ bound to the f and a to the left of the '='. And f in $\mathcal{E}_b(f)$ is bound to the f to the left of the '='. The function being defined, when applied to an argument expr , in $\mathcal{E}_b(f)$, yields a value as prescribed in $\mathcal{E}_d(f, a)$ where expr is substituted for a .

3.6.11 Structured Expressions

- $\mathbf{if } \mathcal{B} \mathbf{ then } \mathcal{E}_c \mathbf{ else } \mathcal{E}_a \mathbf{ end},$

Evaluation of this, the Boolean valued expression $\mathbf{if } \dots \mathbf{ then } \dots \mathbf{ else } \dots \mathbf{ end}$ expression proceeds as follows. First \mathcal{B} is evaluated. If it yields a value other than a *Boolean*, either **chaos** or else, the whole evaluation yields **chaos**. If it yields the truth value, **true**, then evaluation of the entire expression is just the evaluation of \mathcal{E}_c , the *consequence*, else \mathcal{E}_a , the *alternative*.

The generalized conditional, also referred to as the McCarthy⁷ conditional, has the form:

- $\mathcal{B}_1 \rightarrow \mathcal{E}_1, \mathcal{B}_2 \rightarrow \mathcal{E}_2, \dots, \mathcal{B}_m \rightarrow \mathcal{E}_m, _ \rightarrow \mathcal{E}_n,$

The above "abbreviates" $\mathbf{if } \mathcal{B}_1 \mathbf{ then } \mathcal{E}_1 \mathbf{ else if } \mathcal{B}_2 \mathbf{ then } \mathcal{E}_2 \mathbf{ else } \dots \mathbf{ end end}$. If none of the \mathcal{B}_i holds, then the value of the entire expression is that of \mathcal{E}_n . This last clause is optional. Then, if none of the \mathcal{B}_i holds, then the value of the entire expression is **chaos**.

- $\mathbf{case } \mathcal{E} \mathbf{ of } \mathcal{P}_1 \rightarrow \mathcal{E}_1, \mathcal{P}_2 \rightarrow \mathcal{E}_2, \dots, \mathcal{P}_m \rightarrow \mathcal{E}_m, _ \rightarrow \mathcal{E}_n \mathbf{ end}$

Expression \mathcal{E} is first evaluated. If **chaos**, then the evaluation "blows up": chaos! Otherwise the value of \mathcal{E} is matched against patterns \mathcal{P}_i , sequentially, from 1 to m . If a match can be found then that pattern's identifiers are bound to corresponding values in \mathcal{E} and \mathcal{E}_i is evaluated. If none can be matched then the value of the entire expression is that of \mathcal{E}_n . This last clause is optional. Then, if the "catch all", the '_' alternative, is absent, and if none of the \mathcal{B}_i holds, then the value of the entire expression is **chaos**.

3.7 Statements

Statements serve to effect state changes.

⁷[https://en.wikipedia.org/wiki/John_McCarthy_\(computer_scientist\)](https://en.wikipedia.org/wiki/John_McCarthy_(computer_scientist))

3.7.1 Statements – a Motivation

The provision of statements in **AMoL** is motivated primarily by the need to express concurrency, by means of **CSP** [110], see Sect. 3.8. Concurrency is needed in order to model the concurrent behaviour of multiple domain part behaviours, in particular their synchronization and communication. In **AMoL**:

- $\text{ch}\{\$ui_i, ui_j\}! \text{val} ; \mathcal{E}$ and
- $\text{let } v = \text{ch}\{ui_i, ui_j\} ? \text{ in } \mathcal{E}(v)$

are expressions! $\text{ch}\{ui_i, ui_j\}! \text{val}$ is an **AMoL** concurrency (an output) clause, a la **CSP**, of a description of a behaviour identified by ui_i offering to synchronize and communicated with behaviour identified by ui_j . Its evaluation, in the context of behaviours identified by ui_i and ui_j , is expected to lead to behaviour ui_i sending its message, val , to behaviour ui_j . Once communicated $\text{ch}\{ui_i, ui_j\}! \text{val}$ evaluates to the “void” state, denoted by $()$. Thereafter, “;”, \mathcal{E} is evaluated. And the value of $\text{ch}\{ui_i, ui_j\}! \text{val} ; \mathcal{E}$ is then that of \mathcal{E} – “plus” the side-effect of the synchronization and communication!

Imperative **AMoL** clauses, i.e., statements, [otherwise] designate state changes.

That is, it is the presence of the “;” that prompts the issue of “imperativeness”, i.e., of statements. Once we allow “;” we may, as well, open the (“flood”) gate for statements! But, in domain modelling we shall use statements very sparingly!

3.7.2 Imperative Variable Declarations

The imperative programming version of **AMoL**, extends base **AMoL** with one or more *specification units* of the form:

- **variable** $v:T := \mathcal{C}$

Variable v designates a state: something that associates variable names with [contained] values. Clause \mathcal{C} (with $:=$) may be absent in which case the value of state variable v is left undefined, otherwise it is ‘initialized’ to the value of \mathcal{C} .

3.7.3 Imperative Variable Expressions

Imperative variables extend expressions, as defined above, with the atomic:

- $\underline{\mathbf{c}}v$

We assume, now, not only a context, the environment, ρ , but also a state, σ . Environments bind globally [specification unit] declared variables to what we shall call references, i.e., addresses, in the [storage] state. States, then, bind state variable references to values.

$$\text{type ENV} = ID \xrightarrow{\text{m}} \text{VAL} | \text{REF}, \Sigma = \text{REF} \xrightarrow{\text{m}} \text{VAL}$$

Note the use of *slanted* text. It is to signal that this *text* is in a meta-language, here some mathematics, different from **AMoL**. The imperative atomic expression $\underline{\mathbf{c}}v$ is now analyzed into v which stands for, denotes, a “value”, *REF*, of type *refT* (where T is the type of the declared variable). The [proverbial] environment, ρ , maps v into some reference, and the state, σ , maps that reference into a value:

- $\sigma(\rho(v))$.

The imperative atomic expression $\underline{\mathbf{c}}v$ may occur anywhere where an expression may occur. That is, also in applicative **AMoL** expressions.

The state variable v may denote any value as unveiled below: Booleans, numbers, characters, texts, sets, Cartesians, lists and map – and also functions (over these).

In domain modelling we shall use variables very sparingly.

Basically domain models need only two variables: one for holding all parts, and another for holder the unique identifier of all parts. The first variable, σ_{ps} , will be used to “implement” *monitorable*, including *biddable* part attributes, cf. Sect. 2.7.2.3.2 on page 27. The second variable, σ_{uis} , will be used express part mereologies and the axiom that all parts have unique identifiers.

3.7.4 Atomic Statements

There are three kinds of atomic statements: assignment, **skip** and **stop**.

3.7.4.1 The Assignment Statement

has the form:

- $v := C$,

This is the **AMoL** ‘assignment’ statement. Clause C is evaluated and [should] result in a value [of type T]. That value then replaces whatever value “was contained” in v .

3.7.4.2 Statement Interpretation.

Statements are interpreted: “one by one, one after the other’, sequentially.”

That is, one way of, simplifying, looking at **AMoL** texts is by considering their elaboration. The elaborator, \mathbb{E} , applies to **AMoL** texts, environment and states:

- **type** \mathbb{E} : “**AMoL**” $\rightarrow \text{ENV} \rightarrow \Sigma \rightarrow ((\text{VAL}|\text{nil}) \times \Sigma)$
- $\mathbb{E}(\text{txt})(\rho)(\sigma) \equiv \dots$

If the txt is a constant or a variable or an assignment then the semantics specification is:

- * $\mathbb{E}(\text{“id”})(\rho)(\sigma) \equiv (\rho(\text{id}), \sigma)$
- * $\mathbb{E}(\text{“v”})(\rho)(\sigma) \equiv (\sigma(\rho(v)), \sigma)$
- * $\mathbb{E}(\text{“v := C”})(\rho)(\sigma) \equiv (-, \sigma \uparrow [\rho(v) \mapsto \mathbb{E}(C)(\rho)(\sigma)])$

3.7.4.3 The skip Statement

has the form:

- **skip**

Interpreting **skip** results in no change, i.e., “nothing happens”, interpretation “control” skips to the next, if any, statement:

- * $\mathbb{E}(\text{“skip”})(\rho)(\sigma) \equiv (\rho, \sigma)$

3.7.4.4 The stop Statement

has the form:

- **stop**

Interpreting **stop** means to “abort” specification elaboration:

- * $\mathbb{E}(\text{“stop”})(\rho)(\sigma) \equiv (-, -)$

3.7.5 Enumerated Statements

Enumerated statements are of the form:

- $\mathcal{S}_1; \mathcal{S}_2; \dots, \mathcal{S}_n$

One way of explaining the statement list is as follows: First statement \mathcal{S}_1 is interpreted, i.e., “executed” in some state σ . It may abort, i.e., ending up in **chaos**. Then all is **chaos** and elaboration of the AMoL specification in which it occurs is “abandoned”. If interpretation of \mathcal{S}_1 does not abort then its “execution” results in a state change – into σ' . Then statement \mathcal{S}_2 is interpreted in state σ' . Same as for \mathcal{S}_1 . And so forth, through an intended sequence of state changes, from σ , via σ' , ..., to a final state $\sigma' \dots'$.

Below, in a slight informal manner, we suggest a “formalization” of statement list interpretation.

$$\begin{aligned} \mathbb{E}(\text{stmtl})(\rho)(\sigma) \equiv & \\ & \text{case stmtl of} \\ & \quad \langle \rangle \rightarrow (\rho, \sigma), \\ & \quad \langle \text{stmt} \rangle^{\text{stl}} \rightarrow \text{let } (\rho, \sigma') = \mathbb{E}(\text{stmt})(\rho)(\sigma) \text{ in } \mathbb{E}(\text{stl})(\rho)(\sigma') \text{ end} \\ & \text{end} \end{aligned}$$

3.7.6 Conditional Statements

There are three forms of conditional statements.

3.7.6.1 The “if ... then ... else ... end” Statement

- **if** \mathcal{B} **then** \mathcal{S}_c **else** \mathcal{S}_a **end**,

This is the simplest AMoL ‘conditional’ statement. [It “parallels” the AMoL conditional expression.] Boolean expression \mathcal{B} is evaluated. If **true** then ‘consequence’ statement \mathcal{S}_c is elaborated. If **false** ‘alternative’ statement \mathcal{S}_a is elaborated. If \mathcal{B} evaluates to other than a Boolean value, including **chaos**, then **chaos** ensues!

3.7.6.2 The “case ... of ... end” Statement

- **case** \mathcal{E} **of** $\mathcal{P}_1 \rightarrow \mathcal{S}_1, \mathcal{P}_2 \rightarrow \mathcal{S}_2, \dots, \mathcal{P}_m \rightarrow \mathcal{S}_m, - \rightarrow \mathcal{S}_n$ **end**,

\mathcal{P}_i are Patterns, cf. Sect. 3.6.9 on page 50. The pattern clauses are elaborated “sequentially”. If none of the patterns can be made to fit the value of \mathcal{E} then the *escape* statement, ‘ $- \rightarrow \mathcal{S}_m$ ’ is elaborated.

3.7.6.3 The McCarthy Statement

- $\mathcal{B}_1 \rightarrow \mathcal{S}_1, \mathcal{B}_2 \rightarrow \mathcal{S}_2, \dots, \mathcal{B}_m \rightarrow \mathcal{S}_m, - \rightarrow \mathcal{S}_n$,

\mathcal{B}_i are Boolean expressions. The Boolean clauses are elaborated “sequentially”. If none of the Booleans hold the *escape* statement ‘ $- \rightarrow \mathcal{S}_m$ ’ is elaborated.

3.8 Concurrency

Domain behaviours need to interact, i.e., to synchronize and communicate. The AMoL concurrency constructs serve this purpose.

Domain behaviours cannot make do with interaction via *buffers*. In “real life” they synchronize and communicate instantaneously. If some or another form of behaviour interaction via some or

another form of, say, buffers is required, then the domain modeller shall model such interaction, hence these “buffers” etc., explicitly.

AMoL contains a number of clauses “derived/inspired” from CSP. CSP stands for *Communicating Sequential Processes*. CSP was first proposed by Tony Hoare in 1978 [108]. Several textbooks have since been published, e.g., [110, 111, 161, 164].

3.8.1 AMoL Behaviours

We shall distinguish between two concepts:

- *domain behaviours* and
- *processes*.

The concept of *behaviour* is associated with that of domains, whereas the concept of *processes* is associated with that of the formal, mathematical interpretation or computerized “execution” of **AMoL**-specified behaviours.

By an **AMoL**-specified behaviour we shall understand the “execution”, i.e., elaboration, of a named function as introduced in an **AMoL** function specification unit, such that that specification unit function when invoked “becomes” a uniquely named process. Two or more, i.e., m , invocations of the same **AMoL** function specification unit, through their unique identifier, $ui_1, ui_2, \dots, ui_m, m \geq 2$, invocation argument, thus “become” uniquely distinguishable processes. They can, in **AMoL**/CSP, be referred to by these unique identifiers.

You may therefore conceive an **AMoL** induced process as the sequential “execution” (elaboration) of the **AMoL** function specification unit’s function body appropriately initialized with the function invocations’ arguments.

3.8.2 AMoL Behaviour Specification Units

(We refer to Sect. 3.8.3 on page 58.) There is the general form of *function value* specification units, cf. Sect. 3.4.2 on page 41:

- **value** $f:A \rightarrow B, f(a) \equiv C(f,a)$;

and there is now the special *function value* specification form for domain behaviours.

value

behaviour: $UI \rightarrow Mereology \rightarrow Stat_Attrs \rightarrow Mon_Attrs \rightarrow Progr_Attrs \dots$ **Unit**
 behaviour(ui)(m)(sta)(mon)(prgr) $\equiv C(ui,m,sta,mon,prgr)$

The first line above specifies, defines, the *signature* of the behaviour. The second line above specifies the detailed definition of the behaviour, i.e., the *definition body*.

We shall now describe that form in some detail.

[1] Behaviour (function)s are given names, one distinct name for each part sort. [] Behaviour functions have a number of arguments. These are linked directly to the *internal qualities* of manifest *parts*. Cf. Sect. 2.7.2 on page 23. These arguments are shown in Schönfinckel/Curry⁸ form.

[2] We choose as first argument the unique identifier of *part*, $p:P$. This means that we can, and will, apply the behaviour function for *parts*, $p:P$, to all such parts of sort P. If more than one there will then be several instances of that behaviour, each uniquely distinguishable. The unique identifier argument of an invoked behaviour, i.e., a process, will remain “constant”, that is, will not be modified in recursive invocations (“calls”) of, i.e., within that behaviour. We refer to Sect. 2.7.2.1 on page 23.

[3] The second argument is then the mereology of *part*, $p:P$. The value of this mereology expresses the unique identities of the [other, invoked] behaviours with which invocations of this defined

⁸ [165] https://en.wikipedia.org/wiki/Moses_Schönfinckel and <https://en.wikipedia.org/wiki/Currying> and https://en.wikipedia.org/wiki/Haskell_Curry

behaviour can interact. These identifiers are therefore mentioned in *output/input* clauses of the *behaviour definition body*, [9]. If the defined behaviour “creates” or “destroys” domain parts, then the mereologies, of part behaviours, that refer to the “destroyed”, or should refer to the newly “created” parts must be updated! We refer to Sect. 2.7.2.2 on page 25.

[4,5,6] The third, fourth and fifth arguments are then those of *static*, *monitorable* and *programmable* attributes of part $p:P$. We refer to Sect. 2.7.2.3 on page 26.

[4] Static arguments are passed (“called”) *by value*, that is, in invocations, the corresponding argument expressions are evaluated and then treated as constants in the behaviour definition body [9].

[5] Monitorable arguments are passed (“called”) *by name*, that is, in invocations, the corresponding argument expressions are evaluated only when encountered during elaboration of the behaviour definition body [9].

[6] Programmable arguments are passed (“called”) *by value*, that is, in invocations, the corresponding argument expressions are evaluated and become *initial* values. They are then treated as possibly changeable/updateable, i.e., programmed values, in the behaviour definition body [9]. Such updates are manifest within the definition body as well in the more-or-less mandated tail-recursive invocation (‘continuation’) of the invoked behaviour, see below, Sect. 3.8.3 on the following page.

[7] This line specifies two separate things.

First it specifies the sub-channels, $\{\text{ch}[\{ui_i, ui_j\}] | ui_i, ui_j:UI \bullet \dots\}$, over which behaviours, i.e., “their invoked process” may interact with other processes.

Then it specifies, **Unit**, that invocation of the behaviour leaves no value, but [possibly] a state change. We write “state change”. If the behaviour process stops then that state change is denoted by ‘()’. If it goes on, forever, then that is what the literal **Unit** [also] means.

[8–9] These lines define the functionality, the detailed behaviour.

[8] This line expresses the abstract invocation, i.e., “call”, of the behaviour.

[9] And this line expresses the “body” of the behaviour definition, that is, how it should evaluate any invocation.

value

```
[1]  behaviourp:P:
[2]    UIp:P →
[3]    Mereologyp:P →
[4]    Static_Attrsp:P →
[5]    Monitorable_Attrsp:P →
[6]    Programmable_Attrsp:P
[7]    { ch[ { uii, uij } ] | uii, uij:UI • ... } Unit
[8]  behaviourp:P(uip:P)(merp:P)(stap:P)(monp:P)(prgrp:P) ≡
[9]    Cp:P(uip:P, mp:P, stap:P, monp:P, prgrp:P)
```

Usually line [9] is of the schematic form:

```
[10] Cp:P(uip:P, mp:P, stap:P, monp:P, prgrp:P) ≡
[11] let (m'p:P, prgr'p:P) = Ep:P(uip:P, mp:P, stap:P, monp:P, prgrp:P) in
[12]  behaviourp:P(uip:P)(mer'p:P)(stap:P)(monp:P)(prgr'p:P) end
```

That is, a so-called tail-recursive invocation [12] of the [“same”] behaviour, but with possibly updated mereology (rarely), programmable attributes and, not immediately observable from the definition, monitorable attributes.

More specifically:

- [10] Elaboration (i.e., colloquially: “execution”) of the **behaviour** definition body, i.e., of $\mathcal{C}_{p:P}(\dots)$ first
 - [11] elaborates the “auxiliary” function $\mathcal{E}_{p:P}(\dots)$. It may [or may not necessarily] “update” the mereology, mer (of part $p:P$) and may, usually, update one or more programmable attributes, prgr .
 - [12] Once $\mathcal{C}_{p:P}(\dots)$ is so elaborated “elaboration [control]” passes on to the invoking behaviour, $\text{behaviour}(ui)(\text{mer}')(\text{sta})(\text{mon})(\text{prgr}')$.

3.8.3 AMoL Behaviour Invocation

One thing is the definition, in **AMoL** specification units, of Behaviours. Another thing is their invocation (i.e., application) in **AMoL** clauses. Invocations, for manifest parts $p:P$, are of the form:

- $\text{behaviour}(uidp)(\text{mereop})(\text{sta_attrs}(p))(\text{mon_attrs}(p))(\text{prg_attrs}(p))$

where $\text{sta_attrs}(p)$, $\text{mon_attrs}(p)$ and $\text{prg_attrs}(p)$ are of the forms:

- $\text{sta_attrs}(p)$: $A1_{sta}, A2_{sta}, \dots, A_{s_{sta}}$
where $A1_{sta}, A2_{sta}, \dots, A_{s_{sta}}$ are the *static* attributes of parts $p:P$.
- $\text{mon_attrs}(p)$: $\cdot \text{attr_}A1_{mon}, \text{attr_}A2_{mon}, \dots, \text{attr_}A_{m_{mon}}$
where $A1_{mon}, A2_{mon}, \dots, A_{m_{mon}}$ are the *monitorable* attributes of parts $p:P$.
- $\text{prg_attrs}(p)$: $A1_{prg}, A2_{prg}, \dots, A_{p_{prg}}$
where $A1_{prg}, A2_{prg}, \dots, A_{p_{prg}}$ are the *programmable* attributes of parts $p:P$.

that is, sequences of zero, one or more arguments.

The presence of the **attr_s**, prefixing monitorable attribute type identifiers expresses that monitorable attributes are “called by η name”!

If an argument sequence is zero, i.e., (), it may be left out, omitted.

3.8.4 AMoL Channel Specification Units

Behaviours interact. In **AMoL** domain modelling behaviour interaction is expressed in terms of *output/inputs* between two distinct behaviours such that the induced behaviours *synchronize* their *output/inputs*, that is, their *rendez-vous*, while at the same time *communicating* a value: one process *outputs* the value, the other process, *inputs* that value. We say that the *rendez-vous* takes place over *channels*.

The concurrent, or parallel programming version of **AMoL**, thus extends base **AMoL** with a specification unit:

channel { $ch[\{ui_i, ui_j\}] \mid ui_i, ui_j:UI \bullet \mathcal{B}(ui_i, ui_j) \}$ \top

ch names an *array* of $\{ui_i, ui_j\}$ indexed [sub-]channels, one for each possible pair of (ui_i, ui_j) of distinct behaviours, whether or not such distinctly named behaviours exist. The set of all ui_k s are the unique identifiers of all manifest domain parts. $\mathcal{B}(ui_i, ui_j)$ specifies possible constraints on $\{ui_i, ui_j\}$. The type of the values communicated over ch is T , where T is specified in some **AMoL** specification unit.

3.8.5 AMoL Concurrency Initialization

There is a behaviour initialization specification unit:

- $\parallel \{ \mathcal{F}(p) \mid p:P \bullet p \in ps \}$

The $\mathcal{F}(p)$ refer to behaviour invocations, cf. Sect. 3.8.3 on the preceding page. The $\|\{\mathcal{F}(p) \mid p:P \bullet p \in ps\}$ describes the comprehended initialization of a number of processes, one for each part p in a set of parts ps . ps usually refers to a “global” value of, or a global variable which contains, all the parts (and sub-parts, etc.) of a domain. That is, behaviours can only be invoked in *domain initialization units*.⁹ Usually it suffices that a domain description contains just one such behaviour initialization specification unit.

3.8.6 AMoL Concurrency Clauses

There are several forms of AMoL concurrency clauses.

Base, applicative, and imperative AMoL is extended with a subset of CSP [110] clauses, \mathcal{P} . These may occur wherever an AMoL clause, expression, \mathcal{E} , or statement, \mathcal{S} , may occur:

- $ch\{i,j\}!\mathcal{E}$ – output, statement;
- $ch\{i,j\}?\mathcal{E}$ – input, expressions;
- $\mathcal{C} \square \mathcal{C}$ – non-det. internal choice, , any clause; and
- $\mathcal{C} \sqcap \mathcal{C}$ – non-det. external choice, any clause.

Here \mathcal{C} is any expression, statement and concurrency Clause of AMoL.

3.8.7 Output

We assume behaviours identified by their unique identifiers ui_i, ui_j . The AMoL *output* clause:

- $ch\{ui_i, ui_j\}!\mathcal{E}$

expresses that the behaviour, identified by ui_i , *offers* the value of expression \mathcal{E} to behaviour ui_j . The $ch\{ui_i, ui_j\}!\mathcal{E}$ clause is a statement. Once elaborated, that is: once accepted by behaviour ui_j , interpretation [“execution” [control]] control “passes on” to whichever clause “follows” $ch\{ui_i, ui_j\}!\mathcal{E}$. That is, the elaborated semantic value of output is “()”. If not accepted by behaviour ui_j the “rendez-vous” between behaviours ui_i, ui_j does not take place, i.e., they are not synchronized, and behaviour ui_i “stalls”, “stands still, does not progress”!

3.8.8 Input

We assume behaviours identified by their unique identifiers ui_i, ui_j . The AMoL *input* clause:

- $ch\{ui_i, ui_j\}?\mathcal{E}$

$ch\{ui_i, ui_j\}?\mathcal{E}$ clause is an expression. It expresses that behaviour ui_i *offers* to accept a value from the behaviour, identified by ui_j . Once elaborated, that is: once accepted by behaviour ui_j , the value of $ch\{ui_i, ui_j\}?\mathcal{E}$ is the value “received”, i.e., input from behaviour ui_j . If output from behaviour ui_j is not offered, the “rendez-vous” between behaviours ui_i, ui_j does not take place, i.e., they are not synchronized, and behaviour ui_i “stalls”, “stands still, does not progress”!

3.8.9 More on Behaviour Definition Bodies

We refer to items [10–12], Page 58, Sect. 3.8.2:

- ```
[10] $\mathcal{C}(ui,m,sta,mon,prgr) \equiv$
[11] let (m',prgr') = $\mathcal{E}(ui,mer,sta,mon,prgr)$ in
[12] behaviour(ui)(mer')(sta)(mon)(prgr') end
```

The above illustrates but a simplest form of *domain behaviour definition body*. Instead of by lines [11–12]  $\mathcal{C}(ui,m,sta,mon,prgr)$  may be defined in basically either of three forms [13, 14, 15] using the  $\square$  and  $\sqcap$  behaviours operators. These operators will be explained next. First we summarize, illustratively, three forms of composing deterministic external choice,  $\square$ , and non-deterministic internal choice,  $\sqcap$ , clauses: Then we explain, “narrate”, these.

<sup>9</sup>This is a restriction of the use of the  $\|\$  behaviour composition operator wrt. to “general” CSP.

|                                                                                                                                                                                                                                                                                                                                                        |                                                                                                                                                                                                                                                                                                                                              |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>value</b></p> <p>[13] <math>\mathcal{C}(ui,me,sta,mon,prgr) \equiv</math></p> <p>[13.1] <math>\sqcap \{ \mathcal{K}_{D_1}(ui,me,sta,mon,prgr)</math></p> <p>[13.2] <math>\mathcal{K}_{D_2}(ui,me,sta,mon,prgr),</math></p> <p style="text-align: center;">...</p> <p>[13.m] <math>\mathcal{K}_{D_m}(ui,me,sta,mon,prgr) \}</math> <b>or:</b></p> | <p><b>value</b></p> <p>[13] <math>\mathcal{C}(ui,me,sta,mon,prgr) \equiv</math></p> <p>[14.1] <math>\sqcap \{ \mathcal{K}_{N_1}(ui,me,sta,mon,prgr),</math></p> <p>[14.2] <math>\mathcal{K}_{N_2}(ui,me,sta,mon,prgr),</math></p> <p style="text-align: center;">...</p> <p>[14.n] <math>\mathcal{K}_{N_n}(ui,me,sta,mon,prgr) \}</math></p> |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

**or:**

|                                                                                                                                                                                                                                                                                                                          |                                                                                                                                                                                                                                                                                          |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>[15] <math>\mathcal{C}(ui,me,sta,mon,prgr) \equiv</math></p> <p>[13.1] <math>\sqcap \{ \mathcal{K}_{D_1}(ui,me,sta,mon,prgr),</math></p> <p>[13.2] <math>\mathcal{K}_{D_2}(ui,me,sta,mon,prgr),</math></p> <p style="text-align: center;">...</p> <p>[13.m] <math>\mathcal{K}_{D_m}(ui,me,sta,mon,prgr) \}</math></p> | <p>[15] <math>\sqcap</math></p> <p>[14.1] <math>\sqcap \{ \mathcal{K}_{N_1}(ui,me,sta,mon,prgr),</math></p> <p>[14.2] <math>\mathcal{K}_{N_2}(ui,me,sta,mon,prgr),</math></p> <p style="text-align: center;">...</p> <p>[14.n] <math>\mathcal{K}_{N_n}(ui,me,sta,mon,prgr) \}</math></p> |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

**where:**  $\mathcal{K}_{D_i}(ui,m,sta,mon,prgr)$  is, typically<sup>10</sup>, of the form:

**value**

[16]  $\dots ; \text{let } v = \text{ch}[\{ui\_x, ui+y\}] ? \text{ in}$

[16.1]  $\text{let } (me', prgr') = \mathcal{C}_{D_i}(ui,me,sta,mon,prgr) \text{ in}$

[16.2]  $\text{behaviour}(ui)(me')(sta)(mon)(prgr') \text{ end end}$

**and where:**  $\mathcal{K}_{N_j}(ui,m,sta,mon,prgr)$  is, typically<sup>11</sup>, of the form:

[17]  $\dots ; \text{ch}[\{ui\_x, ui+y\}] ! \mathcal{CO}_{N_j}(ui,me,sta,mon,prgr)$

[17.1]  $\text{let } (me', prgr') = \mathcal{C}_{N_j}(ui,me,sta,mon,prgr) \text{ in}$

[17.2]  $\text{behaviour}(ui)(me')(sta)(mon)(prgr') \text{ end}$

To understand the [next] explanation of the above (three) forms of deterministic and non-deterministic choices we must remind the readers that behaviours progress in time. Time enters our understanding of **AMoL** specifications when concurrency is at stake. It is really not time, as such, e.g., “*March 12, 2024: 10:48 am*”, but “sequencing”: that elaboration of [some] clauses of different behaviour descriptions take place “at the same time, or at different times”.

- [13] Here  $\mathcal{C}(ui,me,sta,mon,prgr)$  is expressed in terms of the *external deterministic choice*,  $\sqcap$ , between  $m$  “possibilities”:  $\mathcal{K}_{D_i}(ui,me,sta,mon,prgr)$ : [13.1], [13.2], ..., [13.m].  $\mathcal{K}_{D_i}$ , for  $i = 1, \dots, m$ , are clauses. The selected “choice” is determined ‘externally’, that is, not by  $\mathcal{C}(ui,me,sta,mon,prgr)$ , but by an, or the, context, i.e., environment, in which “other” behaviours offer to synchronize and communicate, either by means of offering output to, or offering to accept input from the behaviour,  $ui$ , which  $\mathcal{C}(ui,me,sta,mon,prgr)$  represents.  $\mathcal{C}(ui,me,sta,mon,prgr)$  is elaborated. That is, we must assume that there are a number of behaviours which offer so and that each  $\mathcal{K}_{D_i}(ui,me,sta,mon,prgr)$  contains corresponding output or input clauses, i.e., [16–17]. If no ‘other behaviours’, external to behaviour  $ui$ , offers to synchronize and communicate “at the time  $\mathcal{K}_{D_i}$  “is first” elaborated, then behaviour  $ui$  is “stalled” until some such external behaviour is ready to offer. If more than one external behaviour offers, then a non-deterministic choice [arbitrarily] selects one of these. The textual, i.e., linear, ordering of the  $\mathcal{K}$  clauses, [13.1–m], is (thus) immaterial.
- [14] Here  $\mathcal{C}(ui,me,sta,mon,prgr)$  is expressed in terms of the *internal non-deterministic choice*,  $\sqcap$ , between  $n$  “possibilities”:  $\mathcal{K}_{N_i}(ui,me,sta,mon,prgr)$ : [14.1], [14.2], ..., [14.m].  $\mathcal{K}_{N_i}$ , for  $i = 1, \dots, n$ , are clauses. The selected “choice” is determined ‘internally’, that is, by  $\mathcal{C}(ui,me,sta,mon,prgr)$ . The  $\mathcal{K}_{N_i}$  clauses are expected to offer output or input clauses, i.e., [16–17]. If

<sup>10</sup> $\mathcal{K}_{D_i}$  can also be of the form  $\mathcal{K}_{N_j}$

<sup>11</sup> $\mathcal{K}_{N_j}$  can also be of the form  $\mathcal{K}_{D_i}$

no ‘behaviours’, external to behaviour  $ui$ , offers to synchronize and communicate “at the time  $\mathcal{K}_{N_i}$  “is first” elaborated, then behaviour  $ui$  is “stalled” until some such external behaviour is ready to offer. If more than one external behaviour offers, then a non-deterministic choice [arbitrarily] selects one of these. The textual, i.e., linear, ordering of the  $\mathcal{K}$  clauses, [14.1–n], is (thus) immaterial.

- [15] Here  $\mathcal{C}(ui,me,sta,mon,prgr)$  is expressed in terms of an ordered “pair” of sets, each of one or more, either *external non-deterministic choices*, respectively *internal non-deterministic choices*. The two pair sets may be interchanged without changing meaning of the while clause. The idea here is that  $\mathcal{C}(ui,me,sta,mon,prgr)$  offers to engage both ( $\alpha$ ) non-deterministically external and ( $\beta$ ) non-deterministically internal choice “rendez-vous” with other behaviours. The choice as to whether  $\alpha$  or  $\beta$  is “a-flip-of-the-coin”: non-deterministic internal.
- [13–15] [13] is a special case of [15] for  $m=0$ , [14] is a special case of [15] for  $n=0$ .
- [16–17] [16] and [17] shows that in order to engage in a “rendez-vous” there must be a mutual pair of output/input clauses in the two synchronizing & communicating processes: where one offers to output, the other offers to accept an input.

### 3.8.10 Parallel Composition

A restricted form of the **AMoL** *parallel composition*, i.e.,  $\parallel$ , clause may be useful anywhere a simple **AMoL** [imperative] clause may appear:

- $\parallel \{C_1, C_2, \dots, C_n\}$  **or**  $C_1 \parallel C_2 \parallel \dots \parallel C_n, n \geq 2$

The restriction is typically this: The  $\mathcal{C}$  clause is an output clause:  $ch[\{ui_i, ui_j\}]! \text{expr}$ .

## 3.9 Summary

We have surveyed the main **AMoL** syntax: (i) the specification units, (ii) the **AMoL** type and value concept, (iii) the applicative expressions, (iv) the imperative statements, and (v) the concurrency clauses.



## Part II

# Conclusion



## Chapter 4

# Conclusion

TO BE WRITTEN





## Chapter 5

# Bibliography

- [1] Martin Abadi and Luca Cardelli. *A Theory of Objects*. Monographs in Computer Science. Springer-Verlag, New York, NY, USA, August 1996.
- [2] Jean-Raymond Abrial. *The B Book: Assigning Programs to Meanings and Modeling in Event-B: System and Software Engineering*. Cambridge University Press, Cambridge, England, 1996 and 2009.
- [3] Mordecai Avriel, Michal Penn, and Naomi Shpirer. Container ship stowage problem: complexity and connection to the coloring of circle graphs. *Discrete Applied Mathematics*, 103(1–3):271–279, 15 July 2000. Faculty of Industrial Engineering and Management, Technion, Israel Institute of Technology, Haifa 3200, Israel.
- [4] Mordecai Avriel, Michal Penn, Naomi Shpirer, and Smadar Witteboon. Stowage planning for container ships to reduce the number of shifts. *Annals of Operations Research*, 76(9):55–71, January 1998.
- [5] H. Bekič, D. Bjørner, W. Henhagl, C. B. Jones, and P. Lucas. A Formal Definition of a PL/I Subset. Technical Report 25.139, IBM Laboratory, Vienna, December 1974.
- [6] Hans Bekič. On the Formal Definition of Programming Languages. In *International Computing Symposium, ACM Europe*, Bonn, Nov. 1970. GDM.
- [7] Hans Bekič. Programming Languages and Their Definition. In Cliff B. Jones, editor, *Lecture Notes in Computer Science, Vol. 177*. Springer, 1984.
- [8] Hans Bekič, Peter Lucas, Kurt Walk, and Many Others. Formal Definition of PL/I, ULD Version I. Technical report, IBM Laboratory, Vienna, 1966.
- [9] Hans Bekič, Peter Lucas, Kurt Walk, and Many Others. Formal Definition of PL/I, ULD Version II. Technical report, IBM Laboratory, Vienna, 1968.
- [10] Hans Bekič, Peter Lucas, Kurt Walk, and Many Others. Formal Definition of PL/I, ULD Version III. IBM Laboratory, Vienna, 1969.
- [11] Claude Berge. *Théorie des Graphes et ses Applications*. Collection Universitaire de Mathématiques. Dunod, Paris, 1958. See [12].
- [12] Claude Berge. *Graphs*, volume 6 of *Mathematical Library*. North-Holland Publ. Co., second revised edition of part 1 of the 1973 english version edition, 1985. See [11].
- [13] Sandford Bessler, Eva Kühn, Richard Mordinyi, and Slobodanka Tomic. Using tuple-spaces to manage the storage and dissemination of spatial-temporal content. *Journal of Computer and System Sciences*, page 10, February 2010. Link: <http://dx.doi.org/10.1016/j.jcss.2010.01.010>.

- [14] D. Bjørner. Stepwise Transformation of Software Architectures. In [73], chapter 11, pages 353–378. Prentice-Hall, 1982.
- [15] D. Bjørner and C. B. Jones. *Formal Specification and Software Development*. Prentice-Hall, 1982.
- [16] Dines Bjørner. Software Development Graphs — A Unifying Concept for Software Development? In K.V. Nori, editor, *Vol. 241 of Lecture Notes in Computer Science: Foundations of Software Technology and Theoretical Computer Science*, pages 1–9. Springer-Verlag, Dec. 1986.
- [17] Dines Bjørner. The Stepwise Development of Software Development Graphs: Meta-Programming VDM Developments. In *See [74]*, volume 252 of *LNCS*, pages 77–96. Springer-Verlag, Heidelberg, Germany, March 1987.
- [18] Dines Bjørner. Specification and Transformation: Methodology Aspects of the Vienna Development Method. In *TAPSOFT'89*, volume 352 of *Lab. Note*, pages 1–35. Springer-Verlag, Heidelberg, Germany, 1989.
- [19] Dines Bjørner. Software Systems Engineering — From Domain Analysis to Requirements Capture: An Air Traffic Control Example. In *2nd Asia-Pacific Software Engineering Conference (APSEC '95)*. IEEE Computer Society, 6–9 December 1995. Brisbane, Queensland, Australia.
- [20] Dines Bjørner. Formal Software Techniques in Railway Systems. In Eckehard Schnieder, editor, *9th IFAC Symposium on Control in Transportation Systems*, pages 1–12, Technical University, Braunschweig, Germany, 13–15 June 2000. VDI/VDE-Gesellschaft Mess- und Automatisierungstechnik, VDI-Gesellschaft für Fahrzeug- und Verkehrstechnik. Invited talk.
- [21] Dines Bjørner. Domain Models of "The Market" — in Preparation for E-Transaction Systems. In *Practical Foundations of Business and System Specifications (Eds.: Haim Kilov and Ken Baclawski)*, The Netherlands, December 2002. Kluwer Academic Press. [www2.imm.dtu.dk/dibj/themarket.pdf](http://www2.imm.dtu.dk/dibj/themarket.pdf).
- [22] Dines Bjørner. Dynamics of Railway Nets: On an Interface between Automatic Control and Software Engineering. In *CTS2003: 10th IFAC Symposium on Control in Transportation Systems*, Oxford, UK, August 4-6 2003. Elsevier Science Ltd. Symposium held at Tokyo, Japan. Editors: S. Tsugawa and M. Aoki. [www2.imm.dtu.dk/dibj/ifac-dynamics.pdf](http://www2.imm.dtu.dk/dibj/ifac-dynamics.pdf).
- [23] Dines Bjørner. New Results and Trends in Formal Techniques for the Development of Software for Transportation Systems. In *FORMS2003: Symposium on Formal Methods for Railway Operation and Control Systems*. Institut für Verkehrssicherheit und Automatisierungstechnik, Techn.Univ. of Braunschweig, Germany, 15–16 May 2003. Conf. held at Techn.Univ. of Budapest, Hungary. Editors: G. Tarnai and E. Schnieder, Germany. [www2.imm.dtu.dk/dibj/dines-amore.pdf](http://www2.imm.dtu.dk/dibj/dines-amore.pdf).
- [24] Dines Bjørner. The Grand Challenge – FAQs of the R&D of a Railway Domain Theory. In *IFIP World Computer Congress, Topical Days: TRain: The Railway Domain*, IFIP, Amsterdam, The Netherlands, 2004. Kluwer Academic Press.
- [25] Dines Bjørner. Towards a Formal Model of CyberRail. In *Building the Information Society, IFIP 18th World Computer Congress, Topical Sessions, 22–27 August, 2004, Toulouse, France — Ed. René Jacquot*, pages 657–664. Kluwer Academic Publishers, August 2004.
- [26] Dines Bjørner. *Software Engineering, Vol. 1: Abstraction and Modelling*. Texts in Theoretical Computer Science, the EATCS Series. Springer, 2006. See [31, 34].
- [27] Dines Bjørner. *Software Engineering, Vol. 2: Specification of Systems and Languages*. Texts in Theoretical Computer Science, the EATCS Series. Springer, 2006. Chapters 12–14 are primarily authored by Christian Krog Madsen. See [32, 35].

- [28] Dines Bjørner. *Software Engineering, Vol. 3: Domains, Requirements and Software Design*. Texts in Theoretical Computer Science, the EATCS Series. Springer, 2006. See [33, 36].
- [29] Dines Bjørner. A Container Line Industry Domain. [www.imm.dtu.dk/db/container-paper.pdf](http://www.imm.dtu.dk/db/container-paper.pdf). Techn. report, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, Denmark, June 2007.
- [30] Dines Bjørner. Domain Engineering. In *The 2007 Lipari PhD Summer School*, Dds. E. Börger and A. Ferro, pages 1–102. University of Catania, Sicily, Italy, 2007. [www.imm.dtu.dk/dibj/lipari-paper.pdf](http://www.imm.dtu.dk/dibj/lipari-paper.pdf).
- [31] Dines Bjørner. *Software Engineering, Vol. 1: Abstraction and Modelling*. Qinghua University Press, 2008.
- [32] Dines Bjørner. *Software Engineering, Vol. 2: Specification of Systems and Languages*. Qinghua University Press, 2008.
- [33] Dines Bjørner. *Software Engineering, Vol. 3: Domains, Requirements and Software Design*. Qinghua University Press, 2008.
- [34] Dines Bjørner. **Chinese:** *Software Engineering, Vol. 1: Abstraction and Modelling*. Qinghua University Press. Translated by Dr Liu Bo Chao et al., 2010.
- [35] Dines Bjørner. **Chinese:** *Software Engineering, Vol. 2: Specification of Systems and Languages*. Qinghua University Press. Translated by Dr Liu Bo Chao et al., 2010.
- [36] Dines Bjørner. **Chinese:** *Software Engineering, Vol. 3: Domains, Requirements and Software Design*. Qinghua University Press. Translated by Dr Liu Bo Chao et al., 2010.
- [37] Dines Bjørner. Domain Engineering. In Paul Boca and Jonathan Bowen, editors, *Formal Methods: State of the Art and New Directions*, Eds. Paul Boca and Jonathan Bowen, pages 1–42, London, UK, 2010. Springer.
- [38] Dines Bjørner. Domain Science & Engineering – *From Computer Science to The Sciences of Informatics, Part I of II: The Engineering Part*. *Kibernetika i sistemny analiz*, 2(4):100–116, May 2010.
- [39] Dines Bjørner. On Development of Web-based Software: A Divertimento of Ideas and Suggestions. Technical, Technical University of Vienna, August–October 2010. [www.imm.dtu.dk/dibj/wfdftp.pdf](http://www.imm.dtu.dk/dibj/wfdftp.pdf).
- [40] Dines Bjørner. The Tokyo Stock Exchange Trading Rules [www.imm.dtu.dk/db/todai/tse-1.pdf](http://www.imm.dtu.dk/db/todai/tse-1.pdf), [www.imm.dtu.dk/db/todai/tse-2.pdf](http://www.imm.dtu.dk/db/todai/tse-2.pdf). R&D Experiment, Techn. Univ. of Denmark, Fredsvej 11, DK-2840 Holte, Denmark, 2010.
- [41] Dines Bjørner. Domain Science & Engineering – *From Computer Science to The Sciences of Informatics Part II of II: The Science Part*. *Kibernetika i sistemny analiz*, 2(3):100–120, June 2011.
- [42] Dines Bjørner. Domains: Their Simulation, Monitoring and Control – A Divertimento of Ideas and Suggestions. In *Rainbow of Computer Science, Festschrift for Hermann Maurer on the Occasion of His 70th Anniversary.*, Festschrift (eds. C. Calude, G. Rozenberg and A. Saloma), pages 167–183. Springer, Heidelberg, Germany, January 2011. [www.imm.dtu.dk/dibj/maurer-bjorner.pdf](http://www.imm.dtu.dk/dibj/maurer-bjorner.pdf).
- [43] Dines Bjørner. Domains: Their Simulation, Monitoring and Control – A Divertimento of Ideas and Suggestions. In *Rainbow of Computer Science, Festschrift for Hermann Maurer on the Occasion of His 70th Anniversary.*, Festschrift (eds. C. Calude, G. Rozenberg and A. Saloma), pages 167–183. Springer, Heidelberg, Germany, January 2011. [www.imm.dtu.dk/dibj/maurer-bjorner.pdf](http://www.imm.dtu.dk/dibj/maurer-bjorner.pdf).

- [44] Dines Bjørner. Documents – a Domain Description. Experimental Research Report 2013-3, DTU Compute and Fredsvej 11, DK-2840 Holte, Denmark, Spring 2013.
- [45] Dines Bjørner. Pipelines – a Domain [www.imm.dtu.dk/ dibj/pipe-p.pdf](http://www.imm.dtu.dk/dibj/pipe-p.pdf). Experimental Research Report 2013-2, DTU Compute and Fredsvej 11, DK-2840 Holte, Denmark, Spring 2013.
- [46] Dines Bjørner. Road Transportation – a Domain Description [www.imm.dtu.dk/ dibj/road-p.pdf](http://www.imm.dtu.dk/dibj/road-p.pdf). Experimental Research Report 2013-4, DTU Compute and Fredsvej 11, DK-2840 Holte, Denmark, Spring 2013.
- [47] Dines Bjørner. A Rôle for Mereology in Domain Science and Engineering. In *Mereology and the Sciences*, Synthese Library (eds. Claudio Calosi and Pierluigi Graziani), pages 323–357, Amsterdam, The Netherlands, October 2014. Springer. [https://www.imm.dtu.dk/ dibj/2011/urbino/urbino-colour.pdf](https://www.imm.dtu.dk/dibj/2011/urbino/urbino-colour.pdf).
- [48] Dines Bjørner. A Credit Card System: Uppsala Draft [www.imm.dtu.dk/ dibj/2016/credit/accs.pdf](http://www.imm.dtu.dk/dibj/2016/credit/accs.pdf). Technical Report: Experimental Research, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, Denmark, November 2016.
- [49] Dines Bjørner. Domains: Their Simulation, Monitoring and Control – A Divertimento of Ideas and Suggestions. Extensive revision of [42]. [www.imm.dtu.dk/ dibj/2016/demos/faoc-demo.pdf](http://www.imm.dtu.dk/dibj/2016/demos/faoc-demo.pdf). Technical report, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, Denmark, 2016.
- [50] Dines Bjørner. Weather Information Systems: Towards a Domain Description [www.imm.dtu.dk/ dibj/2016/wis/wis-p.pdf](http://www.imm.dtu.dk/dibj/2016/wis/wis-p.pdf). Technical Report: Experimental Research, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, Denmark, November 2016.
- [51] Dines Bjørner. Manifest Domains: Analysis & Description [www.imm.dtu.dk/ dibj/2015/faoc/faoc-bjorner.pdf](http://www.imm.dtu.dk/dibj/2015/faoc/faoc-bjorner.pdf). *Formal Aspects of Computing*, 29(2):175–225, March 2017. Online: 26 July 2016.
- [52] Dines Bjørner. Domain analysis & description - the implicit and explicit semantics problem [www.imm.dtu.dk/ dibj/2017/bjorner-impex.pdf](http://www.imm.dtu.dk/dibj/2017/bjorner-impex.pdf). In Régine Laleau, Dominique Méry, Shin Nakajima, and Elena Troubitsyna, editors, Proceedings Joint Workshop on *Handling IMPLICIT and EXPLICIT knowledge in formal system development (IMPEX) and Formal and Model-Driven Techniques for Developing Trustworthy Systems (FM&MDD)*, Xi’An, China, 16th November 2017, volume 271 of *Electronic Proceedings in Theoretical Computer Science*, pages 1–23. Open Publishing Association, 2018.
- [53] Dines Bjørner. Domain Facets: Analysis & Description. Extensive revision of [37]. [www.imm.dtu.dk/ dibj/2016/facets/faoc-facets.pdf](http://www.imm.dtu.dk/dibj/2016/facets/faoc-facets.pdf). Technical report, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, Denmark, May 2018.
- [54] Dines Bjørner. To Every Manifest Domain a CSP Expression [www.imm.dtu.dk/ dibj/2016/mereo/mereo.pdf](http://www.imm.dtu.dk/dibj/2016/mereo/mereo.pdf). *Journal of Logical and Algebraic Methods in Programming*, 1(94):91–108, January 2018.
- [55] Dines Bjørner. Domain Analysis & Description – Principles, Techniques and Modeling Languages. [www.imm.dtu.dk/ dibj/2018/tosem/Bjorner-TOSEM.pdf](http://www.imm.dtu.dk/dibj/2018/tosem/Bjorner-TOSEM.pdf). *ACM Trans. on Software Engineering and Methodology*, 28(2):66 pages, March 2019.
- [56] Dines Bjørner. Domain Analysis & Description – Principles, Techniques and Modelling Languages. [www.imm.dtu.dk/ dibj/2018/tosem/Bjorner-TOSEM.pdf](http://www.imm.dtu.dk/dibj/2018/tosem/Bjorner-TOSEM.pdf). *ACM Trans. on Software Engineering and Methodology*, 28(2):1–67, April 2019. 68 pages.
- [57] Dines Bjørner. Domain Analysis & Description: Sorts, Types, Intents. [www.imm.dtu.dk/ dibj/2019/ty+so/HavelundFestschriftOctober2020.pdf](http://www.imm.dtu.dk/dibj/2019/ty+so/HavelundFestschriftOctober2020.pdf). Technical report, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, Denmark, November 2019. Paper for Klaus Havelund Festschrift, October 2020.

- [58] Dines Bjørner. *Domain Science & Engineering – A Foundation for Software Development*. EATCS Monographs in Theoretical Computer Science. Springer, Heidelberg, Germany, 2021. A revised version of this book is [64].
- [59] Dines Bjørner. *Rigorous Domain Descriptions. A compendium of draft domain description sketches carried out over the years 1995–2021. Chapters cover: Graphs, Railways, Road Transport The “7 Seas”, The “Blue Skies”, Credit Cards Weather Information, Documents, Urban Planning, Swarms of Drones, Container Terminals, A Retailer Market, Shipping, Rivers, Canals, Stock Exchangew, and Web Transactions. This document is currently being edited.* Own: [www.imm.dtu.dk/~dibj/2021/dd/dd.pdf](http://www.imm.dtu.dk/~dibj/2021/dd/dd.pdf), Fredsvej 11, DK-2840 Holte, Denmark, November 15, 2021.
- [60] Dines Bjørner. Shipping. [www.imm.dtu.dk/~dibj/2021/ra1/ra1.pdf](http://www.imm.dtu.dk/~dibj/2021/ra1/ra1.pdf). Technical Report, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, Denmark, April 2021.
- [61] Dines Bjørner. Domain Modelling. Research report, Technical University of Denmark, DK-2800 Lyngby, Denmark, 2023.
- [62] Dines Bjørner. Domain Modelling – A Foundation for Software Development. In Jonathan Bowen et al., editor, *Theories of Programming and Formal Methods: Essays Dedicated to Jifeng He on the Occasion of His 80th Birthday*, Lecture Notes in Computer Science, Festschrift. Springer, Heidelberg, Germany, August 2023. <https://www.imm.dtu.dk/~dibj/2023/FEA/hjf.pdf> and <https://www.imm.dtu.dk/~dibj/2023/final/HeJiFeng.pdf>.
- [63] Dines Bjørner. Domain Modelling – A Primer. A short version of [64]. xii+202 pages<sup>1</sup>, May 2023.
- [64] Dines Bjørner. Domain Science & Engineering – A Foundation for Software Development. Revised edition of [58]. xii+346 pages<sup>2</sup>, January 2023.
- [65] Dines Bjørner. [67] Chap. 10: *Towards a Family of Script Languages – Licenses and Contracts – Incomplete Sketch*, pages 283–328. JAIST Press, March 2009.
- [66] Dines Bjørner. [67] Chap. 7: *Documents – A Rough Sketch Domain Analysis*, pages 179–200. JAIST Press, March 2009.
- [67] Dines Bjørner. *Domain Engineering: Technology Management, Research and Engineering*. A JAIST Press Research Monograph # 4, 536 pages, March 2009.
- [68] Dines Bjørner and Asger Eir. Compositionality: Ontology and Mereology of Domains. Some Clarifying Observations in the Context of Software Engineering in July 2008, eds. Martin Steffen, Dennis Dams and Ulrich Hannemann. In *Festschrift for Prof. Willem Paul de Roever Concurrency, Compositionality, and Correctness*, volume 5930 of *Lecture Notes in Computer Science*, pages 22–59, Heidelberg, July 2010. Springer.
- [69] Dines Bjørner, Chris W. George, and Søren Prehn. Computing Systems for Railways — A Rôle for Domain Engineering. Relations to Requirements Engineering and Software for Control Applications. In *Integrated Design and Process Technology*. Editors: Bernd Kraemer and John C. Petterson, P.O.Box 1299, Grand View, Texas 76050-1299, USA, 24–28 June 2002. Society for Design and Process Science. [www2.imm.dtu.dk/~dibj/pasadena-25.pdf](http://www2.imm.dtu.dk/~dibj/pasadena-25.pdf).
- [70] Dines Bjørner, Christian Gram, Ole N. Oest, and Leif Rystrøm. Dansk Datamatik Center. In *3rd IFIP WG 9.7 Working Conference on History of Nordic Computing*, IFIP Advances in Information and Communication Technology, pages 2–34. Springer, 2010.

<sup>1</sup>This book is currently being translated into Chinese by Dr. Yang ShaoFa, IoS/CAS, Beijing and into Russian by Dr. Mikhail Chupilko, ISP/RAS, Moscow

<sup>2</sup>Due to copyright reasons no URL is given to this document’s possible Internet location. A primer version, omitting certain chapters, is [63]

- [71] Dines Bjørner and Cliff B. Jones, editors. *The Vienna Development Method: The Meta-Language*, volume 61 of *LNCS*. Springer, 1978. This was the first monograph on *Meta-IV*.
- [72] Dines Bjørner and Cliff B. Jones, editors. *The Vienna Development Method: The Meta-Language*, volume 61 of *LNCS*. Springer, Heidelberg, Germany, 1978.
- [73] Dines Bjørner and Cliff B. Jones, editors. *Formal Specification and Software Development*. Prentice-Hall, London, England, 1982.
- [74] Dines Bjørner, Cliff B. Jones, Micheal Mac an Airchinnigh, and Erich J. Neuhold, editors. *VDM – A Formal Method at Work*. Proc. VDM-Europe Symposium 1987, Brussels, Belgium, Springer, Lecture Notes in Computer Science, Vol. 252, March 1987.
- [75] Dines Bjørner, Søren Prehn, and Chris W. George. Formal Models of Railway Systems: Domains. Technical report, Dept. of IT, Technical University of Denmark, Bldg. 344, DK-2800 Lyngby, Denmark, September 23 1999. Presented at the FME Rail Workshop on Formal Methods in Railway Systems, FM'99 World Congress on Formal Methods, Toulouse, France. Available on CD ROM.
- [76] Dines Bjørner, Søren Prehn, and Chris W. George. Formal Models of Railway Systems: Requirements. Technical report, Dept. of IT, Technical University of Denmark, Bldg. 344, DK-2800 Lyngby, Denmark, September 23 1999. Presented at the FME Rail Workshop on Formal Methods in Railway Systems, FM'99 World Congress on Formal Methods, Toulouse, France. Available on CD ROM.
- [77] Nikolaj Bjørner, Maxwell Levatch, Nuno P. Lopes, Andrey Rybalchenko, and Chandrasekar Vuppalapati. Supercharging plant configurations using Z3. In Peter J. Stuckey, editor, *Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 18th International Conference, CPAIOR 2021, Vienna, Austria, July 5-8, 2021, Proceedings*, volume 12735 of *Lecture Notes in Computer Science*, pages 1–25. Springer, 2021.
- [78] Dines Bjørner. Urban Planning Processes. [www.imm.dtu.dk/dibj/2017/up/urban-planning.pdf](http://www.imm.dtu.dk/dibj/2017/up/urban-planning.pdf). Research Note, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, Denmark, July 2017.
- [79] J.A. Bondy and U.S.R. Murty. *Graph Theory with Applications*. American Elsevier, N.Y. and MacMillan, London, 1976.
- [80] Bram Borgman, Eelco van Asperen, and Rommert Dekker. Online rules for container stacking. *OR Spectrum*, 32:687–716, 19 March 2010.
- [81] Roberto Casati and Achille C. Varzi. *Parts and Places: the structures of spatial representation*. MIT Press, 1999.
- [82] Patrick Cousot. *Principles of Abstract Interpretation*. The MIT Press, 2021.
- [83] Patrick Cousot and Rhadia Cousot. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *4th POPL: Principles of Programming and Languages*, pages 238–252. ACM Press, 1977.
- [84] Stefan Craß. A Formal Model of the Extensible Virtual Shared Memory (xvsm) and its Implementation in Haskell – Design and Specification. M.sc., Technische Universität Wien, A-1040 Wien, Karlsplatz 13, Austria, February 5 2010.
- [85] Stefan Craß, Eva Kühn, and Gernot Salzer. Algebraic Foundation of a Data Model for an Extensible Space-based Collaboration Protocol. In Bipin C. Desai, editor, *IDEAS 2009*, pages 301–306, Cetraro, Calabria, Italy, September 16–18 2009.

- [86] Razvan Diaconescu and Kokichi Futatsugi. *CafeOBJ Report: The Language, Proof Techniques, and Methodologies for Object-Oriented Algebraic Specification*. AMAST Series in Computing - Vol. 6. World Scientific Publishing Co., Pte. Ltd., 5 Toh Tuck Link, Singapore 596224, July 1998. 196pp, ISBN 981-02-3513-5, US\$30.
- [87] Opher Dubrovsky, Gregory Levitin, and Michal Penn. A genetic algorithm with a compact solution encoding for the container ship stowage problem. *Journal of Heuristics*, 8(6):585–599, November 2002.
- [88] S. Even. *Graph Algorithms*. Computer Science Press, Md., USA, 1979.
- [89] Bureau Export. A-Z Dictionary of Export, Trade and Shipping Terms. [www.exportbureau.com/-trade\\_shipping\\_terms/dictionary.html](http://www.exportbureau.com/-trade_shipping_terms/dictionary.html), 2007.
- [90] Peter Fettke and Wolfgang Reisig. Modelling service-oriented systems and cloud services with HERAKLIT. *CoRR*, abs/2009.14040, 2020.
- [91] Peter Fettke and Wolfgang Reisig. HERAKLIT – epistemologically motivated modeling of computer-integrated systems. HERAKLIT working paper, v1, December 15, 2020, <http://www.heraklit.org>, 2020.
- [92] Peter Fettke and Wolfgang Reisig. HERAKLIT case study: 8-second hell. HERAKLIT working paper, v1, December 12, 2020, <http://www.heraklit.org>, 2020.
- [93] Peter Fettke and Wolfgang Reisig. HERAKLIT case study: adder. HERAKLIT working paper, v1, December 5, 2020, <http://www.heraklit.org>, 2020.
- [94] Peter Fettke and Wolfgang Reisig. HERAKLIT case study: parallel adder. HERAKLIT working paper, v1, December 5, 2020, <http://www.heraklit.org>, 2020.
- [95] Peter Fettke and Wolfgang Reisig. HERAKLIT case study: retailer. HERAKLIT working paper, v1, December 21, 2020, <http://www.heraklit.org>, 2020.
- [96] Peter Fettke and Wolfgang Reisig. HERAKLIT case study: service system. HERAKLIT working paper, v1, November 20, 2020, <http://www.heraklit.org>, 2020.
- [97] John Fitzgerald and Peter Gorm Larsen. *Modelling Systems – Practical Tools and Techniques in Software Development*. Cambridge University Press, The Edinburgh Building, Cambridge CB2 2RU, UK, 1998. ISBN 0-521-62348-0.
- [98] John Fitzgerald and Peter Gorm Larsen. *Modelling Systems – Practical Tools and Techniques in Software Development*. Cambridge University Press, The Edinburgh Building, Cambridge CB2 2RU, UK, 1998. ISBN 0-521-62348-0.
- [99] Kokichi Futatsugi. Advances of proof scores in CafeOBJ. *Science of Computer Programming*, 224, December 2022.
- [100] Chris W. George, Peter Haff, Klaus Havelund, Anne Elisabeth Haxthausen, Robert Milne, Claus Bendix Nielsen, Søren Prehn, and Kim Ritter Wagner. *The RAISE Specification Language*. The BCS Practitioner Series. Prentice-Hall, Hemel Hempstead, England, 1992.
- [101] Chris W. George, Anne Elisabeth Haxthausen, Steven Hughes, Robert Milne, Søren Prehn, and Jan Storbak Pedersen. *The RAISE Development Method*. The BCS Practitioner Series. Prentice-Hall, Hemel Hempstead, England, 1995.
- [102] Jean-Yves Girard, Y. Lafont, and P. Taylor. *Proofs and Types*, volume 7. Cambridge Univ. Press, Cambridge, UK, Cambridge Tracts in Theoretical Computer Science edition, 1989.

- [103] Michael Hammer and James A. Champy. *Reengineering the Corporation: A Manifesto for Business Revolution*. HarperCollinsPublishers, 77–85 Fulham Palace Road, Hammersmith, London W6 8JB, UK, May 1993. 5 June 2001, Paperback.
- [104] Michael Hammer and Stephen A. Stanton. *The Reengineering Revolution: The Handbook*. HarperCollinsPublishers, 77–85 Fulham Palace Road, Hammersmith, London W6 8JB, UK, 1996. Paperback.
- [105] Michael Reichhardt Hansen and Hans Rischel. *Functional Programming in Standard ML*. Addison Wesley, 1997.
- [106] R. Harper, D. MacQueen, and R. Milner. Standard ML. Technical Report ECS-LFCS-86-2, Lab. f. Found. of Comp. Sci., Dept. of Comp. Sci., Univ. of Edinburgh, Scotland, 1986.
- [107] Frank Harray. *Graph Theory*. Addison Wesley Publishing Co., 1972.
- [108] Charles Anthony Richard Hoare. Communicating Sequential Processes. *Communications of the ACM*, 21(8), Aug. 1978.
- [109] Charles Anthony Richard Hoare. Communicating Sequential Processes. *Communications of the ACM*, 21(8), Aug. 1978.
- [110] Charles Anthony Richard Hoare. *Communicating Sequential Processes*. C.A.R. Hoare Series in Computer Science. Prentice-Hall International, London, England, 1985. Published electronically: [usingcsp.com/cspbook.pdf](http://usingcsp.com/cspbook.pdf) (2004).
- [111] Charles Anthony Richard Hoare. *Communicating Sequential Processes*. C.A.R. Hoare Series in Computer Science. Prentice-Hall International, 1985.
- [112] Charles Anthony Richard Hoare. Communicating Sequential Processes. Published electronically: [usingcsp.com/cspbook.pdf](http://usingcsp.com/cspbook.pdf), 2004. Second edition of [111]. See also [usingcsp.com/](http://usingcsp.com/).
- [113] Gerard J. Holzmann. *The SPIN Model Checker, Primer and Reference Manual*. Addison-Wesley, Reading, Massachusetts, 2003.
- [114] Akio Imai, Kazuya Sasaki, Etsuko Nishimura, and Stratos Papadimitriou. Multi-objective simultaneous stowage and load planning for a container ship with container rehandle in yard stacks. *European Journal of Operational Research*, 171:373–389, 2006.
- [115] Daniel Jackson. *Software Abstractions: Logic, Language, and Analysis*. The MIT Press, Cambridge, Mass., USA, April 2006. ISBN 0-262-10114-9.
- [116] Michael A. Jackson. *Software Requirements & Specifications: a lexicon of practice, principles and prejudices*. ACM Press. Addison-Wesley, Reading, England, 1995.
- [117] Michael A. Jackson and Graham Twaddle. *Business Process Implementation — Building Workflow Systems*. Addison-Wesley, 1997.
- [118] C. B. Jones. *Software Development: A Rigorous Approach*. Prentice-Hall, 1980.
- [119] C. B. Jones. *Systematic Software Development — Using VDM*. Prentice-Hall, 1986.
- [120] C. B. Jones. *Systematic Software Development — Using VDM, 2nd Edition*. Prentice-Hall, 1989.
- [121] J.W. Backus and F.L. Bauer and J.Green and C. Katz and J. McCarthy and P. Naur and A.J. Perlis and H. Rutishauser and K. Samelson and B. Vauquois and J.H. Wegstein and A. van Wijngaarden and M. Woodger. Revised Report on the Algorithmic Language Algol 60 – edited by P. Naur. *The Computer Journal*, 5(4):349367, 1963.



- [122] Eva Kühn, Richard Mordinyi, László Keszthelyi, and Christian Schreiber. Introducing the Concept of Customizable Structured Space for Agent Coordination in the Production of Automation Domain. In Sierra Decker, Sichman and Castelfranchi, editors, *8<sup>th</sup> Intl. Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS 2009)*, volume 625–632 of *Proceedings of Autonomous Agents and Multi-Agent Systems*, Budapest, Hungary, May 10–15 2009. 8.
- [123] Eva Kühn, Richard Mordinyi, László Keszthelyi, Christian Schreiber, Sandford Bessler, and Slobodanka Tomic. Aspect-oriented Space Containers for Efficient Publish/Subscribe Scenarios in Intelligent Transportation Systems. In T. Dillon and P. HereroR. Meersmann, editors, *OTM 2009, Part I*, volume 5870 of *LNCS*, pages 432–448. Springer, 2009.
- [124] Peter J. Landin. The Next 700 Programming Languages. *Communications of the ACM*, 9(3):157–166, 1966.
- [125] J.A.N. Lee and W. Delmore. The Vienna Definition Language, a generalization of instruction definitions. In *SIGPLAN Symp. on Programming Language Definitions*, San Francisco, Aug. 1969.
- [126] Morten Lind. An introduction to multilevel flow modeling. *International Journal of Nuclear Safety and Simulation*, 2(2):22–32, 2011.
- [127] W. Little, H.W. Fowler, J. Coulson, and C.T. Onions. *The Shorter Oxford English Dictionary on Historical Principles*. Clarendon Press, Oxford, England, 1973, 1987. Two vols.
- [128] P. Lucas. Formal Definition of Programming Languages and Systems. In *Proc. IFIP'71*. IFIP World Congress Proceedings, Springer, 1971.
- [129] P. Lucas. On the Semantics of Programming Languages and Software Devices. In Rustin, editor, *Formal Semantics of Programming Languages*. Prentice-Hall, 1972.
- [130] P. Lucas. On the formalization of programming languages: Early history and main approaches. In D. Bjørner and C. B. Jones, editors, [71]. Springer, 1978.
- [131] P. Lucas. Formal Semantics of Programming Languages: VDL. *IBM Journal of Devt. and Res.*, 25(5):549–561, 1981.
- [132] P. Lucas. Main approaches to formal specification. In [14], chapter 1, pages 3–24. Prentice-Hall, 1982.
- [133] P. Lucas. Origins, hopes, and achievements. In [74], pages 1–18. Springer, 1987.
- [134] P. Lucas and K. Walk. On the Formal Description of PL/I. *Annual Review Automatic Programming Part 3*, 6(3), 1969.
- [135] E.C. Luschei. *The Logical Systems of Leśniewski*. North Holland, Amsterdam, The Netherlands, 1962.
- [136] ANSI X3.9-1966. The Fortran programming language. Technical report, American National Standards Institute, Standards on Computers and Information Processing, 1966.
- [137] John McCarthy, Paul W. Abrahams, Daniel J. Edwards, Timothy P. Hart, and Michael I. Levin. *LISP 1.5 Programmer's Manual*. The MIT Press, Cambridge, Mass., 1962.
- [138] J. A. McDermid and P. Whysall. *Formal System Specification and Implementation using Z*. International Series in Computer Science. Prentice Hall, Hemel Hempstead, Hertfordshire, UK, 1992. Withdrawn.
- [139] Usama Mehmood, Radu Grosu, Ashish Tiwari, Nicola Paoletti, Shan Lin, Yang JunXing, Dung Phan, Scott D. Stoller, and Scott A. Smolka. *Declarative vs Rule-based Control for Flocking Dynamics*. In *Proceedings of ACM/SIGAPP Symposium on Applied Computing (SACC 2018)*. ACM Press, April 9–13, 2018. 8 pages.

- [140] J. H. Morris. Types are not Sets. In *Proc. ACM Symposium on Principles of Programming Languages (PoPL)*, pages 120–124. ACM, Boston, 1973.
- [141] Peter D. Mosses, editor. *CASL Reference Manual*, volume 2960 of *LNCS, IFIP Series*. Springer-Verlag, Heidelberg, Germany, 2004. Part I (Summary) and Part II (Syntax): Peter Mosses; Part III (Semantics): Don Sannella, and Andrzej Tarlecki; Parts IV (Logic), V (Refinement) and VI (Libraries): Till Mossakowski.
- [142] Lev Nachmanson. Microsofts Automated Layout Tool. Technical report, MS Research, 2021. <https://github.com/microsoft/automated-graph-layout>.
- [143] Reza Olfati-Saber. Flocking for Multi-agent Dynamic Systems: Algorithms and Theory. *IEEE Transactions on Automatic Control*, 51(3):401–420, 13 March 2006. <http://ieeexplore.ieee.org/document/1605401/>; DOI: 10.1109/TAC.2005.864190; Thayer School of Engineering, Dartmouth College, Hanover, NH, USA.
- [144] Oystein Ore. *Graphs and their Uses*. The Mathematical Association of America, 1963.
- [145] International Labour Organisation. Portworker Development Programme: PDP Units. Enumerate PDP units. , April 2002.
- [146] Benjamin Pierce. *Types and Programming Languages*. The MIT Press, 2002.
- [147] Karl R. Popper. *Conjectures and Refutations. The Growth of Scientific Knowledge*. Routledge and Kegan Paul Ltd. (Basic Books, Inc.), 39 Store Street, WC1E 7DD, London, England (New York, NY, USA), 1963, . . . ,1981.
- [148] B. F. Potter, J. E. Sinclair, and D. Till. *An Introduction to Formal Specification and Z*. Prentice Hall International Series in Computer Science, 1991.
- [149] Martin Pěnička and Dines Bjørner. From Railway Resource Planning to Train Operation — a Brief Survey of Complementary Formalisations. In *Building the Information Society, IFIP 18th World Computer Congress, Topical Sessions, 22–27 August, 2004, Toulouse, France — Ed. Renée Jacquot*, pages 629–636. Kluwer Academic Publishers, August 2004.
- [150] Martin Pěnička, Albena Kirilova Strupchanska, and Dines Bjørner. Train Maintenance Routing. In *FORMS'2003: Symposium on Formal Methods for Railway Operation and Control Systems*. L'Harmattan Hongrie, 15–16 May 2003. Conf. held at Techn.Univ. of Budapest, Hungary. Editors: G. Tarnai and E. Schnieder, Germany. [www2.imm.dtu.dk/~dibj/martin.pdf](http://www2.imm.dtu.dk/~dibj/martin.pdf).
- [151] K.V. Ramani. An interactive simulation model for the logistics planning of container operations in seaports. *SIMULATION*, 66(5):291–300, 1996.
- [152] Wolfgang Reisig. *Petri Nets: An Introduction*, volume 4 of *EATCS Monographs in Theoretical Computer Science*. Springer Verlag, May 1985.
- [153] Wolfgang Reisig. *A Primer in Petri Net Design*. Springer Verlag, March 1992. 120 pages.
- [154] Wolfgang Reisig. The Expressive Power of Abstract State Machines. *Computing and Informatics*, 22(1–2), 2003.
- [155] Wolfgang Reisig. *Petrinetze: Modellierungstechnik, Analysemethoden, Fallstudien*. Leitfäden der Informatik. Vieweg+Teubner, 1st edition, 15 June 2010. 248 pages; ISBN 978-3-8348-1290-2.
- [156] Wolfgang Reisig. *Understanding Petri Nets Modeling Techniques, Analysis Methods, Case Studies*. Springer, 2013. 230+XXVII pages, 145 illus.
- [157] Craig Reynolds. *Flocks, Herds and Schools: A Distributed Behavioral Model*. *SIGGRAPH Computer Graphics*, 21(4), August 1987. <https://doi.org/10.1145/37402.37406>.

- [158] Craig Reynolds. *Steering Behaviors for Autonomous Characters*. In *Proceedings of Game Developers Conference*, pages 763–782, 1999.
- [159] Craig Reynolds. OpenSteer, *Steering Behaviours for Autonomous Characters*, 2004. <http://opensteer.sourceforge.net>.
- [160] John C. Reynolds. Types, abstraction and parametric polymorphism. In R.E.A. Mason, editor, *Proc. IFIP World Computer Congress*, pages 512–523. Elsevier Sci.Publ. (North-Holland), 1983.
- [161] A. W. Roscoe. *Theory and Practice of Concurrency*. C.A.R. Hoare Series in Computer Science. Prentice-Hall, 1997. <http://www.comlab.ox.ac.uk/people/bill.roscoe/publications/68b.pdf>.
- [162] Douglas T. Ross. Toward foundations for the understanding of type. In *Proceedings of the 1976 conference on Data: Abstraction, definition and structure*, pages 63–65, New York, NY, USA, 1976. ACM. <http://doi.acm.org/10.1145/800237.807120>.
- [163] David A. Schmidt. *The Structure of Typed Programming Languages*. MIT Press, 1994. ISBN 0262193493.
- [164] Steve Schneider. *Concurrent and Real-time Systems — The CSP Approach*. Worldwide Series in Computer Science. John Wiley & Sons, Ltd., Baffins Lane, Chichester, West Sussex PO19 1UD, England, January 2000.
- [165] M. Schönfinkel. On the Building Blocks of Mathematical Logic. In [178]. Harvard University Press, 1967.
- [166] D.S. Scott. Lattice theory, data types and semantics. In R. Rustin, editor, *Symposium on Formal Semantics*, pages 67–106. Prentice-Hall, 1972.
- [167] Kai Sørlander. *Det Uomgængelige – Filosofiske Deduktioner [The Inevitable – Philosophical Deductions, with a foreword by Georg Henrik von Wright]*. Munksgaard · Rosinante, Copenhagen, Denmark, 1994. 168 pages.
- [168] Kai Sørlander. *Under Evighedens Synsvinkel [Under the viewpoint of eternity]*. Munksgaard · Rosinante, Copenhagen, Denmark, 1997. 200 pages.
- [169] Kai Sørlander. *Den Endegyldige Sandhed [The Final Truth]*. Rosinante, Copenhagen, Denmark, 2002. 187 pages.
- [170] Kai Sørlander. *Indføring i Filosofien [Introduction to The Philosophy]*. Informations Forlag, Copenhagen, Denmark, 2016. 233 pages.
- [171] Kai Sørlander. *Den rene fornufts struktur [The Structure of Pure Reason]*. Ellekær, Slagelse, Denmark, 2022. See [172].
- [172] Kai Sørlander. *The Structure of Pure Reason*. Publisher to be decided, 2023. This is an English translation of [171] – done by Dines Bjørner in collaboration with the author.
- [173] J. M. Spivey. *Understanding Z: A Specification Language and its Formal Semantics*, volume 3 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, January 1988.
- [174] J. M. Spivey. *The Z Notation: A Reference Manual*. Prentice Hall International Series in Computer Science, 2nd edition, 1992.
- [175] Dirk Steenken, Stefan Voß, and Robert Stahlbock. Container terminal operation and operations research - a classification and literature review. *OR Spectrum*, 26(1):3–49, January 2004.

- [176] Albena Kirilova Strupchanska, Martin Pěnička, and Dines Bjørner. Railway Staff Rostering. In *FORMS2003: Symposium on Formal Methods for Railway Operation and Control Systems*. L'Harmattan Hongrie, 15–16 May 2003. Conf. held at Techn.Univ. of Budapest, Hungary. Editors: G. Tarnai and E. Schnieder, Germany. [www2.imm.dtu.dk/~dibj/albena.pdf](http://www2.imm.dtu.dk/~dibj/albena.pdf).
- [177] Tetsuo Tamai. Social Impact of Information System Failures. *Computer, IEEE Computer Society Journal*, 42(6):58–65, June 2009.
- [178] J. van Heijenoort. *From Frege to Gödel — a Source Book in Mathematical Logic*. Harvard University Press, 1967.
- [179] Achille C. Varzi. *On the Boundary between Mereology and Topology*, pages 419–438. Hölder-Pichler-Tempsky, Vienna, 1994.
- [180] I.D. Wilson and P.A. Roach. Container stowage planning: a methodology for generating computerised solutions. *Journal of the Operational Research Society*, 51(11):1248–1255, 1 November 2000. Palgrave Macmillan. University of Glamorgan, UK.
- [181] I.D. Wilson, P.A. Roach, and J. A. Ware. Container stowage pre-planning: using search to generate solutions, a case study. *Knowledge-Based Systems*, 14(3–4):137–145, June 2001.
- [182] J. C. P. Woodcock. *Using Standard Z*. International Series in Computer Science. Prentice Hall, Hemel Hempstead, Hertfordshire, UK, 1993. In preparation.
- [183] James Charles Paul Woodcock and James Davies. *Using Z: Specification, Proof and Refinement*. Prentice Hall International Series in Computer Science, London, England, 1996.

## Part III

# APPENDIX

- The appendix contains three kinds of descriptions:
  - In part IV on page 81 four conceptual models.
  - In part V on page 169 eleven concrete models.
  - In part VI on page 499 four “system” models.
- We refer to these parts’ introductions.



## Part IV

# Conceptual Domain Models

In Chapters A–D we present models of four conceptual models. By a model of a conceptual domain we mean a description which abstracts properties of a number of concrete domains – or of a natural domain, that is, some non-man-made domain.

We briefly characterize these here.

- **Chapter A: Graphs** pages 83–112  
Draft from February 2021. We cover some aspects, in “our” style of domain modelling, of “classical” graph theory. This chapter overlaps with chapters B, C, and D.
- **Chapter B: Rivers** pages 113–120  
Draft from April 2021. Having drafted chapter A, I singled out ‘rivers’ for a separate inquiry.
- **Chapter C: Canals** pages 121–153  
First drafts from 2007. Having drafted chapters A and B, I singled out ‘canals’ for a separate inquiry.
- **Chapter D: The 7 Seas** pages 155–167  
Draft from August 2021. Having drafted chapters A, B and C, I singled out “the 7 seas” for a separate inquiry.





# Appendix A

## A Graph Domain

### Contents

---

|            |                                                     |           |
|------------|-----------------------------------------------------|-----------|
| <b>A.1</b> | <b>Introduction</b>                                 | <b>85</b> |
| A.1.1      | Critique of Classical Mathematical Modeling of Nets | 85        |
| A.1.2      | The Thesis                                          | 85        |
| A.1.3      | Structure of This Report                            | 85        |
| <b>A.2</b> | <b>Examples of Networks</b>                         | <b>86</b> |
| A.2.1      | Overland Transport Nets                             | 86        |
| A.2.1.1    | Road Nets                                           | 86        |
| A.2.1.2    | Rail Nets                                           | 86        |
| A.2.1.3    | Pipeline Nets                                       | 87        |
| A.2.2      | Natural Trees with Roots                            | 87        |
| A.2.3      | Waterways                                           | 87        |
| A.2.3.1    | Canals                                              | 87        |
| A.2.3.2    | Rivers                                              | 87        |
| A.2.3.3    | General                                             | 88        |
| A.2.3.4    | Visualisation of Rivers and Canals                  | 89        |
| A.2.3.4.1  | Rivers                                              | 89        |
| A.2.3.4.2  | Deltas                                              | 89        |
| A.2.3.4.3  | Canals and Water Systems                            | 89        |
| A.2.3.4.4  | Locks                                               | 90        |
| A.2.4      | Conclusion                                          | 90        |
| <b>A.3</b> | <b>Classical Mathematical Models</b>                | <b>90</b> |
| A.3.1      | Graphs                                              | 91        |
| A.3.1.1    | General Graphs                                      | 91        |
| A.3.1.1.1  | Some Mathematics!                                   | 91        |
| A.3.1.1.2  | Some Graphics!                                      | 92        |
| A.3.1.2    | Unique Identification of Vertices and Edges         | 92        |
| A.3.1.3    | Paths                                               | 92        |
| A.3.1.4    | Directed Graphs                                     | 92        |
| A.3.1.5    | Acyclic Graphs                                      | 93        |
| A.3.1.6    | Connected Graphs and Trees                          | 93        |
| A.3.1.7    | Vertex In- and Out-Degrees of Directed Graphs       | 93        |
| <b>A.4</b> | <b>Our General Graph Model</b>                      | <b>94</b> |
| A.4.1      | The External Qualities                              | 94        |
| A.4.1.1    | A “Global” Graph                                    | 94        |

|            |                                                                           |            |
|------------|---------------------------------------------------------------------------|------------|
| A.4.1.2    | <b>Varieties of Endurants</b> . . . . .                                   | 94         |
| A.4.1.2.1  | <b>Road Net Endurants</b> . . . . .                                       | 95         |
| A.4.1.2.2  | <b>Rail Endurants</b> . . . . .                                           | 95         |
| A.4.1.2.3  | <b>Pipeline Endurants</b> . . . . .                                       | 95         |
| A.4.1.2.4  | <b>River Net Endurants</b> . . . . .                                      | 96         |
| A.4.2      | <b>Internal Qualities</b> . . . . .                                       | 97         |
| A.4.2.1    | <b>Unique Identifiers</b> . . . . .                                       | 97         |
| A.4.2.2    | <b>Auxiliary Functions</b> . . . . .                                      | 98         |
| A.4.2.2.1  | <b>Extraction Functions: Unique Identifies</b> . . . . .                  | 98         |
| A.4.2.2.2  | <b>Retrieval Functions</b> . . . . .                                      | 98         |
| A.4.2.3    | <b>Wellformedness</b> . . . . .                                           | 98         |
| A.4.2.4    | <b>Unique Identifier Examples</b> . . . . .                               | 98         |
| A.4.2.4.1  | <b>Road Net Identifiers</b> . . . . .                                     | 98         |
| A.4.2.4.2  | <b>Rail Net Identifiers</b> . . . . .                                     | 99         |
| A.4.2.4.3  | <b>Pipeline Net Identifiers</b> . . . . .                                 | 99         |
| A.4.2.4.4  | <b>River Net Identifiers</b> . . . . .                                    | 99         |
| A.4.2.5    | <b>Mereologies</b> . . . . .                                              | 100        |
| A.4.2.5.1  | <b>Mereology of Undirected Graphs</b> . . . . .                           | 100        |
| A.4.2.5.2  | <b>Wellformedness of Mereologies</b> . . . . .                            | 100        |
| A.4.2.5.3  | <b>Mereology of Directed Graphs</b> . . . . .                             | 100        |
| A.4.2.5.4  | <b>In- and Out-Degrees</b> . . . . .                                      | 101        |
| A.4.2.5.5  | <b>Paths of Undirected Graphs</b> . . . . .                               | 101        |
| A.4.2.5.6  | <b>Paths of Directed Graphs</b> . . . . .                                 | 102        |
| A.4.2.5.7  | <b>Connectivity</b> . . . . .                                             | 102        |
| A.4.2.5.8  | <b>Acyclic Graphs, Trees and Forests</b> . . . . .                        | 103        |
| A.4.2.5.9  | <b>Forest</b> . . . . .                                                   | 103        |
| A.4.2.5.10 | <b>Mereology Examples</b> . . . . .                                       | 103        |
| A.4.2.6    | <b>Attributes</b> . . . . .                                               | 107        |
| A.4.2.6.1  | <b>Graph Labeling</b> . . . . .                                           | 107        |
| A.4.2.6.2  | <b>General Net Attributes</b> . . . . .                                   | 108        |
| A.4.2.6.3  | <b>Road Net Attributes</b> . . . . .                                      | 108        |
| A.4.2.7    | <b>Summing Up</b> . . . . .                                               | 110        |
| A.4.2.7.1  | <b>A Summary of The Example Endurant Models</b> . . . . .                 | 110        |
| A.4.2.7.2  | <b>Initial Conclusion on Labeled Graphs and Example Domains</b> . . . . . | 111        |
| <b>A.5</b> | <b>The Nets Domain</b> . . . . .                                          | <b>112</b> |
| A.5.1      | <b>Some Introductory Definitions</b> . . . . .                            | 112        |

---

We study formalisations of graphs as they are found in the conventional *Graph Theory* literature, but formalisations as we would formalise graphs in the style of this compendium. The title of this chapter, *A Graph Domain*, shall indicate that we shall present *graphs*, not in the conventional mathematical style, but according to the principles, techniques and tools of [58]. That is, both as mathematical entities and as, albeit abstract, i.e., not necessarily manifest, phenomena of the world. As such we shall endow vertices and edges of graphs with unique identifiability, mereology – to model the edge/vertex relations, and attributes – to model vertex and edge labeling, i.e., to model properties of vertices and edges, including directedness!

## A.1 Introduction

### A.1.1 Critique of Classical Mathematical Modeling of Nets

Classical mathematical modeling of (road and rail) transport nets, river systems, canal systems, etc., misses some, to us, important points.

The point being that the more-or-less individual elements of these systems, the links (edges) and hubs (nodes, vertices) each have their unique identity, their mereology and their attributes, and that it is these internal qualities of edges and nodes that capture the “real” meaning of the nets.

In the mathematical models graph edges and vertices have no internal qualities: they are treated merely as syntactic entities.

We strive, in *domain analysis & description* [58], to model *first* the **syntactic** properties of manifest phenomena, then the **semantic** properties. Naturally we cannot model their **pragmatics**!

### A.1.2 The Thesis

The thesis of this compendium is that the *domain analysis & description* principles, techniques and tools as brought forward in [51, 54, 55, 57, 58] is a more proper way to model nets.

### A.1.3 Structure of This Report

- In Sect. A.2 we casually pictorialise a number of domains whose compositions basically amount to graphs. These examples are:
  - **Road Nets** [Sect. A.2.1.1 on the next page],
  - **Railways** [Sect. A.2.1.2 on the following page],
  - **Pipelines** [Sect. A.2.1.3 on page 87],
  - **Rivers** [Sect. A.2.3.4.1 on page 89] and
  - **Canals** [Sect. A.2.3.4.3 on page 89].
- In Sect. A.3 we prepare the ground by presenting a minimum account of graphs as they are usually first introduced in textbooks.

Correlated narratives and formalisations for these domains are shown, spread all over this compendium as follows:

- **Road Nets:** Sections:
  - A.2.1.1 [Pictures],
  - A.4.1.2.1 [Endurants],
  - A.4.2.4.1 [Unique Identifiers],
  - A.4.2.5.10 [Mereology] and
  - A.4.2.6.3 [Attributes].
- **Railways:** In the compendium-proper we pictorialise railways in Sect. refnets-ex:Rail Nets [Pictures]. In all:
  - A.2.1.2 [Pictures],
  - F.1 [Endurants],
  - A.4.2.4.2 [Unique Identifiers], and
  - F.1.2.2 [Mereology].
- **Pipelines:** Sections:

- A.2.1.3 [Pictures],
  - A.4.1.2.3 [Endurants],
  - A.4.2.4.3 [Unique Identifiers], and
  - A.4.2.5.10 [Mereology].
- **Rivers:** Sections:
    - A.2.3.4.1 [Pictures],
    - A.4.1.2.4 [Endurants],
    - A.4.2.4.4 [Unique Identifiers], and
    - A.4.2.5.10 [Mereology].
  - **Canals:** Other than Sects. A.2.3.4.3 this compendium does not yet illustrate a systematic canal system description.

## A.2 Examples of Networks

We shall consider a widest set of networks,

### A.2.1 Overland Transport Nets

By overland transport nets we mean such which are either placed on the ground, or underground, as tunnels, or through mountains, also as tunnels, or placed on bridges over valleys, etc.

#### A.2.1.1 Road Nets

Road nets are for the conveyance of automobiles: private cars, buses, trucks, etc.



Figure A.1: Left: The Netherlands. R: Scotland



Figure A.2: L & R: European Road Infrastructure

#### A.2.1.2 Rail Nets

Rail nets are for the conveyance of passenger and freight trains.

Rail nets and train traffic on these are narrated and formalised in Chapter F:

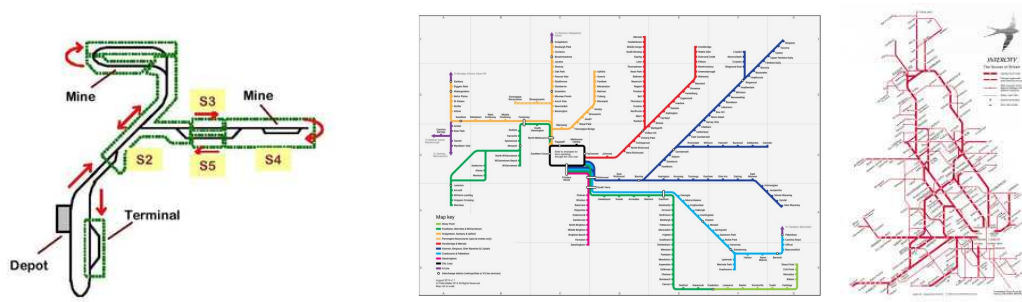


Figure A.3: Example Railway Nets

**A.2.1.3 Pipeline Nets**

Pipelines are for the conveyance of fluids: water, natural gas, hydrogen, oil, etc.

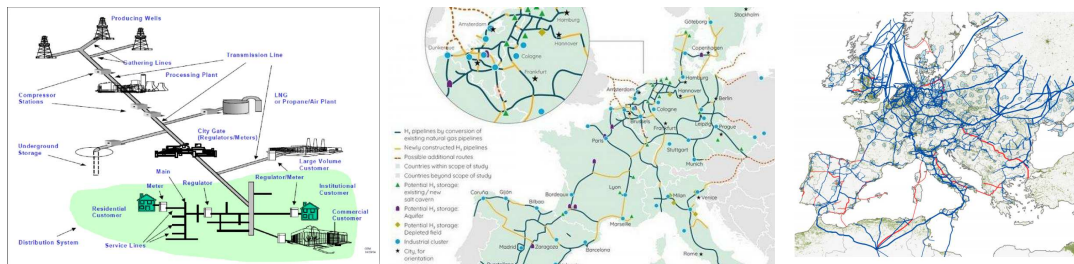


Figure A.4: Oil or Gas Field; European Gas and Hydrogen Pipelines

**A.2.2 Natural Trees with Roots**

Figures A.6 on the following page, A.7 on page 89, and A.8 on page 89 illustrate our point.

**A.2.3 Waterways**

By waterways we mean rivers, canals, lakes and oceans – such as are navigable by vessels: barges, boats and ships.

**A.2.3.1 Canals**

Canals are artificial or human-made channels or waterways. They are used for navigation, transporting water, crop irrigation, or drainage purposes. Therefore, a canal can be considered an artificial version of a river. Canals are constructed to connect existing rivers, seas, or lakes and to explicitly convey barges etc.

**A.2.3.2 Rivers**

Rivers, on the other hand, are naturally flowing watercourses, and typically flow until discharging their water into a lake, sea, ocean, or another river. However, occasionally some rivers do not discharge their water into lakes, seas, oceans, or other rivers. Rivers that do not empty into another body of water might flow into the ground or simply dry up before reaching another body of water. Additionally, small rivers can also be referred to as streams, rivulets, creeks, rills, or brooks.



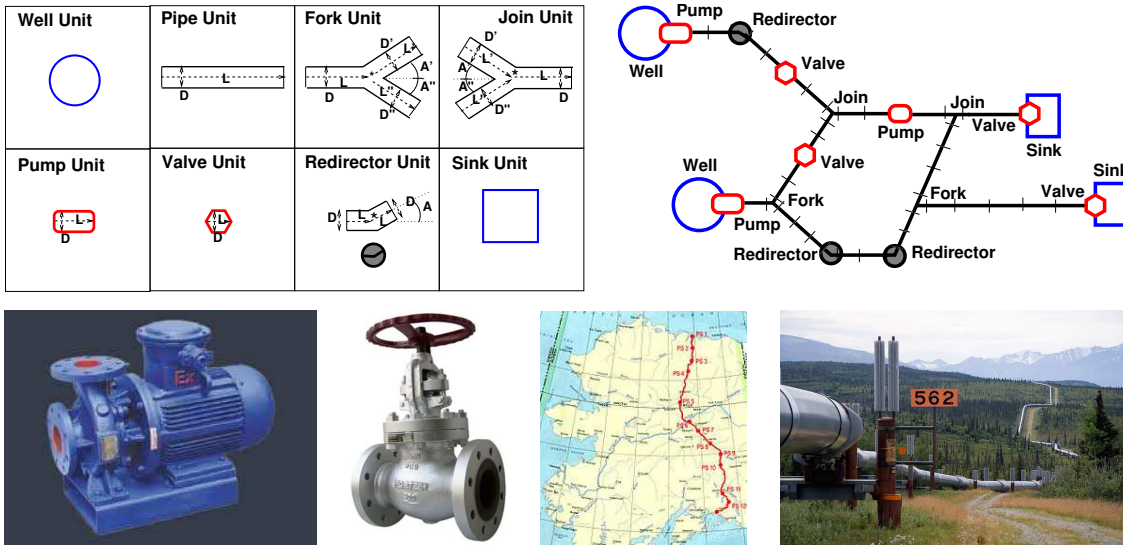


Figure A.5: Oil unit graphics; a simple oil pipeline.  
 A pump; a valve; the Trans-Alaska Pipeline System (TAPS); TAPS pipes, re-directors and ‘heat pipes’.



Figure A.6: A Japanese Maple [Portland, OR, US] and an Angel Oak Tree [SC, US]

**Disclaimer:** At present (“great”) lakes and the oceans (there are two!) are not included in this modeling effort.

**A.2.3.3 General**

Canals are artificial or human-made channels or waterways that are used for navigation, transporting water, crop irrigation, or drainage purposes. Therefore, a canal can be considered an artificial version of a river. Canals are artificial or human-made channels or waterways that are used for navigation, transporting water, crop irrigation, or drainage purposes. Therefore, a canal can be considered an artificial version of a river.

Rivers, on the other hand, are naturally flowing watercourses, and typically flow until discharging their water into a lake, sea, ocean, or another river, while canals are constructed to connect existing rivers, seas, or lakes. However, occasionally some rivers do not discharge their water into lakes, seas, oceans, or other rivers. Rivers that do not empty into another body of water might flow into the ground or simply dry up before reaching another body of water. Additionally, small rivers can also be referred to as streams, rivulets, creeks, rills, or brooks.

The natural water system of the earth includes 71% ocean with land continents being traversed by brooks, rivers, lakes and river deltas.



Figure A.7: Drawings of Banyan Trees



Figure A.8: A Dragon Tree [Yemen] and an Aspen Tree Root [Colorado, US]

Headwaters are streams and rivers (tributaries) that are the source of a stream or river.

A tributary is a river or stream that flows into another stream, river, or lake.

A delta is a large, silty area at the mouth of a river at which the river splits into many different slow-flowing channels that have muddy banks. New land is created at deltas. Deltas are often triangular-shaped, hence the name (the Greek letter 'delta' is shaped like a triangle).

The trunk is the main course of river.

**Confluence:** In geography, a confluence (also: conflux) occurs where two or more flowing bodies of water join together to form a single flow. A confluence can occur in several configurations: at the point where a tributary joins a larger river (main stem); or where two streams meet to become the source of a river of a new name; or where two separated channels of a river (forming a river island) rejoin at the downstream end.

**Towns and Harbours:** In this report we model towns. That is, we therefore also model that towns have harbours – allowing river (and canal) vessels to berth (a place for mooring in a harbour) for cargo loading, unloading and resting.

#### A.2.3.4 Visualisation of Rivers and Canals

**A.2.3.4.1 Rivers** Figures A.9 on the next page and A.10 on the following page illustrate a number of rivers.

**A.2.3.4.2 Deltas** We illustrate four deltas, Fig. A.11 on page 91:

**A.2.3.4.3 Canals and Water Systems** We illustrate just four ship/barge/boat and water level control canal systems, Figs. A.12, A.13, A.14 on page 92 and A.14 on page 92.

The rightmost figure of Fig. A.14 is from the Dutch *Rijkswaterstaat*: [www.rijkswaterstaat.nl/english/](http://www.rijkswaterstaat.nl/english/).



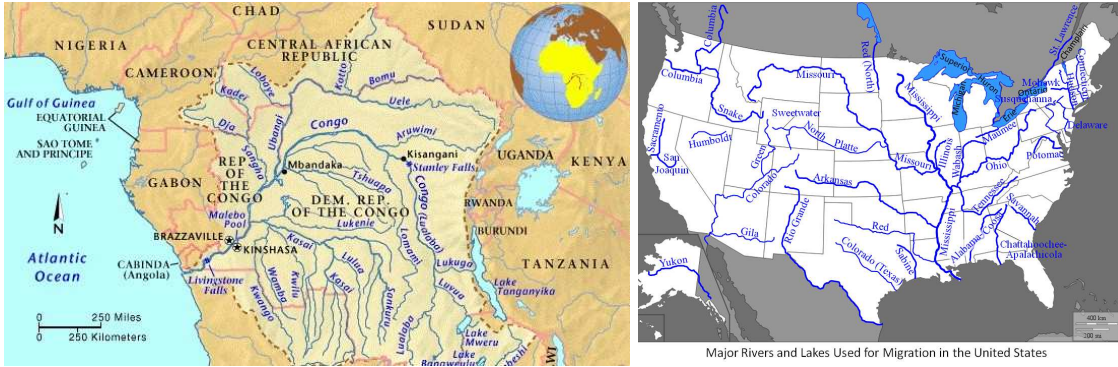


Figure A.9: The Congo and the US Rivers

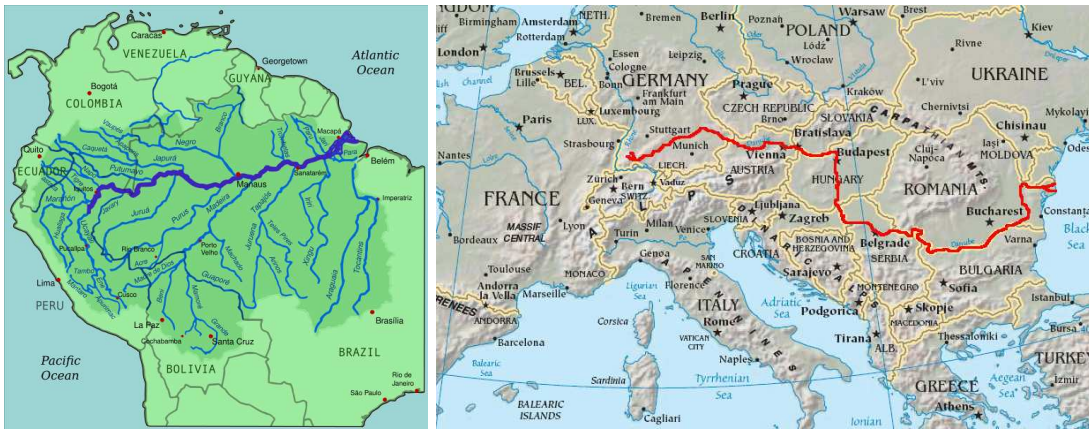


Figure A.10: The Amazon and The Danube Rivers

**A.2.3.4.4 Locks** A lock is a device used for raising and lowering boats, ships and other watercraft between stretches of water of different levels on river and canal waterways. The distinguishing feature of a lock is a fixed chamber in which the water level can be varied. Locks are used to make a river more easily navigable, or to allow a canal to cross land that is not level. Later canals used more and larger locks to allow a more direct route to be taken.<sup>1</sup>

We illustrate a number of locks: Figs. A.15 on page 93 and A.16 on page 93.

**A.2.4 Conclusion**

**A.3 Classical Mathematical Models**

We refer to standard textbooks in Graph Theory:

- **Claude Berge:** **Graphs** [11, 12, 1958–1978, 1st–2nd ed.]
- **Oystein Ore:** **Graphs and their Uses** [144, 1963]
- **Frank Harry:** **Graph Theory** [107, 1972]
- **J.A. Bondy and U.S.R. Murty:** **Graph Theory with Applications** [79, 1976]
- **S. Even:** **Graph Algorithms** [88, 1979]

<sup>1</sup>[https://en.wikipedia.org/wiki/Lock\\_\(water\\_navigation\)](https://en.wikipedia.org/wiki/Lock_(water_navigation))



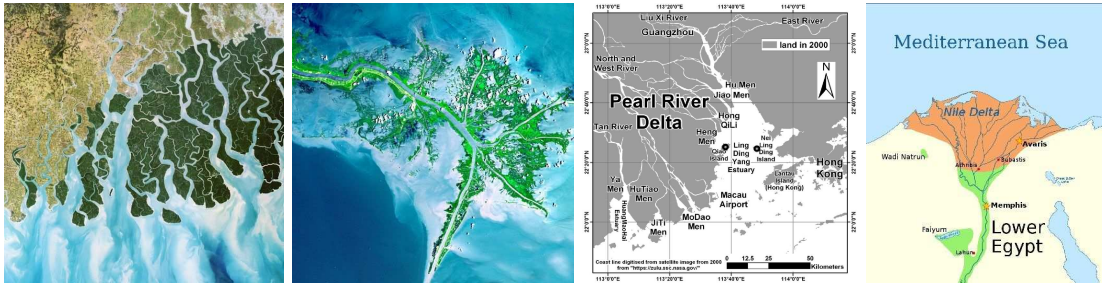


Figure A.11: The Ganges, Mississippi, Pearl and the Nile Deltas

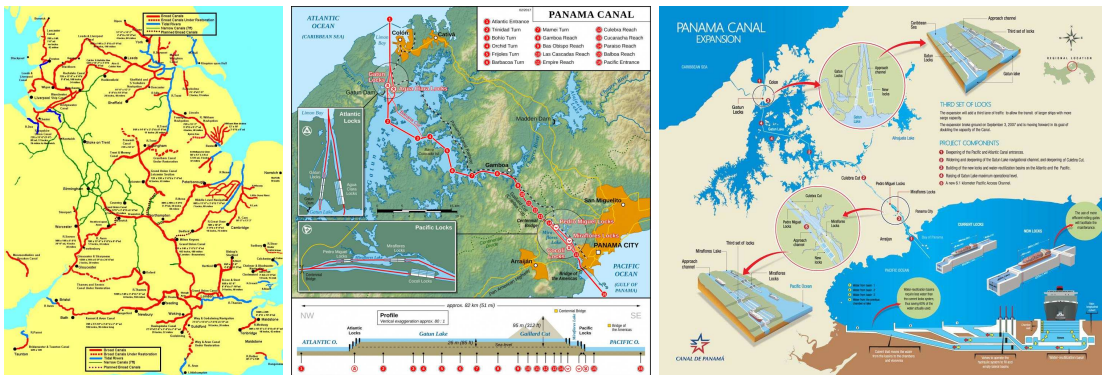


Figure A.12: UK Canals and The Panama Canal

or these Wikipedia Web pages:

- a. **Graph Theory**  
[en.m.wikipedia.org/wiki/Graph\\_theory](https://en.m.wikipedia.org/wiki/Graph_theory)
- b. **Graphs: Discrete Mathematics**  
[en.m.wikipedia.org/wiki/Graph\\_\(discrete\\_mathematics\)](https://en.m.wikipedia.org/wiki/Graph_(discrete_mathematics))
- c. **The Hamiltonian Path Problem**  
[en.m.wikipedia.org/wiki/Hamiltonian\\_path\\_problem](https://en.m.wikipedia.org/wiki/Hamiltonian_path_problem)
- d. **Glossary of Graph Theory**  
[en.wikipedia.org/wiki/Glossary\\_of\\_graph\\_theory\\_terms](https://en.wikipedia.org/wiki/Glossary_of_graph_theory_terms)

### A.3.1 Graphs

#### A.3.1.1 General Graphs

We refer to [en.wikipedia.org/wiki/Glossary\\_of\\_graph\\_theory\\_terms#A](https://en.wikipedia.org/wiki/Glossary_of_graph_theory_terms#A).

**A.3.1.1.1 Some Mathematics!** From (a.): in one restricted but very common sense of the term, a graph is an ordered pair

- $G = (V, E)$ , where
- $V$ , is a set of vertices (also called nodes or points), and
- $E \subseteq \{\{x, y\} \mid x, y \in V\}$  is a set of edges (also called links or lines), which are unordered pairs of vertices.

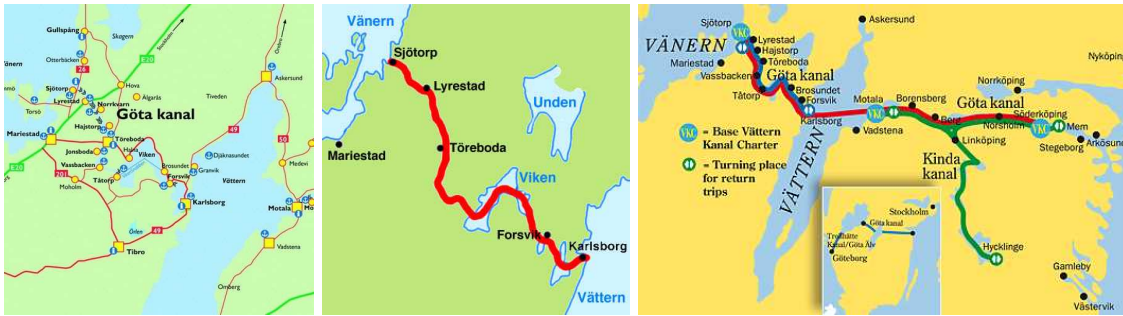


Figure A.13: The Swedish Göta Kanal

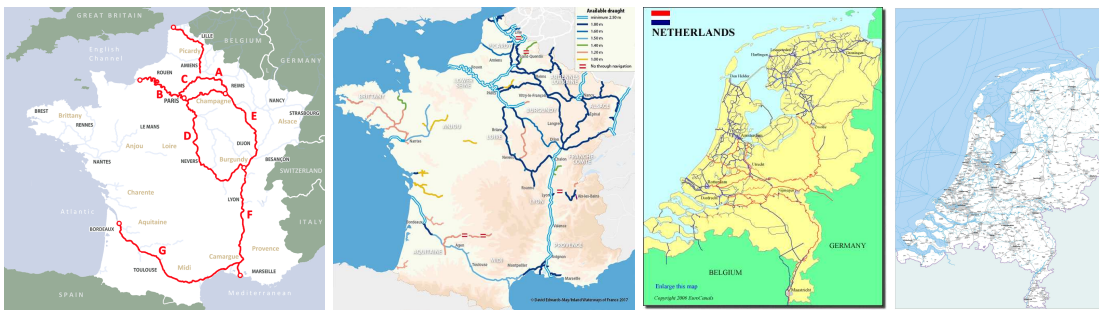


Figure A.14: French and Dutch Rivers and Canals

- If  $x = y$  then the edge  $s$  is a  $1$ -loop, cf. upper leftmost edge of **G0** of Fig. A.17 on page 94.

To avoid ambiguity, this type of object may be called precisely an undirected simple graph, cf. graph **G0** of Fig. A.17.

**A.3.1.1.2 Some Graphics!** Figure A.17 shows five similarly “shaped” graphs. Figure A.18 shows how these could have been drawn differently.

**A.3.1.1.2 Unique Identification of Vertices and Edges**

There is no way it can be avoided<sup>2</sup>. It simply makes no sense to not bring in that vertices and edges are uniquely identified. So we identify vertices and edges, cf. graph **G1** of Fig. A.17 on page 94. When, in classical graph theory, labeling of vertices and edges is introduced it is either for convenience of reference or for property attribution, as we shall later see.

With unique identification there is no problem with multiple edges between any pair of vertices.

**A.3.1.1.3 Paths**

A *vertex path* is a sequence,  $\langle v_i, v_j, \dots, v_k, v_{k+1}, \dots, v_l \rangle$ , of two or more vertices such that vertex  $v_k$  is *adjacent* to vertex  $v_{k+1}$  if there is an edge between them. Similar notion of *edge paths* and *vertex-edge-vertex paths* can be defined.

Graphs thus define possibly infinite sets of possibly infinite paths.

**A.3.1.1.4 Directed Graphs**

Directed graphs have directed edges, shown, in graph pictures, by affixing arrows to edges, see graph **G2** of Fig. A.17 on page 94.

<sup>2</sup>Sections 2.2.2.1 and 2.2.2.2 of [58, Bjørner] makes this clear: Unique identifiability is an unavoidable fact of any world.

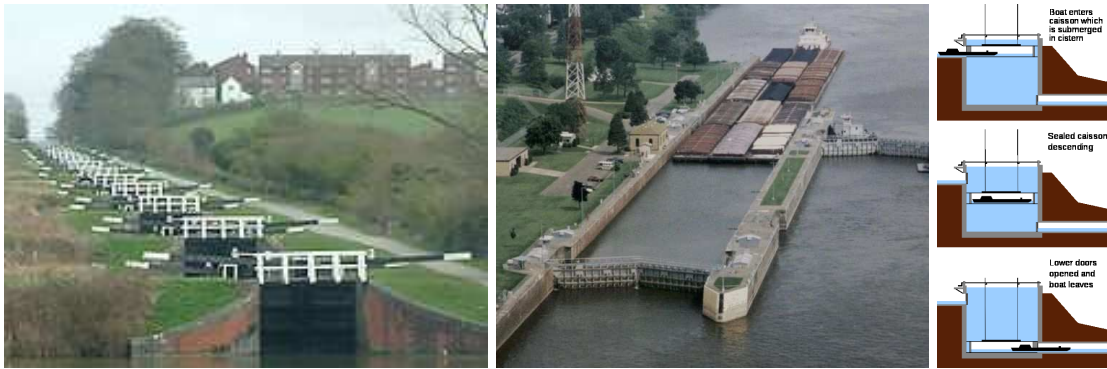


Figure A.15: Inland Canal Locks



Figure A.16: Harbour Canal Locks

- $G$  and  $V$  is as before, but
- $E \subseteq \{(x, y) \mid x, y \in V, \}$ .

Directed graphs still define possibly infinite sets of possibly infinite paths. The vertex sequence  $\langle v_a, v_c, v_d, v_f, v_f \rangle$  is a path of graph **G2** of Fig. A.17 on the next page.

### A.3.1.5 Acyclic Graphs

An *acyclic* graph is a graph none of whose vertex paths contain any vertex at most once. Graph **G3** of Fig. A.17 on the following page is an acyclic graph.

### A.3.1.6 Connected Graphs and Trees

A graph is *connected* if and only if for any two its vertices  $v_i, v_j$  there exists a path from  $v_i$  to  $v_j$ . A graph that is connected and is acyclic is a *tree*, cf. graph **G4** of Fig. A.17 on the next page.

### A.3.1.7 Vertex In- and Out-Degrees of Directed Graphs

By the *in-degree* of a vertex of a [directed] graph is meant the number of edges incident upon that vertex. By the *out-degree* of a vertex of a [directed] graph is meant the number of edges emanating from that vertex. In an un-directed graph the in- and out-degrees of any vertex are identical. In an acyclic graph there necessarily must be one or more vertices whose in-degrees are zero. And in an acyclic graph there necessarily must be one or more vertices whose out-degrees are zero.

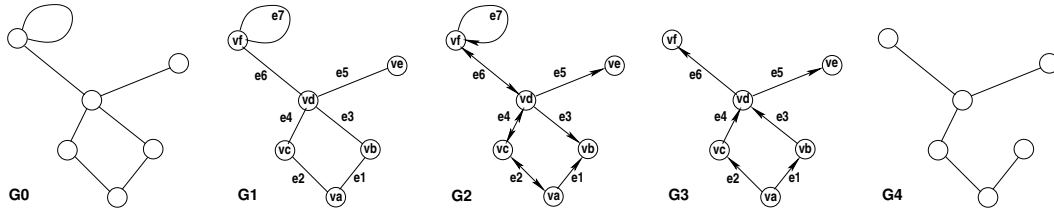


Figure A.17: Graphs

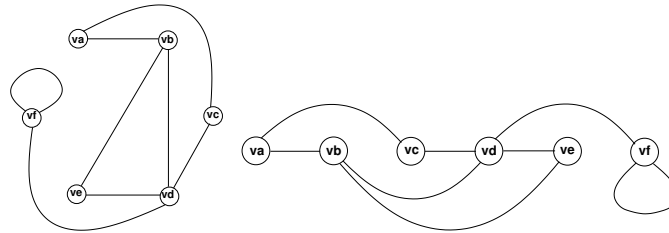


Figure A.18: Graphs

## A.4 Our General Graph Model

### A.4.1 The External Qualities

We refer to [58, Chapter 4].

- 36. Our domain is that of graphs.
- 37. From graphs one can observe sets of vertices,
- 38. and edges.

**type**

- 36.  $G$
- 37.  $V$
- 38.  $E$

**value**

- 37.  $obs\_Vs: G \rightarrow V\text{-set}$
- 38.  $obs\_Es: G \rightarrow E\text{-set}$

Please notice that nothing is said about how vertices and edges relate. That is an issues of mereology, cf. [58, Sect. 5.3.1].

#### A.4.1.1 A “Global” Graph

- 39. For ease of reference we can postulate a[n arbitrary] graph.

**value**

- 39.  $g:G$

#### A.4.1.2 Varieties of Endurants

Some domains warrant explication (e.g., renaming) of the vertices and edges or “collapsing” these into sets over a variety of units.

**A.4.1.2.1 Road Net Endurants**

- 40. A road as a pair of hubs and links.
- 41. Substitute vertices for *hubs*,  $H$ , i.e., street intersections,
- 42. and edges for *links*,  $L$ , i.e., street segment with no intersections.

**type**

- 40.  $RN = H\text{-set} \times L\text{-set}$  [  $\simeq G$  for Graphs ]
- 41.  $H$  [  $\simeq V$  for Graphs ]
- 42.  $L$  [  $\simeq E$  for Graphs ]

**A.4.1.2.2 Rail Endurants**

- 43. So a graph, i.e., a railway net,  $RN$ , consists of a set of rail units.
- 44. A rail units is
  - (a) either a simple, linear [or curved] unit,  $LU$ ,
  - (b) or a switch,  $SU$ ,
  - (c) or a cross-over,  $XU$ ,
  - (d) or a cross-over switch,  $CS$ ,
  - (e) or ...

We refer to Fig. F.1 on page 193 of Sect. F.1.1.1 on page 192.

**type**

- 43.  $RN = RU\text{-set}$  [  $\simeq G$  for Graphs ]
- 44.  $RU == LU \mid SU \mid XU \mid XS \mid SC$
- 44a.  $LU :: LiU$
- 44b.  $SU :: SiU$
- 44c.  $XU :: XiU$
- 44d.  $CS :: CiS$
- 44e. ...

Again; here we say nothing more about these units.

**A.4.1.2.3 Pipeline Endurants**

- 45. So a graph, i.e., a pipeline net,  $PN$ , consists of a set of pipeline units,  $PLU$ .
- 46. A pipeline units is
  - (a) either a source (a well),  $WU$ ,
  - (b) or a pump,  $PU$ ,
  - (c) or a pipe,  $LU$ ,
  - (d) or a valve,  $VU$ ,
  - (e) or a fork,  $FU$ ,
  - (f) or a join,  $JU$ ,
  - (g) or a sink,  $SU$ .
- 47. All pipeline units are distinct.



**type**

- 45. PN
- 46.  $PLU == WU \mid PU \mid LU \mid VU \mid FU \mid JU \mid SU$
- 46a.  $WU :: W$
- 46b.  $PU :: P$
- 46c.  $LU :: L$
- 46d.  $VU :: V$
- 46e.  $FU :: F$
- 46f.  $JU :: J$
- 46g.  $SU :: S$

**value**

- 46.  $obs\_PLUs: PN \rightarrow PLU\text{-set}$

**axiom**

- 47.  $WU \cap PU = \{\} \wedge WU \cap LU = \{\} \wedge WU \cap VU = \{\} \wedge WU \cap FU = \{\} \wedge WU \cap JU = \{\} \wedge WU \cap SU = \{\} \wedge$
- 47.  $PU \cap LU = \{\} \wedge PU \cap VU = \{\} \wedge PU \cap FU = \{\} \wedge PU \cap JU = \{\} \wedge PU \cap SU = \{\} \wedge$
- 47.  $LU \cap VU = \{\} \wedge LU \cap FU = \{\} \wedge LU \cap JU = \{\} \wedge LU \cap SU = \{\} \wedge$
- 47.  $VU \cap FU = \{\} \wedge VU \cap JU = \{\} \wedge VU \cap SU = \{\} \wedge$
- 47.  $FU \cap JU = \{\} \wedge FU \cap SU = \{\} \wedge$
- 47.  $JU \cap SU = \{\}$

Again; here we say nothing more about these units.

**A.4.1.2.4 River Net Endurants**

- 48. A river net is modeled as a graph, more specifically as a tree. The *root* of that river net tree is the mouth (or delta) of the river net. The *leaves* of that river net tree are the sources of respective trees. Paths from leaves to the root define *flows* of water.
- 49. We can thus, from a river net observe vertices
- 50. and edges.
- 51. River vertices model either a *source*: **so:SO**, a *mouth*: **mo:MO**, or possibly some *confluence*: **ko:KO**.  
A river may thus be “punctuated” by zero or more confluences, **k:KO**.  
A confluence defines the joining a ‘main’ river with zero<sup>3</sup> or more rivers into that ‘main’ river.  
We can talk about the “upstream” and the “downstream” of rivers from their confluence.
- 52. River edges model *stretches*: **st:ST**.  
A stretch is a linear sequences of simple, **se:SE**, or composite **ce:CE**, river elements.
- 53. River elements are either simple: (ch) river channels, which we shall call *river channels*: **CH**, or (la) lakes: **LA**, or (lo) locks: **LO**, or (wa) waterfalls (or *rapids*): **WA**, or (da) dams: **DA**, or (to) towns (cities, villages): **to:TO**<sup>4</sup>; or composite, **ce:CE**: a dam with a lock, (**da:DA,la:LA**), a town with a lake, (**to:TO,la:LA**), etcetera; even a town with a lake and a confluence, **to:TO,la:LA,ko:KO**. Etcetera.

**type**

- 48. RiN
- 49. V
- 50. E
- 51. SO, MO, KO
- 52.  $ST = (SE|CE)^*$
- 53. CH, LA, LO, WA, KO, DA, TO

<sup>3</sup>Normally, though, one would expect, not zero, but one

<sup>4</sup>Towns is here really a synonym for river harbours, places along the river (or a canal) where river vessels can stop (moor) for the loading and unloading of cargo and for resting.

- 53.  $SE = CH \mid LA \mid LO \mid WA \mid DA \mid TO$
- 53.  $DaLo, WaLo, ToLa, ToLaKo, \dots$
- 53.  $CE = DaLo \mid WaLo \mid ToLa \mid ToLaKo \mid \dots$
- value**
- 51.  $obs\_Vs: RiN \rightarrow V\text{-set}$
- 51. **axiom**
- 51.  $\forall g:G, vs:V\text{-set} \bullet vs \in obs\_Vs(g) \Rightarrow vs \neq \{\}$
- 51.  $\wedge \forall v:V \bullet v \in vs \Rightarrow is\_SO(v) \vee is\_KO(v) \vee is\_MO(v)$
- 52.  $obs\_Es: RiN \rightarrow E\text{-set}$
- 52. **axiom**
- 52.  $\forall g:G, es:E\text{-set} \bullet es \in obs\_Es(g) \Rightarrow es \neq \{\}$
- 52.  $\wedge \forall e:E \bullet e \in es \Rightarrow is\_ST(e)$
- 52.  $obs\_ST: E \rightarrow ST$
- 48.  $xtr\_In\_Degree\_0\_Vertices: RiN \rightarrow SO\text{-set}$
- 48.  $xtr\_Out\_Degree\_0\_Vertex: RiN \rightarrow MO$

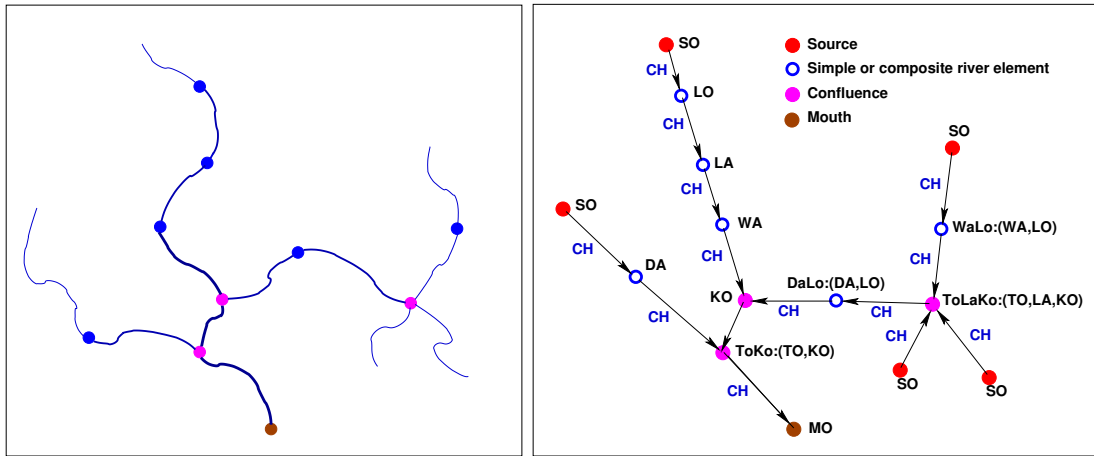


Figure A.19: The “Composition” of a River Net: Right Tree is an abstraction of the Left Tree

### A.4.2 Internal Qualities

We refer to [58, Chapter 5]

#### A.4.2.1 Unique Identifiers

We refer to [58, Sect. 5.2]

- 54. Each vertex has a unique identifier.
- 55. Each edge has a unique identifier.

**type**

- 54.  $V\_UI$
- 55.  $E\_UI$

**value**

- 54.  $uid\_V: V \rightarrow V\_UI$
- 55.  $uid\_E: E \rightarrow E\_UI$

### A.4.2.2 Auxiliary Functions

#### A.4.2.2.1 Extraction Functions: Unique Identifiers

56. We can calculate the set of all unique vertex identifiers of a graph,
57. and all unique edge identifiers of a graph,
58. and all unique identifiers of vertices and edges of a graph.

#### value

56.  $\text{xtr\_V\_UIs}: G \rightarrow \text{V\_UI-set}$ ,  $\text{xtr\_V\_UIs}(g) \equiv \{ \text{uid\_V}(v) \mid v:V \bullet v \in \text{obs\_Vs}(g) \}$
57.  $\text{xtr\_E\_UIs}: G \rightarrow \text{EI-set}$ ,  $\text{xtr\_E\_UIs}(g) \equiv \{ \text{uid\_E}(e) \mid e:E \bullet e \in \text{obs\_Es}(g) \}$
58.  $\text{xtr\_U\_UIs}: G \rightarrow (\text{V|EI})\text{-set}$ ,  $\text{xtr\_UIs}(g) \equiv \text{xtr\_V\_UIs}(g) \cup \text{xtr\_E\_UIs}(g)$

#### A.4.2.2.2 Retrieval Functions

59. Given a unique vertex identifier of a graph one can retrieve, from the graph, the vertex of that identification.
60. Given a unique edge identifier of a graph one can retrieve, from the graph, the edge of that identification.

#### value

59.  $\text{retr\_V}: \text{V\_UI} \rightarrow G \xrightarrow{\sim} V$
59.  $\text{retr\_V}(v\_ui)(g) \equiv \text{let } v:V \bullet v \in \text{obs\_Vs}(g) \wedge v\_ui = \text{uid\_V}(v) \text{ in } v \text{ end, pre: } e\_ui \in \text{xtr\_E\_UIs}(g)$
60.  $\text{retr\_E}: \text{EI} \rightarrow G \xrightarrow{\sim} E$
60.  $\text{retr\_E}(ei)(g) \equiv \text{let } e:E \bullet e \in \text{obs\_Es}(g) \wedge e\_ui = \text{uid\_E}(e) \text{ in } e \text{ end, pre: } e\_ui \in \text{xtr\_E\_UIs}(g)$

### A.4.2.3 Wellformedness

61. Vertex and edge identifiers are all distinct.
62. Each vertex and each edge has a distinct unique identifier.

#### axiom

61.  $\forall g:G \bullet \text{xtr\_V\_UIs}(g) \cap \text{xtr\_E\_UIs}(g) = \{ \}$
62.  $\text{card obs\_Vs}(g) = \text{card xtr\_V\_UIs}(g) \wedge \text{card obs\_Es}(g) = \text{card xtr\_E\_UIs}(g)$

### A.4.2.4 Unique Identifier Examples

We give four examples: roads, rails, pipelines and rivers.

#### A.4.2.4.1 Road Net Identifiers

Very simple,

63. substitute vertex identifiers, VI, with hub identifiers, HI, and
64. substitute edge identifiers, EI, with link identifiers, LI,

in type and unique observer function definitions.

#### type

63. HI [  $\equiv$  VI for Graphs ]
64. LI [  $\equiv$  EI for Graphs ]



**A.4.2.4.2 Rail Net Identifiers**

65. With every rail net unit we associate a unique identifier.  
 66. That is, no two rail net units have the same unique identifier.

**type**

65. UI

**value**65. uid\_NU: NU  $\rightarrow$  UI**axiom**66.  $\forall ui_i, ui_j: UI \bullet ui_i = ui_j \equiv uid\_NU(ui_i) = uid\_NU(ui_j)$ **A.4.2.4.3 Pipeline Net Identifiers**

67. With pipeline units a type WU, PU, LU, VU, FU, JU and SU we associate a single unique identifier sort: UI.

67. UI == WU\_UI | PU\_UI | LU\_UI | VU\_UI | FU\_UI | JU\_UI | SU\_UI

**A.4.2.4.4 River Net Identifiers** We shall associate unique identifiers both with vertices, edges and vertex and edge elements.

68. River net vertices and edges have unique identifiers.  
 69. River net sources, confluences and mouths have unique identifiers.  
 70. River net stretches have unique identifiers.  
 71. River net channels, lakes, locks, waterfalls, dams and towns as well as combinations of these, that is, simple and composite river entities have unique identifiers.

**type**

68. V\_UI, E\_UI

69. SO\_UI, KO\_UI, MO\_UI

70. ST\_UI

71. CH\_UI, LA\_UI, LO\_UI, WA\_UI, DA\_UI, TO\_UI, DaLo\_UI, WaLo\_UI, ToLa\_UI, ToLaKo\_UI, ...

**value**68. uid\_V: V  $\rightarrow$  V\_UI, uid\_E: E  $\rightarrow$  E\_UI69. uid\_SO: SO  $\rightarrow$  SO\_UI, uid\_KO: KO  $\rightarrow$  KO\_UI, uid\_MO: MO  $\rightarrow$  MO\_UI,70. uid\_ST: ST  $\rightarrow$  ST\_UI71. uid\_CH: CH  $\rightarrow$  CH\_UI, uid\_LA: LA  $\rightarrow$  LA\_UI, uid\_LO: LO  $\rightarrow$  LO\_UI, uid\_WA: WA  $\rightarrow$  WA\_UI,71. uid\_DA: DA  $\rightarrow$  DA\_UI, uid\_TO: TO  $\rightarrow$  TO\_UI,71. uid\_DaLo: DaLo  $\rightarrow$  DaLo\_UI, uid\_WaLo: WaLo  $\rightarrow$  WaLo\_UI, uid\_ToLa: ToLa  $\rightarrow$  ToLa\_UI,71. uid\_ToLaKo: ToLaKo  $\rightarrow$  ToLaKo\_UI, ...

72. All these identifiers are distinct.

The  $\cap$  operator takes the pairwise intersection of the types in its argument list and examines them for disjointedness.

**axiom**72.  $\cap(V\_UI, E\_UI, SO\_UI, KO\_UI, MO\_UI, ST\_UI, CH\_UI,$ 

72. LA\_UI, LO\_UI, WA\_UI, DA\_UI, TO\_UI, DaLo\_UI, WaLo\_UI, ToLa\_UI, ToLaKo\_UI)

73. There are [many] other constraints, please state them !

73. [ left as exercise to the reader ! ]

#### A.4.2.5 Mereologies

We refer to [58, Sect. 5.3]. We shall formalise a number of mereologies:

- of undirected graphs — typically road, air and sea transport nets,
- and “general” directed graphs —

##### A.4.2.5.1 Mereology of Undirected Graphs

74. The mereology of a vertex is the set of unique identifiers of the edges incident upon the vertex.
75. The mereology of an edges is the one-or two element set of the unique identifiers of the [1-loop] vertex, respectively the vertices which the edge is connecting.

##### type

74.  $V\_Mer = E\_UI\text{-set}$

75.  $E\_Mer = V\_UI\text{-set}$

##### value

74.  $mereo\_V: V \rightarrow V\_Mer$

75.  $mereo\_E: E \rightarrow E\_Mer$

##### axiom

74.  $\forall g:G, v:V \bullet v \in obs\_Vs(g) \Rightarrow mereo\_V(v) \subseteq_{xtr} E\_UIs(g)$

75.  $\forall g:G, e:E \bullet e \in obs\_Es(g) \Rightarrow mereo\_E(e) \subseteq_{xtr} V\_UIs(g)$

##### A.4.2.5.2 Wellformedness of Mereologies

76. The vertex mereology must record unique edge identifiers of the graph.
77. The edge mereology must record unique vertex identifiers of the graph.
78. If a vertex mereology identify edges then these edge mereologies must identify that vertex, and, vice versa
79. If an edge mereology identify vertices then these vertex mereologies must identify that edge.

##### axiom

76.  $\forall g:G, v:V \bullet v \in obs\_Vs(g) \Rightarrow mereo\_V(v) \subseteq_{xtr} E\_UIs(g)$

77.  $\forall g:G, e:E \bullet e \in obs\_Es(g) \Rightarrow mereo\_E(e) \subseteq_{xtr} V\_UIs(g)$

78.  $\forall g:G, v:V \bullet v \in obs\_Vs(g) \Rightarrow \forall ei \in mereo\_V(v) \Rightarrow uid\_V(v) \in mereo\_E(ei)$

79.  $\forall g:G, e:E \bullet e \in obs\_Es(g) \Rightarrow \forall vi \in mereo\_E(e) \Rightarrow uid\_E(e) \in mereo\_V(vi)$

##### A.4.2.5.3 Mereology of Directed Graphs

80. The mereology of a vertex is a pair of the set of unique identifiers of the edges incident upon the vertex and the set of unique identifiers of the edges emanating from the vertex –
81. and these must all be of the graph.
82. The mereology of an edge is a one or two element set of pairs of vertex identifiers –
83. and these must all be of the graph.

**type**80.  $V\_Mer = E\_UI\text{-set} \times E\_UI\text{-set}$ 82.  $E\_Mer = (V\_UI \times V\_UI)\text{-set}$ **value**80.  $mereo\_V: V \rightarrow G \rightarrow V\_Mer$ 82.  $mereo\_E: E \rightarrow G \rightarrow E\_Mer$ **axiom**81.  $\forall g:G, v:V \bullet v \in obs\_Vs(g) \Rightarrow$ 81. **let**  $(e\_uis\_i, e\_uis\_e) = V\_Mer(v)$  **in**  $e\_uis\_i \cup e\_uis\_e \subseteq xtr\_E\_UIs(g)$  **end**83.  $\forall g:G, e:E \bullet e \in obs\_Es(g) \Rightarrow$ 83. **let**  $p\_v\_uis = V\_Mer(e)$  **in**83. **let**  $v\_uis = \{ v\_uis\_i, v\_uis\_e \mid (v\_uis\_i, v\_uis\_e): (V\_UI \times V\_UI) \bullet (v\_uis\_i, v\_uis\_e) \in p\_v\_uis \}$  **in**83.  $v\_uis \subseteq xtr\_V\_UIs(g)$  **end end****A.4.2.5.4 In- and Out-Degrees**

84. The in-degree of a vertex of a directed graph is the number of edges incident upon that vertex.

85. The out-degree of a vertex of a graph is the number of edges emanating that vertex.

84.  $in\_degree\_V: V \rightarrow G \xrightarrow{\sim} \mathbf{Nat}$ 84.  $in\_degree(v)(vs, es) \equiv \mathbf{let} (uis\_i, \_) = mereo\_V(v)$  **in**  $\mathbf{card} uis\_i$  **end, pre**  $v \in vs$ 85.  $out\_degree\_V: V \rightarrow G \xrightarrow{\sim} \mathbf{Nat}$ 85.  $out\_degree(v)(vs, es) \equiv \mathbf{let} (\_, uis\_e) = mereo\_V(v)$  **in**  $\mathbf{card} uis\_e$  **end, pre**  $v \in vs$ **A.4.2.5.5 Paths of Undirected Graphs** We shall only illustrate *vertex-edge-vertex paths* for given graphs,  $g$ .

86. A vertex-edge-vertex path is a sequence of zero or more edges.

87. That is, the empty sequence,  $\langle \rangle$ , is a vertex-edge-vertex path, [the first basis clause].88. If  $e$  is an edge of  $g$ , then the two elements  $\langle (vi, ej, vk) \rangle$ ,  $\langle (vk, ej, vi) \rangle$ , where  $ej$  is the unique identifier of  $e$  whose mereology is  $\{vi, vj\}$ , are vertex-edge-vertex paths.89. In  $\langle (vi, ej, vk) \rangle$  we refer to  $vi$  is the first vertex identifier and  $vk$  as the second. Vice versa in  $\langle (vk, ej, vi) \rangle$ .**value**89.  $fVIIEP: EP \rightarrow VI$ ,  $fVIIEP(ep: \langle (vi, ej, vk) \rangle \hat{\ } ep') \equiv vi$ , **pre:**  $ep \neq \langle \rangle$ 89.  $IVIIEP: EP \rightarrow VI$ ,  $IVIIEP(ep: ep' \hat{\ } \langle (vi, ej, vk) \rangle) \equiv vk$ , **pre:**  $ep \neq \langle \rangle$ 90. If  $p$  and  $p'$  are paths of  $g$  such that the last vertex identifier of the last element of  $p$  is the same as the first vertex identifier of the first element of  $p'$ , then the sequence  $p$  followed by the sequence  $p'$  is a vertex-edge-vertex path of  $g$  [the inductive clause].

91. Only such paths which can be constructed by the above rules are edge paths [the extremal clause].

**type**86.  $EP = E^\omega$ 86.  $edge\_paths: G \rightarrow EP\text{-set}$ 86.  $edge\_paths(g) \equiv$

```

87. let ps = {⟨⟩}
88. ∪ {⟨(vi,uid_E(e),vk),⟨(vk,uid_E(e),vi)⟩|e:E•e ∈ xtr_Es(g)∧{vi,vk}⊆mereo_E(e)}
90. ∪ {p^p'|p,p':EP•{p,p'}⊆ps∧∀IIIEP(9)=fVIfEP(p')} in
91. ps end

```

#### A.4.2.5.6 Paths of Directed Graphs

86. A vertex-edge-vertex path is a sequence of zero or more edges.
87. That is, the empty sequence,  $\langle \rangle$ , is a vertex-edge-vertex path, [the first basis clause].
92. If  $e$  is an edge of  $g$ , and if  $(vi,vj)$  is in the mereology of  $e$ , then the  $\langle (vi, ej, vk) \rangle$ , where  $ej$  is the unique identifier of  $e$  is a vertex-edge-vertex path.
90. If  $p$  and  $p'$  are paths of  $g$  such that the last vertex identifier of the last element of  $p$  is the same as the first vertex identifier of the first element of  $p'$ , then the sequence  $p$  followed by the sequence  $p'$  is a vertex-edge-vertex path of  $g$  [the inductive clause].
91. Only such paths which can be constructed by the above rules are edge paths [the extremal clause].

#### type

```

86. EP = Eω
86. edge_paths: G → EP-set
86. edge_paths(g) ≡
87. let ps = {⟨⟩}
88. ∪ {⟨(vi,uid_E(e),vk)⟩|e:E•e ∈ xtr_Es(g)∧(vi,vk)∈mereo_E(e)}
90. ∪ {p^p'|p,p':EP•{p,p'}⊆ps∧∀IIIEP(9)=fVIfEP(p')} in
91. ps end

```

Notice that the difference in the two definitions of (overload-named) `edge_paths` differ only in the last terms of items 88 and 92.

#### A.4.2.5.7 Connectivity

93. For every pair of vertices we can calculate the set of all paths connecting these in a graph.

```

93. all_connected_paths: (V×V) → G → EP-set
93. all_connected_paths(vi,vj) ≡
93. { ep | ep:EP • ep ∈ edge_paths(g) • ep[1] = (uid_V(vi),_,_) , ep[len ep] = (_,_,uid_V(vj)) }

```

94. Two vertices,  $v_i, v_j$ , of a graph,  $g$ , are *connected* if there is a path from  $v_i$  to  $v_j$  in  $g$ .

#### value

```

94. are_connected: (V×V) → G $\tilde{\rightarrow}$ Bool
94. are_connected(vi,vj)(g) ≡ all_connected_paths(vi,vj) ≠ {}

```

95. A graph *is\_connected* if there is a path from every vertex to every other vertex.

#### value

```

95. is_connected: G → Bool
95. is_connected(g) ≡ ∀ vi,vj:V • {vi,vj} ∈ obs_Vs(g) • are_connected(vi,vj)(g)

```

**A.4.2.5.8 Acyclic Graphs, Trees and Forests**

- 96. A cycle is a path which begins and ends at the same vertex.
- 97. An acyclic graph is a graph having no graph cycles.
- 98. A bipartite graph (or bi-graph) is a graph whose vertices can be divided into two disjoint and independent sets,  $V', V''$ , such that every edge connects a vertex in  $V'$  to one in  $V''$ . Acyclic graphs are bipartite.
- 99. By a tree we<sup>5</sup> shall understand a connected, acyclic graph such that there are no two distinct paths from any given pair of in-degree-0 and out-degree-0 vertices.
- 100. A disjoint graph is a set of two or more graphs such that no two of these graphs,  $G, g'$ , have vertices in  $g$  with edges to  $g'$ .
- 101. A forest is a disconnected set of trees, hence form a disjoint graph of distinct trees.

```

97. is_a_cycle: EP → Bool
97. is_a_cycle(ep) ≡ let (vi,_,_) = ep[1], (_,_,vi') = ep[len ep] in vi = vi' end
97. is_acyclic: G → Bool
97. is_acyclic(g) ≡ !∃ ep:EP • ep ∈ edge_paths(g) ∧ is_a_cycle(ep)
98. is_bipartite: G → Bool
98. is_bipartite(g) ≡ ... [exercise for the reader]
99. is_a_tree: G → Bool
99. is_a_tree(g) ≡ ... [exercise for the reader]
100. is_disjoint_graph: G → Bool
100. is_disjoint_graph(g) ≡ ... [exercise for the reader]
101. is_a_forest: G → Bool
101. is_a_forest(g) ≡ ... [exercise for the reader]

```

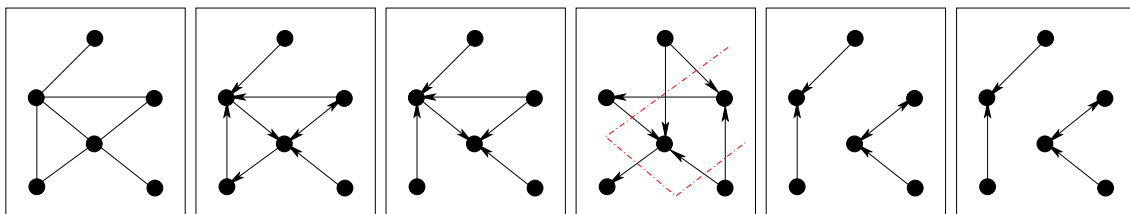


Figure A.20: Undirected, Directed, Acyclic, Bipartite, Tree and Disjoint Graphs

**A.4.2.5.9 Forest** A forest is an undirected graph without cycles (a disjoint union of un-rooted trees), or a directed graph formed as a disjoint union of rooted trees.

**A.4.2.5.10 Mereology Examples** We present mereology examples of both undirected and directed graphs.

**Mereology of Undirected Graph Examples:** We present mereology examples of road nets and railway tracks.

- **Road Nets**

---

<sup>5</sup>Our definition is OK, but there are more encompassing definitions of trees.

The mereology of road nets follow that of undirected graphs:

102. substitute V for H and VI for HI, and
103. substitute E for L and EI for LI.

We refer to Sect. A.4.2.5.10.

102.  $H\_Mer = L\_UI\text{-set} \times L\_UI\text{-set}$
103.  $L\_Mer = (H\_UI \times H\_UI)\text{-set}$ , **axiom**  $\forall lm:L\_Mer \bullet \text{card}lm \in \{0,1,2\}$

- **Rail Nets**

We refer to Chapter F.

**Mereology of directed Graph Examples:** In some circumstances we may model mereologies of directed graphs in terms of attributes. An example is that of road nets. Road nets, usually, can be considered undirected graphs. But discrete dynamically set and reset traffic signals as well as road signs may render streets and their intersection, i.e., links and hubs, “directed”. We then model this “directedness”, as we shall see, in Sect. A.4.2.6.3 on page 108, in terms of *programmable attributes*.

- **Pipeline Nets**

We refer to Sects. A.2.1.3 on page 87, A.4.1.2.3 on page 95 and A.4.2.4.3 on page 99.

104. Wells have exactly one connection to an output unit – which is usually a pump.
105. Pipes, pumps, valves and re-directors have exactly one connection from an input unit and one connection to an output unit.
106. Forks have exactly one connection from an input unit and exactly two connections to distinct output units.
107. Joins have exactly two connections from distinct input units and one connection to an output unit.
108. Sinks have exactly one connection from an input unit – which is usually a valve.
109. Thus we model the mereology of a pipeline unit as a pair of disjoint sets of unique pipeline unit identifiers.

**type**

109  $PM' = (UI\text{-set} \times UI\text{-set})$ ,  $PM = \{ \{(iuis,ouis): PM' \bullet iuis \cap ouis = \{ \} \} \}$

**value**

109 mereo\_PE: PE  $\rightarrow$  PM

The well-formedness inherent in narrative lines 104–108 are formalised:

**axiom** [Well–formedness of Pipeline Systems, PL (0)]

$\forall pl:PL, pe:PE \bullet pe \in \text{all\_pipeline\_uits}(pl) \Rightarrow$   
     **let** (iuis,ouis)=mereo\_PE(pe) **in**  
     **case** (card iuis,card ouis) **of**  
 104       (0,1)  $\rightarrow$  is\_We(pe),  
 105       (1,1)  $\rightarrow$  is\_Pi(pe)  $\vee$  is\_Pu(pe)  $\vee$  is\_Va(pe),  
 106       (1,2)  $\rightarrow$  is\_Fo(pe),  
 107       (2,1)  $\rightarrow$  is\_Jo(pe),  
 108       (1,0)  $\rightarrow$  is\_Si(pe),  $\_ \rightarrow$  **false**  
     **end end**

To express full well-formedness we need express that pipeline nets are acyclic. To do so we first define a function which calculates all routes in a net.

Two pipeline units,  $pe_i$  with unique identifier  $\pi_i$ , and  $pe_j$  with unique identifier  $\pi_j$ , that are connected, such that an outlet marked  $\pi_j$  of  $p_i$  “feeds into” inlet marked  $\pi_i$  of  $p_j$ , are said to **share** the connection (modeled by, e.g.,  $\{(\pi_i, \pi_j)\}$ )

110. The observed pipeline units of a pipeline system define a number of routes (or pipelines):

**Basis Clauses:**

111. The null sequence,  $\langle \rangle$ , of no units is a route.

112. Any one pipeline unit,  $pe$ , of a pipeline system forms a route,  $\langle pe \rangle$ , of length one.

113. ] **Inductive Clauses:**

114. Let  $r_i \hat{\ } \langle pe_i \rangle$  and  $\langle pe_j \rangle \hat{\ } r_j$  be two routes of a pipeline system.

115. Let  $pe_{i_{ui}}$  and  $pe_{j_{ui}}$  be the unique identifiers  $pe_i$ , respectively  $pe_j$ .

116. If one of the output connectors of  $pe_i$  is  $pe_{i_{ui}}$

117. and one of the input connectors of  $pe_j$  is  $pe_{j_{ui}}$ ,

118. then  $r_i \hat{\ } \langle pe_i, pe_j \rangle \hat{\ } r_j$  is a route of the pipeline system.

**Extremal Clause:**

119. Only such routes which can be formed by a finite number of applications of the clauses form a route.

**type**

110.  $R = PE^\omega$

**value**

110 routes:  $PL \xrightarrow{\sim} R\text{-infset}$

110 routes(pl)  $\equiv$

110 **let** cpes = pipeline\_units(pl) **in**

111 **let** rs =  $\{\langle \rangle\}$

112  $\cup \{\langle pe \rangle \mid pe:PE \bullet pe \in cpes\} \cup$

118  $\cup \{r_i \hat{\ } \langle pe_i \rangle \hat{\ } \langle pe_j \rangle \hat{\ } r_j \mid pe_i, pe_j:PE \bullet \{pe_i, pe_j\} \subseteq cpes$

114  $\wedge r_i \hat{\ } \langle pe_i \rangle, \langle pe_j \rangle \hat{\ } r_j: R \bullet \{r_i \hat{\ } \langle pe_i \rangle, \langle pe_j \rangle \hat{\ } r_j\} \subseteq rs$

115,116  $\wedge pe_{i_{ui}} = uid\_PE(pe_i) \wedge pe_{i_{ui}} \in xtr\_oUOs(pe_i)$

115,117  $\wedge pe_{j_{ui}} = uid\_PE(pe_j) \wedge pe_{j_{ui}} \in xtr\_iUls(pe_j)\}$  **in**

119 **rs end end**

xtr\_iUls:  $PE \rightarrow UI\text{-set}$ ,  $xtr\_iUls(u) \equiv \text{let } (iuis, \_) = mereo\_PE(pe) \text{ in } iuis \text{ end}$

xtr\_oUls:  $PE \rightarrow UI\text{-set}$ ,  $xtr\_oUls(u) \equiv \text{let } (\_, ouis) = mereo\_PE(pe) \text{ in } ouis \text{ end}$

120. The observed pipeline units of a pipeline system forms a net subject to the following constraints:

- (a) unit output connectors, if any, are connected to unit input connectors;
- (b) unit input connectors, if any, are connected to unit output connectors;
- (c) there are no cyclic routes;
- (d) nets has all their connectors connected, that is, “starts” with wells
- (e) and “ends” with sinks.

**value**

```

120. wf_Net: PL → Bool
120. wf_Net(pl) ≡
120. let cpes = all_pipeline_units{pl} in
120. ∀ pe:PE • pe ∈ cpes ⇒ let (iuis,ouis) = mereo_PE(pe) in
120. axiom 104.–108.
120a. ∧ ∀ pe.:UI•pe_ui ∈ iuis ⇒
120a. ∃ pe':PE•pe'≠pe∧pe'isin cpes∧uid_PE(pe')=pe_ui∧pe_ui∈extr_iUIs(pe')
120b. ∧ ∀ pe_ui:UI•pe_ui ∈ ouis ⇒
120b. ∃ pe':PE•pe'≠pe∧pe'isin cpes∧uid_PE(pe')=pe_ui∧pe_ui∈extr_oUIs(pe')
120c. ∧ ∀ r:R•r ∈ routes(pl) ⇒
120c. ∼∃ i,j:Nat•i≠j∧{i,j}∈ inds r∧r(i)=r(j)
120d. ∧ ∃ we:We • we ∈ us ∧ r(1) = mkWe(we)
120e. ∧ ∃ si:Si • si ∈ us ∧ r(len r) = mkSi(si)
110. end end

```

- **River Nets**

121. The mereology of a river vertex is a pair: a set of unique identifiers,  $E\_UI$ , of river edges, i.e., stretches, linear sequences of simple and composite river elements, incident upon the vertex, and a set of unique identifiers,  $E\_UI$ , of river edges emanating from the vertex. If the vertex is a source then the first element of this pair is empty. If the vertex is a mount then the second element of this pair is empty. For a confluence vertex both elements of the pair are non-empty.

122. The mereology of a river edge, that is, the linear sequence of simple and composite river elements between two adjacent vertices, is a pair: the first element is a unique identifier of a river vertex and so is the second element of the pair.

We present the river net mereology in two forms. The first was with respect to its graph rendition. The second is with respect to its river element rendition.

123. The mereology of a source is just the single unique identifier of the first simple or composite river element of the stretch emanating from the source.

124. The mereology of a confluence is a triplet: the single unique identifier of the last simple or composite river element of the stretch of the main river incident upon the source, a set of unique identifier of the last simple or composite river element of the stretches of the tributary rivers incident upon the source, and the single unique identifier of the first simple or composite river element of the main river stretch emanating from the confluence.

125. The mereology of a mouth is just the single unique identifier of the last simple or composite river element of the stretch incident upon the mouth

126. The mereologies of simple and composite river elements are pairs: of the unique identifier of the river elements, including sources and confluences, upstream adjacent to the river element being “mereologised”, and of the unique identifier of the river elements, including confluences and mouths, downstream adjacent to the river element being “mereologised”.

```

121. Mer_V = E_UI-set × E_UI-set
122. Mer_E = V_UI × V_UI
123. Mer_SO = SE_UI | CE_UI
124. Mer_KO = (SE_UI|CE_UI) × (SE_UI|CE_UI)-set × (SE_UI|CE_UI)
125. Mer_MO = SE_UI | CE_UI
126. Mer_RE = (SO_UI|CO_UI|SE_UI|CE_UI) × (SE_UI|CE_UI|CO_UI|MO_UI)

```



- 127. The unique vertex and edge identifiers must be identifiers of the vertices and edges of a graph.
- 128. Similarly, the unique source, confluence and mouth identifiers must be identifiers of respective sources, confluences and mouths of a graph.
- 129. And likewise for simple and composite element identifiers.
- 130. No two sources, confluences, mouths, simple and composite elements have identical unique identifiers.
- 131. There are other constraints, please state them !

**axiom**

- 127. [ left as exercise to the reader ! ]
- 128. [ left as exercise to the reader ! ]
- 129. [ left as exercise to the reader ! ]
- 130. [ left as exercise to the reader ! ]
- 131. [ left as exercise to the reader ! ]

**A.4.2.6 Attributes**

We refer to [58, Sect. 5.4]

*Attributes of discrete endurants ascribe to them such properties that endow these, typical manifest entities with substance.* *External* qualities of **endurants** allow us to reason about atomicity and compositions, whether as Cartesian-like products, as sets or as sequences; but not much more! The *internal quality* of **unique identification** allows us to speak of, i.e., analyze and describe multiplicities of same sort endurants. The *internal quality* of **mereology** allows us to relate discrete endurants either topologically or otherwise. But it is the *internal quality* of possessing one or more **attributes**, i.e., properties — usually many more than we may actually care to define, that really sets different sort parts “apart” (!) and allows us to reason more broadly, more domain-specifically, about endurants.

**A.4.2.6.1 Graph Labeling** It is quite common, in fact usually normal, to so-called “label” vertices and edges of graphs; that is, either none, or all, rarely only some proper subset. Such labeling is used for two distinct purposes: Either such labeling occur in only one of these forms, sometimes, though, in both; the situation is confusing. In our approach we clearly analyze labeling into two separate forms: the unique identification of distinct parts, and the ascription of attributes, sometimes the same, or “overlapping”<sup>6</sup> to more than one part, or even part sort. One should take care of the following: whereas distinct parts “receive” distinct unique identification, such distinct parts may be ascribed the same **attribute value**.

One may classify attributes in two different ways: into either *static*, *monitorable* and *programmable* as introduced by M.A. Jackson, [116], and as slightly “simplified” in [58, Sect. 5.4.2.3]; or as either measurable (by for example electro-, chemical or mechanical instruments) or referable (one can talk about histories of events) or both! Anyone part may be ascribed attributes of any mix and composition of these classifications.

If you sense some uneasiness about the issue of graph labeling as it is treated in for example operations, where graphs are a stable work horse, then you are right !

---

<sup>6</sup>By “overlapping” assignment of attribute to different parts we mean that two or more parts may be assigned the same **attribute type**.

**A.4.2.6.2 General Net Attributes** Let us informally recall some general facts about the concept of attributes such as we introduce them in [58, Sect. 5.4].

132. We can speak of the set of **names** of attribute types. If  $A$  is the type of an attributes, the  $\eta A$  is the name of that type.
133. For every part sort,  $P$ , we can thus speak of the set, or a suitably chosen, to be modeled, subset of attributes types in terms of their names.
134. Of course, different attribute names must designate distinct, i.e., non-overlapping attribute values of that type.

Likewise informally:

**type**

132.  $\eta A, A$

**value**

132.  $\text{name\_of\_attribute}: A \rightarrow \eta A, \eta \text{name\_of\_attribute}(A) \equiv \eta A$

133.  $\text{attributes}: P \rightarrow \eta A\text{-set}$

133.  $\text{attributes}(p) \equiv \{\eta A_i, \eta A_j, \dots, \eta A_k\}$

**axiom**

134.  $\forall p:P, \text{anms}:\{\eta A_i, \eta A_j, \dots, \eta A_k\}:\eta A\text{-set}:\text{anms} \subseteq \text{attributes}(p) \Rightarrow \forall i,j,\dots,k \bullet A_i \cap A_j = \{\} \wedge A_j \cap A_k = \{\} \wedge \dots$

**A.4.2.6.3 Road Net Attributes**

**Link Attributes:**

135. Standard “bookkeeping” link attributes are *road name*, *length*, name of *administrative authority* and others. These are static attributes.
136. Standard “control” attributes model the dynamically settable direction of flow along a link: its current link state as well as the space of all such link states.
137. Standard “event history” attributes model the time-stamped chronologically ordered sequence, for example latest first, of automobiles entering, stopping along (say parking) and leaving a link. A first element in such a list denotes “entering”. The last element “leaving”. Any element in-between, pairwise, “stopping” (for example for parking) and “starting” (resume driving).

**Hub Attributes:**

138. Standard “bookkeeping” hub attributes are *road intersection name*, name of *administrative authority* and others. These are static attributes.
139. Standard “control” attributes model the dynamically settable direction of flow along into and out of a hubs: its current hub state as well as the space of all such hub states.
140. Standard “event history” attributes model the time-stamped chronologically ordered sequence, for example latest first, of automobiles entering, stopping along (say parking) and leaving a hub. A first element in such a list denotes “entering”. The last element “leaving”. Any element in-between, pairwise, “stopping” (for example for parking) and “starting” (resume driving).
141. We assume a sort of automobile identifiers.

**type**

- 135. Road\_Name, Length, Admin\_Auth, ...
- 136.  $L\Sigma = (H\_UI \times H\_UI)\text{-set}$ ; **axiom**  $\forall l\sigma:L\Sigma \cdot \text{card } l\sigma \in \{0,1,2\}$ ;  $L\Omega = L\Sigma\text{-set}$
- 137.  $L\_History = A\_UI \xrightarrow{m} \text{TIME}^*$
- 138. Intersection\_Name, Admin\_Auth, ...
- 139.  $H\Sigma = (L\_UI \times L\_UI)\text{-set}$
- 139.  $H\Omega = H\Sigma\text{-set}$
- 140.  $H\_History = A\_UI \xrightarrow{m} \text{TIME}^*$
- 141. A\_UI

**value**

- 135. attr\_Road\_Name::  $L \rightarrow \text{Road\_Name}$ , attr\_Length:  $L \rightarrow \text{Length}$ , attr\_Admin\_Auth:  $L \rightarrow \text{Admin\_Auth}$ , ...
- 136. attr\_L\Sigma:  $L \rightarrow L\Sigma$ , attr\_L\Omega:  $L \rightarrow L\Omega$
- 137. attr\_L\_History:  $L \rightarrow L\_History$
- 138. attr\_Intersection\_name::  $H \rightarrow \text{Intersection\_name}$ , attr\_Admin\_Auth:  $H \rightarrow \text{Admin\_Auth}$ , ...
- 139. attr\_H\Sigma:  $H \rightarrow H\Sigma$ , attr\_H\Omega:  $H \rightarrow H\Omega$
- 140. attr\_H\_History:  $H \rightarrow H\_History$

We omit narrating and formalizing attributes for `Road_Surface_Temperature`, `Road_Maintenance_Condition`, etc., etc.

**Elucidation of Road Net History Attributes**

The above was a terse rendition. Below we elucidate, in two steps.

- **All Events are Historized!**

The above “story” on road net history attributes was a “lead-in”! To get you started on the notion of event histories. They are not recorded by anyone. They do occur. That is a fact. We can talk about them. So they are attributes. But they occur without our consciously talking about them. So they are chronicled.

- **More Detailed Road Unit Histories**

Also, the “story” was simplified. Here is a slightly more detailed history rendition of:

- 142. Attributed vents related to automobiles on roads.
- 143. Automobiles enter a link.
- 144. Automobiles stop along the link at a
- 145. fraction of the distance between the entered and the intended destination hubs.
- 146. Automobiles Restart.
- 147. Automobiles may make U-turns along a link at fraction of the distance between the entered and the originally intended destination hubs.
- 148. Eventually automobiles leave a link, entering a hub.
- 149. Same story for automobiles at a hub.

**type**

- 142.  $L\_Hist = A\_UI \xrightarrow{m} (A\_L\_Event \times \text{TIME})^*$
- 142.  $A\_L\_Event == \text{Enter} \mid \text{Stop} \mid \text{ReStart} \mid \text{U\_Turn} \mid \text{Leave}$
- 143.  $\text{Enter} :: H\_UI$
- 144.  $\text{Stop} :: H\_UI \times \text{Frac} \times H\_UI$
- 145.  $\text{Frac} = \text{Real}$ ; **axiom**  $\forall f:\text{Frac} \cdot 0 < f < 1$
- 146.  $\text{ReStart} :: \dots$
- 147.  $\text{U\_Turn} :: H\_UI \times \text{Frac} \times H\_UI$
- 148.  $\text{Leave} :: H\_UI$

- **Requirements: Recording Events**

So all events are chronicled. Not by the intervention of any device, but by “the sheer force of fate”! So be it — in the domain. But if you are to develop software for a road net application: be it a road pricing system, or a traffic control system, or other – something related to automobile and road events, then recording these events may be necessary. If so, you have to develop requirements from, for example, a domain description of this kind. We refer to [58, *Chapter 9: Requirements*]. More specifically you have to extend the domain, [58, *Sect. 9.4.4: Domain Extension*] – sensors that record the position of cars<sup>7</sup>. And this sensing may fail, and thus an implementation of the recording of hub and link histories may leave “holes” – and the requirements must then prescribe which kind of safeguards the thus extended road net system must provide.

#### A.4.2.7 Summing Up

**A.4.2.7.1 A Summary of The Example Endurant Models** We summarise “the tip of the icebergs” by recording here the main domains, but now in a concrete form; that is, with concrete types for main sorts instead of abstract types with observers.

River nets form graphs. Similarly can be done for all the examples. First we recall graphs.

- **Graphs:** See Items. 36 on page 94, 37 on page 94, 38 on page 94, 54 on page 97, 55 on page 97, 74 on page 100, 80 on page 100, 75 on page 100 and 82 on page 100.

**type** [Endurants]

36.  $G = V\text{-set} \times E\text{-set}$

37.  $V$

38.  $E$

**type** [Unique Identifiers]

54.  $V\_UI$

55.  $E\_UI$

**type** [Mereology]

74.  $V\_Mer = E\_UI\text{-set}$ ; 80.  $V\_Mer = E\_UI\text{-set} \times E\_UI\text{-set}$  [ Un-directed; Directed Graphs ]

75.  $E\_Mer = V\_UI\text{-set}$ ; 82.  $E\_Mer = (V\_UI \times V\_UI)\text{-set}$  [ Un-directed; Directed Graphs ]

- **Roads:** See Items 43 on page 95, 40 on page 95, 42 on page 95, 102 on page 104 and 103 on page 104.

**type** [Endurants]

40.  $RN = H\text{-set} \times L\text{-set}$  [  $\simeq G$  for Graphs ]

41.  $H$  [  $\simeq V$  for Graphs ]

42.  $L$  [  $\simeq E$  for Graphs ]

**type** [Unique Identifiers]

63.  $HI$  [  $\equiv VI$  for Graphs ]

64.  $LI$  [  $\equiv EI$  for Graphs ]

**type** [Mereology]

102.  $H\_Mer = LI\text{-set} \times LI\text{-set}$  [  $\simeq V\_Mer$  for Graphs ]

103.  $L\_Mer = (HI \times HI)\text{-set}$ , **axiom**  $\forall lm: L\_Mer \cdot \text{card}lm \in \{0,1,2\}$  [  $\equiv E\_Mer$  for Graphs ]

- **Rails:** See Chapter F.
- **Pipelines:** See Items 45 on page 95, 46 on page 95, 67 on page 99 and 109 on page 104.

<sup>7</sup>These sensors may be photo-electric or electronic and placed at suitable points along the road net, or they may be satellite borne. To work properly we assume that automobiles emit such signals that let their identity be recorded.

**type** [Endurants]  
 45.  $PN = PLU\text{-set}$  [  $\simeq G$  for Graphs ]  
 46.  $PLU == WU \mid PU \mid LU \mid VU \mid FU \mid JU \mid SU$  [  $\simeq (V|E)$  for Graphs ]  
**type** [Unique Identifiers]  
 67.  $UI == WU\_UI \mid PU\_UI \mid LU\_UI \mid VU\_UI \mid FU\_UI \mid JU\_UI \mid SU\_UI$   
**type** [Mereology]  
 109  $PM'=(UI\text{-set} \times UI\text{-set}), PM=\{(iuis,ouis):PM' \bullet iuis \cap ouis=\{\}\}$  [  $\simeq V\_Mer \cup E\_Mer$  for Graphs ]

- **Rivers:** See Items 48 on page 96, 49 on page 96, 50 on page 96, Sect. A.4.2.4.4 on page 99, 121 on page 106 and 122 on page 106.

**type** [Endurants]  
 48.  $RiN$   
 49.  $V$   
 50.  $E$   
**type** [Unique Identifiers]  
 68.  $V\_UI, E\_UI$   
**type** [Mereology]  
 121.  $Mer\_V = E\_UI\text{-set} \times E\_UI\text{-set}$   
 122.  $Mer\_E = V\_UI \times V\_UI$

**A.4.2.7.2 Initial Conclusion on Labeled Graphs and Example Domains** We have shown basic models of abstract undirected and directed graphs. And we have shown four examples:

- road nets,
- rail nets,
- pipeline nets and
- river nets.

Road, rail, pipeline and river elements are all uniquely identified. The road and river nets were basically modeled as as graphs with vertices (hubs, respectively sources, confluences and mouths) and edges (links, respectively stretches of simple and composite river elements). The rail and pipeline nets we modeled as sets of rail and pipeline units with the mereology implying edges.

Labels, such as they are “practiced” in conventional graph theory, are introduced by may of attributes. Attributes were also used to model dynamically varying “directedness” of edges.

We can conclude the following

- There is now a firm foundation for the labeling of graphs:
  - *the origin of vertex and edge labeling is*
    - \* *the unique identifiers and/or*
    - \* *the attributes**of the endurant parts that vertices and edges designate; and*
  - *there really can be no vertex or edge labeling unless the origin is motivated in*
    - \* *the unique identification and/or*
    - \* *the attribution**of the vertex and edge parts.*

## A.5 The Nets Domain

### A.5.1 Some Introductory Definitions

**Definition:** By a **net domain**, or, for short, just a **net**, we shall understand a domain of the kind illustrated in Sect. A.4, that is, a domain the mereology of whose main parts model graphs ■

**Definition:** By a **dynamic net domain**, or, for short, just a **dynamic net**, we shall understand a *net* whose *mereology* – or a corresponding attribute notion – may change ■

**Definition:** By a **nets domain**, or, for short, just **nets**, (notice the suffix ‘s’, we shall understand a domain each of whose instances is a *dynamic net domain* ■

MORE TO COME

# Appendix B

## Rivers

### Contents

---

|                                               |            |
|-----------------------------------------------|------------|
| <b>B.1 Introduction</b>                       | <b>113</b> |
| B.1.1 Waterways                               | 113        |
| B.1.2 Visualization of Rivers                 | 114        |
| B.1.2.1 Rivers                                | 114        |
| B.1.2.2 Deltas                                | 114        |
| B.1.3 Structure of This Report                | 114        |
| <b>B.2 External Qualities – The Endurants</b> | <b>115</b> |
| <b>B.3 Internal Qualities</b>                 | <b>116</b> |
| B.3.1 Unique Identifiers                      | 116        |
| B.3.2 Mereologies                             | 117        |
| B.3.3 Routes                                  | 118        |
| B.3.4 Attributes                              | 119        |
| <b>B.4 Conclusion</b>                         | <b>120</b> |

---

Presently this document represents a technical-scientific note. It is technical in that much of the material can be found in other technical notes of mine. It is – perhaps – scientific in that I am searching for a nice, well, beautiful, way of modeling rivers.

## B.1 Introduction

### B.1.1 Waterways

By waterways we mean rivers, canals, lakes and oceans – such as are navigable by vessels: barges, boats and ships.

Rivers are naturally flowing watercourses, and typically flow until discharging their water into a lake, sea, ocean, or another river, while canals are constructed to connect existing rivers, seas, or lakes. However, occasionally some rivers do not discharge their water into lakes, seas, oceans, or other rivers. Rivers that do not empty into another body of water might flow into the ground or simply dry up before reaching another body of water. Additionally, small rivers can also be referred to as streams, rivulets, creeks, rills, or brooks.

The natural water system of the earth includes 71% ocean with land continents being traversed by brooks, rivers, lakes and river deltas.

Headwaters are streams and rivers (tributaries) that are the source of a stream or river.

A tributary is a river or stream that flows into another stream, river, or lake.

A delta is a large, silty area at the mouth of a river at which the river splits into many different slow-flowing channels that have muddy banks. New land is created at deltas. Deltas are often triangular-shaped, hence the name (the Greek letter 'delta' is shaped like a triangle).



The trunk is the main course of river.

**Confluence:** In geography, a confluence (also: conflux) occurs where two or more flowing bodies of water join together to form a single flow. A confluence can occur in several configurations: at the point where a tributary joins a larger river (main stem); or where two streams meet to become the source of a river of a new name; or where two separated channels of a river (forming a river island) rejoin at the downstream end.

**Towns and Harbours:** In this report we model towns. That is, we therefore also model that towns have harbours – allowing river (and canal) vessels to berth (a place for mooring in a harbour) for cargo loading, unloading and resting.

### B.1.2 Visualization of Rivers

#### B.1.2.1 Rivers

Figures B.1 and B.2 illustrate a number of rivers.

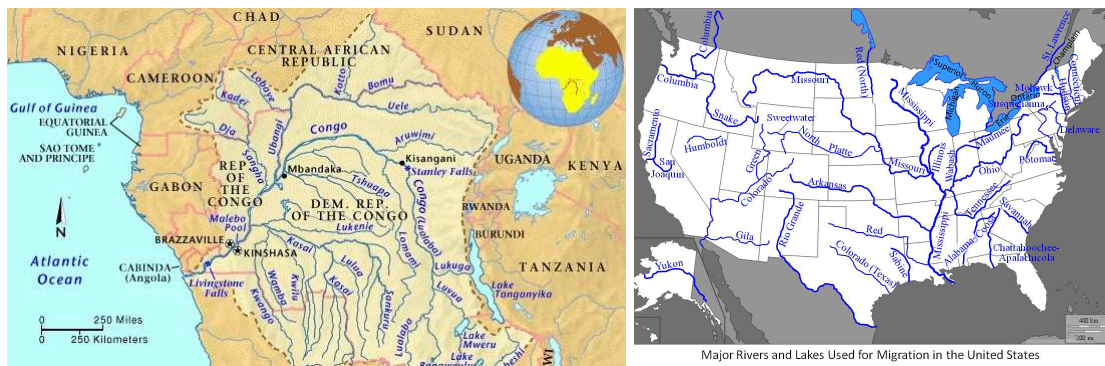


Figure B.1: The Congo and the US Rivers



Figure B.2: The Amazon and The Danube Rivers

#### B.1.2.2 Deltas

We illustrate four deltas, Fig. B.3 on the next page:

### B.1.3 Structure of This Report

Rivers are narrated and formalized in Sects.:



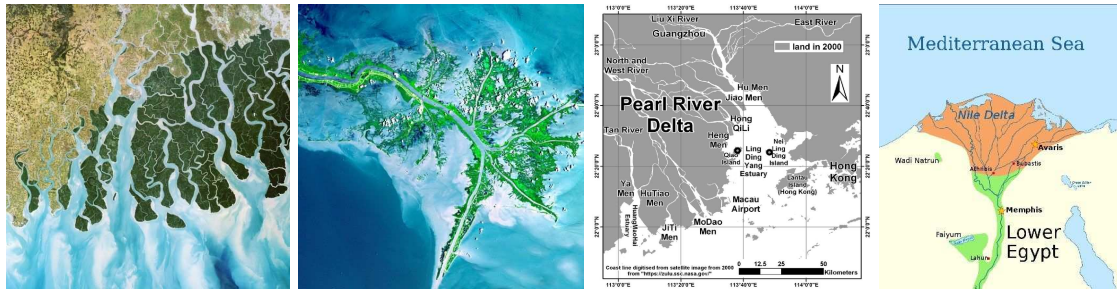


Figure B.3: The Ganges, Mississippi, Pearl and the Nile Deltas

- B.2 [Endurants],
- B.3.1 [Unique Identifiers],
- B.3.2 [Mereology], and
- B.3.4 [Attributes].

We omit from this compendium references to a number of ‘River Terminologies’.

## B.2 External Qualities – The Endurants

150. A river net is modeled as a graph, more specifically as a tree. The *root* of that river net tree is the mouth (or delta) of the river net. The *leaves* of that river net tree are the sources of respective trees. Paths from leaves to the root define *flows* of water.
151. We can thus, from a river net observe vertices
152. and edges.
153. River vertices model either a *source*: **so:SO**, a *mouth*: **mo:MO**, or possibly some *confluence*: **ko:KO**.  
A river may thus be “punctuated” by zero or more confluences, k:KO.  
A confluence defines the joining a ‘main’ river with zero<sup>1</sup> or more rivers into that ‘main’ river.  
We can talk about the “upstream” and the “downstream” of rivers from their confluence.
154. River edges model *stretches*: **st:ST**.  
A stretch is a linear sequences of simple, **se:SE**, or composite **ce:CE**, river elements.
155. River elements are either simple: (ch) river channels, which we shall call *river channels*: **CH**, or (la) lakes: **LA**, or (lo) locks: **LO**, or (wa) waterfalls (or *rapids*): **WA**, or (da) dams: **DA**, or (to) towns (cities, villages): **to:TO**<sup>2</sup>; or composite, **ce:CE**: a dam with a lock, (**da:DA,la:LA**), a town with a lake, (**to:TO,la:LA**), etcetera; even a town with a lake and a confluence, **to:TO,la:LA,ko:KO**. Etcetera.

### type

150. RiN  
151. V  
152. E  
153. SO, MO, KO  
154. ST = (SE|CE)\*  
155. CH, LA, LO, WA, KO, DA, TO

<sup>1</sup>Normally, though, one would expect, not zero, but one

<sup>2</sup>Towns is here really a synonym for river harbours, places along the river (or a canal) where river vessels can stop (moor) for the loading and unloading of cargo and for resting.

155.  $SE = CH \mid LA \mid LO \mid WA \mid DA \mid TO$   
 155.  $DaLo, WaLo, ToLa, ToLaKo, \dots$   
 155.  $CE = DaLo \mid WaLo \mid ToLa \mid ToLaKo \mid \dots$   
**value**  
 153.  $obs\_Vs: RiN \rightarrow V\text{-set}$   
 153. **axiom**  
 153.  $\forall g:G, vs:V\text{-set} \bullet vs \in obs\_Vs(g) \Rightarrow vs \neq \{\}$   
 153.  $\wedge \forall v:V \bullet v \in vs \Rightarrow is\_SO(v) \vee is\_KO(v) \vee is\_MO(v)$   
 154.  $obs\_Es: RiN \rightarrow E\text{-set}$   
 154. **axiom**  
 154.  $\forall g:G, es:E\text{-set} \bullet es \in obs\_Es(g) \Rightarrow es \neq \{\}$   
 154.  $\wedge \forall e:E \bullet e \in es \Rightarrow is\_ST(e)$   
 154.  $obs\_ST: E \rightarrow ST$   
 150.  $xtr\_In\_Degree\_0\_Vertices: RiN \rightarrow SO\text{-set}$   
 150.  $xtr\_Out\_Degree\_0\_Vertex: RiN \rightarrow MO$

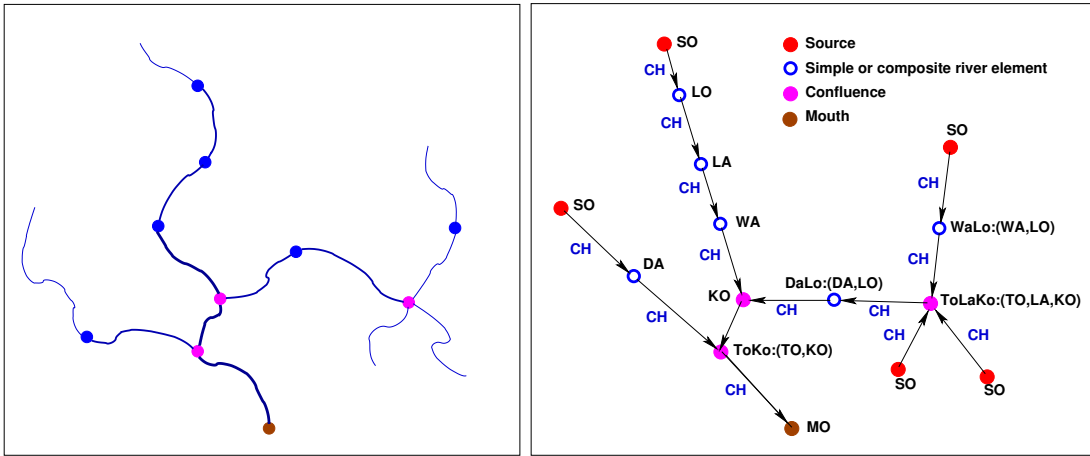


Figure B.4: The “Composition” of a River Net: Right Tree is an abstraction of the Left Tree

## B.3 Internal Qualities

We refer to [58, Chapter 5]

### B.3.1 Unique Identifiers

We shall associate unique identifiers both with vertices, edges and vertex and edge elements.

156. River net vertices and edges have unique identifiers.  
 157. River net sources, confluences and mouths have unique identifiers.  
 158. River net stretches have unique identifiers.  
 159. River net channels, lakes, locks, waterfalls, dams and towns as well as combinations of these, that is, simple and composite river entities have unique identifiers.

**type**

156.  $V\_UI, E\_UI$

157. SO\_UI, KO\_UI, MO\_UI  
 158. ST\_UI  
 159. CH\_UI, LA\_UI, LO\_UI, WA\_UI, DA\_UI, TO\_UI, DaLo\_UI, WaLo\_UI, ToLa\_UI, ToLaKo\_UI, ...  
**value**  
 156. uid\_V: V→V\_UI, uid\_E: E→ E\_UI  
 157. uid\_SO: SO→SO\_UI, uid\_KO: KO→KO\_UI, uid\_MO: MO→MO\_UI,  
 158. uid\_ST: ST→ST\_UI  
 159. uid\_CH: CH→CH\_UI, uid\_LA: LA→LA\_UI, uid\_LO: LO→LO\_UI, uid\_WA: WA→WA\_UI,  
 159. uid\_DA: DA→DA\_UI, uid\_TO: TO→TO\_UI,  
 159. uid\_DaLo: DaLo→DaLo\_UI, uid\_WaLo: WaLo→WaLo\_UI, uid\_ToLa: ToLa→ToLa\_UI,  
 159. uid\_ToLaKo: ToLaKo→ToLaKo\_UI, ...

160. All these identifiers are distinct.

The  $\mathfrak{m}$  operator takes the pairwise intersection of the types in its argument list and examines them for disjointness.

**axiom**

160.  $\mathfrak{m}(V\_UI, E\_UI, SO\_UI, KO\_UI, MO\_UI, ST\_UI, CH\_UI,$   
 160.  $LA\_UI, LO\_UI, WA\_UI, DA\_UI, TO\_UI, DaLo\_UI, WaLo\_UI, ToLa\_UI, ToLaKo\_UI)$

161. There are [many] other constraints, please state them !

161. [ left as exercise to the reader ! ]

### B.3.2 Mereologies

162. The mereology of a river vertex is a pair: a set of unique identifiers, E\_UI, of river edges, i.e., stretches, linear sequences of simple and composite river elements, incident upon the vertex, and a set of unique identifiers, E\_UI, of river edges emanating from the vertex. If the vertex is a source then the first element of this pair is empty. If the vertex is a mount then the second element of this pair is empty. For a confluence vertex both elements of the pair are non-empty.
163. The mereology of a river edge, that is, the linear sequence of simple and composite river elements between two adjacent vertices, is a pair: the first element is a unique identifier of a river vertex and so is the second element of the pair.

We present the river net mereology in two forms. The first was with respect to its graph rendition. The second is with respect to its river element rendition.

164. The mereology of a source is just the single unique identifier of the first simple or composite river element of the stretch emanating from the source.
165. The mereology of a confluence is a triplet: the single unique identifier of the last simple or composite river element of the stretch of the main river incident upon the source, a set of unique identifier of the last simple or composite river element of the stretches of the tributary rivers incident upon the source, and the single unique identifier of the first simple or composite river element of the main river stretch emanating from the confluence.
166. The mereology of a mouth is just the single unique identifier of the last simple or composite river element of the stretch incident upon the mouth

167. The mereologies of simple and composite river elements are pairs: of the unique identifier of the river elements, including sources and confluences, upstream adjacent to the river element being “mereologised”, and of the unique identifier of the river elements, including confluences and mouths, downstream adjacent to the river element being “mereologised”.
162.  $Mer\_V = E\_UI\text{-set} \times E\_UI\text{-set}$
163.  $Mer\_E = V\_UI \times V\_UI$
164.  $Mer\_SO = SE\_UI \mid CE\_UI$
165.  $Mer\_KO = (SE\_UI \mid CE\_UI) \times (SE\_UI \mid CE\_UI)\text{-set} \times (SE\_UI \mid CE\_UI)$
166.  $Mer\_MO = SE\_UI \mid CE\_UI$
167.  $Mer\_RE = (SO\_UI \mid CO\_UI \mid SE\_UI \mid CE\_UI) \times (SE\_UI \mid CE\_UI \mid CO\_UI \mid MO\_UI)$
168. The unique vertex and edge identifiers must be identifiers of the vertices and edges of a graph.
169. Similarly, the unique source, confluence and mouth identifiers must be identifiers of respective sources, confluences and mouths of a graph.
170. And likewise for simple and composite element identifiers.
171. No two sources, confluences, mouths, simple and composite elements have identical unique identifiers.
172. There are other constraints, please state them!

**axiom**

168. [ left as exercise to the reader ! ]
169. [ left as exercise to the reader ! ]
170. [ left as exercise to the reader ! ]
171. [ left as exercise to the reader ! ]
172. [ left as exercise to the reader ! ]

**B.3.3 Routes**

173. A vertex-edge-vertex path is a sequence of zero or more edges. We define the `edge_paths` function – recursively.
174. That is, the empty sequence,  $\langle \rangle$ , is a vertex-edge-vertex path, [the first basis clause].
175. If  $e$  is an edge of  $g$ , and if  $(vi, vj)$  is in the mereology of  $e$ , then the  $\langle (vi, ej, vk) \rangle$ , where  $ej$  is the unique identifier of  $e$  is a vertex-edge-vertex path.
176. If  $p$  and  $p'$  are paths of  $g$  such that the last vertex identifier of the last element of  $p$  is the same as the first vertex identifier of the first element of  $p'$ , then the sequence  $p$  followed by the sequence  $p'$  is a vertex-edge-vertex path of  $g$  [the inductive clause].
177. Only such paths which can be constructed by the above rules are edge paths [the extremal clause].

**type**

173.  $EP = E^\omega$
173. `edge_paths`:  $G \rightarrow EP\text{-set}$
173. `edge_paths(g)`  $\equiv$
174. **let**  $ps = \{ \langle \rangle \}$
175.  $\cup \{ \langle (vi, uid\_E(e), vk) \rangle \mid e: E \bullet e \in \text{xtr\_Es}(g) \wedge (vi, vk) \in \text{mereo\_E}(e) \}$
176.  $\cup \{ p \hat{\ } p' \mid p, p': EP \bullet \{ p, p' \} \subseteq ps \wedge \forall i \exists j (EP(9) = i \wedge j \in EP(p')) \}$  **in**
173. **ps end**

### B.3.4 Attributes

This author is not “an expert” on neither geographical matters relating to rivers, lakes, etc., nor on the management of rivers: flood control, river traffic, etc. So, please, do not expect a very illuminating set of river attribute examples. All the attribute specifications are “tuned” to the purpose of the ensuing domain description: whether for one or another form of river system study or eventual software system realisation.

- |                                                                                                                              |                                                                        |
|------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------|
| 178. River entities have geodetical positions –                                                                              | 184. Locks have ... et cetera                                          |
| 179. all three dimensions: longitude, latitude and altitude <sup>3</sup> .                                                   | 185. Waterfalls ...                                                    |
| 180. River entities cover geodetical areas <sup>4</sup> .                                                                    | 186. Dams ...                                                          |
| 181. River entities have normal, low, high and overflow water levels <sup>5</sup> .                                          | 187. Towns ...                                                         |
| 182. River channels have “extent” in the form, for example of a precise description <sup>6</sup> of its course. <sup>7</sup> | 188. Sources ...                                                       |
| 183. Lakes have a precise [three dimensional] description of their form, ...                                                 | 189. Confluences ...                                                   |
|                                                                                                                              | 190. Mouths ...                                                        |
|                                                                                                                              | 191. Compositions of these have respective unions of these attributes. |

#### type

178.  $\text{GeoPos} = \text{Long} \times \text{Lat} \times \text{Alt}$   
 179. Long, Lat, Alt  
 181. Area  
 181.  $\text{LoWL} = \dots, \text{NoWL} = \dots, \text{HiWL} = \dots, \text{OfWL} = \dots$   
 182. Course = ...  
 183. LakeForm = ...  
 185. ...; 186. ...; 187. ...; 188. ...; 189. ...; 190. ...; 191. ...

#### value

178.  $\text{attr\_GeoPos}: (\text{SO}|\text{KO}|\text{MO}|\text{SE}|\text{CE}) \rightarrow \text{GeoPos}$   
 179.  $\text{attr\_Long}: \text{GeoPos} \rightarrow \dots, \text{attr\_Lat}: \text{GeoPos} \rightarrow \dots, \text{attr\_Alt}: \text{GeoPos} \rightarrow \dots$   
 181.  $\text{attr\_Area}: (\text{SO}|\text{KO}|\text{MO}|\text{SE}|\text{CE}) \rightarrow \text{Area}$   
 181.  $\text{attr\_}(\text{LoWL}|\text{NoWL}|\text{HiWL}|\text{OfWL}): (\text{SO}|\text{KO}|\text{MO}|\text{SE}|\text{CE}) \rightarrow \text{LoWL}|\text{NoWL}|\text{HiWL}|\text{OfWL}$   
 182.  $\text{attr\_CH}: \text{CH} \rightarrow \text{Course}$   
 183.  $\text{attr\_LakeForm}: \text{LA} \rightarrow \text{LakeForm}$   
 184.  $\text{attr\_} \dots: \text{LO} \rightarrow \dots; 185. \text{attr\_} \dots: \text{WF} \rightarrow \dots; 186. \text{attr\_} \dots: \text{DA} \rightarrow \dots; 187. \text{attr\_} \dots: \text{TO} \rightarrow \dots;$   
 188.  $\text{attr\_} \dots: \text{SO} \rightarrow \dots; 189. \text{attr\_} \dots: \text{KO} \rightarrow \dots; 190. \text{attr\_} \dots: \text{MO} \rightarrow \dots; 191. \text{attr\_} \dots: \dots \rightarrow \dots$

We illustrate the issue of river attributes primarily to show you the sheer size and complexity of the task!

192. River entities have positions “within” their areas<sup>8</sup>.  
 193. No two distinct river entities have conflicting (?) areas<sup>9</sup>.

<sup>3</sup>These are facts: How we represent them is a matter for geographers. Also: What is really meant by the ‘position’ of a source, or a river channel, etc.? Also that is left for others to care about!

<sup>4</sup>See Footnote 3.

<sup>5</sup>See Footnote 3.

<sup>6</sup>See Footnote 3. In any **domain description**, yes, a precise description – whether “computable” [i.e., realizable] or not!

<sup>7</sup>– in a subsequent **requirements prescription** the domain description’s “precise” form is replaced by, for example, a reasonably detailed [and computable] three dimensional *Bézier curve* specification [[en.wikipedia.org/wiki/B%C3%A9zier\\_curve](http://en.wikipedia.org/wiki/B%C3%A9zier_curve)].

<sup>8</sup>See Footnote 3.

<sup>9</sup>For example: their areas do not overlap. See Footnote 3.

194. Two mereologically immediately adjacent river entities have bordering areas<sup>10</sup>.

195.

196.

Axiom 193 is rather “sweeping”. It implies, of course, that river channels do not cross one another; that two or more non-channel river entities similarly do not “interfere” with one another, i.e., are truly “separate”.

## B.4 Conclusion

TO BE WRITTEN

---

<sup>10</sup>See Footnote 3.

# Appendix C

## Canals

### Contents

---

|            |                                                   |            |
|------------|---------------------------------------------------|------------|
| <b>C.1</b> | <b>Introduction</b>                               | <b>122</b> |
| <b>C.2</b> | <b>Visualisation of Canals</b>                    | <b>122</b> |
| C.2.1      | Canals and Water Systems                          | 122        |
| C.2.2      | Locks                                             | 122        |
| <b>C.3</b> | <b>The Endurants</b>                              | <b>123</b> |
| C.3.1      | Some Introductory Remarks                         | 123        |
| C.3.1.1    | The Dutch Polder System                           | 123        |
| C.3.1.2    | Natural versus Artefactual Domains                | 124        |
| C.3.1.3    | Editorial Remarks                                 | 124        |
| C.3.1.4    | A Broad Sketch Narrative of Canal System Entities | 125        |
| C.3.1.5    | A Plan for The Canal System Description           | 126        |
| C.3.1.6    | No Structures                                     | 127        |
| C.3.1.7    | Sequences of Presentation                         | 128        |
| C.3.1.8    | Naming Conventions                                | 128        |
| C.3.2      | External Qualities                                | 128        |
| C.3.2.1    | Endurant Sorts                                    | 128        |
| C.3.2.2    | Some Calculations                                 | 131        |
| C.3.3      | Internal Qualities                                | 134        |
| C.3.3.1    | Unique Identifiers                                | 134        |
| C.3.3.1.1  | Unique Identifier Sorts                           | 134        |
| C.3.3.1.2  | Some Calculations                                 | 134        |
| C.3.3.1.3  | An Axiom                                          | 136        |
| C.3.3.1.4  | Another Representation of UI Values               | 136        |
| C.3.3.1.5  | An Extract Function                               | 137        |
| C.3.3.2    | Mereologies                                       | 137        |
| C.3.3.2.1  | Mereology Types                                   | 137        |
| C.3.3.2.2  | The Mereology Axiom                               | 141        |
| C.3.3.2.3  | Well-formed Mereologies                           | 141        |
| C.3.3.3    | Routes                                            | 147        |
| C.3.3.3.1  | Preliminaries                                     | 147        |
| C.3.3.3.2  | All Routes                                        | 148        |
| C.3.3.3.3  | Connected Canal Systems                           | 148        |
| C.3.3.3.4  | A Canal System Axiom                              | 149        |
| C.3.3.4    | Attributes                                        | 149        |
| C.3.3.4.1  | Spatial and Temporal Attributes                   | 149        |

C.3.3.4.2 Canal System, Net and Polder Attributes . . . . . 151  
 C.3.3.4.3 Canal Hub and Link Attributes . . . . . 151  
 C.3.3.5 Well-formedness of Attributes . . . . . 153  
 C.3.4 Speculations . . . . . 153  
 C.4 Conclusion . . . . . 153

Presently this document represents a technical-scientific note. It is technical in that much of the material can be found in other technical notes of mine. It is – perhaps – scientific in that I am searching for a nice, well, beautiful, way of modeling canals, such as for example those of the Dutch Rijkswaterstaat<sup>1</sup>. I am fascinated with Holland’s tackling of their land/water/river/ocean levels.

## C.1 Introduction

Canals are artificial or human-made channels or waterways that are used for navigation, transporting water, crop irrigation, or drainage purposes. Therefore, a canal can be considered an artificial version of a river. Canals are artificial or human-made channels or waterways that are used for navigation, transporting water, crop irrigation, or drainage purposes. Therefore, a canal can be considered an artificial version of a river.

## C.2 Visualisation of Canals

### C.2.1 Canals and Water Systems

We illustrate just four ship/barge/boat and water level control canal systems, Figs. C.1, C.2, and C.3 on the facing page.

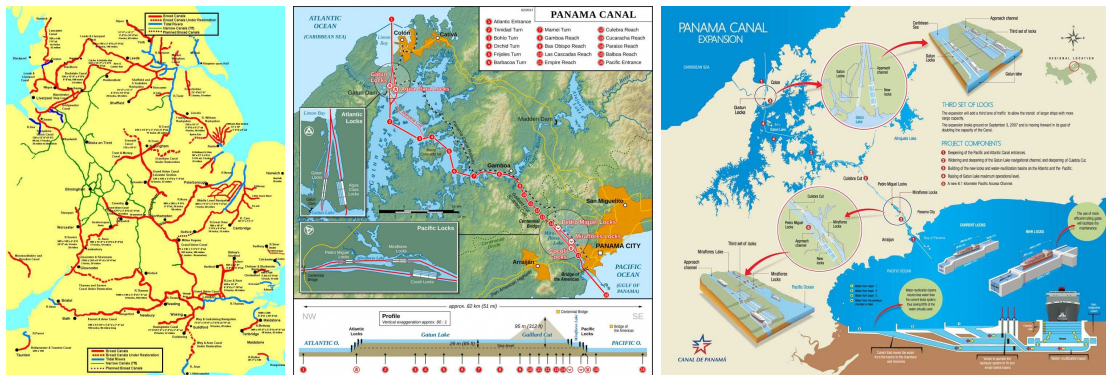


Figure C.1: UK Canals and The Panama Canal

The rightmost figure of Fig. C.3 is from the Dutch *Rijkswaterstaat*: [www.rijkswaterstaat.nl/english/](http://www.rijkswaterstaat.nl/english/).

### C.2.2 Locks

A lock is a device used for raising and lowering boats, ships and other watercraft between stretches of water of different levels on river and canal waterways. The distinguishing feature of a lock is a fixed chamber in which the water level can be varied. Locks are used to make a river more easily navigable, or to allow a canal to cross land that is not level. Later canals used more and larger locks to allow a more direct route to be taken.<sup>2</sup>

<sup>1</sup><https://www.rijkswaterstaat.nl/> The Dutch canal system is first and foremost, it appears, for the control of water levels, secondly for ship/barge/boat navigation.

<sup>2</sup>[https://en.wikipedia.org/wiki/Lock\\_\(water\\_navigation\)](https://en.wikipedia.org/wiki/Lock_(water_navigation))



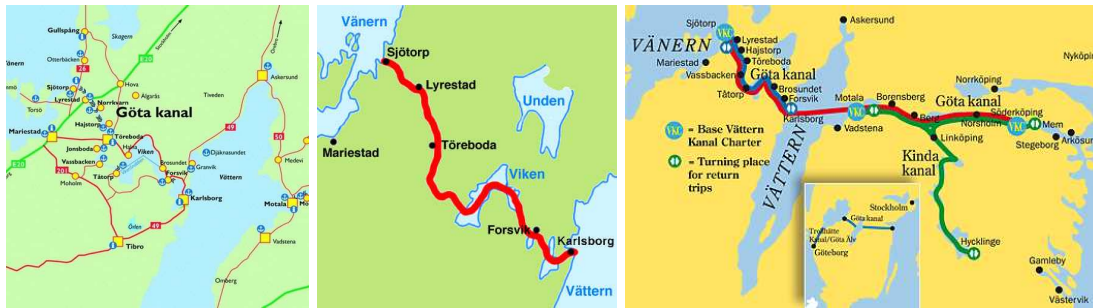


Figure C.2: The Swedish Göta Kanal

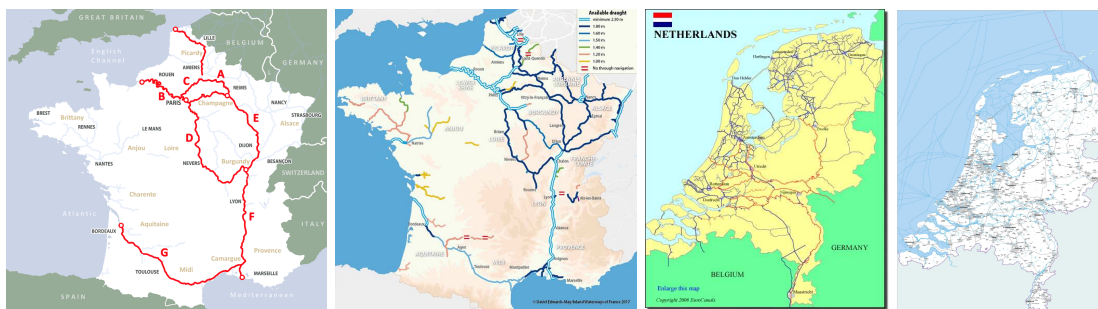


Figure C.3: French and Dutch Rivers and Canals

We illustrate a number of locks: Figs. C.4 and C.5 on the following page.

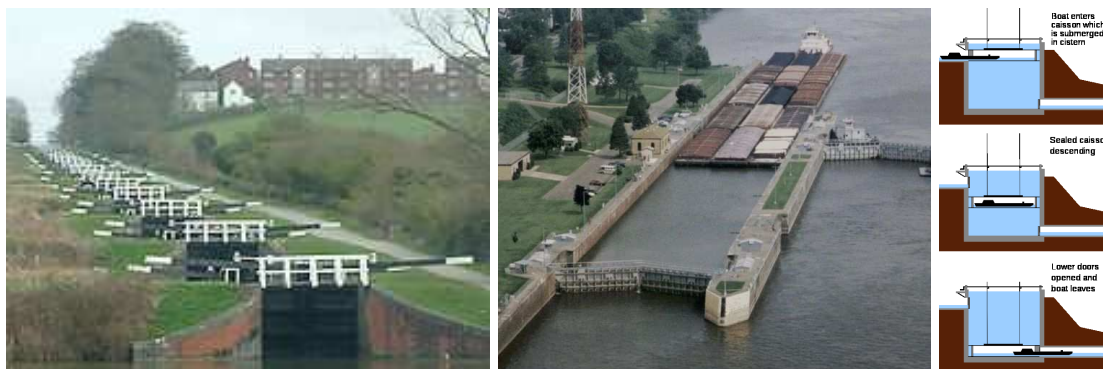


Figure C.4: Inland Canal Locks

## C.3 The Endurants

As an example we wish our model to include the Dutch system of *polders, pumps, canals, locks, dikes, flood barriers, lakes, storm barriers* and the *ocean*.<sup>3</sup>

### C.3.1 Some Introductory Remarks

#### C.3.1.1 The Dutch Polder System

We refer to Figs. C.7 to C.10 on pages 125–127.

<sup>3</sup>[www.fao.org/fileadmin/templates/giahs/PDF/Dutch-Polder-System\\_2010.pdf](http://www.fao.org/fileadmin/templates/giahs/PDF/Dutch-Polder-System_2010.pdf)

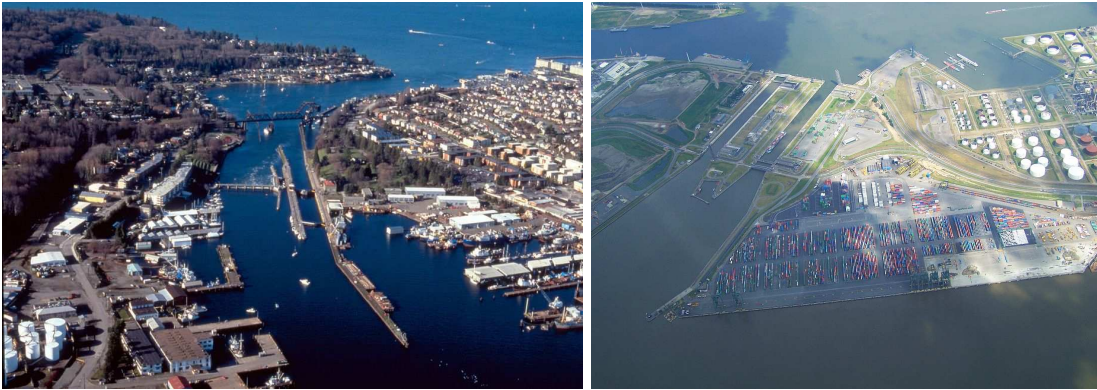


Figure C.5: Harbour Canal Locks



Figure C.6: The Dutch Polder System

### C.3.1.2 Natural versus Artefactual Domains

In contrast to river nets modeled earlier in this compendium, a system of mostly natural endurants, canal systems of *polders*, *pumps*, *canals*, *locks*, *dikes*, *flood barriers*, *lakes*, *storm barriers* and the *ocean* are, in a sense, dominated by man-made, i.e., artefactual endurants.

### C.3.1.3 Editorial Remarks

In order to develop an appropriate domain analysis & description of a reasonably comprehensive and representative canal domain I need answers to the following questions – and may more that can be derived from answer to these questions:

- **Canal Flow:** Are canals generally stagnant, or do canal water flow, that is, do canal flow have a preferred direction?
- **Canal Graphs:** Do a canal form a[n undirected] graph, i.e., can canals be confluent with other canals?
- **Canals and Rivers:** It is assumed that canals can be confluent with rivers. When canals join a river is it always with a lock of the canal onto the river – and the river flow is basically not interfered with by that canal, or otherwise?
- **Canal Levels:** Can a canal pass a river overhead? Or otherwise? Same for canals and roads. Do canals run through mountains or over valleys?
- **Canal Locks:** It is assumed that an otherwise “unhindered” stretch of canal can have one or more locks. Yes or no?

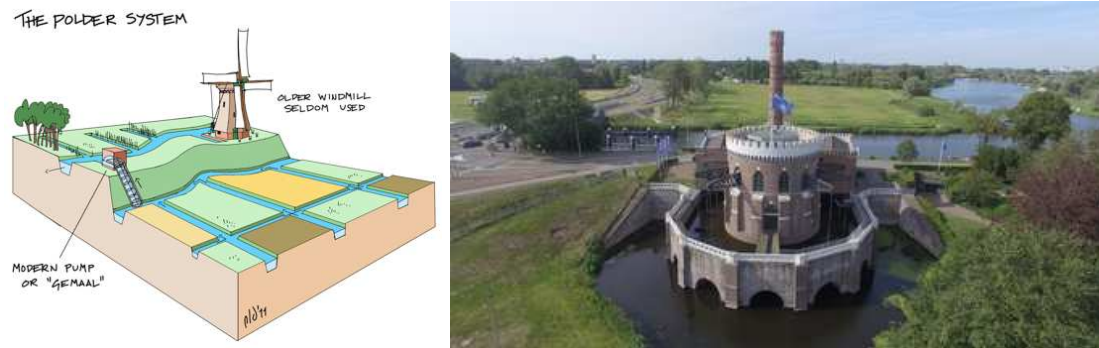


Figure C.7: A Polder Schematic and The De Cruquius Pump



Figure C.8: A Polder. Another Polder Schematic

- **Canal Pumps:** It is assumed that there are two kinds of canal pumps: those in connection with locks, and those **not** in connection with any locks. Yes or no? I need information about the latter.
- **Polders and Pumps:** Are polders predominantly characterisable in terms of their land area and the pumps that keep these dry?
- **Pumps and Canals:** Do polder pumps always operate in the context of canals?

#### C.3.1.4 A Broad Sketch Narrative of Canal System Entities

- We take our departure point in the polders: So a polder-etc.-canal system contains **polders** and **polder pumps** take the water out of the polders and “puts” it in higher level **canals**.
- **Canals** are modeled as an undirected [general] graph whose vertices are **canal entities** and whose edges are given by the mereology of these entities – as to how they are topologically connected.
- The following are **canal entities**:
  - **canal channels:** like river channels, only artefactual;
  - **canal locks:** the locks as illustrated earlier;
  - **canal pumps:** pumps water into locks – rather than using water from higher level canals;
  - **canal gates:** protects the interior from ocean storm surges.



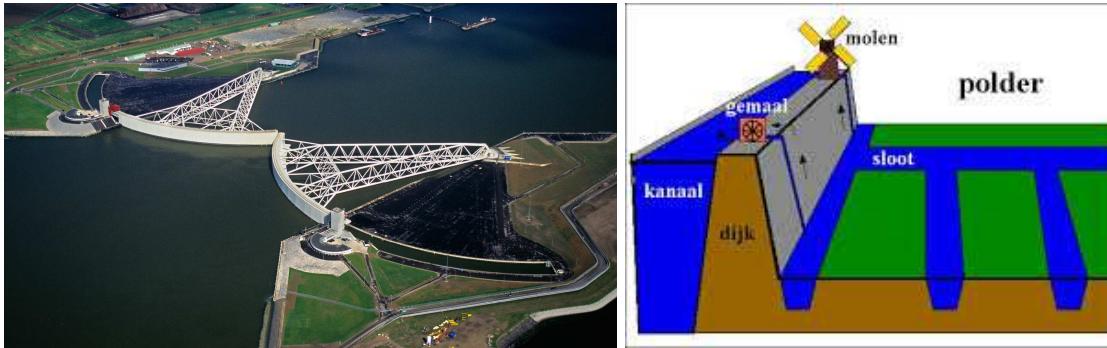


Figure C.9: A Barrier. A Final Polder Schematic

### C.3.1.5 A Plan for The Canal System Description

Our plan is to analyse & describe

- external qualities of canal system endurants, Sect. C.3.2.
- internal qualities of canal system, Sect. C.3.3.1
- internal qualities of canal system, Sect. C.3.3.2
- internal qualities of canal system, Sect. C.3.3.4

For each of these categories we analyse & describe

- **sorts** and **types** of these entities: endurants, unique identifiers, mereologies and attributes;
- **observer functions**, i.e., **obs\_...**, **uid\_...** and **attr\_...** for the observance of endurants, their unique identifiers, their mereologies and their attributes;
- **auxiliary functions** and
- **well-formedness predicates** **is\_wf\_...**

The external and internal quality definitions should be so conceived by the domain analyser & describer as to capture an essence, if not “the essence”, of endurants. But they can never capture the essence “completely”. As for the relation between *context free grammars* and *context sensitive grammars*, we must therefore introduce the notion of **well-formedness axioms**. The axioms constrain the relations between external and the various categories of internal qualities. More specifically:

197. The well-formedness of a canal system,  $is\_wf\_CS$  [ 197] is the conjunction of the well-formedness of canal system identifiers,  $is\_wf\_CS\_Identities$  [ 259 on page 135], mereologies,  $is\_wf\_CS\_Mereology$  [ 301a on page 141], and attributes,  $is\_wf\_CS\_Attributes$  [ 363 on page 153].

#### type

197. CS

#### value

197.  $is\_wf\_CS: CS \rightarrow \mathbf{Bool}$

197.  $is\_wf\_CS(cs) \equiv is\_wf\_CS\_Identities(cs) \wedge is\_wf\_CS\_Mereologies(cs) \wedge is\_wf\_CS\_Attributes(cs)$

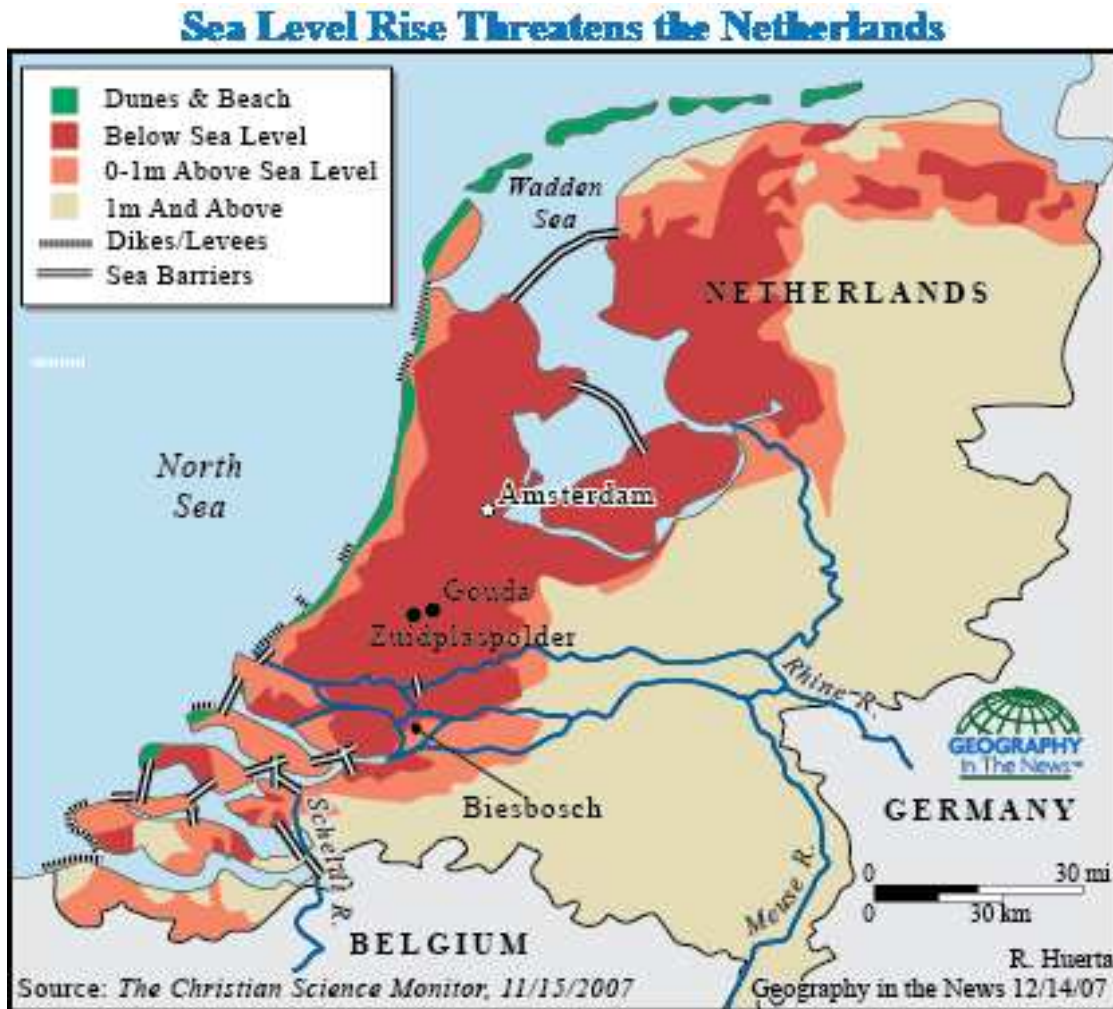


Figure C.10: The Land-Water Levels of The Netherlands

### C.3.1.6 No Structures

In this (long and detailed) example domain analysis & description I shall not use the pragmatic “device” of *structures* [cf, [58, Sect.4.10]]. Everything will be painstakingly analysed and described.

Some clarifying comments are in order:

- Compound endurants are either
  - Cartesian<sup>4</sup> or
  - sets.
- In analysing Cartesians, say  $c$ , into composite endurants, we analyse  $c$  into a number of components,  $c_i, c_j, \dots, c_k$ , of respective sorts,  $C_i, C_j, \dots, C_k$ , by means of observers  $\text{obs}_{C_i}, \text{obs}_{C_j}, \dots, \text{obs}_{C_k}$ .
  - The Cartesians,  $C$ , in this report, all have:
    - \* unique identifiers,

<sup>4</sup>Cartesian is spelled with a large ‘C’, after René Descartes, the French mathematician (1596–1650) [https://da.wikipedia.org/wiki/René\\_Descartes](https://da.wikipedia.org/wiki/René_Descartes).

- \* mereologies and
- \* attributes.
- So do each of the  $C_i, C_j, \dots, C_k$ .
- In analysing an endurant,  $E$ , into sets, say  $s$  or sort  $S$ , we first analyses  $E$  into a separately observable endurant  $Ss$ , i.e., **obs\_Ss**, which we then, at the same time define as  $Ss = \mathbf{S-set}$ .
- An  $Ss$  endurant thus has all the internal qualities:
  - a unique identifier,
  - a mereology
  - and attributes.

### C.3.1.7 Sequences of Presentation

The sequence in which endurant sorts are introduced is “repeated” in the sequences in which unique identifier sorts and mereology types are introduced. Thus the sequences of narrative and formal

- endurant items,
  - sort items, Items 198–212 on page 130,
  - value items, Items 219–238 on pages 131–133 and
  - “alternative” value items, Items  $\nu$ 219– $\nu$ 238 on pages 133–134,

are “repeated” in

- unique identifier
  - sort items, Items 239–257 on page 134,
  - value items, Items 259–274 on pages 135–136 and
  - “alternative” value items, Items  $\nu$ 212– $\nu$ 226 on pages 136–137,

and in

- mereology
  - type items, Items 283 to 299 on pages 137–141 and
  - well-formedness items, Items 301 to 318 on pages 141–147.

### C.3.1.8 Naming Conventions

Some care has been taken in order to name endurants, including sets of and predicates and functions over these; their unique identifiers and typed sets and values of and predicates and functions over these; their mereologies and typed sets and values of and predicates and functions over these; and their attributes and typed sets and values of and predicates and functions over these.

## C.3.2 External Qualities

### C.3.2.1 Endurant Sorts

The narrative(s) that follow serves two purposes:

- a formal purpose: the identification of endurants, and
- an informal purpose: in “casually familiarising” the reader as the the rôle of these endurants.

The former purpose is the only one to formalise. The latter purpose informally “herald” things to come – motivating, in a sense, theses “things”, the internal qualities and, if we had included a treatment of canal perdurants, the behaviours of these canal elements seen as behaviours.

All the elements mentioned below consist of both discrete endurants and fluids, i.e., water. In contrast to the treatment of such conjoins in [58, Sect. 4.13.3] we shall, in an informal digression from the principles, techniques and tool of the *analysis & description calculi* of [58, Chapters 4–5], omit “half the story”! It will be partly “restored” in our treatment of *canal attributes*, Sect. C.3.3.4.

In this section we shall narrate all the different endurant sorts, Items 198–218 (Pages 129–130), before we formalise them (Pages 130–130). We beg the readers forebearance in possibly having to thumb between narrative (page)s and formalisations (page)s.

198. **Canal systems**, CS, are given.

199. From a canal system one can observe a **canal net**, CN.

200. From a canal system one can observe a **polder aggregate**, PA.

Observing two endurants of a composite endurant is as if the composite is a Cartesian product of two. Hence the “(…,...)” of Fig. C.11 on page 131.

201. From canal nets one can observe **canal hub aggregates**, CA\_HA, and

202. **canal link aggregates**, CA\_LA.

203. From a polder aggregate one can observe a **polder set**, Ps, of polders, P. One observes the set, not its elements.

204. From a canal hub aggregate one can observe a **hub set**, CA\_Hs, of hubs, CA\_H. One observes the set, not its elements.

205. From a canal link aggregate one can observe a **canal link set**, CA\_Ls, of canal links, CA\_L. One observes the set, not its elements.

206. Polders are considered atomic. A **polder** is a low-lying tract of land that forms an artificial hydrological entity, enclosed by embankments known as dikes. The three types of polder<sup>5</sup> are:

- Land reclaimed from a body of water, such as a lake or the seabed.
- Flood plains separated from the sea or river by a dike.
- Marshes separated from the surrounding water by a dike and subsequently drained; these are also known as koogs.

207. Canal hubs are considered atomic and are:

208. either **canal begin/ends** (that is, where there is no continuation of a canal: where it ends “blind”, or where begins “suddenly”<sup>6</sup>), CA\_BE,

---

<sup>5</sup>The ground level in drained marshes subsides over time. All polders will eventually be below the surrounding water level some or all of the time. Water enters the low-lying polder through infiltration and water pressure of groundwater, or rainfall, or transport of water by rivers and canals. This usually means that the polder has an excess of water, which is pumped out or drained by opening sluices at low tide. Care must be taken not to set the internal water level too low. Polder land made up of peat (former marshland) will sink in relation to its previous level, because of peat decomposing when exposed to oxygen from the air.

Polders are at risk from flooding at all times, and care must be taken to protect the surrounding dikes. Dikes are typically built with locally available materials, and each material has its own risks: sand is prone to collapse owing to saturation by water; dry peat is lighter than water and potentially unable to retain water in very dry seasons. Some animals dig tunnels in the barrier, allowing water to infiltrate the structure; the muskrat is known for this activity and hunted in certain European countries because of it. Polders are most commonly, though not exclusively, found in river deltas, former fenlands, and coastal areas.

<sup>6</sup>A canal “end” is a canal channel which is “connected” only at one end to a canal channel.

209. or **canal confluences** (of three or more canals<sup>7</sup>), CA\_CO,
210. or **canal outlets**, CA\_OU (where canals join a *river*, or a *lake*, or an *ocean*). These sorts are all considered atomic.
211. **Canal links** are aggregates.
212. From canal links we choose to observe a set of **canal link elements**, CA\_LE<sup>8</sup>. (Canal links are such, through their mereology, see Sect. C.3.3.2, that they form two reversible sequences between connecting edges.)
213. Canal link elements are considered atomic and are
214. either **canal channels**, CA\_CH<sup>9</sup>,
215. or **canal locks**, CA\_LO<sup>10</sup>,
216. or **canal lock pumps**, CA\_LO\_PU<sup>11</sup>,
217. or **canal polder pumps**, CA\_PO\_PU<sup>12</sup>.
218. We do not further describe **canal outlets, rivers, lakes and oceans**.

|                                |                                             |
|--------------------------------|---------------------------------------------|
| <b>type</b>                    | 212. CA_LEs = CA_LE-set                     |
| 198. CS                        | 213. CA_LE == CA_CH CA_LO CA_LO_PU CA_PO_PU |
| 199. CN                        | 214. CA_CH :: ...                           |
| 200. PA                        | 215. CA_LO :: ...                           |
| 201. CA_HA                     | 217. CA_PO_PU :: ...                        |
| 202. CA_LA                     | <b>value</b>                                |
| 203. Ps = P-set, P             | 199. obs_CN: CS → CN                        |
| 204. CA_Hs = CA_H-set          | 200. obs_PA: CS → PA                        |
| 205. CA_Ls = CA_L-set          | 201. obs_CA_HA: CS → CA_HA                  |
| 206. P                         | 202. obs_CA_LA: CN → CA_LA                  |
| 207. CA_H == CA_BE CA_CO CA_OU | 203. obs_Ps: PA → Ps                        |
| 208. CA_BE :: ...              | 204. obs_CA_Hs: CA_HA → CA_Hs               |
| 209. CA_CO :: ...              | 205. obs_CA_Ls: CA_LA → CA_Ls               |
| 210. CA_OU :: ...              | 212. obs_CA_LEs: CA_L → CA_LEs              |
| 211. CA_L                      |                                             |

Figure C.11 on the facing page shows the taxonomy of a wide class of canal systems.

Figure C.12 on page 132 shows the schematisation of a specific canal system.

Figure C.13 on page 132 shows the individual endurants of a canal system for that shown in Fig. C.12 on page 132. Given what we have formalised so far, i.e., formula 198–205, this is really all we can “diagram”. The “part” list of Fig. C.13 on page 132 cannot show other than that there are these parts, but not how they are connected – that is first revealed when we ascribe mereologies – and that there are canal channels, not, for example, their length – that is first revealed when we ascribe attributes, such as length.

<sup>7</sup>Without loss of generality we model only confluences of three canals.

<sup>8</sup>We could have chosen other abstractions, for example, to observe a sequence of elements. More on this later.

<sup>9</sup>A canal channel offers a “straight”, un-interrupted “stretch” of water – like does a river channel.

<sup>10</sup>A canal lock) is always connected to two distinct canal link elements. Canal locks still act like a waterway, as does a canal channel.

<sup>11</sup>Canal lock pumps are like canal locks, but with pumps. A *canal lock pump* is connected to a canal lock and the two canal link elements connected by the lock. It takes water either from the lower lying canal link element and pumps it up into the lock chamber, or from the lock chamber and pumps it up to the higher level canal link element. Canal locks are without pumps. The canal link elements mentioned here are usually canal channels.

<sup>12</sup>A canal polder pump is a pump that takes water from a polder and deposits it in a canal which is at a higher level than the polder.



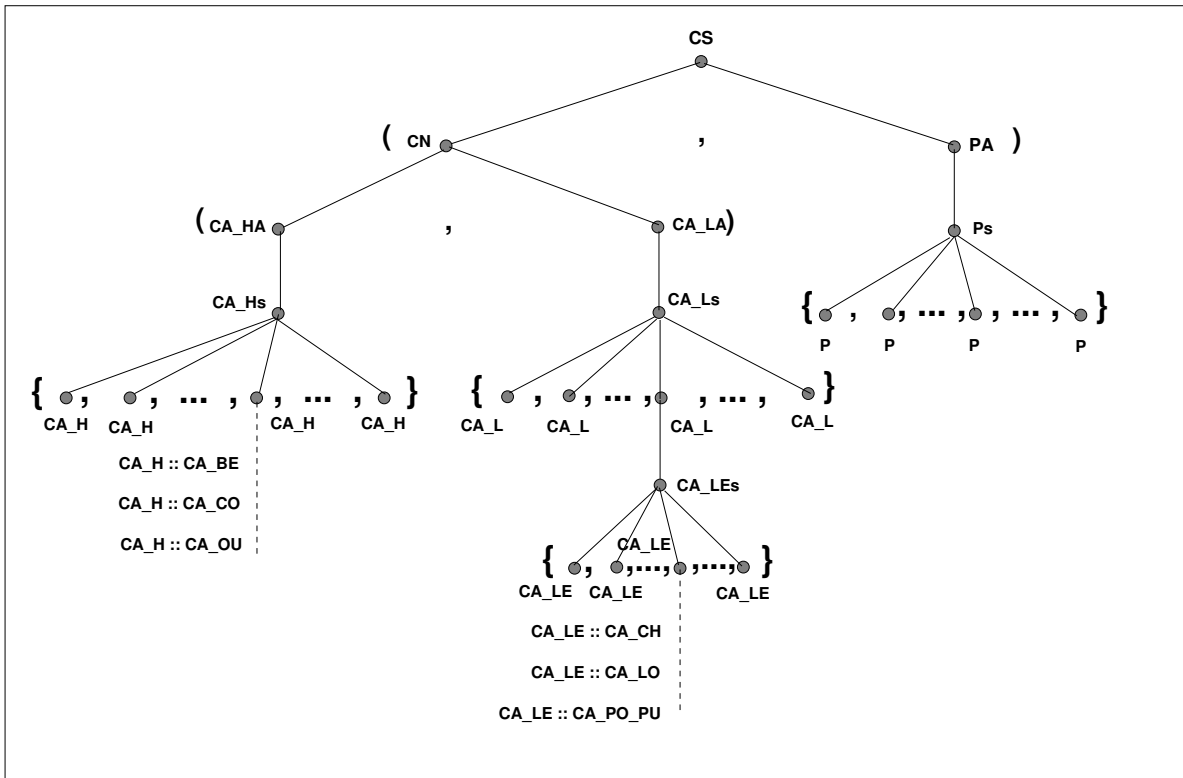


Figure C.11: Canal System Ontology

**C.3.2.2 Some Calculations**

We refer to Fig. C.14 on page 133. We shall list the enduring parts – and later on their unique identifiers – in the left-to-right order of a breadth-first traversal of the canal ontology.

- 219. Let  $cs$  be a “global” canal system.<sup>13</sup>
- 220. Canal nets and polders can be seen as consisting of the following enduring parts, modeled as a map,  $map_{ends}$ :
  - 221. the canal system,  $cs_{end}$ ,
  - 222. the canal net,  $cn_{end}$ ,
  - 223. the polder aggregate,  $pa_{end}$ ,
  - 224. the canal net hub aggregate,  $ca_{ha}_{end}$ ,
  - 225. the canal net link aggregate,  $ca_{la}_{end}$ ,
  - 226. the set of polders,  $ps_{end}$ ,
  - 227. the set of hubs,  $ca_{hs}_{end}$ ,
  - 228. the set of canal links,  $ca_{ls}_{end}$ ,
  - 229. the set of polders,  $pos_{end}$ ,

<sup>13</sup>Introducing  $cs$  allows us to refer to it and its “derivatives”, “all over”, and thus “universally prefix quantify” many axioms.

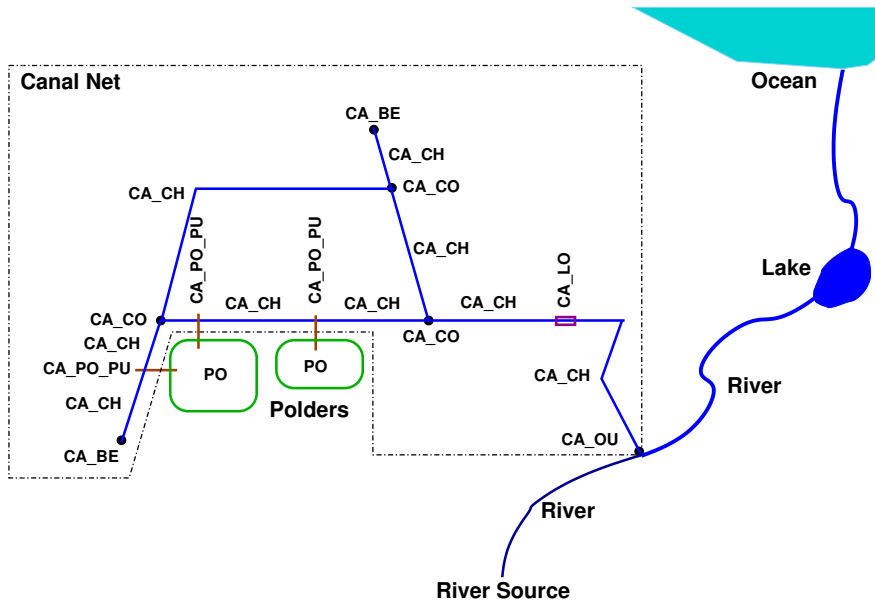


Figure C.12: A Schematised Specific Canal System: Canal Net + Polders

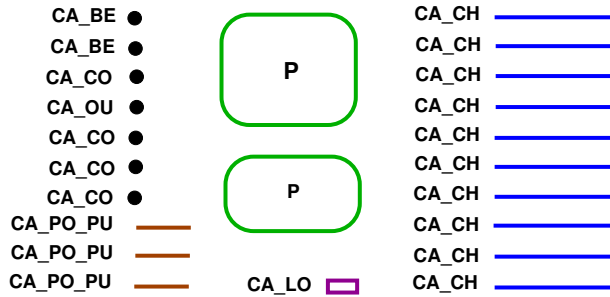


Figure C.13: Component Endurants of the Canal System of Fig. C.12

230. the set of hubs,  $ca\_hs_{end}$ , have the following kinds of hubs:

231. canal begin/ends,  $ca\_bes_{end}$ ,

232. canal confluences,  $ca\_cos_{end}$ , and

233. canal outlets,  $ca\_ous_{end}$ ,

234. the set of canal link elements,  $ca\_les_{end}$ ,

235. the canal link elements are of the following kinds:

236. canal channels,  $ca\_chs_{end}$ ,

237. canal locks,  $ca\_los_{end}$ ,

238. canal polder pumps,  $ca\_po\_pus_{end}$ .

**value**

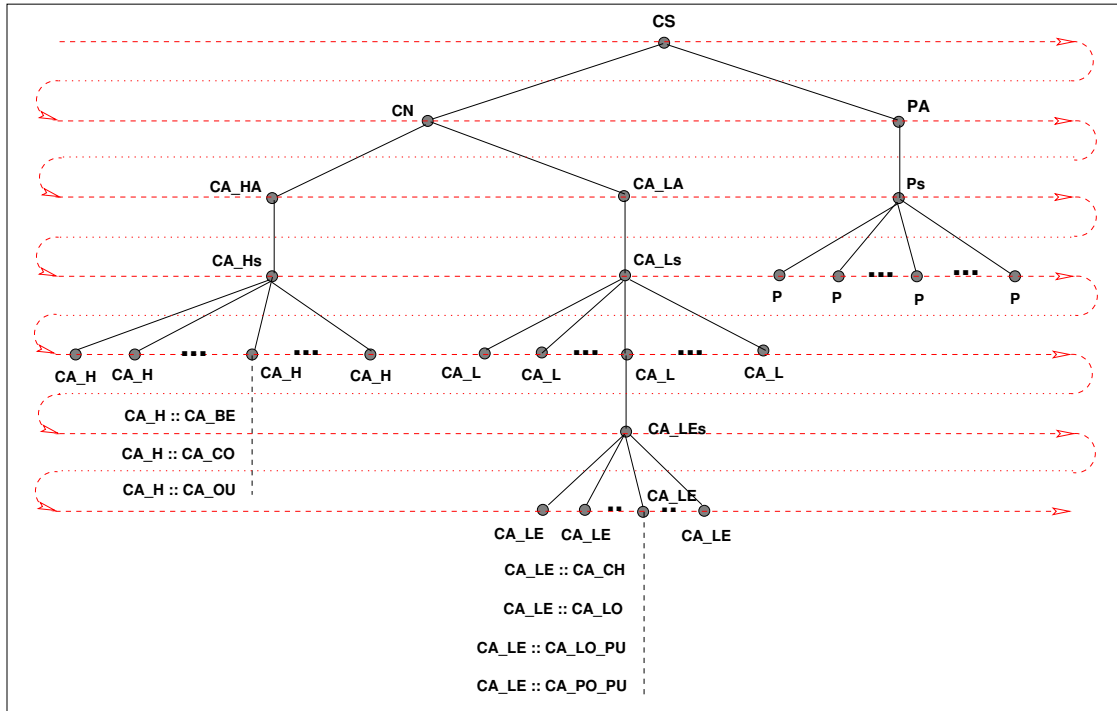


Figure C.14: A Breadth-first Left-to-Right [Top-down] Canal Ontology Traversal

```

219. cs:CS
220. map_ends: MAP_END14
220. map_ends = []
221. cs_end ↦ {cs},
222. cn_end ↦ {obs_CN(map_ends(cs_end))},
223. pa_end ↦ {obs_PA(map_ends(cs))},
224. ca_ha_end ↦ {obs_CA_HA(map_ends(cs))},
225. ca_la_end ↦ {obs_CA_LA(map_ends(cn_end))},
226. ps_end ↦ obs_Ps(map_ends(pa_end)),
227. ca_hs_end ↦ obs_CA_Hs(map_ends(ca_pa_end)),
228. ca_ls_end ↦ obs_CA_Ls(map_ends(ca_ps_end)),
229. pos_end ↦ map_ends(ca_ps_end),
231. ca_bes_end ↦ map_ends(ca_hs_end) \ CA_BE,
232. ca_cos_end ↦ map_ends(ca_hs_end) \ CA_CO,
233. ca_ous_end ↦ map_ends(ca_hs_end) \ CA_OU,
234. ca_les_end ↦ obs_CA_Ls(map_ends(ca_ps_end)),
236. ca_chs_end ↦ ∪ map_ends(ca_les_end) \ CA_CH,
237. ca_los_end ↦ ∪ map_ends(ca_les_end) \ CA_LO,
238. ca_po_pus_end ↦ ∪ map_ends(ca_les_ps_end) \ CA_PO_PU

```

We, in a name-overloading fashion, define – note the  $\nu$  prefix of the formula item numbers:

```

value
 ν 221. cs_end = cs,
 ν 222. cn_end = obs_CN(map_ends(cs_end)),
 ν 223. pa_end = obs_PA(map_ends(cs)),

```

<sup>14</sup>We invite the reader to formulate the MAP\_END type. As you can see from Items 209–220, it is a map from some sort of names to sets of durants.

|               |                   |                                                                           |
|---------------|-------------------|---------------------------------------------------------------------------|
| $\nu_{224}$ . | $ca\_ha_{end}$    | $= \text{obs\_CA\_HA}(\text{map\_ends}(cs)),$                             |
| $\nu_{225}$ . | $ca\_la_{end}$    | $= \text{obs\_CA\_LA}(\text{map\_ends}(cn_{end})),$                       |
| $\nu_{226}$ . | $ps_{end}$        | $= \text{obs\_Ps}(\text{map\_ends}(pa_{end})),$                           |
| $\nu_{227}$ . | $ca\_hs_{end}$    | $= \text{obs\_CA\_Hs}(\text{map\_ends}(ca\_pa_{end})),$                   |
| $\nu_{228}$ . | $ca\_ls_{end}$    | $= \text{obs\_CA\_Ls}(\text{map\_ends}(ca\_la_{end})),$                   |
| $\nu_{229}$ . | $pos_{end}$       | $= \text{map\_ends}(ca\_ps_{end}),$                                       |
| $\nu_{231}$ . | $ca\_bes_{end}$   | $= \text{map\_ends}(ca\_hs_{end}) \setminus \text{CA\_BE},$               |
| $\nu_{232}$ . | $ca\_cos_{end}$   | $= \text{map\_ends}(ca\_hs_{end}) \setminus \text{CA\_CO},$               |
| $\nu_{233}$ . | $ca\_ous_{end}$   | $= \text{map\_ends}(ca\_hs_{end}) \setminus \text{CA\_OU},$               |
| $\nu_{234}$ . | $ca\_cles_{end}$  | $= \text{obs\_CA\_LEs}(\text{map\_ends}(ca\_ls_{end})),$                  |
| $\nu_{236}$ . | $ca\_chs_{end}$   | $= \cup \text{map\_ends}(ca\_les_{end}) \setminus \text{CA\_CH},$         |
| $\nu_{237}$ . | $ca\_los_{end}$   | $= \cup \text{map\_ends}(ca\_les_{end}) \setminus \text{CA\_LO},$         |
| $\nu_{238}$ . | $ca\_popus_{end}$ | $= \cup \text{map\_ends}(ca\_les\_ps_{end}) \setminus \text{CA\_PO\_PU},$ |

### C.3.3 Internal Qualities

#### C.3.3.1 Unique Identifiers

##### C.3.3.1.1 Unique Identifier Sorts

|                                                         |                                                        |
|---------------------------------------------------------|--------------------------------------------------------|
| 239. Canal systems have unique identifiers .            | 249. canal begin/ends ,                                |
| 240. Canal nets have unique identifiers .               | 250. canal confluences and                             |
| 241. Polder aggregates have unique identifiers .        | 251. canal outlets .                                   |
| 242. Canal hub aggregates have unique identifiers .     | 252. Canal links have unique identifiers .             |
| 243. Canal link aggregates have unique identifiers .    | 253. Canal link element sets have unique identifiers . |
| 244. Polder sets (of polders) have unique identifiers . | 254. Canal link elements have unique identifiers:      |
| 245. Canal hub sets have unique identifiers .           | 255. canal channels ,                                  |
| 246. Canal link sets have unique identifiers .          | 256. canal locks and                                   |
| 247. Polders have unique identifiers.                   | 257. canal polder pumps .                              |
| 248. Canal hubs have unique identifiers:                |                                                        |
| <b>type</b>                                             | 256. CA_LO_UI                                          |
| 239. CS_UI                                              | 257. CA_PO_PU_UI                                       |
| 240. CN_UI                                              | <b>value</b>                                           |
| 241. PA_UI                                              | 239. uid_CS: CS $\rightarrow$ CS_UI                    |
| 242. CA_HA_UI                                           | 240. uid_CN: CN $\rightarrow$ CN_UI                    |
| 243. CA_LA_UI                                           | 241. uid_PA: PA $\rightarrow$ PA_UI                    |
| 244. Ps_UI                                              | 242. uid_CA_HA: CA_HA $\rightarrow$ CA_HA_UI           |
| 245. CA_Hs_UI                                           | 243. uid_CA_LA: CA_LA $\rightarrow$ CA_LA_UI           |
| 246. CA_Ls_UI                                           | 244. uid_Ps: Ps $\rightarrow$ Ps_UI                    |
| 247. P_UI                                               | 245. uid_CA_Hs: CA_Hs $\rightarrow$ CA_Hs_UI           |
| 248. CA_H_UI =                                          | 246. uid_CA_Ls: CA_Ls $\rightarrow$ CA_Ls_UI           |
| 248. CA_BE_UI CA_CO_UI CA_OU                            | 247. uid_P: P $\rightarrow$ P_UI                       |
| 249. CA_BE_UI                                           | 249. uid_CA_BE: CA_BE $\rightarrow$ CA_BE_UI           |
| 250. CA_CO_UI                                           | 250. uid_CA_CO: CA_CO $\rightarrow$ CA_CO_UI           |
| 251. CA_OU_UI                                           | 251. uid_CA_OU: CA_OU $\rightarrow$ CA_OU_UI           |
| 252. CA_L_UI                                            | 252. uid_CA_L: CA_L $\rightarrow$ CA_L_UI              |
| 253. CA_LEs_UI                                          | 253. uid_CA_LEs: CA_LEs $\rightarrow$ CA_LEs_UI        |
| 254. CA_LE_UI = CA_CH_UI                                | 255. uid_CA_CH: CA_CA_CH $\rightarrow$ CA_CH_UI        |
| 254.  CA_LO_UI CA_LO_PU_UI CA_PO_PU_UI                  | 256. uid_CA_LO: CA_CA_LO $\rightarrow$ CA_CA_LO_UI     |
| 255. CA_CH_UI                                           | 257. uid_CA_PO_PU: CA_LE $\rightarrow$ CA_PO_PU_UI     |

##### C.3.3.1.2 Some Calculations

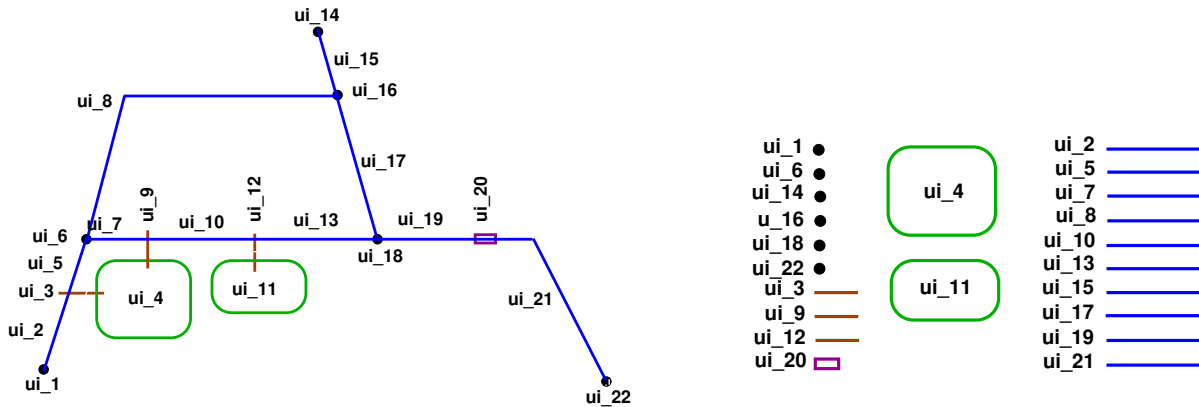


Figure C.15: Unique Identifiers of the Canal System of Figs. C.12 and C.13

258. We can calculate the following sets of unique identifiers, seen as a map from some kind of RSL names to sets of unique identifiers:

259. the canal system singleton set of its unique identifier, ,

260. the canal net singleton set of its unique identifier, ,

261. the polder aggregate singleton set of its unique identifier, ,

262. the canal hub aggregate singleton set of its unique identifier, ,

263. the canal link aggregate singleton set of its unique identifier, ,

264. the polder set singleton set of its unique identifier, ,

265. the hub set singleton set of its unique identifier, ,

266. the link set singleton set of its unique identifier, ,

267. the set of polder unique identifiers, ,

268. the set of canal begin/end unique identifiers, ,

269. the set of canal confluence unique identifiers, ,

270. the set of canal outlet unique identifiers, ,

271. the set of canal link set unique identifiers, ,

272. the set of canal link unique identifiers, ,

273. the set of canal channel unique identifiers, ,

274. the set of canal lock unique identifiers, ,

275. the set of canal lock pump unique identifiers, and

276. the set of canal polder pump unique identifiers, .

To define the next map we make use of the following generic function:

277. It applies to a set of endurants of sort X and yields the set of unique identifiers of the members of that set.

**type**277.  $\text{uid}_X: X\text{-set} \rightarrow X\text{-UI-set}$ **value**277.  $\text{uid}_X(xs) \equiv \{\text{uid}_X(x) \mid x: X \bullet x \in xs\}$ **value**258.  $\text{map\_uids}: \text{MAP\_UI}^{15}$ 258.  $\text{map\_uids} = [$ 

|                          |           |                                                             |
|--------------------------|-----------|-------------------------------------------------------------|
| 259. $cs_{uid}$          | $\mapsto$ | $\text{uid\_CS}(\text{map\_end}(cs_{end}))$ ,               |
| 260. $cn_{uid}$          | $\mapsto$ | $\text{uid\_CN}(\text{map\_ends}(cn_{end}))$ ,              |
| 261. $pa_{uid}$          | $\mapsto$ | $\text{uid\_PA}(\text{map\_ends}(pa_{end}))$ ,              |
| 262. $ca\_ha_{uid}$      | $\mapsto$ | $\text{uid\_CA\_HA}(\text{map\_ends}(ca\_ha_{end}))$ ,      |
| 263. $ca\_la_{uid}$      | $\mapsto$ | $\text{uid\_CA\_LA}(\text{map\_ends}(ca\_la_{end}))$ ,      |
| 264. $ps_{uid}$          | $\mapsto$ | $\text{uid\_P}(\text{map\_ends}(ps_{end}))$ ,               |
| 265. $ca\_hs_{uid}$      | $\mapsto$ | $\text{uid\_CA\_Hs}(\text{map\_ends}(ca\_hs_{end}))$ ,      |
| 266. $ca\_ls_{uid}$      | $\mapsto$ | $\text{uid\_CA\_Ls}(\text{map\_ends}(ca\_ls_{end}))$ ,      |
| 267. $pos_{uid}$         | $\mapsto$ | $\text{uid\_P}(\text{map\_ends}(ps_{end}))$ ,               |
| 268. $ca\_bes_{uid}$     | $\mapsto$ | $\text{uid\_BE}(\text{map\_ends}(ca\_bes_{end}))$ ,         |
| 269. $ca\_cos_{uid}$     | $\mapsto$ | $\text{uid\_CO}(\text{map\_ends}(ca\_cos_{end}))$ ,         |
| 270. $ca\_ous_{uid}$     | $\mapsto$ | $\text{uid\_OU}(\text{map\_ends}(ca\_ous_{end}))$ ,         |
| 271. $ca\_les_{uid}$     | $\mapsto$ | $\text{uid\_CA\_LEs}(\text{map\_ends}(ca\_ls_{end}))$ ,     |
| 273. $ca\_chs_{uid}$     | $\mapsto$ | $\text{uid\_CA\_CH}(\text{map\_ends}(ca\_chs_{end}))$ ,     |
| 274. $ca\_los_{uid}$     | $\mapsto$ | $\text{uid\_CA\_LO}(\text{map\_ends}(ca\_los_{end}))$ ,     |
| 276. $ca\_po\_pus_{uid}$ | $\mapsto$ | $\text{uid\_PO\_PU}(\text{map\_ends}(ca\_po\_pus_{end}))$ ] |

**C.3.3.1.3 An Axiom**278. Let  $end_{parts}$  stand for the set of all composite and atomic canal system endurants,279. and  $end_{uids}$  the set of all their unique identifiers.280. The number of endurants parts equals the number of endurant part unique identifiers,  $\text{is\_wf\_CS\_Identities}(cs)$ .**value**278.  $end_{parts} = \cup \text{rng proper\_map\_ends}$ 279.  $end_{uids} = \cup \text{rng map\_uids}$ **axiom**280.  $\text{is\_wf\_CS\_Identities}: \text{CS} \rightarrow \text{Bool}$ 280.  $\text{is\_wf\_CS\_Identities}(cs) \equiv \text{card } end_{parts} = \text{card } end_{uids}$ 

**C.3.3.1.4 Another Representation of UI Values** We, in a somewhat name-overloading fashion, similarly define:

**value**

|            |                |     |                                      |
|------------|----------------|-----|--------------------------------------|
| $\nu 259.$ | $cs_{uid}$     | $=$ | $\text{uid\_CS}(cs_{end})$ ,         |
| $\nu 260.$ | $cn_{uid}$     | $=$ | $\text{uid\_CN}(cn_{end})$ ,         |
| $\nu 261.$ | $pa_{uid}$     | $=$ | $\text{uid\_PA}(pa_{end})$ ,         |
| $\nu 262.$ | $ca\_ha_{uid}$ | $=$ | $\text{uid\_CA\_HA}(ca\_ha_{end})$ , |
| $\nu 263.$ | $ca\_la_{uid}$ | $=$ | $\text{uid\_CA\_LA}(ca\_la_{end})$ , |
| $\nu 264.$ | $ps_{uid}$     | $=$ | $\text{uid\_Ps}(ps_{end})$ ,         |

<sup>15</sup>We invite the reader to formulate the MAP\_UI type. As you can see from Items 209–220, it is a map from some sort of names to sets of unique identifiers.

$\nu 265.$   $ca\_hs_{uid}$  =  $uid\_Hs(ca\_pa_{end})$ ,  
 $\nu 266.$   $ca\_ls_{uid}$  =  $uid\_Ls(ca\_ps_{end})$ ,  
 $\nu 267.$   $pos_{uid}$  =  $uid\_P(ps_{end})$ ,  
 $\nu 268.$   $ca\_bes_{uid}$  =  $uid\_BE(ca\_bes_{end})$ ,  
 $\nu 269.$   $ca\_cos_{uid}$  =  $uid\_CO(ca\_cos_{end})$ ,  
 $\nu 270.$   $ca\_ous_{uid}$  =  $uid\_OU(ca\_ous_{end})$ ,  
 $\nu 271.$   $ca\_cles_{uid}$  =  $uid\_CA\_LEs(ca\_les_{end})$ ,  
 $\nu 272.$   $ca\_chs_{uid}$  =  $uid\_CA\_CH(ca\_chs_{end})$ ,  
 $\nu 273.$   $ca\_los_{uid}$  =  $uid\_CA\_LO(ca\_los_{end})$ ,  
 $\nu 275.$   $ca\_po\_pus_{uid}$  =  $uid\_PO\_PU(ca\_po\_pus_{end})$

### C.3.3.1.5 An Extract Function

281. Given 278.  $end_{parts}$  and 279.  $end_{uids}$ , we can, from any known unique identifier obtain its corresponding part:

**value**

281.  $get\_part: UI \rightarrow END$

281.  $get\_part(ui) \equiv \mathbf{let} \ p:P \cdot p \in end_{parts} \cdot uid\_P(p)=ui \mathbf{in} \ p \mathbf{end}; \mathbf{pre:} \ ui \in \mathbf{rng} \ end_{uids}$

### C.3.3.2 Mereologies

**C.3.3.2.1 Mereology Types** We shall focus only on the **topological mereologies** of canal system enduring. These can be “read off” the ontology tree of Fig. C.11 on page 131. Had we included the modeling of vessels that ply the waters of canals, then the mereologies of most canal enduring would also include sets of vessel identifiers.

As for the definitions of enduring, cf. Items 198 on page 129 to 217 on page 130, and the unique identifiers, cf. Items 239 on page 134 to 257 on page 134, we define the mereologies for each category of enduring. These mereologies are defined using the unique identifiers of the enduring immediately “above” and “below” them in the ontology “tree” of Fig. C.11 on page 131.

**Common Hub and Link Types:** From the unique identifier section we take over types defined in Items 241 and 242 on page 134

282. while introducing a set of their identifiers:

**type**

241.  $CA\_HE\_UI = CA\_BE\_UI|CA\_CO\_UI|CA\_OU\_UI$

242.  $CA\_LE\_UI = CA\_CH\_UI|CA\_LO\_UI|LO\_PU\_UI|PO\_PU\_UI$

282.  $CA\_LE\_UI\_H = (CL\_HE\_UI|CP\_LE\_UI)\text{-set}$

**Canal Systems:**

283. The mereology of a *canal system* is a pair of the unique identifiers of the canal net and of the polder aggregate.

**type**

283.  $CS\_Mer = CN\_UI \times PA\_UI$

**value**

283.  $mereo\_CS: CS \rightarrow mereo\_CS$

**Canal Nets:**

284. The mereology of a *canal net* aggregate is a pair of the unique identifier of the canal system, of which it is a part, and a pair of the set of the unique identifiers of the canal hub aggregate and the canal link aggregate of the net.

**type****value**

$$284. \text{CN\_Mer} = \text{CS\_UI} \times (\text{CA\_HA} \times \text{CA\_LA})$$

**value**

$$284. \text{mereo\_CN}: \text{CN} \rightarrow \text{CN\_Mer}$$

**Polder Aggregates:**

285. The mereology of a *polder* aggregate is a pair of the unique identifier of the canal system, of which it is a part, and the unique identifier of the polder set it “spawns”.

**type**

$$285. \text{PA\_Mer} = \text{CS\_UI} \times \text{Ps\_UI}$$

**value**

$$285. \text{mereo\_PA}: \text{PA} \rightarrow \text{PA\_Mer}$$

**Canal Hub Aggregates:**

286. The mereology of a *hub aggregate* is a pair of the unique identifier of the canal net it belongs to and the hub set it “spawns”.

**type**

$$286. \text{CA\_HA\_Mer} = \text{CN\_UI} \times \text{CA\_Hs}$$

**value**

$$286. \text{mereo\_CA\_HA}: \text{HA} \rightarrow \text{CA\_HA\_Mer}$$

**Canal Link Aggregates:**

287. The mereology of a *link aggregate* is a pair of the unique identifier of the canal net it belongs to and a set of the unique identifiers of the links that it “spawns”.

**type**

$$287. \text{CA\_LA\_Mer} = \text{CN\_UI} \times \text{CA\_Ls}$$

**value**

$$287. \text{mereo\_CA\_LA}: \text{LA} \rightarrow \text{CA\_LA\_Mer}$$

**Sets of Polders:**

288. The mereology of a *polder set* is a pair of the unique identifier of the polder aggregate it belongs to and a set of the unique identifiers of the polders that it “spawns”.

**type**

$$288. \text{Ps\_Mer} = \text{PA\_UI} \times \text{P\_UI-set}$$

**value**

$$288. \text{mereo\_Ps}: \text{Ps} \rightarrow \text{Pa\_Mer}$$

**Sets of Hubs:**

289. The mereology of a *hub set* is a pair of the unique identifier of the hub aggregate it belongs to and a set of the unique identifiers of the hubs that it “spawns”.



**type**

289.  $CA\_Hs\_Mer = CA\_HA\_UI \times CA\_H\_UI\_set$

**value**

289.  $mereo\_CA\_Hs: CA\_Hs \rightarrow CA\_Hs\_Mer$

**Sets of Links:**

290. The mereology of a *link set* is a pair of the unique identifier of the link aggregate it belongs to and a set of the unique identifiers of the links that it “spawns”.

**type**

290.  $CA\_Ls\_Mer = CS\_LA\_UI \times CA\_L\_UI\_set$

**value**

290.  $mereo\_CA\_Ls: Ls \rightarrow CA\_Ls\_Mer$

**Polders:**

291. The mereology of a *polder* is a pair of the unique identifier of the polder aggregate and a set of unique identifiers of *canal polder pumps*.

**type**

291.  $P\_Mer = Ps\_UI$

**value**

291.  $mereo\_P: P \rightarrow P\_Mer$

**Hubs:**

- Hubs are not individually “recognisable” as such. They are either begin/ends, confluences or outlets; cf. Item 207 on page 129.
- The mereologies of hubs thus “translates” into the mereology of either begin/ends, confluences or outlets.

– **Begin/End**

292. The mereology of a *canal begin/end* is a pair: the unique identifier of the canal hub set it belongs to and the singleton set of the unique identifier of the first canal link element for which it is the begin/end.

**type**

292.  $CA\_BE\_Mer = CA\_Hs\_UI \times s:CA\_LE\_UI\_set$  **axiom**  $\forall (\_ ,s):CA\_BE\_Mer \bullet card\ s=1$

**value**

292.  $mereo\_CA\_BE: CA\_BE \rightarrow CA\_BE\_Mer$

– **Confluence**

293. The mereology of a *canal confluence* is a pair: the unique identifier of the canal hub set it belongs and set of two or more canal element unique identifiers, one for each canal link incident upon the canal confluence.

**type**

293.  $CA\_CO\_Mer = CA\_Hs\_UI \times s:CL\_E\_UI\_set$  **axiom**  $\forall (\_ ,s):CA\_CO\_Mer \bullet card\ s \geq 2$

**value**

293.  $mereo\_CA\_CO: CA\_CO \rightarrow mereo\_CA\_CO$

– **Outlet**

294. The mereology of an *outlet* is a pair: the unique identifier of the canal hub set it belongs and the singleton set of the unique identifier of the last canal link element for which it is the outlet.

**type**

294.  $CA\_OU\_Mer = CA\_Hs\_UI \times s:CL\_E\_UI\text{-set}$  **axiom**  $\forall (\_,s):CA\_OU\_Mer \bullet \text{card } s=1$

**value**

294. mereo\_CA\_OU:  $CA\_OU \rightarrow CA\_OU\_Mer$

**Canal Links:**

295. The mereology of a *canal link* are triples: the unique identifier of the canal link set to which it belongs, a two element set of the canal hubs that the link is linking, and a list (i.e., an ordered sequence) of the unique identifiers of the one or more canal link elements of the link.

**type**

295.  $CA\_L\_Mer = CA\_Ls\_UI \times CA\_H\_UI\text{-set} \times s:CA\_LE\_UI^*$

295. **axiom**  $\forall (\_,s,l):CA\_L\_Mer \bullet \text{card } s=2 \wedge \text{len } l \geq 1$

**value**

295. mereo\_CA\_L:  $CA\_L \rightarrow CA\_L\_Mer$

**Sets of Canal Link Elements:**

296. The mereology of any *canal link element* includes a pair: the unique identifier of the canal link to which it belongs and a two element set, one element is the unique identifier of either a canal hub or a[another] canal link element, the second element is the unique identifier of either a [next] canal link element or a canal hub – these we call CLE\_UI\_P.

**type**

296.  $CA\_LE\_Mer\_Common = CL\_UI \times seuis:(CA\_H\_UI|CA\_LE\_UI)\text{-set}$

296. **axiom**  $\forall (clui,ch Luis):CA\_LE\_Mer \bullet \text{card } ch Luis = 2$

**Canal Link Elements:**

- Canal link elements are not individually “recognisable” as such. They are either canal channels, canal locks, canal locks with pumps or are canal polder pumps; cf. Item 213 on page 130.

– **Canal Channels**

297. The mereology of any *canal channel* is as the mereology included in any canal element mereology, cf. Item 296.

**type**

297.  $CA\_CH\_Mer = se:CA\_LE\_Mer\_Common$

**value**

297. mereo\_CA\_CH:  $CA\_CH \rightarrow CA\_CH\_Mer$

– **Canal Locks**

298. The mereology of any *canal lock* is as the mereology included in any canal element mereology, cf. Item 296.

**type**  
 298. CA\_LO\_Mer = se:CA\_LE\_Mer\_Common  
**value**  
 298. mereo\_CA\_LO: CA\_LO  $\rightarrow$  CA\_LO\_Mer

– **Canal Polder Pumps**

299. The mereology of any *canal polder pump*, is a pair: in addition to the mereology of any canal link element – which is now first element of the pair, has the second element being the unique identifier of a polder.

**type**  
 299. CA\_PO\_PU\_Mer = se:CA\_LE\_Mer\_Common  $\times$  P\_UI  
**value**  
 299. mereo\_CA\_PO\_PU: CA\_PO\_PU  $\rightarrow$  CA\_PO\_PU\_Mer

**C.3.3.2.2 The Mereology Axiom** It is You, the domain analysers & describers, who decide on the mereologies of a domain! You may wish to emphasize topological aspects of a domain; or you may wish to emphasize “co-ordination” relations between topologically “unrelatable” parts; or you may choose a mix of these; it all, also, depends on which aspects You wish to emphasize when transcendently deducing [certain] parts into behaviours. Therefore the **mereology axiom** to be expressed reflects Your choice. Here we have chosen to emphasize the topological aspects of the canal domain. We use the term *well-formedness* of the mereology of an endurant. But do not be misled! It is not a property that we impose on the domain endurant. It is a fact. We cannot escape from that fact. Later, in the requirements engineering of a possible software product for a domain, You may decide to implement data structures to reflect mereologies, in which case you shall undoubtedly need to prove that your choice of data structures, their initialisation and update does indeed satisfy the axioms of the domain model.

300. For a canal system to be mereologically, cum topologically well-formed means that the canal system mereology is well-formed.

**axiom**  
 300. is\_wf\_CS\_Mereology( $cs_{end}$ )

**C.3.3.2.3 Well-formed Mereologies Canal Systems:**

301. Canal system well-formednes, is\_wf\_CS\_Mereology,  
 (a) besides the appropriateness of its own mereology,  
 (b) is secured by the well-formedness of the canal net aggregate and polder aggregate, is\_wf\_CN\_Mereology and is\_wf\_PA\_Mereology.

**value**  
 301. is\_wf\_CS\_Mereology: CS  $\rightarrow$  **Bool**  
 301. is\_wf\_CS\_Mereology( $cs_{end}$ )  $\equiv$   
 301a. **let** (cn\_ui,pa\_ui) = mereo\_CS( $cs_{end}$ ) **in**  
 301a. cn\_ui =  $cn_{uid}$   $\wedge$  pa\_ui =  $pa_{uid}$  **end**  $\wedge$   
 301b. is\_wf\_CN\_Mereology( $cn_{end}$ )  $\wedge$  is\_wf\_PA\_Mereology( $pa_{end}$ )

**Canal Nets:**

302. Well-formedness of canal nets, is\_wf\_CN\_Mereology,

- (a) besides the appropriateness of its own mereology, `is_wf_CS_Mereology`,
- (b) is secured by the well-formedness of link and the hub aggregates, `is_wf_CA_LA_Mereology`, and all links, `is_wf_CA_HA_Mereology`.

**value**

302. `is_wf_CN_Mereology`: `CN`  $\rightarrow$  **Bool**

**axiom**

302. `is_wf_CN_Mereology`( $cn_{end}$ )  $\equiv$

302a. **let** (`cn_ui`,`ca_ha_ui`,`ca_la_ui`) = `mereo_CN`( $cn_{end}$ ) **in**

302a. `cn_ui` =  $cn_{uid}$   $\wedge$  `ca_ha_ui` =  $ca_{hauid}$   $\wedge$  `ca_la_ui` =  $ca_{lauid}$  **end**  $\wedge$

302b. `is_wf_CA_HA_Mereology`( $ca_{ha_{end}}$ )  $\wedge$  `is_wf_CA_LA_Mereology`( $ca_{la_{end}}$ )

**Polder Aggregates:**

303. Well-formedness of polder aggregates, `is_wf_PA_Mereology`,

- (a) besides the appropriateness of its own mereology,
- (b) is secured by the well-formedness of the polder set `is_wf_Ps_Mereology`.

**type****value**

303. `is_wf_PA_Mereology`: `PA`  $\rightarrow$  **Bool**

303. `is_wf_PA_Mereology`( $pa_{end}$ )  $\equiv$

303a. **let** (`cs_ui`,`ps_ui`) = `mereo_PA`( $pa_{end}$ ) **in**

303a. `cs_ui` =  $cs_{uid}$   $\wedge$  `ps_ui` =  $ps_{uid}$  **end**  $\wedge$

303b. `is_wf_Ps_Mereology`( $ps_{end}$ )

**Canal Hub Aggregates:**

304. Well-formedness of canal hub aggregates, `is_wf_CA_HA_Mereology`,

- (a) besides the appropriateness of its own mereology,
- (b) is secured by the well-formedness of its set of hubs.

**value**

304. `is_wf_CA_HA_Mereology`: `CA_HA`  $\rightarrow$  **Bool**

304. `is_wf_CA_HA_Mereology`(`hub`)  $\equiv$

304a. **let** (`cnui`,`cahsui`) = `mereo_CA_HA`(`hub`) **in**

304a. `cnui` =  $cn_{uid}$   $\wedge$  `cahsui` =  $ca_{hsuid}$  **end**  $\wedge$

304b. `is_wf_CA_Hs`( $ca_{hs_{end}}$ )

**Canal Link Aggregates:**

305. Well-formedness of canal link aggregates, `is_wf_CA_LA_Mereology`,

- (a) besides the appropriateness of its own mereology,
- (b) is secured by the well-formedness of its set of links.

**value**

305. `is_wf_CA_LA_Mereology`: `CA_LA`  $\rightarrow$  **Bool**

305. `is_wf_CA_LA_Mereology`(`la`)  $\equiv$

305a. **let** (`cnui`,`clsui`) = `mereo_CA_LA`(`la`) **in**

305a. `cnui` =  $cn_{uid}$   $\wedge$  `clsui` =  $cls_{uid}$  **end**  $\wedge$

305b. `is_wf_CA_Ls`( $cls_{end}$ )

**Sets of Polders:**

306. Well-formedness of sets of polders,  $\text{is\_wf\_Ps\_Mereology}$ ,
- (a) besides the appropriateness of its own mereology,
  - (b) is secured by the well-formedness of its individual polders.

**value**

306.  $\text{is\_wf\_Ps\_Mereology}: \text{Ps} \rightarrow \mathbf{Bool}$   
 306.  $\text{is\_wf\_Ps\_Mereology}(ps) \equiv$   
 306a. **let** (pau<sub>i</sub>, puis) = mereo\_Ps( $ps_{end}$ ) **in**  
 306a.  $\text{pau}_i = ca\_pa_{uid} \wedge \text{puis} = ca\_pos_{uid}$  **end**  $\wedge$   
 306b.  $\forall po:PO \bullet po \in pos_{end} \Rightarrow \text{is\_wf\_P}(po)$

**Sets of Hubs:**

307. Well-formedness of a hub set  $\text{is\_wf\_CA\_Hs\_Mereology}$ ,
- (a) besides the appropriateness of its own mereology,
  - (b) is secured by the well-formedness of its individual hubs.

**value**

307.  $\text{is\_wf\_CA\_Hs\_Mereology}: \text{CA\_Hs} \rightarrow \mathbf{Bool}$   
 307.  $\text{is\_wf\_CA\_Hs\_Mereology}(ca\_hs_{end}) \equiv$   
 307. **let** (caha<sub>i</sub>, cahuis) = mereo\_CA\_Hs( $ca\_bes_{end} \cup ca\_cos_{end} \cup ca\_ous_{end}$ ) **in**  
 307.  $\text{caha}_i = ca\_ha_{uid} \wedge \text{cahuis} = ca\_hs_{uid}$  **end**  $\wedge$   
 307a.  $\forall \text{hub}:CA\_H \bullet \text{hub} \in ca\_hs_{end} \Rightarrow \text{is\_wf\_CA\_H}(\text{hub})$

**Sets of Links:**

308. Well-formedness of sets of links
- (a) besides the appropriateness of its own mereology,
  - (b) is secured by the well-formedness of all of its individual links.

**value**

308.  $\text{is\_wf\_CA\_Ls\_Mereology}: \text{mereo\_CA\_Ls} \rightarrow \mathbf{Bool}$   
 308.  $\text{is\_wf\_CA\_Ls\_Mereology}(ca\_ls_{end}) \equiv$   
 308a. **let** (cs\_la<sub>ui</sub>, ca\_ls<sub>ui</sub>) = mereo\_CA\_Ls( $ca\_ls_{end}$ ) **in**  
 308a.  $\text{cs\_la}_{ui} = \wedge \text{ca\_ls}_{ui} = ca\_ls_{uid}$  **end**  $\wedge$   
 308b.  $\forall \text{link}:CA\_L \bullet \text{link} \in ca\_ls_{end} \Rightarrow \text{is\_wf\_CA\_L}(\text{link})$

**Polders:**

309. Well-formedness of polders,  $\text{is\_wf\_P\_Mereology}$ , depends jst on the appropriateness of its own mereology.

**value**

309.  $\text{is\_wf\_Mereology\_Polder}: \text{mereo\_P} \rightarrow \mathbf{Bool}$   
 309.  $\text{is\_wf\_Mereology\_Polder}(p) \equiv$   
 309. **let** ps<sub>ui</sub> = mereo\_P( $p$ ) **in**  
 309.  $\text{ps}_{ui} = ps_{uid}$  **end**  $\equiv$

**Hubs:**

- 207 Hubs are not individually “recognisable” as such. They are either begin/ends, confluences or outlets; cf. Item 207 on page 129.
310. The well-formedness of hubs thus “translates” into the well-formedness of either begin/ends, confluences or outlets.

**type**

207.  $CA\_H == CA\_BE|CA\_CO|CA\_OU$

**value**

310.  $is\_wf\_Mereology\_H: H \rightarrow \mathbf{Bool}$   
 310.  $is\_wf\_Mereology\_H(h) \equiv$   
 310.  $is\_CA\_BE(h) \rightarrow is\_wf\_Mereology\_CA\_BE(h),$   
 310.  $is\_CA\_CO(h) \rightarrow is\_wf\_Mereology\_CA\_CO(h),$   
 310.  $\_ \rightarrow is\_wf\_Mereology\_CA\_OU(h)$

- **Begin/End**

311. Well-formedness of the mereology of begin/end hubs,  $is\_wf\_Mereology\_CA\_BE$ , depends just on the appropriateness of their own mereology.

**value**

311.  $is\_wf\_Mereology\_CA\_BE: CA\_BE \rightarrow \mathbf{Bool}$   
 311.  $is\_wf\_Mereology\_CA\_BE(be) \equiv \equiv$   
 311. **let** (cahsui,cleuis) = mereo\_CA\_BE(be) **in**  
 311.  $cahsui \in ca\_hs\_uid \wedge cleuis \in ca\_les\_uid$  **end**

- **Confluence**

312. Well-formedness of the mereology of confluence hubs,  $is\_wf\_Mereology\_CA\_CO$ , depends just on the appropriateness of their own mereology.

**value**

312.  $is\_wf\_Mereology\_CA\_CO: CA\_CO \rightarrow \mathbf{Bool}$   
 312.  $is\_wf\_Mereology\_CA\_CO(co) \equiv$   
 312. **let** (cahsui,cleuis) = mereo\_CA\_CO(co) **in**  
 312.  $cahsui \in ca\_hs\_uid \wedge cleuis \in ca\_les\_uid$  **end**

- **Outlet**

313. Well-formedness of the mereology of outlet hubs,  $is\_wf\_Mereology\_CA\_OU$ , depends just on the appropriateness of their own mereology.

313.  $is\_wf\_Mereology\_CA\_OU: CA\_CO \rightarrow \mathbf{Bool}$   
 313.  $is\_wf\_Mereology\_CA\_OU(ou) \equiv$   
 313. **let** (cahsui,cleuis) = mereo\_CA\_OU(ou) **in**  
 313.  $cahsui \in ca\_hs\_uid \wedge cleuis \in ca\_les\_uid$  **end**

**Canal Links:**

314. The well-formedness of canal links depends on
- (a) the appropriateness of its own mereology, that is, that its unique identifier references are indeed to canal system identifiers,

- (b) the well-formedness of the set of link elements that can be observed from a canal link, that is, that they form a sequence of canal link elements – connecting two canal hubs, and
- (c) the (“remaining”) well-formedness of the canal link elements.

```

314. is_wf_Mereology_CA_L: CA_L → Bool
314. is_wf_Mereology_CA_L(link) ≡
314a. let (calsui,cahuis,caleuil) = mereo_CAL_L(link) in
314a. calsui = ∧ cahuis = ∧ caleuil = ∧
314b. wf_Link_Es(obs_CA_LEs(link))(cahuis)(caleuil) ∧
314c. ∀ le:CA_LE•le ∈ obs_CA_LEs(link) ⇒ is_wf_Mereology_CA_LE(le) end

```

#### Well-formed Sets of Canal Link Elements:

The introduction of the wf\_Link\_Es predicate represents a slight deviation from the introduction of the usual is\_wf\_Mereology predicates.

315. The wf\_Link\_Es predicate applies to a set of link elements, link, and a unique identifier list, uil, of unique link element identifiers. The predicate holds if the set, link: CA\_LE-**set**, of link elements not only can be ordered in the sequence indicated by uil.

- (a) The length of the unique identifier list, uil, must match the cardinality of the set link.
- (b) Let linkl be the list of link elements prescribed by uil.
  - i. The elements of a list “alternate” as follows:
    - A. canal locks have either canal hubs or canal channels as immediate neighbours<sup>16</sup>;
    - B. Canal locks and polder pumps cannot be adjacent.
    - C. It is allowed for two or more canal channels to be adjacent.
    - D. Thus canal links may have either canal channels or canal locks as first/last elements.
  - ii. Now there are the following cases of “neighbour” mereologies to observe:
  - iii. For a singleton list, linkl, its only element must connect the two distinct hubs identified in cahuis.
  - iv. for a two-element unique identifier list, ⟨luil,ruil⟩ one of their common mereology identifiers are shared, i.e., their elements are connected, and the other common mereology identifiers are those of canal hubs, i.e. end-points.
  - v. For lists of length three or more elements
    - A. the first and last elements must have end-points,
    - B. and for all elements in-between it must be the case that the neighbour identifiers
    - C. of the previous and the following link elements
    - D. must share identifiers with the quantified element
    - E. and share identifier with

#### value

```

315. wf_Link_Es: CA_LE-set → CA_H_UI-set × CA_LE_UI* → Bool
315. wf_Link_Es(link)(euis:{l_ca_h_ui,r_ca_h_ui})(uil) ≡ [axiom card euis = 2]
315a. card link = len uil ∧
315b. let linkl = ⟨ le | i:Nat, ce:C_LE•
315b. 1 ≤ i ≤ len uil ∧ le ∈ link ∧ uid_CA_LE(le) = uil[i] ⟩ in
315(b)i. is_wf_neighbours(linkl) ∧
315(b)ii. case linkl of

```

<sup>16</sup>That is, a sequence of locks, such as illustrated in Fig. C.4 on page 123, is here considered a single lock whose attributes “reveals” its “multiplicity”.

```

315(b)iii. ⟨ui⟩ →
315(b)iii. cahais = seuis(mereo_LE(get_part(ui))),
315(b)iii. axiom: let {lui, rui} = seuis(mereo_LE(get_part(ui))) in
315(b)iii. wf_end_points(lui, rui)(euis) end
315(b)iv. ⟨lui, rui⟩ →
315(b)iv. wf_end_points(lui, rui)(euis),
315(b)v. ⟨lui⟩^link^⟨rui⟩ → [axiom: len linkl ≥ 3, i.e., link ≠ ⟨⟩]
315(b)vA. wf_end_points(lui, rui)(euis) ∧
315(b)vB. ∀ i: Nat • 1 < i < len linkl ⇒
315(b)vC. let {uim1, uim1} = seuis(mereo_CA_LE(get_part(linkl[i-1])),
315(b)vC. {uip1, uip1} = seuis(mereo_CA_LE(get_part(linkl[i+1]))) in
315(b)vD. axiom: lui ∈ {uim1, uim1} ∧ rui ∈ {uip1, uip1}
315(b)vE. let uism1 = {uim1, uim1} \ {lui}, uisp1 = {uip1, uip1} \ {rui} in
315(b)vE. link[i] = uism1 = uisp1 end
315. end end end

315(b)i. is_wf_neighbours: CA_LE* → Bool
315(b)i. is_wf_neighbours(linkl) ≡
315(b)i. ∀ i: Nat • {i, i+1} ⊆ inds linkl ⇒
315(b)iA. is_CA_LO_UI(linkl[i]) ⇒ ~ (is_CA_LO_UI(linkl[i+1]) Vis_PO_PU_UI(linkl[i+1]))

315(b)iv. is_shared: UI × UI-set × UI-set → Bool
315(b)iv. is_shared(ui, luis, ruis) ≡ ui ∈ luis ∩ ruis
315(b)iv.
315(b)iv. shared: UI-set × UI-set → UI
315(b)iv. shared(luis, ruis) ≡ luis ∩ ruis
315(b)iv. pre: ∃ ui: UI • is_shared(ui, luis, ruis)
315(b)iv.
315(b)iv. wf_end_points: (UI × UI) → CA_H_UI-set → Bool
315(b)iv. wf_end_points(lui, rui)(euis) ≡ [axiom: card euis = 2]
315(b)iv. let {llui, lrui} = seuis(mereo_LE(get_part(lui))),
315(b)iv. {rlui, rrui} = seuis(mereo_LE(get_part(rui))) in
315(b)iv. if ∃ ui: CA_LE_UI • is_shared(ui, {llui, lrui}, {rlui, rrui})
315(b)iv. then let ui = shared({llui, lrui}, {rlui, rrui}) in
315(b)iv. {llui, lrui, rlui, rrui} \ {ui} = euis ∧
315(b)iv. euis ⊆ ca_bes_uid ∪ ca_cus_uid ∪ ca_ous_uid end
315(b)iv. else false end end,

```

### Canal Link Elements:

...

MORE TO COME

### • Canal Channels

316. is\_wf\_CA\_CH\_Mereology,

- (a)
- (b)
- (c)

316.

316.



- 316a.
- 316b.
- 316c.

- **Canal Locks**

- 317.  $is\_wf\_CA\_LO\_Mereology,$ 
  - (a)
  - (b)
  - (c)

- 317.
- 317.
- 317a.
- 317b.
- 317c.

- **Canal Polder Pumps**

- 318. (a)
- (b)
- (c)

- 318.
- 318.
- 318a.
- 318b.
- 318c.

### C.3.3.3 Routes

#### C.3.3.3.1 Preliminaries

- 319. By an end-identifier we mean the unique identifier of a begin/end or an outlet.
- 320. By a middle-identifier we mean the unique identifier of a confluence, channel, lock, lock with pump or a polder pump.
- 321. By a unit identifier we mean either an end-identifier or a middle-identifier.
- 322. By a canal route we mean a sequence of one or more unique identifiers of atomic canal entities, two if one of the identifiers is that of a begin/end or an outlet unit.

Notice that adjacent canal route identifiers be distinct. But a triplet of adjacent canal route identifiers may have the same first and last elements. It is allowed that a route, so-to-speak, goes forward and backward. There is, in a sense, no preferred directions in canal systems.

#### type

- 319.  $E\_UI = CA\_BE\_UI|CA\_OU\_UI$
- 320.  $M\_UI = CA\_CO\_UI|CA\_CH\_UI|CA\_LO\_UI|CA\_LO\_PU\_UI|CA\_PO\_PU\_UI$
- 321.  $UI = E\_UI|M\_UI$
- 322.  $CR = UI^*$

#### axiom

- 322.  $\forall cr:CR, i: \mathbf{Nat} \bullet \{i, i+1\} \subseteq \mathbf{inds} \ cr \Rightarrow cr[i] \neq cr[i+1]$

323. Let `uid_MU` be a “common” unique identifier observer of middle units.
324. Let `mereo_MU` be a “common” mereology observer of middle units other than polder pumps.
325. From middle units, i.e., confluences, channels, locks, lock with pumps and polder pumps we can extract simple, one-, two- or three element canal routes.

**type**

323. `uid_MU = uid_CA_CO|uid_CA_CH|uid_CA_LO|uid_CA_LO_PU|uid_CA_PO_PU`

324. `mereo_MU = mereo_CA_CO|mereo_CA_CH|mereo_CA_LO|uid_CA_LO_PU`

325. `MU = CA_CO|CA_CH|CA_LO|CA_LO_PU|CA_PO_PU`

**value**

325. `xtr_M_UI_CRs: MU → CR-set`

325. `xtr_M_UI_CRs(mu) ≡`

325.     **let** `mu_ui = uid_MU(mu),`

325.         `{ui1,ui2} =`

325.             `is_CA_PO_PU(mu) →`

325.                 **let** `(_,cuis,_) = mereo_CA_PO_PU(mu) in cuis end`

325.             `_ → let (__,cuis) = mereo_MU(mu) in cuis end`

325.     `{⟨mu_ui⟩,⟨ui1,mu_ui⟩,⟨ui2,mu_ui⟩,⟨mu_ui,ui1⟩,⟨mu_ui,ui2⟩,⟨ui1,mu_ui,ui2⟩,⟨ui2,mu_ui,ui1⟩}`

325.     **end**

**C.3.3.3.2 All Routes**

326. By means of `xtr_M_UI_CRs` we can extract, `xtr_CRs(mus)`, the infinite set of canal routes from any set, `mus`, of middle canal elements.
327. First we calculate initial, i.e., simple routes.
328. Then for every two routes, a “left” and a “right” route, in the set of routes being recursively defined, such that the last element of the left route is identical to the first element of the right route, the route formed by concatenating the left and right routes “around” the shared element is a route.
329. The set of routes of a canal system is the least fix-point solution the the equation of Item 328.
330. No two adjacent identifiers are the same.

**type**

325. `MU = CA_CO|CA_CH|CA_LO|CA_LO_PU|CA_PO_PU`

**value**

326. `xtr_CRs: MU-set → CR-infset`

326. `xtr_CRs(mus) ≡`

327.     **let** `icrs = ∪{xtr_M_UI_CRs(mu)|mu:MU•mu ∈ mus} in`

328.     **let** `crs = icrs ∪ {lr^⟨ui⟩^r||r,⟨ui⟩,rr:CR•{lr^⟨ui⟩,⟨ui⟩^rr}∈icrs} in`

329.     `crs`

330.     **axiom:**  $\forall cr:CR, i:\mathbf{Nat} \bullet cr \text{ isn } crs \wedge \{i,i+1\} \subseteq \mathbf{inds } cr \Rightarrow cr[i] \neq cr[i+1]$

326.     **end end**

**C.3.3.3.3 Connected Canal Systems**

331. Canal systems, such as we shall understand them, are connected.
332. That is, there is a route from any canal element to any other other canal element.
333. Let `mus` be the set of all middle elements of a canal system.

334. Let  $rs$  be the infinite set of all routes of  $mus$ .

335. Now, for any two unique identifiers of middle elements there must be a route in  $rs$ .

**value**

331.  $is\_connected\_CS: CS \rightarrow \mathbf{Bool}$

332.  $is\_connected\_CS(cs) \equiv \mathbf{in}$

333. **let**  $mus = ca\_cos_{end} \cup ca\_chs_{end} \cup ca\_los_{end} \cup ca\_lo\_pus_{end} \cup ca\_po\_pus_{end}$  **in**

334. **let**  $rs = xtr\_CRs(mus)$  **in**

335.  $\forall ui: M\_UI \bullet \{fui, tui\} \subseteq uid\_MU(mus) \Rightarrow \exists r: R \bullet r \in rs \text{ and } r[1] = r[\mathbf{len} \ r]$

332. **end end**

#### C.3.3.3.4 A Canal System Axiom

336. Canal systems are connected.

**axiom**

336.  $\forall cs: CS \bullet is\_connected\_CS(cs)$

#### C.3.3.4 Attributes

We shall treat the issue of canal part attributes, not, as is usual, one-by-one, sort-by-sort, but more-or-less “collectively”, across canal hubs and links. And we do so category-by-category of attribute kinds: spatial, temporal and other.

##### C.3.3.4.1 Spatial and Temporal Attributes Spatial Attributes:

Natural and artefactual, that is, man-made durants reside in space. We have dealt with space, i.e.,  $SPACE$ , in [58, Sects. 2.2 and 3.4]. Subsidiary spatial concepts are those of  $VOLUME$ ,  $AREA$ ,  $CURVE$  (or  $LINE$ ), and  $POINT$ . All canal system durants possess, whether we choose to model them or not, such spatial attributes. We shall not here be bothered by any representation, let alone computational representations, of spatial attributes. They are facts. Any properties that two  $AREAS$ ,  $a_i$  and  $a_j$  may have in common – like **bordering**, **overlapping disjoint** or **properly contained** – are facts and should, as such be expressed in terms of **axioms**. They are not properties that can, hence must, be proven. Once a *domain description*, involving spatial concepts is the base for a *requirements prescription*, then, if these spatial concepts are not *projected* out of the evolving requirements, they must, eventually, be prescribed – or assumed to have – computable representations. In that case axioms concerning spatial quantities are turned into **proof obligations** that must, eventually, be **discharged**.

Let us establish the following spatial attributes, common to all canal parts:

337. **Location:** A single  $POINT$  in  $SPACE$  characterised by its longitude, latitude and altitude, the latter height above or depth below sea level, including 0. How these are measured is of no concern in this model.

338. **Extent:** An  $AREA$ , i.e., a plane in  $SPACE$ , i.e., a dense set of  $POINTS$  according to some topology.

339. **Volume:** A proper subset  $SPACE$ , i.e., a three dimensional dense set of  $POINTS$   $SPACE$ , according to some topology.

340. The Location of a canal part is always **embedded** in its Extent.

341. The Extent of a canal part is always **embedded** in its Volume

**type**

- 337. Location
- 338. Extent
- 339. Volume

**value**

- 337. attr\_Location: CS|CN|PA|CA\_HA|CA\_LA|CA\_LE  $\rightarrow$  Location
- 338. attr\_Extent: CS|CN|PA|CA\_HA|CA\_LA|CA\_LE  $\rightarrow$  Extent
- 339. attr\_Volume: CS|CN|PA|CA\_HA|CA\_LA|CA\_LE  $\rightarrow$  Volume
- 339. **is\_embedded**: Location  $\times$  Extent  $\rightarrow$  **Bool**, **is\_embedded**: Extent  $\times$  Volume  $\rightarrow$  **Bool**

**axiom**

- 340.  $\forall e:(CS|CN|PA|CA_HA|CA_LA|CA_LE)\bullet\mathbf{is\_embedded}(\mathit{attr\_Location}(e),\mathit{attr\_Extent}(e))$
- 341.  $\forall e:(CS|CN|PA|CA_HA|CA_LA|CA_LE)\bullet\mathbf{is\_embedded}(\mathit{attr\_Extent}(e),\mathit{attr\_Volume}(e))$

Let us establish the following **\*\*\***s, common to some canal parts:

- 342. Let us addume the sort notions of Latitude, Longitude and Altitude,
- 343. And let us assume “sea level” Altitude value "0".
- 344. A projected extent is an extent all of whose **altitude** elements are zero (0), i.e., “at sea level”.
- 345. We assume functions, **latitude**, **logitude**, **altitude**, that extract respective elements of a point.
- 346. No two distinct hubs and link elements can share neither location, area nor volume – so they are **disjoint**.

Canal channels may share projected extents.

**type**

- 342. Latitude, Longitude, Altitude

**value**

- 343. "0": Altitude
- 344. projected\_Extent: Extent  $\rightarrow$  Extent
- 345. latitude: POINT  $\rightarrow$  Latitude, longitude: POINT  $\rightarrow$  Longitude, altitude: POINT  $\rightarrow$  Altitude

**axiom**

- 346.  $\forall e,e':(CS|CN|PA|CA_HA|CA_LA|CA_LE): e\neq e' \Rightarrow \mathbf{disjoint}(\mathit{attr\_Volume}(e),\mathit{attr\_Volume}(e'))$
- 342.  $\forall e,e':CA\_CH \bullet$
- 343.
- 344.

**Temporal Attributes:**

Natural and artefactual, that is, mane-made endurants reside in time. We have dealt with space, i.e., **TIME**, in [58, *Sects. 2.2 and 3.5*]. Subsidiary spatial concepts are those of **TIME** and **TIME INTERVALS**. All canal system endurants possess, whether we choose to model them or not, such temporal attributes. We shall not here be bothered by any representation, let alone computational representations, of temporal attributes. They are facts. Any properties that two **TIME INTERVALS**,  $ti_i$  and  $ti_j$  may have in common, like **bordering** or **overlapping**, are facts and should, as such be expressed in terms of **axioms**<sup>17</sup>. They are not properties that can, hence must, be proven. Once a *domain description*, involving temporal concepts is the base for a *requirements prescription*, then, if these temporal concepts are not *projected* out of the evolving requirements, they must, eventually, be prescribed – or assumed to have – computable representations. In that case axioms concerning temporal quantities are turned into **proof obligations** that must, eventually, be **discharged**.

**Event Attributes**

<sup>17</sup>We refer here to the **TIME** and **TIME INTERVAL** operators of [58, *Sects. 2.2 and 3.5*]

Some events can, for example, be talked about, by humans. They, so-to-speak, belong to an event-category: “*von hörensagen*”. Examples are: “*a canal lock opened at time  $\tau$* ”; “*a polder pump stopped pumping at time  $\tau'$* ”; and “*a canal vessel passed a certain canal channel point at time  $\tau''$* ”. Let refer to the event as  $e:E$ . If, for an endurant,  $p$  of sort  $P$ , they are relevant to an analysis & description of a domain, then they must be noted, for example in the form of an attribute named, say, `history_E`:

**type** `history_E` = `TIME`  $\multimap$  `E`

### Continuous Time Attributes

Mostly one models discrete time phenomena. But often phenomena are continuous time varying. Examples are: “*the canal water level*”, “*the canal water temperature*”, and “*the position of a vessel along a canal*”. If, for an endurant,  $p$  of sort  $P$ , such a phenomenon,  $e:E$ , is relevant to an analysis & description of a domain, then it must be noted, for example in the form of an attribute named, say, `history_E`:

**type** `history_E` = `TIME`  $\rightarrow$  `E`

#### C.3.3.4.2 Canal System, Net and Polder Attributes

347. Canal systems have location and extent.

348. So do canal nets and

349. polder aggregates.

350. Canal nets and polder aggregates are **bordering**<sup>18</sup>.

351. Canal nets and polder aggregates are properly **embedded**<sup>19</sup> in canal systems.

352. Etc.

#### value

347. `attr_Location`: `CS`  $\rightarrow$  `Location`; `attr_Extent`: `CS`  $\rightarrow$  `Extent`

348. `attr_Location`: `CN`  $\rightarrow$  `Location`; `attr_Extent`: `CN`  $\rightarrow$  `Extent`

349. `attr_Location`: `PA`  $\rightarrow$  `Location`; `attr_Extent`: `PA`  $\rightarrow$  `Extent`

#### axiom

350.  $\forall cs:CS \bullet \mathbf{are\_bordering}(\mathbf{attr\_Extent}(\mathbf{obs\_CN}(cs)), \mathbf{attr\_Extent}(\mathbf{obs\_PA}(cs)))$

351.  $\forall cs:CS \bullet \mathbf{is\_embedded}(\mathbf{attr\_Extent}(\mathbf{obs\_CN}(cs)), cs) \wedge \mathbf{is\_embedded}(cs, \mathbf{attr\_Extent}(\mathbf{obs\_PA}(cs)))$

352. ...

**C.3.3.4.3 Canal Hub and Link Attributes** Two kinds of attributes shared across hubs and links, therefore their elements, stand out: *water levels* and *ambient* and *water temperatures*.

#### Water Temperatures:

353. Let there be given a notion of water temperature.

Generally, over time, one can associate with any canal hub and link element,

354. high,

355. normal and

<sup>18</sup>We leave it to a chosen `Topology` to define the **are\_bordering** predicate

<sup>19</sup>We leave it to a chosen `Topology` to define the **is\_embedded** predicate

356. low water

water temperatures, and specifically, at any time,

357. current water temperatures.

**type**

353.  $Wa\_Temp$

354.  $Hi\_Temp = TIME \rightarrow Wa\_Temp$

355.  $No\_Temp = TIME \rightarrow Wa\_Temp$

356.  $Lo\_Temp = TIME \rightarrow Wa\_Temp$

357.  $Cu\_Temp = Wa\_Temp$

**value**

354.  $attr\_Hi\_Temp: H \rightarrow Hi\_Temp$ ,  $attr\_LE\_Temp: LE \rightarrow Hi\_Temp$

355.  $attr\_No\_Temp: H \rightarrow No\_Temp$ ,  $attr\_LE\_Temp: LE \rightarrow No\_Temp$

356.  $attr\_Lo\_Temp: H \rightarrow Lo\_Temp$ ,  $attr\_LE\_Temp: LE \rightarrow Lo\_Temp$

357.  $attr\_Cu\_Temp: H \rightarrow Cu\_Temp$ ,  $attr\_LE\_Temp: LE \rightarrow Cu\_Temp$

The  $Hi\_Temp$ ,  $No\_Temp$  and  $Lo\_Temp$  attributes are normally continuous functions over time. They are facts. One does not have to “go out” and measure them! We do not have to think of representations for the  $Hi\_Temp$ ,  $No\_Temp$  and  $Lo\_Temp$  attributes.

**Water Levels:**

358. Let there be given a notion of water level.

Generally, over time, one can associate with any canal hub and link element,

359. high,

360. normal and

361. low

water levels, and specifically, at any time,

362. current water level.

**type**

358.  $Wa\_Lev$

359.  $Hi\_WL = TIME \rightarrow Wa\_Lev$

360.  $No\_WL = TIME \rightarrow Wa\_Lev$

361.  $Lo\_WL = TIME \rightarrow Wa\_Lev$

362.  $Cu\_WL = Wa\_Lev$

**value**

359.  $attr\_Hi\_WL: H \rightarrow Hi\_WL$ ,  $attr\_LE\_WL: LE \rightarrow Hi\_WL$

360.  $attr\_No\_WL: H \rightarrow No\_WL$ ,  $attr\_LE\_WL: LE \rightarrow Hi\_WL$

361.  $attr\_Lo\_WL: H \rightarrow Lo\_WL$ ,  $attr\_LE\_WL: LE \rightarrow Hi\_WL$

362.  $attr\_Cu\_WL: H \rightarrow Cu\_WL$ ,  $attr\_LE\_WL: LE \rightarrow Hi\_WL$

The  $Hi\_WL$ ,  $No\_WL$  and  $Lo\_WL$  attributes are normally continuous functions<sup>20</sup> over time. Remarks on  $Hi\_WL$ ,  $No\_WL$  and  $Lo\_WL$  attributes similar to those of the  $Hi\_Temp$ ,  $No\_Temp$  and  $Lo\_Temp$  attributes as to continuity and representations apply.

MORE TO COME

<sup>20</sup>– barring cyclones, tornados and the like!

**C.3.3.5 Well-formedness of Attributes**

363. There is a predicate, `is_wf_CS_Attributes`.

364.

365.

366.

367.

MORE TO COME

**C.3.4 Speculations**

TO BE WRITTEN

**C.4 Conclusion**

TO BE WRITTEN





# Appendix D

## The 7 Seas

### Contents

---

|            |                                      |            |
|------------|--------------------------------------|------------|
| <b>D.1</b> | <b>Introduction</b>                  | <b>156</b> |
| <b>D.2</b> | <b>Endurants</b>                     | <b>156</b> |
| D.2.1      | <b>External Qualities</b>            | 156        |
| D.2.1.1    | <b>Informal Introduction</b>         | 156        |
| D.2.1.2    | <b>Formal Introduction</b>           | 156        |
| D.2.1.2.1  | <b>Parts and Fluids</b>              | 157        |
| D.2.1.2.2  | <b>The 7 Seas State</b>              | 158        |
| D.2.2      | <b>Internal Qualities</b>            | 159        |
| D.2.2.1    | <b>Unique Identifiers</b>            | 159        |
| D.2.2.1.1  | <b>Observers</b>                     | 159        |
| D.2.2.1.2  | <b>All Unique Identifiers</b>        | 159        |
| D.2.2.1.3  | <b>Axiom</b>                         | 160        |
| D.2.2.1.4  | <b>Extraction of Atomic Elements</b> | 160        |
| D.2.2.2    | <b>Mereology</b>                     | 161        |
| D.2.2.2.1  | <b>Types, Observers and Axioms</b>   | 161        |
| D.2.2.2.2  | <b>A Remark</b>                      | 163        |
| D.2.2.2.3  | <b>A Domain Axiom</b>                | 163        |
| D.2.2.3    | <b>Attributes</b>                    | 163        |
| D.2.2.3.1  | <b>Seas</b>                          | 164        |
| D.2.2.3.2  | <b>Rivers</b>                        | 164        |
| D.2.2.3.3  | <b>Canals and Straits</b>            | 165        |
| D.2.2.3.4  | <b>Continents</b>                    | 165        |
| D.2.2.3.5  | <b>Harbours</b>                      | 166        |
| D.2.2.3.6  | <b>Vessels</b>                       | 166        |
| <b>D.3</b> | <b>Perdurants</b>                    | <b>167</b> |
| D.3.1      | <b>Channels</b>                      | 167        |
| D.3.2      | <b>Behaviours</b>                    | 167        |
| D.3.2.1    | <b>Signatures</b>                    | 167        |
| D.3.2.2    | <b>Definitions</b>                   | 167        |
| D.3.2.3    | <b>System</b>                        | 167        |
| <b>D.4</b> | <b>Conclusion</b>                    | <b>167</b> |

---

## D.1 Introduction

In this model we shall treat waterways, not as fluids, but as solids! That is, we may consider waterways as parts, and hence, by transcendental deductions, as possibly having behaviours. Similarly we shall consider many composite endurants, not as elements of structures, but as parts, while not considering their internal qualities, that is, not considering their possible behaviours.

## D.2 Endurants

### D.2.1 External Qualities

#### D.2.1.1 Informal Introduction

Waterways include seas, rivers and navigable “k”anals. One can take the view that there are the following eight seas: the *Arctic Ocean*, the *North Atlantic Ocean*, the *South Atlantic Ocean*, the *Indian Ocean*, the *North Pacific Ocean*, the *South Pacific Ocean*, the *Southern* (or *Antarctic*) *Ocean*, and the *Kaspian Sea*. Another view “collapses” the north and south into one, leaving just 6 oceans and seas. Yet a third view is that there are just 2 oceans and seas: The *Kaspian Sea* and the others – since they are all “tightly” connected! The *Kaspian Sea* cannot be reached by ship or boat from the ocean[s]! *The Mediterranean* and *The Black Seas* are both considered segments of *The Atlantic Ocean*. *The Arab Sea* is considered a segment of *The Indian Ocean*. Etcetera.

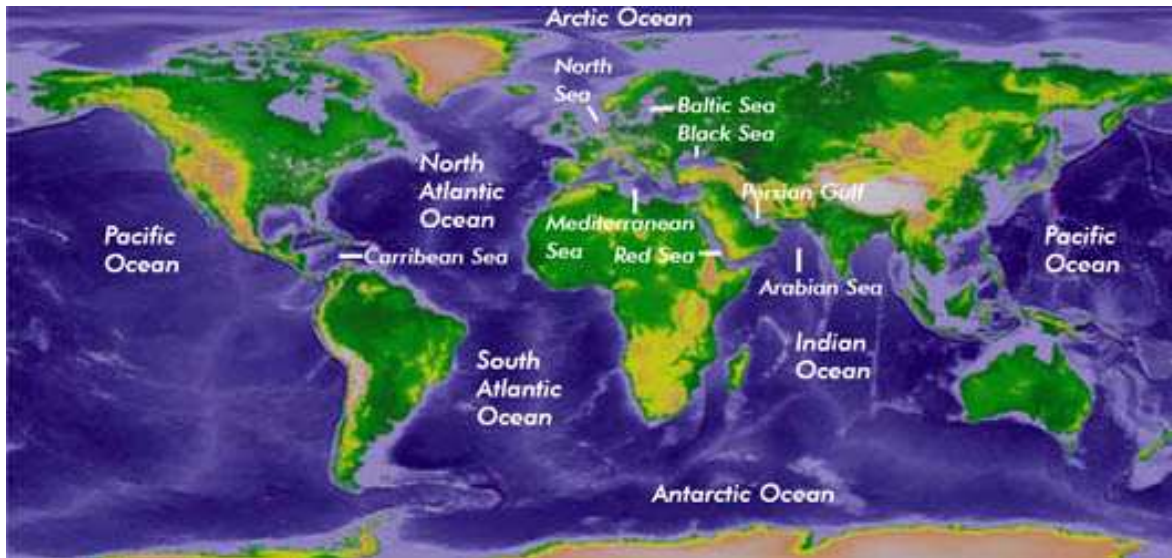


Figure D.1: A World Map of Oceans and Seas

By *navigable rivers*, “k”anals and status mean such rivers, “k”anals and straits that are connected to the seas and can be navigated by boats and ships. Such areas of rivers and “k”anals that are not navigable by ocean-going boats and ships are area-wise elements of “their” continents. Notice that we “lump” “k”anals and straits:

By *continents* we loosely mean some connected land area.

By *harbours* we mean places at the edge of continents, seas, rivers, “k”anals and straits where vessels can berth, unload and load cargo and/or passengers.

By *vessels* we mean ocean-going ships and boats. Without loss of generality we omit consideration of such vessels as floats, barges, etc.

#### D.2.1.2 Formal Introduction



Figure D.2: The Mediterranean and Arab Seas



Figure D.3: The Black Sea and the Kaspian Ocean

#### D.2.1.2.1 Parts and Fluids

368. “The 7 Seas” is a structure composite of the waterways, the continents, the harbours and the vessels.
369. The waterways aggregate consists of an structure composite of a fluids: seas, rivers and “k”anal/straits aggregates.
370. The seas aggregate is a set of seas.
371. The rivers aggregate is a set of [atomic] rivers.
372. The “k”anal/straits aggregate is a set of [atomic] “k”anals and straits.
373. The continents aggregate is a set of [atomic] continents.
374. The harbour aggregate is a set of [atomic] harbours.
375. The Vessel aggregate is a set of [atomic] vessels.

#### type

368. 7Seas, WA, CA, HA, VA  
 369. SA, RA, KA  
 370. S<sub>s</sub> = S-set  
 371. R<sub>s</sub> = R-set  
 372. K<sub>s</sub> = K-set  
 373. C<sub>s</sub> = C-set  
 374. H<sub>s</sub> = H-set  
 375. V<sub>s</sub> = V-set

#### value



Figure D.4: The Mississippi and the Amazon Rivers



Figure D.5: The Yang Tse and the Danube Rivers

368. obs\_WA: 7Seas→WA, obs\_CA: 7Seas→CA, obs\_HA: 7Seas→HA, obsVA: 7Seas→VA  
 369. obs\_SA: WA → SA, obs\_RA: WA → RA, obs\_KA: WA → KA  
 370. obs\_Ss: SA → Ss  
 371. obs\_Rs: RA → Rs  
 372. obs\_Ks: KA → Ks  
 373. obs-Cs: CA → Cs  
 374. obs\_Hs: HA → Hs  
 375. obs\_Vs: VA → Vs

#### D.2.1.2.2 The 7 Seas State

376. By “The 7 Seas state” we mean the collection of all atomic “The 7 Seas” endurants – a collection which is the distributed union of all continents, rivers, canals, continents, harbours and vessels.

#### value

368. 7seas:7Seas  
 370. ss:Ss = obs\_Ss(obs\_SA(obs\_WA(7seas)))  
 371. rs:Rs = obs\_Rs(obs\_RA(obs\_WA(7seas)))  
 372. ks:Ks = obs\_Ks(obs\_KA(obs\_WA(7seas)))  
 373. cs:Cs = obs-Cs(obs\_CA(7seas))  
 374. hs:Hs = obs\_Hs(obs\_HA(7seas))





Figure D.6: The Mediterranean and Arab Seas

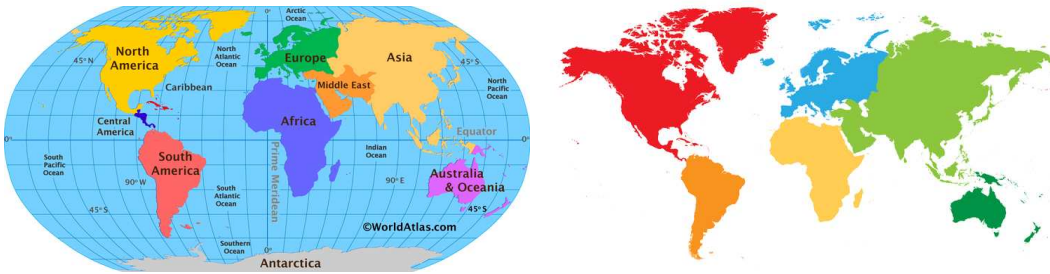


Figure D.7: Continents: *Central America, The Caribbean and Middle East*

- 375.  $vs:Vs = \text{obs\_Vs}(\text{obs\_VA}(7\text{seas}))$
- 376.  $7\sigma:(S|R|K|C|H|V)\text{-set} = ss \cup rs \cup ks \cup cs \cup hs \cup vs$

Please not the *type font* names for the state values.

### D.2.2 Internal Qualities

#### D.2.2.1 Unique Identifiers

##### D.2.2.1.1 Observers

377.

**type**

377. SI, RI, KI, CI, HI, VI

**value**

377.  $\text{uid\_S}: S \rightarrow SI, \text{uid\_R}: R \rightarrow RI, \text{uid\_K}: K \rightarrow KI, \text{uid\_C}: C \rightarrow CI, \text{uid\_H}: H \rightarrow HI, \text{uid\_V}: V \rightarrow VI$

##### D.2.2.1.2 All Unique Identifiers

378. We can calculate the sets of all sea, river, canal, continent, harbor and vessel identifiers,

379. as well as the set of all atomic part and fluid identifiers of the 7 Seas domain.

**value**

- 378.  $\text{sis}:SI\text{-set} = \{\text{uid\_S}(s)|s:S \bullet s \in ss\}$
- 378.  $\text{ris}:RI\text{-set} = \{\text{uid\_R}(r)|r:R \bullet r \in rs\}$
- 378.  $\text{kis}:KI\text{-set} = \{\text{uid\_K}(k)|k:K \bullet k \in ks\}$
- 378.  $\text{cis}:CI\text{-set} = \{\text{uid\_C}(c)|c:C \bullet c \in cs\}$
- 378.  $\text{his}:HI\text{-set} = \{\text{uid\_H}(h)|h:H \bullet h \in hs\}$
- 378.  $\text{vis}:VI\text{-set} = \{\text{uid\_V}(v)|v:V \bullet v \in vs\}$
- 379.  $7is:(S|R|K|C|H|V)\text{-set} = \text{sis} \cup \text{ris} \cup \text{kis} \cup \text{cis} \cup \text{his} \cup \text{vis}$



Figure D.8: The Panama and Suez Canals. The Gibraltar and Malacca Straits

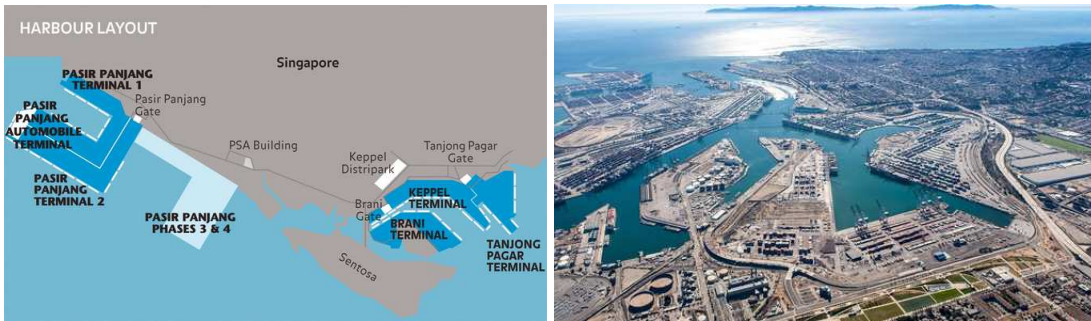


Figure D.9: Singapore and Los Angeles Harbours

### D.2.2.1.3 Axiom

380. All atomic parts and separate fluids have unique identifiers.

#### axiom

380.  $\text{card } 7\sigma = \text{card ais}$

### D.2.2.1.4 Extraction of Atomic Elements

381. From a sea identifier we can, via the domain state  $ss$ , obtain the seal.

382. From a river identifier we can, via the domain state  $rs$ , obtain the river.

383. From a canal identifier we can, via the domain state  $ks$ , obtain the canal.

384. From a continent identifier we can, via the domain state  $cs$ , obtain the continent.

385. From a harbour identifier we can, via the domain state  $hs$ , obtain the harbour.

386. From a vessel identifier we can, via the domain state  $vs$ , obtain the vessel.

#### value

381.  $\text{xtr}_S: SI \rightarrow S; \text{xtr}_S(si) \equiv \text{let } s:S \cdot s \in ss \wedge \text{uid}_S(s) = si \text{ in } s \text{ end}$

382.  $\text{xtr}_R: RI \rightarrow R; \text{xtr}_R(ri) \equiv \text{let } r:R \cdot r \in rs \wedge \text{uid}_R(r) = ri \text{ in } r \text{ end}$

383.  $\text{xtr}_K: KI \rightarrow K; \text{xtr}_K(ki) \equiv \text{let } k:K \cdot k \in ks \wedge \text{uid}_K(k) = ki \text{ in } k \text{ end}$

384.  $\text{xtr}_C: CI \rightarrow C; \text{xtr}_C(ci) \equiv \text{let } c:C \cdot c \in cs \wedge \text{uid}_C(c) = ci \text{ in } c \text{ end}$

385.  $\text{xtr}_H: HI \rightarrow H; \text{xtr}_H(hi) \equiv \text{let } h:H \cdot h \in hs \wedge \text{uid}_H(h) = hi \text{ in } h \text{ end}$

386.  $\text{xtr}_V: VI \rightarrow V; \text{xtr}_V(vi) \equiv \text{let } v:V \cdot v \in vs \wedge \text{uid}_V(v) = vi \text{ in } v \text{ end}$



Figure D.10: Rotterdam and Shanghai Harbours



Figure D.11: Miscellaneous Vessels, I

### D.2.2.2 Mereology

#### D.2.2.2.1 Types, Observers and Axioms Seas

387. The mereology of a sea is a triplet of the sets of unique identifiers of

- the vessels that may sail on it,
- the continents that borders it and
- the harbours that confront it.

#### type

387.  $MS = VI\text{-set} \times CI\text{-set} \times HI\text{-set}$

#### value

387.  $\text{mereo}_S: S \rightarrow MS$

#### axiom

387.  $\forall s:S: s \in ss \Rightarrow \text{let } (vis,cis,his) = \text{mereo}_S(s) \text{ in } vis \subseteq vis \wedge cis \subseteq cis \wedge his \subseteq his \text{ end}$

#### Rivers

388. The mereology of a river is the triplet of

- the non-empty set of unique identifiers of the continents it is embedded in,
- the [one] unique identifier of the sea (or ocean) it is connected to, and
- the set of unique identifiers of the vessels that may sail on that river.

#### type

388.  $MR = CI\text{-set} \times SI \times VI\text{-set}$

#### value

388.  $\text{mereo}_R: R \rightarrow MR$

#### axiom

388.  $\forall r:R: r \in rs \Rightarrow \text{let } (cis,si,vis) = \text{mereo}_R(r) \text{ in } \{ \} \neq cis \subseteq cis \wedge si \in sis \wedge vis \subseteq vis \text{ end}$

#### Canals and Straits



Figure D.12: Miscellaneous Vessels, II



Figure D.13: Miscellaneous Vessels, III

389. The mereology of a canal or a strait is the triplet of

- a set of one or two unique identifiers of the seas that the canal or strait connects,
- the set of unique identifiers of the harbours it offers,
- the set of unique identifiers of the vessels that may sail through the canal or strait.

**type**

389.  $MK = SI\_set \times HI\_set \times VI\_set$

**value**

389.  $mereo\_K: K \rightarrow MK$

**axiom**

389.  $\forall r:K: k \in ks \Rightarrow \text{let } (sis, cis, vis) = \text{mereo\_K}(k) \text{ in } 1 \leq \text{card } sis \leq 2 \wedge sis \subseteq sis \wedge his \in his \wedge vis \subseteq vis \text{ end}$

**Continents**

390. The mereology of a continent is the triplet of

- the set of unique identifiers of the [other<sup>1</sup>] continents that the continent borders with,
- the set of unique identifiers of the harbours on that continent, and
- the set of unique identifiers of the rivers flowing through that continent.

**type**

390.  $MC = CI\_set \times HI\_set \times RI\_set$

**value**

390.  $mereo\_C: C \rightarrow MC$

**axiom**

390.  $\forall c:C: c \in cs \Rightarrow \text{let } (cis, his, ris) = \text{mereo\_C}(c) \text{ in } cis \subseteq cis \wedge his \subseteq his \wedge ris \subseteq ris \text{ end}$

**Harbours**

391. The mereology of a harbour is the triplet of

<sup>1</sup>The **axiom** (389) does not model “the other” clause!



- the unique identifier of the continent to which the harbour belongs, and
- the set of unique identifiers of the vessels that may berth at that harbour.

**type**391.  $MH = CI \times VI\text{-set}$ **value**391.  $\text{mereo}_H: H \rightarrow MH$ **axiom**391.  $\forall h:H \cdot h \in hs \Rightarrow \text{let } (ci,vis) = \text{mereo}_H(j) \text{ in } ci \in cis \wedge vis \in vis \text{ end}$ **Vessels**

392. The mereology of a vessel is the pair of

- the set of unique identifiers of the seas on which the vessel may sail, and
- the set of unique identifiers of the harbours at which the vessel may berth,

**type**392.  $MV = SI\text{-set} \times HI\text{-set}$ **value**392.  $\text{mereo}_V: V \rightarrow MV$ **axiom**392.  $\forall v:V \cdot v \in vis \Rightarrow \text{let } (sis,his) = \text{mereo}_V(v) \text{ in } sis \subseteq sis \wedge his \subseteq his \text{ end}$ 

**D.2.2.2.2 A Remark** Please note that we have not [yet] had a need to describe the sea and land AREAS of seas and continents.

**D.2.2.2.3 A Domain Axiom** The axioms of Sect. D.2.2.2.1 pertains to the individual atomic elements of the domain, not to their occurrence in the context of the aggregates to which they are elements.

393. The mereology of a sea of a domain states the unique identifiers of the vessels that may sail on it, so we must, vice-versa, expect that the mereology of the identified vessels likewise identify that sea as one on which it may sail.

**axiom**393.  $\forall s:S \cdot s \in ss \Rightarrow$ 393.  $\text{let } (vis,cis,his) = \text{mereo}_S(s) \text{ in}$ 393.  $\forall vi:VI \cdot vi \in vis \Rightarrow$ 393.  $\text{let } v:V \cdot v = \text{xtr}_V(vi) \text{ in}$ 393.  $\text{let } (sis,his) = \text{mereo}_V(v) \text{ in}$ 393.  $\text{uid}_S(s) \in sis \text{ end end end}$ 

We leave it to the reader to narrate and formalise similar “cross-mereology” axioms for [all other] relevant “pairs” of different sort atomic elements of the domain.

**D.2.2.3 Attributes**

Seas, rivers, canals, continents and harbours have spatial attributes of kind SURFACE, LINE and POINT. We refer to [58, Sect. 3.4].

**D.2.2.3.1 Seas**

- 394. We ascribe names to seas.
- 395. Seas spread over contiguous surface (**SURFACE**).
- 396. Seas have borders/edges (**LINE**).
- 397.
- 398.
- 399.
- 400.

**type**

- 394. SeaName
- 395. SeaSurface = **SURFACE**
- 396. SeaBorder = **LINE**
- 397.
- 398.
- 399.

**value**

- 394. attr\_SeaName: S → SeaName
- 395. attr\_SeaSurface: S → SeaSurface
- 396. attr\_SeaBorder: S → SeaBorder
- 397. attr\_: →
- 398. attr\_: →
- 399. attr\_: →

**D.2.2.3.2 Rivers**

- 401.
- 402.
- 403.
- 404.
- 405.
- 406.
- 407.

**type**

- 401.
- 402.
- 403.
- 404.
- 405.
- 406.

**value**

- 401. attr\_: →
- 402. attr\_: →
- 403. attr\_: →

- 404. attr.: →
- 405. attr.: →
- 406. attr.: →

**D.2.2.3.3 Canals and Straits**

- 408.
- 409.
- 410.
- 411.
- 412.
- 413.
- 414.

**type**

- 408.
- 409.
- 410.
- 411.
- 412.
- 413.

**value**

- 408. attr.: →
- 409. attr.: →
- 410. attr.: →
- 411. attr.: →
- 412. attr.: →
- 413. attr.: →

**D.2.2.3.4 Continents**

- 415.
- 416.
- 417.
- 418.
- 419.
- 420.
- 421.

**type**

- 415.
- 416.
- 417.
- 418.
- 419.

420.

**value**

415. attr.: →

416. attr.: →

417. attr.: →

418. attr.: →

419. attr.: →

420. attr.: →

**D.2.2.3.5 Harbours**

422.

423.

424.

425.

426.

427.

428.

**type**

422.

423.

424.

425.

426.

427.

**value**

422. attr.: →

423. attr.: →

424. attr.: →

425. attr.: →

426. attr.: →

427. attr.: →

**D.2.2.3.6 Vessels**

429. Vessels have names.

430. Vessels have kind: passenger, ordinary freight, crude oil, container, ...

431. Vessels, at any one “point” in time has a position.

432. Vessels, when sailing, follow a route.

433. Vessel positions are well-formed if they are on the current route.

434. Vessels have a speed

435. and a velocity.

436. A vessel is **on course** if its position (at some time) is on that vessel’s route.

**type**

429. VesselName  
430. VesselKind = ...  
431. VesselPos = TIME × POSITION  
432. VesselRoute = BezierCurve  
434. VesselSpeed  
434. VesselVelocity

**value**

429. attr\_VesselName: V → VesselName  
430. attr\_VesselKind: V → VesselKind  
431. attr\_VesselPos: V → VesselPos  
432. attr\_VesselRoute: V → VesselRoute  
434. attr\_VesselSpeed: V → Speed  
435. attr\_VesselVelocity: V → Velocity  
436. Vessel\_on\_course: V → **Bool**  
436. Vessel\_on\_course(v) ≡ **let** (vp,\_) = attr\_VesselPos(v) **in** Position\_on\_curve(vp,attr\_VesselRoute(v)) **end**  
436. Position\_on\_curve: POSITION × Bezier → **Bool**

## D.3 Perdurants

### D.3.1 Channels

### D.3.2 Behaviours

#### D.3.2.1 Signatures

#### D.3.2.2 Definitions

#### D.3.2.3 System

## D.4 Conclusion



# Part V

## Concrete Domain Models

In Chapters E–P we present 11 concrete domain models. By a concrete domain we mean a domain which primarily covers properties of a man-made domain.

We briefly characterize these here.

- **Chapter E: Road Transport** pages 171–189  
Chapter is based on drafts from as early as 20 years ago. Chapter 2 has already brought many excerpts of this domain model.
- **Chapter F: Rail Systems** pages 191–209  
First drafts from 1993! This, obviously, goes back many years. The first railway domain modelling was done by the late Søren Prehn.<sup>2</sup>
- **Chapter G: Simple Credit Card Systems** pages 211–221  
This draft model was worked out, with PhD students at a two week course in May 2016 at Uppsala University, Sweden.
- **Chapter H: Simple Consumer Market Systems** pages 223–260  
Draft from January 2021. See the first text of chapter H, page 224. Haim Kilov challenged me, in 2001/2002, to work out a first model of the market, [21]. Haim then kindly published it.
- **Chapter I: Pipelines** pages 261–276  
Draft from 2008. At a PhD course, in November 2008, I asked the students to select a domain and they chose this, a pipeline domain.<sup>3</sup>
- **Chapter J: Shipping** pages 277–303  
I worked out an early draft for an Isola Lipari PhD Summer School in Italy in 2007 [30] [www.imm.dtu.dk/~dibj/lipari-paper.pdf](http://www.imm.dtu.dk/~dibj/lipari-paper.pdf), <http://www.imm.dtu.dk/~dibj/container-paper.pdf>. In April 2022 I worked out, from scratch, a much simpler domain model [60] [www.imm.dtu.dk/~dibj/2021/ral/ral.pdf](http://www.imm.dtu.dk/~dibj/2021/ral/ral.pdf) based, illustratively on the Royal Arctic Lines (hence the file name `ral.pdf`), a Danish/Greenland shipping line.

---

<sup>2</sup>Søren was an M.Sc. student of mine, one of the first I hired into the **Dansk Datamatik Center** (1980–1989). A brilliant software engineer and a fine computer scientist. My first choice of staff to join me (1992–1994) at the UN University’s International Institute for Software Technology, UNU/IIST in Macau – of which I was the first and founding UN Director (1992–1997). Søren sadly passed away in the Spring of 2006. God Bless his soul.

<sup>3</sup>It later transpired that one of the PhD students was the son of a top director at the Austrian Mineral oil Company ÖMG.

- **Chapter K: Container Terminals** pages 305–360  
Draft from a PhD course at ECNU: East China Normal University, Shanghai, Fall 2018. Students actually got to visit SECT (Shanghai East Container Terminal) of the Danish A.P.Møller Maersk company.
- **Chapter M: Document Systems** pages 363–384  
Draft from summer 2017.
- **Chapter N: Swarms of Drones** pages 385–415  
Draft from November 2017.
- **Chapter O: Assembly Lines** pages 417–453  
Draft from Summer 2021.
- **Chapter P: Nuclear Power Plants** pages 455–497  
Draft from July 2023.



# Appendix E

## Road Transport

### Contents

---

|            |                                                           |            |
|------------|-----------------------------------------------------------|------------|
| <b>E.1</b> | <b>The Road Transport Domain</b>                          | <b>172</b> |
| E.1.1      | Naming                                                    | 172        |
| E.1.2      | Rough Sketch                                              | 172        |
| <b>E.2</b> | <b>External Qualities</b>                                 | <b>172</b> |
| E.2.1      | A Road Transport System, II – Abstract External Qualities | 172        |
| E.2.2      | Transport System Structure                                | 173        |
| E.2.3      | Atomic Road Transport Parts                               | 173        |
| E.2.4      | Compound Road Transport Parts                             | 173        |
| E.2.4.1    | The Composites                                            | 173        |
| E.2.4.2    | The Part Parts                                            | 174        |
| E.2.5      | The Transport System State                                | 174        |
| <b>E.3</b> | <b>Internal Qualities</b>                                 | <b>174</b> |
| E.3.1      | Unique Identifiers                                        | 174        |
| E.3.1.1    | Extract Parts from Their Unique Identifiers               | 175        |
| E.3.1.2    | All Unique Identifiers of a Domain                        | 175        |
| E.3.1.3    | Uniqueness of Road Net Identifiers                        | 176        |
| E.3.2      | Mereology                                                 | 176        |
| E.3.2.1    | Mereology Types and Observers                             | 176        |
| E.3.2.2    | Invariance of Mereologies                                 | 177        |
| E.3.2.2.1  | Invariance of Road Nets                                   | 177        |
| E.3.2.2.2  | Possible Consequences of a Road Net Mereology             | 178        |
| E.3.2.2.3  | Fixed and Varying Mereology                               | 178        |
| E.3.3      | Attributes                                                | 178        |
| E.3.3.1    | Hub Attributes                                            | 178        |
| E.3.3.2    | Invariance of Traffic States                              | 179        |
| E.3.3.3    | Link Attributes                                           | 179        |
| E.3.3.4    | Bus Company Attributes                                    | 179        |
| E.3.3.5    | Bus Attributes                                            | 180        |
| E.3.3.6    | Private Automobile Attributes                             | 180        |
| E.3.3.7    | Intentionality                                            | 181        |
| <b>E.4</b> | <b>Perdurants</b>                                         | <b>182</b> |
| E.4.1      | Channels and Communication                                | 182        |
| E.4.1.1    | Channel Message Types                                     | 182        |
| E.4.1.2    | Channel Declarations                                      | 183        |

|           |                                            |     |
|-----------|--------------------------------------------|-----|
| E.4.2     | <b>Behaviours</b>                          | 183 |
| E.4.2.1   | <b>Road Transport Behaviour Signatures</b> | 183 |
| E.4.2.2   | <b>Behaviour Definitions</b>               | 185 |
| E.4.2.2.1 | <b>Automobile Behaviour at a Hub</b>       | 185 |
| E.4.2.2.2 | <b>Automobile Behaviour On a Link</b>      | 186 |
| E.4.2.2.3 | <b>Hub Behaviour</b>                       | 187 |
| E.4.2.2.4 | <b>Link Behaviour</b>                      | 187 |
| E.5       | <b>System Initialisation</b>               | 188 |
| E.5.1     | <b>Initial States</b>                      | 188 |
| E.5.2     | <b>Initialisation</b>                      | 188 |

---

## E.1 The Road Transport Domain

Our universe of discourse in this chapter is the road transport domain.

### E.1.1 Naming

`type` RTS

### E.1.2 Rough Sketch

The road transport system that we have in mind consists of a road net and a set of vehicles such that the road net serves to convey vehicles. We consider the road net to consist of hubs, i.e., street intersections, or just street segment connection points, and links, i.e., street segments between adjacent hubs. We consider vehicles to additionally include departments of motor vehicles (DMVs), bus companies, each with zero, one or more buses, and vehicle associations, each with zero, one or more members who are owners of zero, one or more vehicles<sup>1</sup> ■

## E.2 External Qualities

**A Road Transport System, I – Manifest External Qualities:** Our intention is that the manifest external qualities of a road transport system are those of its roads, their **hubs**<sup>2</sup>i.e., road (or street) intersections, and their **links**, i.e., the roads (streets) between hubs, and **vehicles**, i.e., automobiles – that ply the roads – the buses, trucks, private cars, bicycles, etc. ■

### E.2.1 A Road Transport System, II – Abstract External Qualities

Examples of what could be considered abstract external qualities of a road transport domain are: the aggregate of all hubs and all links, the aggregate of all buses, say into bus companies, the aggregate of all bus companies into public transport, and the aggregate of all vehicles into a department of vehicles. Some of these aggregates may, at first be treated as abstract. Subsequently, in our further analysis & description we may decide to consider some of them as concretely manifested in, for example, actual departments of roads.

<sup>1</sup>This “rough” narrative fails to narrate what hubs, links, vehicles, DMVs, bus companies, buses and vehicle associations are. In presenting it here, as we are, we rely on your a priori understanding of these terms. But that is dangerous! The danger, if we do not painstakingly narrate and formalise what we mean by all these terms, then readers (software designers, etc.) may make erroneous assumptions.

<sup>2</sup>We have **highlighted** certain enduring sort names – as they will re-appear in rather many upcoming examples.

### E.2.2 Transport System Structure

A transport system is modeled as structured into a *road net structure* and an *automobile structure*. The *road net structure* is then structured as a pair: a *structure of hubs* and a *structure of links*. These latter structures are then modeled as set of hubs, respectively links.

We could have modeled the road net *structure* as a *composite part* with *unique identity*, *mereology* and *attributes* which could then serve to model a road net authority. And we could have modeled the automobile *structure* as a *composite part* with *unique identity*, *mereology* and *attributes* which could then serve to model a department of vehicles ■

### E.2.3 Atomic Road Transport Parts

From one point of view all of the following can be considered atomic parts: hubs, links<sup>3</sup>, and automobiles.

### E.2.4 Compound Road Transport Parts

#### E.2.4.1 The Composites

437. There is the *universe of discourse*, UoD.      438. a *road net*, RN, and

It is structured into      439. a *fleet of vehicles*, FV.

Both are structures. ....

|                                                                   |                                    |
|-------------------------------------------------------------------|------------------------------------|
| <b>type</b>                                                       | <b>value</b>                       |
| 437 UoD <b>axiom</b> $\forall uod:UoD \cdot is\_structure(uod)$ . | 438 obs_RN: UoD $\rightarrow$ RN   |
| 438 RN <b>axiom</b> $\forall rn:RN \cdot is\_structure(rn)$ .     | 439 obs_FV: UoD $\rightarrow$ FV ■ |
| 439 FV <b>axiom</b> $\forall fv:FV \cdot is\_structure(fv)$ .     |                                    |

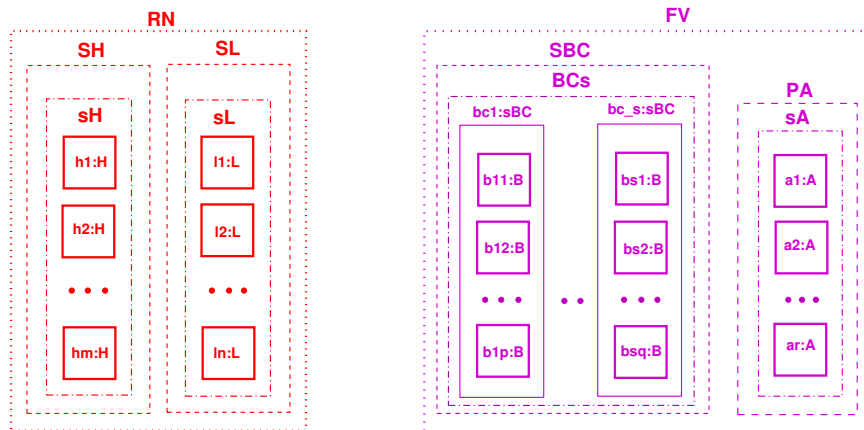


Figure E.1: A Road Transport System: Compounds and Structures

<sup>3</sup>Hub  $\equiv$  street intersection; link  $\equiv$  street segments with no intervening hubs.

### E.2.4.2 The Part Parts

- 440. The structure of hubs is a set,  $sH$ , of atomic hubs,  $H$ .
- 441. The structure of links is a set,  $sL$ , of atomic links,  $L$ .
- 442. The structure of buses is a set,  $sBC$ , of composite bus companies,  $BC$ .
- 443. The composite bus companies,  $BC$ , are sets of buses,  $sB$ .
- 444. The structure of private automobiles is a set,  $sA$ , of atomic automobiles,  $A$ .

type

- 440  $H, sH = H\text{-set axiom } \forall h:H \cdot \text{is\_atomic}(h)$
- 441  $L, sL = L\text{-set axiom } \forall l:L \cdot \text{is\_atomic}(l)$
- 442  $BC, BCs = BC\text{-set axiom } \forall bc:BC \cdot \text{is\_composite}(bc)$
- 443  $B, Bs = B\text{-set axiom } \forall b:B \cdot \text{is\_atomic}(b)$
- 444  $A, sA = A\text{-set axiom } \forall a:A \cdot \text{is\_atomic}(a)$

value

- 440  $\text{obs\_sH}: SH \rightarrow sH$
- 441  $\text{obs\_sL}: SL \rightarrow sL$
- 442  $\text{obs\_sBC}: SBC \rightarrow BCs$
- 443  $\text{obs\_Bs}: BCs \rightarrow Bs$
- 444  $\text{obs\_sA}: SA \rightarrow sA$  ■

## E.2.5 The Transport System State

- 445. Let there be given a universe of discourse,  $rts$ . It is an example of a state.

From that state we can calculate other states.

- 446. The set of all hubs,  $hs$ .
- 447. The set of all links,  $ls$ .
- 448. The set of all hubs and links,  $hls$ .
- 449. The set of all bus companies,  $bcs$ .
- 450. The set of all buses,  $bs$ .
- 451. The set of all private automobiles,  $as$ .
- 452. The set of all parts,  $ps$ .

value

- 445  $rts:UoD$
- 446  $hs:H\text{-set} \equiv :H\text{-set} \equiv \text{obs\_sH}(\text{obs\_SH}(\text{obs\_RN}(rts)))$
- 447  $ls:L\text{-set} \equiv :L\text{-set} \equiv \text{obs\_sL}(\text{obs\_SL}(\text{obs\_RN}(rts)))$
- 448  $hls:(H|L)\text{-set} \equiv hs \cup ls$
- 449  $bcs:BC\text{-set} \equiv \text{obs\_BCs}(\text{obs\_SBC}(\text{obs\_FV}(\text{obs\_RN}(rts))))$
- 450  $bs:B\text{-set} \equiv \cup\{\text{obs\_Bs}(bc) \mid bc:BC \cdot bc \in bcs\}$
- 451  $as:A\text{-set} \equiv \text{obs\_BCs}(\text{obs\_SBC}(\text{obs\_FV}(\text{obs\_RN}(rts))))$
- 452  $ps:(UoB|H|L|BC|B|A)\text{-set} \equiv rts \cup hls \cup bcs \cup bs \cup as$

## E.3 Internal Qualities

### E.3.1 Unique Identifiers

453. We assign unique identifiers to all parts. (b) All links have distinct identifiers.
454. By a road identifier we shall mean a link or a hub identifier. (c) All bus companies have distinct identifiers.
455. By a vehicle identifier we shall mean a bus or an automobile identifier. (d) All buses of all bus companies have distinct identifiers.
456. Unique identifiers uniquely identify all parts. (e) All automobiles have distinct identifiers.
- (a) All hubs have distinct [unique] identifiers. (f) All parts have distinct identifiers.

|                                   |      |                                |
|-----------------------------------|------|--------------------------------|
| <b>type</b>                       | 456a | uid_H: $H \rightarrow H\_UI$   |
| 453 H_UI, L_UI, BC_UI, B_UI, A_UI | 456b | uid_L: $H \rightarrow L\_UI$   |
| 454 R_UI = H_UI   L_UI            | 456c | uid_BC: $H \rightarrow BC\_UI$ |
| 455 V_UI = B_UI   A_UI            | 456d | uid_B: $H \rightarrow B\_UI$   |
| <b>value</b>                      | 456e | uid_A: $H \rightarrow A\_UI$   |

### E.3.1.1 Extract Parts from Their Unique Identifiers

457. From the unique identifier of a part we can retrieve,  $\wp$ , the part having that identifier.

|                                                                                                                                      |
|--------------------------------------------------------------------------------------------------------------------------------------|
| <b>type</b>                                                                                                                          |
| 457 $P = H \mid L \mid BC \mid B \mid A$                                                                                             |
| <b>value</b>                                                                                                                         |
| 457 $\wp: H\_UI \rightarrow H \mid L\_UI \rightarrow L \mid BC\_UI \rightarrow BC \mid B\_UI \rightarrow B \mid A\_UI \rightarrow A$ |
| 457 $\wp(ui) \equiv \text{let } p:(H L BC B A) \bullet p \in ps \wedge \text{uid\_P}(p) = ui \text{ in } p \text{ end}$              |

### E.3.1.2 All Unique Identifiers of a Domain

We can calculate:

458. the set,  $h_{uis}$ , of unique *hub* identifiers;
459. the set,  $l_{uis}$ , of unique *link* identifiers;
460. the map,  $hl_{uim}$ , from unique *hub* identifiers to the set of unique *link* identifiers of the links connected to the zero, one or more identified hubs,
461. the map,  $lh_{uim}$ , from unique *link* identifiers to the set of unique *hub* identifiers of the two hubs connected to the identified link;
462. the set,  $r_{uis}$ , of all unique hub and link, i.e., *road* identifiers;
463. the set,  $bc_{uis}$ , of unique *bus company* identifiers;
464. the set,  $b_{uis}$ , of unique *bus* identifiers;
465. the set,  $a_{uis}$ , of unique private *automobile* identifiers;
466. the set,  $v_{uis}$ , of unique bus and automobile, i.e., *vehicle* identifiers;
467. the map,  $bc_{b_{uim}}$ , from unique *bus company* identifiers to the set of its unique *bus* identifiers; and
468. the (bijective) map,  $bc_{b_{uim}}$ , from unique *bus* identifiers to their unique *bus company* identifiers.

**value**

- 458  $h_{uis}:H\_UI\text{-set} \equiv \{uid\_H(h)|h:H \bullet h \in hs\}$   
 459  $l_{uis}:L\_UI\text{-set} \equiv \{uid\_L(l)|l:L \bullet l \in ls\}$   
 462  $r_{uis}:R\_UI\text{-set} \equiv h_{uis} \cup l_{uis}$   
 460  $hl_{uim}:(H\_UI \xrightarrow{m} L\_UI\text{-set}) \equiv$   
 460  $[h\_ui \mapsto luis | h\_ui:H\_UI, luis:L\_UI\text{-set} \bullet h\_ui \in h_{uis} \wedge (\_, luis, \_) = mereo\_H(\eta(h\_ui))] \text{ [cf. Item 475]}$   
 461  $lh_{uim}:(L+UI \xrightarrow{m} H\_UI\text{-set}) \equiv$   
 461  $[l\_ui \mapsto huis | h\_ui:L\_UI, huis:H\_UI\text{-set} \bullet l\_ui \in l_{uis} \wedge (\_, huis, \_) = mereo\_L(\eta(l\_ui))] \text{ [cf. Item 476]}$   
 463  $bc_{uis}:BC\_UI\text{-set} \equiv \{uid\_BC(bc)|bc:BC \bullet bc \in bcs\}$   
 464  $b_{uis}:B\_UI\text{-set} \equiv \cup \{uid\_B(b)|b:B \bullet b \in bs\}$   
 465  $a_{uis}:A\_UI\text{-set} \equiv \{uid\_A(a)|a:A \bullet a \in as\}$   
 466  $v_{uis}:V\_UI\text{-set} \equiv b_{uis} \cup a_{uis}$   
 467  $bc_{uim}:(BC\_UI \xrightarrow{m} B\_UI\text{-set}) \equiv$   
 467  $[bc\_ui \mapsto buis | bc\_ui:BC\_UI, bc:BC \bullet bc \in bcs \wedge bc\_ui = uid\_BC(bc) \wedge (\_, \_, buis) = mereo\_BC(bc) ]$   
 468  $bbc_{uim}:(B\_UI \xrightarrow{m} BC\_UI) \equiv$   
 468  $[b\_ui \mapsto bc\_ui | b\_ui:B\_UI, bc\_ui:BC\_UI \bullet bc\_ui = \mathbf{domb}bc_{uim} \wedge b\_ui \in bbc_{uim}(bc\_ui) ]$

**E.3.1.3 Uniqueness of Road Net Identifiers**

We must express the following axioms:

469. All hub identifiers are distinct.  
 470. All link identifiers are distinct.  
 471. All bus company identifiers are distinct.  
 472. All bus identifiers are distinct.  
 473. All private automobile identifiers are distinct.  
 474. All part identifiers are distinct.

**axiom**

- 469  $\mathbf{card} hs = \mathbf{card} h_{uis}$   
 470  $\mathbf{card} ls = \mathbf{card} l_{uis}$   
 471  $\mathbf{card} bcs = \mathbf{card} bc_{uis}$   
 472  $\mathbf{card} bs = \mathbf{card} b_{uis}$   
 473  $\mathbf{card} as = \mathbf{card} a_{uis}$   
 474  $\mathbf{card} \{h_{uis} \cup l_{uis} \cup bc_{uis} \cup b_{uis} \cup a_{uis}\}$   
 474  $= \mathbf{card} h_{uis} + \mathbf{card} l_{uis} + \mathbf{card} bc_{uis} + \mathbf{card} b_{uis} + \mathbf{card} a_{uis} \quad \blacksquare$

**E.3.2 Mereology****E.3.2.1 Mereology Types and Observers**

475. The mereology of hubs is a pair: (i) the set of all bus and automobile identifiers<sup>4</sup>, and (ii) the set of unique identifiers of the links that it is connected to and the set of all unique identifiers of all vehicles (buses and private automobiles).<sup>5</sup>
476. The mereology of links is a pair: (i) the set of all bus and automobile identifiers, and (ii) the set of the two distinct hubs they are connected to.
477. The mereology of a bus company is a set the unique identifiers of the buses operated by that company.

478. The mereology of a bus is a pair: (i) the set of the one single unique identifier of the bus company it is operating for, and (ii) the unique identifiers of all links and hubs<sup>6</sup>.
479. The mereology of an automobile is the set of the unique identifiers of all links and hubs<sup>7</sup>.

**type**

```
475 H_Mer = V_UI-set × L_UI-set
476 L_Mer = V_UI-set × H_UI-set
477 BC_Mer = B_UI-set
478 B_Mer = BC_UI × R_UI-set
479 A_Mer = R_UI-set
```

**value**

```
475 mereo_H: H → H_Mer
476 mereo_L: L → L_Mer
477 mereo_BC: BC → BC_Mer
478 mereo_B: B → B_Mer
479 mereo_A: A → A_Mer
```

**E.3.2.2 Invariance of Mereologies**

For mereologies one can usually express some invariants. Such invariants express “*law-like properties*”, facts which are indisputable.

**E.3.2.2.1 Invariance of Road Nets** The observed mereologies must express identifiers of the state of such for road nets:

**axiom**

```
475 ∀ (vuis,luis):H_Mer • luis ⊆ luiss ∧ vuis = vuiss
476 ∀ (vuis,huis):L_Mer • vuis = vuiss ∧ huis ⊆ huiss ∧ cardhuis = 2
477 ∀ buis:H_Mer • buis = buiss
478 ∀ (bc_ui,ruis):H_Mer • bc_ui ∈ bcuiss ∧ ruis = ruiss
479 ∀ ruis:A_Mer • ruis = ruiss
```

480. For all hubs,  $h$ , and links,  $l$ , in the same road net,
481. if the hub  $h$  connects to link  $l$  then link  $l$  connects to hub  $h$ .

**axiom**

```
480 ∀ h:H,l:L • h ∈ hs ∧ l ∈ ls ⇒
480 let (__,luis)=mereo_H(h), (__,huis)=mereo_L(l)
481 in uid_L(l) ∈ luis ≡ uid_H(h) ∈ huis end
```

482. For all links,  $l$ , and hubs,  $h_a, h_b$ , in the same road net,
483. if the  $l$  connects to hubs  $h_a$  and  $h_b$ , then  $h_a$  and  $h_b$  both connects to link  $l$ .

**axiom**

```
482 ∀ h_a,h_b:H,l:L • {h_a,h_b} ⊆ hs ∧ l ∈ ls ⇒
482 let (__,luis)=mereo_H(h), (__,huis)=mereo_L(l)
483 in uid_L(l) ∈ luis ≡ uid_H(h) ∈ huis end
```

<sup>4</sup>This is just another way of saying that the meaning of hub mereologies involves the unique identifiers of all the vehicles that might pass through the hub `is_of_interest` to it.

<sup>5</sup>The link identifiers designate the links, zero, one or more, that a hub is connected to `is_of_interest` to both the hub and that these links is `interested` in the hub.

<sup>6</sup>— that the bus might pass through

<sup>7</sup>— that the automobile might pass through

### E.3.2.2.2 Possible Consequences of a Road Net Mereology

484. are there [isolated] units from which one can not “reach” other units ?
485. does the net consist of two or more “disjoint” nets ?
486. et cetera.

We leave it to the reader to narrate and formalise the above properly.

**E.3.2.2.3 Fixed and Varying Mereology** Let us consider a road net. If hubs and links never change “affiliation”, that is: hubs are in fixed relation to zero one or more links, and links are in a fixed relation to exactly two hubs then the mereology is a *fixed mereology*. f, on the other hand hubs may be inserted into or removed from the net, and/or links may be removed from or inserted between any two existing hubs, then the mereology is a *varying mereology*.

## E.3.3 Attributes

### E.3.3.1 Hub Attributes

We treat some attributes of the hubs of a road net.

487. There is a hub state. It is a set of pairs,  $(l_f, l_t)$ , of link identifiers, where these link identifiers are in the mereology of the hub. The meaning of the hub state in which, e.g.,  $(l_f, l_t)$  is an element, is that the hub is open, “green”, for traffic *f* from link  $l_f$  to link  $l_t$ . If a hub state is empty then the hub is closed, i.e., “red” for traffic from any connected links to any other connected links.
488. There is a hub state space. It is a set of hub states. The current hub state must be in its hub state space. The meaning of the hub state space is that its states are all those states that the hub can attain.
489. Since we can think rationally about it, it can be described, hence we can model, as an attribute of hubs, a history of its traffic: the recording, per unique bus and automobile identifier, of the time ordered presence in the hub of these vehicles. Hub history is an *event history*.

type

$$487 \text{ H}\Sigma = (\text{L\_UI} \times \text{L\_UI})\text{-set}$$

axiom

$$487 \forall h:\text{H} \bullet \text{obs\_H}\Sigma(h) \in \text{obs\_H}\Omega(h)$$

type

$$488 \text{ H}\Omega = \text{H}\Sigma\text{-set}$$

489 H\_Traffic

$$489 \text{ H\_Traffic} = (\text{A\_UI} | \text{B\_UI}) \xrightarrow{\overline{m}} (\text{TIME} \times \text{VPos})^*$$

axiom

$$489 \forall ht:\text{H\_Traffic}, ui:(\text{A\_UI} | \text{B\_UI}) \bullet$$

$$489 \quad ui \in \mathbf{dom} \text{ ht} \Rightarrow \text{time\_ordered}(\text{ht}(ui))$$

value

$$487 \text{ attr\_H}\Sigma: \text{H} \rightarrow \text{H}\Sigma$$

$$488 \text{ attr\_H}\Omega: \text{H} \rightarrow \text{H}\Omega$$

$$489 \text{ attr\_H\_Traffic}: \text{H} \rightarrow \text{H\_Traffic}$$

value

$$489 \text{ time\_ordered}: (\text{TIME} \times \text{VPos})^* \rightarrow \mathbf{Bool}$$

$$489 \text{ time\_ordered}(\text{tvpl}) \equiv \dots$$

In Item 489 we model the time-ordered sequence of traffic as a discrete sampling, i.e.,  $\xrightarrow{\overline{m}}$ , rather than as a continuous function,  $\rightarrow$ .



**E.3.3.2 Invariance of Traffic States**

490. The link identifiers of hub states must be in the set,  $l_{ui}s$ , of the road net's link identifiers.

**axiom**

490  $\forall h:H \bullet h \in hs \Rightarrow$

490 **let**  $h\sigma = \text{attr\_H}\Sigma(h)$  **in**  $\forall (l_{ui}i, l_{ui}i'):(L\_UI \times L\_UI) \bullet (l_{ui}i, l_{ui}i') \in h\sigma \Rightarrow \{l_{ui}i, l_{ui}i'\} \subseteq l_{ui}s$  **end**

**E.3.3.3 Link Attributes**

We show just a few attributes.

491. There is a link state. It is a set of pairs,  $(h_f, h_t)$ , of distinct hub identifiers, where these hub identifiers are in the mereology of the link. The meaning of a link state in which  $(h_f, h_t)$  is an element is that the link is open, “green”, for traffic from hub  $h_f$  to hub  $h_t$ . Link states can have either 0, 1 or 2 elements.

492. There is a link state space. It is a set of link states. The meaning of the link state space is that its states are all those the which the link can attain. The current link state must be in its state space. If a link state space is empty then the link is (permanently) closed. If it has one element then it is a one-way link. If a one-way link,  $l$ , is imminent on a hub whose mereology designates that link, then the link is a “trap”, i.e., a “blind cul-de-sac”.

493. Since we can think rationally about it, it can be described, hence it can model, as an attribute of links a history of its traffic: the recording, per unique bus and automobile identifier, of the time ordered positions along the link (from one hub to the next) of these vehicles.

494. The hub identifiers of link states must be in the set,  $h_{ui}s$ , of the road net's hub identifiers.

**type**

491  $L\Sigma = H\_UI\text{-set}$

**axiom**

491  $\forall l\sigma:L\Sigma \bullet \text{card } l\sigma = 2$

491  $\forall l:L \bullet \text{obs\_L}\Sigma(l) \in \text{obs\_L}\Omega(l)$

**type**

492  $L\Omega = L\Sigma\text{-set}$

493  $L\_Traffic$

493  $L\_Traffic = (A\_UI|B\_UI) \xrightarrow{m} (\mathbb{T} \times (H\_UI \times \text{Frac} \times H\_UI))^*$

493  $\text{Frac} = \text{Real}$ , **axiom**  $\text{frac:Frac} \bullet 0 < \text{frac} < 1$

**value**

491  $\text{attr\_L}\Sigma: L \rightarrow L\Sigma$

492  $\text{attr\_L}\Omega: L \rightarrow L\Omega$

493  $\text{attr\_L\_Traffic}: L \rightarrow L\_Traffic$

**axiom**

493  $\forall lt:L\_Traffic, ui:(A\_UI|B\_UI) \bullet ui \in \text{dom } ht \Rightarrow \text{time\_ordered}(ht(ui))$

494  $\forall l:L \bullet l \in ls \Rightarrow$

494 **let**  $l\sigma = \text{attr\_L}\Sigma(l)$  **in**  $\forall (h_{ui}i, h_{ui}i'):(H\_UI \times K\_UI) \bullet$

494  $(h_{ui}i, h_{ui}i') \in l\sigma \Rightarrow \{h_{ui}i, h_{ui}i'\} \subseteq h_{ui}s$  **end**

**E.3.3.4 Bus Company Attributes**

Bus companies operate a number of lines that service passenger transport along routes of the road net. Each line being serviced by a number of buses.

495. Bus companies create, maintain, revise and distribute [to the public (not modeled here), and to buses] bus time tables, not further defined.

**type**

495 BusTimTbl

**value**

495 attr\_BusTimTbl: BC  $\rightarrow$  BusTimTbl

There are two notions of time at play here: the indefinite “real” or “actual” time; and the definite calendar, hour, minute and second time designation occurring in some textual form in, e.g., time tables.

**E.3.3.5 Bus Attributes**

We show just a few attributes.

496. Buses run routes, according to their line number,  $ln:LN$ , in the
497. bus time table,  $btt:BusTimTbl$  obtained from their bus company, and and keep, as inert attributes, their segment of that time table.
498. Buses occupy positions on the road net:
- (a) either *at a hub* identified by some  $h\_ui$ ,
  - (b) or *on a link*, some *fraction*,  $f:Fract$ , down an *identified link*,  $l\_ui$ , from one of its *identified connecting hubs*,  $fh\_ui$ , in the direction of the other *identified hub*,  $th\_ui$ .
499. Et cetera.

**type**

496 LN

497 BusTimTbl

498 BPos == atHub | onLink

498a atHub ::  $h\_ui:H\_UI$

498b onLink ::  $fh\_ui:H\_UI \times l\_ui:L\_UI \times frac:Fract \times th\_ui:H\_UI$

498b Fract = **Real**, axiom  $frac:Fract \cdot 0 < frac < 1$

499 ...

**value**

497 attr\_BusTimTbl: B  $\rightarrow$  BusTimTbl

498 attr\_BPos: B  $\rightarrow$  BPos

**E.3.3.6 Private Automobile Attributes**

We illustrate but a few attributes:

500. Automobiles have static number plate registration numbers.
501. Automobiles have dynamic positions on the road net:
- [498a] either *at a hub* identified by some  $h\_ui$ ,
  - [498b] or *on a link*, some *fraction*,  $frac:Fract$  down an *identified link*,  $l\_ui$ , from one of its *identified connecting hubs*,  $fh\_ui$ , in the direction of the other *identified hub*,  $th\_ui$ .

**type**

```

500 RegNo
501 APos == atHub | onLink
498a atHub :: h_ui:H_UI
498b onLink :: fh_ui:H_UI × l_ui:L_UI × frac:Fract × th_ui:H_UI
498b Fract = Real, axiom frac:Fract • 0 < frac < 1

```

**value**

```

500 attr_RegNo: A → RegNo
501 attr_APos: A → APos

```

Obvious attributes that are not illustrated are those of velocity and acceleration, forward or backward movement, turning right, left or going straight, etc. The *acceleration*, *deceleration*, *even velocity*, or *turning right*, *turning left*, *moving straight*, or *forward* or *backward* are seen as *command actions*. As such they denote actions by the automobile — such as *pressing the accelerator*, or *lifting accelerator pressure* or *braking*, or *turning the wheel* in one direction or another, etc. As actions they have a kind of counterpart in the velocity, the acceleration, etc. attributes. Observe that bus companies each have their own distinct *bus time table*, and that these are modeled as *programmable*, Item 495 on the facing page, page 180. Observe then that buses each have their own distinct *bus time table*, and that these are modeled as *inert*, Item 497 on the preceding page, page 180. In Items pp. 178 and pp. 179, we illustrated an aspect of domain analysis & description that may seem, and at least some decades ago would have seemed, strange: namely that if we can think, hence speak, about it, then we can model it “as a fact” in the domain. The case in point is that we include among hub and link attributes their histories of the timed whereabouts of buses and automobiles.<sup>8</sup>

**E.3.3.7 Intentionality**

- 502. Seen from the point of view of an automobile there is its own traffic history, *A\_Hist*, which is a (time ordered) sequence of timed automobile’s positions;
- 503. seen from the point of view of a hub there is its own traffic history, *H\_Traffic* Item pp. 178, which is a (time ordered) sequence of timed maps from automobile identities into automobile positions; and
- 504. seen from the point of view of a link there is its own traffic history, *L\_Traffic* Item pp. 179, which is a (time ordered) sequence of timed maps from automobile identities into automobile positions.

The *intentional “pull”* of these manifestations is this:

- 505. The union, i.e. proper merge of all automobile traffic histories, *AllATH*, must now be identical to the same proper merge of all hub, *AllHTH*, and all link traffic histories, *AllLTH*.

**type**

```

502 A_Hi = (T × APos)*
489 H_Trf = A_UI ↗ (TIME × APos)*
493 L_Trf = A_UI ↗ (TIME × APos)*
505 AllATH = TIME ↗ (AUI ↗ APos)
505 AllHTH = TIME ↗ (AUI ↗ APos)
505 AllLTH = TIME ↗ (AUI ↗ APos)

```

**axiom**

```

505 let allA = mrg_AllATH({(a, attr_A_Hi(a)) | a: A • a ∈ a.s}),

```

<sup>8</sup>In this day and age of road cameras and satellite surveillance these traffic recordings may not appear so strange: We now know, at least in principle, of technologies that can record approximations to the hub and link traffic attributes.

```

505 allH=mrg_AllHTH({attr_H-Trf(h)|h:H•h ∈ hs}),
505 allL=mrg_AllLTH({attr_L-Trf(l)|l:L•h ∈ ls}) in
505 allA = mrg_HLT(allH,allL) end

```

We leave the definition of the four **merge** functions to the reader! We endow each automobile with its history of timed positions and each hub and link with their histories of timed automobile positions. These histories are facts! They are not something that is laboriously recorded, where such recordings may be imprecise or cumbersome<sup>9</sup>. The facts are there, so we can (but may not necessarily) talk about these histories as facts. It is in that sense that the purpose (‘**transport**’) for which man let automobiles, hubs and link be made with their ‘**transport**’ intent are subject to an *intentional “pull”*. *It can be no other way: if automobiles “record” their history, then hubs and links must together “record” identically the same history!*

**Intentional Pull – General Transport:** These are examples of human intents: they create *roads* and *automobiles* with the intent of *transport*, they create *houses* with the intents of *living*, *offices*, *production*, etc., and they create *pipelines* with the intent of *oil* or *gas transport* ■

## E.4 Perdurants

In this section we transcendently “morph” **parts** into **behaviours**. We analyse that notion and its constituent notions of **actors**, **channels** and **communication**, **actions** and **events**.

The main transcendental deduction of this chapter is that of associating with each part a behaviour. This section shows the details of that association. Perdurants are understood in terms of a notion of *state* and a notion of *time*.

**State Values versus State Variables:** Item 452 on page 174 expresses the **value** of all parts of a road transport system:

```
452. ps:(UoB|H|L|BC|B|A)-set ≡ rtsUhlsUbcUbsUas.
```

506. We now introduce the set of variables, one for each part value of the domain being modeled.

```
506. { variable vp:(UoB|H|L|BC|B|A) | vp:(UoB|H|L|BC|B|A) • vp∈ps }
```

**Buses and Bus Companies** A bus company is like a “root” for its fleet of “sibling” buses. But a bus company may cease to exist without the buses therefore necessarily also ceasing to exist. They may continue to operate, probably illegally, without, possibly, a valid bus driving certificate. Or they may be passed on to either private owners or to other bus companies. We use this example as a reason for not endowing a “block structure” concept on behaviours.

### E.4.1 Channels and Communication

#### E.4.1.1 Channel Message Types

We ascribe types to the messages offered on channels.

507. Hubs and links communicate, both ways, with one another, over channels, **hl\_ch**, whose indexes are determined by their mereologies.

508. Hubs send one kind of messages, links another.

509. Bus companies offer timed bus time tables to buses, one way.

510. Buses and automobiles offer their current, timed positions to the road element, hub or link they are on, one way.

<sup>9</sup>or thought technologically in-feasible – at least some decades ago!

**type**

```

508 H_L_Msg, L_H_Msg
507 HL_Msg = H_L_Msg | L_F_Msg
509 BC_B_Msg = T × BusTimTbl
510 V_R_Msg = T × (BPos|APos)

```

**E.4.1.2 Channel Declarations**

511. This justifies the channel declaration which is calculated to be:

**channel**

```

511 { hl_ch[h_ui,l_ui]:H_L_Msg
511 | h_ui:H_UI,l_ui:L_UI • i ∈ h_uis ∧ j ∈ lh_uim(h_ui) }
511 ∪
511 { hl_ch[h_ui,l_ui]:L_H_Msg
511 | h_ui:H_UI,l_ui:L_UI • l_ui ∈ l_uis ∧ i ∈ lh_uim(l_ui) }

```

We shall argue for bus company-to-bus channels based on the mereologies of those parts. Bus companies need communicate to all its buses, but not the buses of other bus companies. Buses of a bus company need communicate to their bus company, but not to other bus companies.

512. This justifies the channel declaration which is calculated to be:

**channel**

```

512 { bc_b_ch[bc_ui,b_ui] | bc_ui:BC_UI, b_ui:B_UI
512 • bc_ui ∈ bc_uis ∧ b_ui ∈ b_uis } : BC_B_Msg

```

We shall argue for vehicle to road element channels based on the mereologies of those parts. Buses and automobiles need communicate to all hubs and all links.

513. This justifies the channel declaration which is calculated to be:

**channel**

```

513 { v_r_ch[v_ui,r_ui] | v_ui:V_UI,r_ui:R_UI
513 • v_ui ∈ v_uis ∧ r_ui ∈ r_uis } : V_R_Msg

```

**E.4.2 Behaviours****E.4.2.1 Road Transport Behaviour Signatures**

We first decide on names of behaviours. In the translation schemas we gave schematic names to behaviours of the form  $\mathcal{M}_P$ . We now assign mnemonic names: from part names to names of transcendentally interpreted behaviours and then we assign signatures to these behaviours.

Hub Behaviour Signature

514.  $\text{hub}_{h_{ui}}$ :

- (a) there is the usual “triplet” of arguments: unique identifier, mereology and static attributes;
- (b) then there are the programmable attributes;
- (c) and finally there are the input/output channel references: first those allowing communication between hub and link behaviours,
- (d) and then those allowing communication between hub and vehicle (bus and automobile) behaviours.

**value**

514  $\text{hub}_{h_{ui}}$ :  
 514a  $h_{ui}:H\_UI \times (vuis, luis, \_):H\_Mer \times H\Omega$   
 514b  $\rightarrow (H\Sigma \times H\_Traffic)$   
 514c  $\rightarrow \mathbf{in, out} \{ h\_l\_ch[h_{ui}, l_{ui}] \mid l_{ui}:L\_UI \bullet l_{ui} \in luis \}$   
 514d  $\{ ba\_r\_ch[h_{ui}, v_{ui}] \mid v_{ui}:V\_UI \bullet v_{ui} \in vuis \}$  **Unit**  
 514a **pre:**  $vuis = v_{uis} \wedge luis = l_{uis}$

## Link Behaviour Signature

515.  $\text{link}_{l_{ui}}$ :

- (a) there is the usual “triplet” of arguments: unique identifier, mereology and static attributes;
- (b) then there are the programmable attributes;
- (c) and finally there are the input/output channel references: first those allowing communication between hub and link behaviours,
- (d) and then those allowing communication between link and vehicle (bus and automobile) behaviours.

**value**

515  $\text{link}_{l_{ui}}$ :  
 515a  $l_{ui}:L\_UI \times (vuis, huis, \_):L\_Mer \times L\Omega$   
 515b  $\rightarrow (L\Sigma \times L\_Traffic)$   
 515c  $\rightarrow \mathbf{in, out} \{ h\_l\_ch[h_{ui}, l_{ui}] \mid h_{ui}:H\_UI: h_{ui} \in huis \}$   
 515d  $\{ ba\_r\_ch[l_{ui}, v_{ui}] \mid v_{ui}:(B\_UI|A\_UI) \bullet v_{ui} \in vuis \}$  **Unit**  
 515a **pre:**  $vuis = v_{uis} \wedge huis = h_{uis}$

## Bus Company Behaviour Signature

516.  $\text{bus\_company}_{bc_{ui}}$ :

- (a) there is here just a “doublet” of arguments: unique identifier and mereology;
- (b) then there is the one programmable attribute;
- (c) and finally there are the input/output channel references allowing communication between the bus company and buses.

**value**

516  $\text{bus\_company}_{bc_{ui}}$ :  
 516a  $bc_{ui}:BC\_UI \times (\_, \_, \text{buis}):BC\_Mer$   
 516b  $\rightarrow \text{BusTimTbl}$   
 516c  $\mathbf{in, out} \{ bc\_b\_ch[bc_{ui}, b_{ui}] \mid b_{ui}:B\_UI \bullet b_{ui} \in buis \}$  **Unit**  
 516a **pre:**  $buis = b_{uis} \wedge huis = h_{uis}$

## Bus Behaviour Signature

517.  $\text{bus}_{b_{ui}}$ :

- (a) there is here just a “doublet” of arguments: unique identifier and mereology;
- (b) then there are the programmable attributes;
- (c) and finally there are the input/output channel references: first the input/output allowing communication between the bus company and buses,
- (d) and the input/output allowing communication between the bus and the hub and link behaviours.

**value**

```

517 busbui:
517a bui:B_UI × (bcui,_,ruis):B_Mer
517b → (LN × BTT × BPOS)
517c → out bcb_ch[bcui,bui],
517d {bar_ch[rui,bui]|rui:(H_UI|L_UI)•ui∈vuis} Unit
517a pre: ruis = ruis ∧ bcui ∈ bcuis

```

Automobile Behaviour Signature

518. automobile<sub>a<sub>ui</sub></sub>:

- (a) there is the usual “triplet” of arguments: unique identifier, mereology and static attributes;
- (b) then there is the one programmable attribute;
- (c) and finally there are the input/output channel references allowing communication between the automobile and the hub and link behaviours.

**value**

```

518 automobileaui:
518a aui:A_UI × (__,_,ruis):A_Mer × rn:RegNo
518b → apos:APos
518c in,out {bar_ch[aui,rui]|rui:(H_UI|L_UI)•rui∈ruis} Unit
518a pre: ruis = ruis ∧ aui ∈ auis ■

```

**E.4.2.2 Behaviour Definitions**

We only illustrate automobile, hub and link behaviours.

**E.4.2.2.1 Automobile Behaviour at a Hub** We define the behaviours in a different order than the treatment of their signatures. We “split” definition of the **automobile** behaviour into the behaviour of **automobiles** when positioned at a hub, and into the behaviour **automobiles** when positioned at on a link. In both cases the behaviours include the “idling” of the automobile, i.e., its “not moving”, standing still.

519. We abstract automobile behaviour at a Hub (hui).

520. The vehicle remains at that hub, “idling”,

521. informing the hub behaviour,

522. or, internally non-deterministically,

- (a) moves onto a link, tli, whose “next” hub, identified by th<sub>ui</sub>, is obtained from the mereology of the link identified by tl<sub>ui</sub>;
- (b) informs the hub it is leaving and the link it is entering of its initial link position,
- (c) whereupon the vehicle resumes the vehicle behaviour positioned at the very beginning (0) of that link,

523. or, again internally non-deterministically,

524. the vehicle “disappears — off the radar” !

```

519 automobileaui(aui,({},(ruis,vuis),{}),rn)
519 (apos:atH(flui,hui,tlui)) ≡
520 (bar_ch[aui,hui] ! (recordTIMEatH(flui,hui,tlui)));
521 automobileaui(aui,({},(ruis,vuis),{}),rn)(apos)
522 []
522a (let ({fhui,thui},ruis')=mereo.L(ϕ(tlui)) in
522a assert: fhui=hui ∧ ruis=ruis'
519 let onl = (tlui,hui,0,thui) in
522b (bar_ch[aui,hui] ! (recordTIMEonL(onl)) ||
522b bar_ch[aui,tlui] ! (recordTIMEonL(onl))) ;
522c automobileaui(aui,({},(ruis,vuis),{}),rn)
522c (onL(onl)) end end)
523 []
524 stop

```

#### E.4.2.2.2 Automobile Behaviour On a Link

525. We abstract automobile behaviour on a Link.

- (a) Internally non-deterministically, either
  - i. the automobile remains, “idling”, i.e., not moving, on the link,
  - ii. however, first informing the link of its position,
- (b) or
  - i. **if** if the automobile’s position on the link *has not yet reached the hub*, **then**
    - A. then the automobile moves an arbitrary small, positive **Real**-valued *increment* along the link
    - B. informing the hub of this,
    - C. while resuming being an automobile ate the new position, or
  - ii. **else**,
    - A. while obtaining a “next link” from the mereology of the hub (where that next link could very well be the same as the link the vehicle is about to leave),
    - B. the vehicle informs both the link and the imminent hub that it is now at that hub, identified by **th<sub>ui</sub>**,
    - C. whereupon the vehicle resumes the vehicle behaviour positioned at that hub;
- (c) or
- (d) the vehicle “disappears — off the radar” !

```

525 automobileaui(aui,({},ruis,{}),rno)
525 (vp:onL(fhui,lui,f,thui)) ≡
525(a)ii (bar_ch[thui,aui]!atH(lui,thui,nextlui) ;
525(a)i automobileaui(aui,({},ruis,{}),rno)(vp)
525b []
525(b)i (if notyet_athub(f)
525(b)i then
525(b)iA (let incr = increment(f) in
519 let onl = (tlui,hui,incr,thui) in
525(b)iB bar_ch[lui,aui] ! onL(onl) ;
525(b)iC automobileaui(aui,({},ruis,{}),rno)
525(b)iC (onL(onl))
525(b)i end end)
525(b)ii else

```



```

525(b)iiA (let nxt_lui:L_UI•nxt_lui ∈ mereo-H(\wp (th_ui)) in
525(b)iiB ba_r_ch[thui,auil]atH(l_ui,th_ui,nxt_lui) ;
525(b)iiC automobileaui(a_ui,({},ruis,{}),rno)
525(b)iiC (atH(l_ui,th_ui,nxt_lui)) end)
525(b)i end)
525c []
525d stop
525(b)iA increment: Fract → Fract

```

#### E.4.2.2.3 Hub Behaviour

526. The hub behaviour

- (a) non-deterministically, externally offers
- (b) to accept timed vehicle positions —
- (c) which will be at the hub, from some vehicle,  $v_{ui}$ .
- (d) The timed vehicle hub position is appended to the front of that vehicle's entry in the hub's traffic table;
- (e) whereupon the hub proceeds as a hub behaviour with the updated hub traffic table.
- (f) The hub behaviour offers to accept from any vehicle.
- (g) A **post** condition expresses what is really a **proof obligation**: that the hub traffic,  $ht'$  satisfies the **axiom** of the enduring hub traffic attribute Item pp. 178.

#### value

```

526 hubhui(h_ui,(luis,vuis),h ω)(h σ ,ht) ≡
526a []
526b { let m = ba_r_ch[h_ui,v_ui] ? in
526c assert: m=(_,atHub(_,h_ui,_))
526d let ht' = ht † [h_ui ↦ ⟨m⟩^ht(h_ui)] in
526e hubhui(h_ui,(luis,vuis),(h ω))(h σ ,ht')
526f | v_ui:V_UI•v_ui∈vuis end end }
526g post: $\forall v_{ui}:V_UI \bullet v_{ui} \in \mathbf{dom} \ ht' \Rightarrow \mathbf{time_ordered}(ht'(v_{ui}))$

```

#### E.4.2.2.4 Link Behaviour

527. The link behaviour non-deterministically, externally offers

528. to accept timed vehicle positions —

529. which will be on the link, from some vehicle,  $v_{ui}$ .

530. The timed vehicle link position is appended to the front of that vehicle's entry in the link's traffic table;

531. whereupon the link proceeds as a link behaviour with the updated link traffic table.

532. The link behaviour offers to accept from any vehicle.

533. A **post** condition expresses what is really a **proof obligation**: that the link traffic,  $lt'$  satisfies the **axiom** of the enduring link traffic attribute Item pp. 179.

```

527 linklui(lui,(⟦, (huis,vuis),⟦),lω)(lσ,lt) ≡
527 ||
528 { let m = ba_r_ch[lui,vui] ? in
529 assert: m=(⟦,onLink(⟦, lui,⟦))
530 let lt' = lt † [lui ↦ ⟨m⟩^lt(lui)] in
531 linklui(lui,(huis,vuis),hω)(hσ,lt')
532 | vui:V_UI•vui∈vuis end end }
533 post: ∀ vui:V_UI•vui ∈ dom lt' ⇒ time_ordered(lt'(vui))

```

## E.5 System Initialisation

### E.5.1 Initial States

value

```

hs:H-set ≡ ≡ obs_sH(obs_SH(obs_RN(rts)))
ls:L-set ≡ ≡ obs_sL(obs_SL(obs_RN(rts)))
bcs:BC-set ≡ obs_BCs(obs_SBC(obs_FV(obs_RN(rts))))
bs:B-set ≡ ∪{obs_Bs(bc)|bc:BC•bc ∈ bcs}
as:A-set ≡ obs_BCs(obs_SBC(obs_FV(obs_RN(rts))))

```

### E.5.2 Initialisation

We are reaching the end of this domain modeling example. Behind us there are narratives and formalisations. Based on these we now express the signature and the body of the definition of a “system build and execute” function.

534. The system to be initialised is

- the parallel compositions (||) of
- the distributed parallel composition (||{...|...}) of all hub behaviours,
- the distributed parallel composition (||{...|...}) of all link behaviours,
- the distributed parallel composition (||{...|...}) of all bus company behaviours,
- the distributed parallel composition (||{...|...}) of all bus behaviours, and
- the distributed parallel composition (||{...|...}) of all automobile behaviours.

value

```

534 initial_system: Unit → Unit
534 initial_system() ≡
534b || { hubhui(hui,me,hω)(htrf,hσ)
534b | h:H•h ∈ hs, hui:H_UI•hui=uid_H(h), me:HMetL•me=mereo_H(h),
534b htrf:H_Traffic•htrf=attr_H_Traffic_H(h),
534b hω:HΩ•hω=attr_HΩ(h), hσ:HΣ•hσ=attr_HΣ(h) ∧ hσ ∈ hω }
534a ||
534c || { linklui(lui,me,lω)(ltrf,lσ)
534c | l:L•l ∈ ls, lui:L_UI•lui=uid_L(l), me:LMet•me=mereo_L(l),
534c ltrf:L_Traffic•ltrf=attr_L_Traffic_H(l),
534c lω:LΩ•lω=attr_LΩ(l), lσ:LΣ•lσ=attr_LΣ(l) ∧ lσ ∈ lω }
534a ||
534d || { bus_companybcui(bcui,me)(btt)
534d | bc:BC•bc ∈ bcs, bcui:BC_UI•bcui=uid_BC(bc), me:BCMet•me=mereo_BC(bc),
534d btt:BusTimTbl•btt=attr_BusTimTbl(bc) }

```

```

534a ||
534e || { busbui(bui,me)(ln,btt,bpos)
534e || b:B•b ∈ bs, bui:B_UI•bui=uid_B(b), me:BMet•me=mereo_B(b), ln:LN:pln=attr_LN(b),
534e || btt:BusTimTbl•btt=attr_BusTimTbl(b), bpos:BPos•bpos=attr_BPos(b) }
534a ||
534f || { automobileaui(aui,me,rn)(apos)
534f || a:A•a ∈ as, aui:A_UI•aui=uid_A(a), me:AMet•me=mereo_A(a),
534f || rn:RegNo•rn=attr_RegNo(a), apos:APos•apos=attr_APos(a) } ■

```



# Appendix F

## Rail Systems

### Contents

---

|             |                                         |            |
|-------------|-----------------------------------------|------------|
| <b>F.1</b>  | <b>Endurants – Rail Nets and Trains</b> | <b>192</b> |
| F.1.1       | External Qualities                      | 192        |
| F.1.1.1     | Rail Nets                               | 192        |
| F.1.1.1.1   | The Endurants                           | 192        |
| F.1.1.1.2   | All Net Units                           | 193        |
| F.1.1.2     | Trains                                  | 193        |
| F.1.1.2.1   | The Endurants                           | 193        |
| F.1.1.2.2   | All Trains                              | 194        |
| F.1.2       | Internal Qualities                      | 194        |
| F.1.2.1     | Unique Identifiers                      | 194        |
| F.1.2.1.1   | Rail Units                              | 194        |
| F.1.2.1.2   | All Net Unit Unique Identifiers         | 194        |
| F.1.2.1.3   | Trains                                  | 194        |
| F.1.2.1.4   | Retrieve Net Units                      | 195        |
| F.1.2.2     | Mereology                               | 195        |
| F.1.2.2.1   | Rail Units                              | 195        |
| F.1.2.2.2   | Well-formed Mereologies                 | 196        |
| F.1.2.2.3   | Trains                                  | 196        |
| F.1.2.2.4   | Routes                                  | 196        |
| F.1.2.2.4.1 | Route Types                             | 196        |
| F.1.2.2.4.2 | Initial Routes                          | 197        |
| F.1.2.2.4.3 | Next Route Elements                     | 197        |
| F.1.2.2.4.4 | Previous Route Elements                 | 197        |
| F.1.2.2.4.5 | All Routes                              | 198        |
| F.1.2.2.4.6 | Isolated Rail Net Units                 | 198        |
| F.1.2.2.4.7 | A Delineation: Train Stations           | 198        |
| F.1.2.2.4.8 | All Stations of a Railway System        | 199        |
| F.1.2.2.4.9 | Rail Lines                              | 199        |
| F.1.2.3     | Attributes                              | 200        |
| F.1.2.3.1   | Rail Nets                               | 200        |
| F.1.2.3.2   | Open Routes                             | 202        |
| F.1.2.3.3   | Station Names                           | 202        |
| F.1.2.3.4   | Trains                                  | 202        |
| F.1.2.3.5   | An Intentional Pull                     | 203        |

|            |                                                 |            |
|------------|-------------------------------------------------|------------|
| F.1.2.3.6  | <b>History Attributes</b> . . . . .             | 203        |
| F.1.2.3.7  | <b>The Intentional Pull Revisited</b> . . . . . | 204        |
| <b>F.2</b> | <b>Transcendental Deduction</b> . . . . .       | <b>204</b> |
| F.2.1      | <b>General</b> . . . . .                        | 204        |
| F.2.2      | <b>A Note on TIME</b> . . . . .                 | 205        |
| F.2.3      | <b>Train Traffic</b> . . . . .                  | 205        |
| F.2.3.1    | <b>Well-formed Train Traffics</b> . . . . .     | 205        |
| <b>F.3</b> | <b>Perdurants</b> . . . . .                     | <b>206</b> |
| F.3.1      | <b>Channels</b> . . . . .                       | 207        |
| F.3.2      | <b>Behaviour Signatures</b> . . . . .           | 207        |
| F.3.3      | <b>Behaviour Definitions</b> . . . . .          | 208        |
| F.3.3.1    | <b>Rail Unit Behaviours</b> . . . . .           | 208        |
| F.3.3.2    | <b>Train Behaviour</b> . . . . .                | 208        |
| <b>F.4</b> | <b>Closing</b> . . . . .                        | <b>209</b> |

This model evolved over many years. A first, beautiful model was developed in 1993 by the late Søren Prehn<sup>1</sup>. Over the years variations of this model went into several papers [20, 22–25, 69, 75, 76, 149, 150, 176]. We refer to *Railways* – a compendium [imm.dtu.dk/~dibj/train-book.pdf](http://imm.dtu.dk/~dibj/train-book.pdf). The current model is a complete rewrite of earlier models. These earlier models were not based on the enduring/perdurant, the atomic/compound [set and composite] externalities and the unique identifier, mereology and attribute paradigms. The present model is.

The example is quite extensive. Anything smaller really makes no sense: does not bring across the issues of what it takes to describe a domain nor the scale of domain descriptions.

The example is that of a railway system’s net of rail units and trains.

## F.1 Endurants – Rail Nets and Trains

### F.1.1 External Qualities

#### F.1.1.1 Rail Nets

##### F.1.1.1.1 The Endurants

535. The example is that of a railway system.

536. We focus on the railway net [and, later, trains]. They can be observed from the railway system.

537. The railway net embodies a set of [railway] net units.

538. A net unit is either a straight or curved **linear** unit, or a simple switch, i.e., a **turnout**, unit<sup>2</sup> or a simple cross-over, i.e., a **rigid** crossing unit, or a single switched cross-over, i.e., a **single** slip unit, or a double switched cross-over, i.e., a **double** slip unit, or a **terminal** unit.

We refer to Figure F.1 on the next page.

#### type

535. RS

536. RN

#### value

<sup>1</sup>Søren Prehn was a brilliant student of mine 1975–1980. He became a leading member of Dansk Datamatik Center [70], and later the CR company in Denmark, from 1980 onward. He spent a 2 year sabbatical from CR with me at the UNU/IIST, the United Nations International Institute for Software Technology in Macau, 1992–1994. Sadly he passed away in the spring of 2006.

<sup>2</sup>[https://en.wikipedia.org/wiki/Railroad\\_switch](https://en.wikipedia.org/wiki/Railroad_switch)

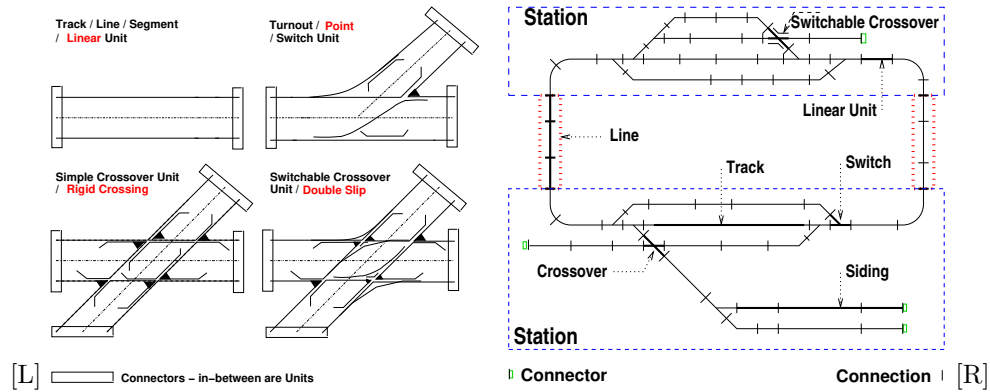


Figure F.1: Left: Four net units; Right: A railway net

536.  $\text{obs\_RN}: \text{RS} \rightarrow \text{RN}$

**type**

537.  $\text{NUs} = \text{NU-set}$

537.  $\text{NU} = \text{LU} \mid \text{PU} \mid \text{RU} \mid \text{SU} \mid \text{DU} \mid \text{TU}$

**value**

538.  $\text{obs\_NUs}: \text{RN} \rightarrow \text{NU-set}$

### F.1.1.1.2 All Net Units

539. From a railway system net one can observe, i.e., extract, all the rail net units.

540. We let  $rs$  denote the value of of an arbitrary chosen railway system,

541. and we let  $nus$  denote the value of the set of all railway units.

**value**

539.  $\text{xtr\_NUs}: \text{RS} \rightarrow \text{NU-set}$

539.  $\text{xtr\_NUs}(rs) \equiv \text{obs\_NUs}(\text{obs\_RN}(rs))$

540.  $rs: \text{RS}$

541.  $nus = \text{obs\_NUs}(rs)$

### F.1.1.2 Trains

#### F.1.1.2.1 The Endurants

542. We shall, simplifying, consider trains as atomic parts.

543. From a railway system one can observe a finite, let us decide, non-empty set of trains.

**type**

542.  $\text{Train}$

543.  $\text{TS} = \text{Train-set}$

**value**

543.  $\text{obs\_TS}: \text{RS} \rightarrow \text{TS}$

**axiom**

543.  $\forall rs: \text{RS} \bullet \text{obs\_TS}(rs) \neq \{\}$

### F.1.1.2.2 All Trains

544. We let *trains* denote the value of the set of all trains.

#### value

544.  $trains = \{ t \mid t:Train \bullet obs\_TS(rs) \}$

## F.1.2 Internal Qualities

### F.1.2.1 Unique Identifiers

#### F.1.2.1.1 Rail Units

545. With every rail net unit we associate a unique identifier.

546. That is, no two rail net units have the same unique identifier.

#### type

545. UI

#### value

545.  $uid\_NU: NU \rightarrow UI$

#### axiom

546.  $\forall ui\_i, ui\_j: UI \bullet ui\_i = ui\_j \equiv uid\_NU(ui\_i) = uid\_NU(ui\_j)$

#### F.1.2.1.2 All Net Unit Unique Identifiers

547. From a railway system net one can observe, i.e., extract, the set of all the unique rail unit identifiers of all the rail net units.

548. We let *uis* denote the set of all railway units of the arbitrarily chosen railway system cum railway net.

#### value

547.  $xtr\_UIs: RS \rightarrow UI\text{-set}$

547.  $xtr\_UIs(rs) \equiv \{ uid\_NU(nu) \mid nu:NU \bullet nu \in obs\_NUs(obs\_RN(rs)) \}$

548.  $uis = xtr\_UIs(rs)$

#### F.1.2.1.3 Trains

549. Trains have unique identifiers.

550. We let *tris* denote the set of all train identifiers.

551. No two distinct trains have the same unique identifier.

552. Train identifiers are distinct from rail net unit identifiers.

#### type

549. TI

#### value

549.  $uid\_Train: Train \rightarrow TI$

550.  $tris = \{ uid\_Train(t) \mid t:Train \bullet t \in trains \}$

#### axiom

551. either:  $card\ trains = card\ tris$

551. or:  $\forall rs:RS \bullet$

551.  $\quad \forall train\_a, train\_b:Train \bullet \{train\_a, train\_b\} \subseteq obs\_TS(rs) \Rightarrow$

551.  $\quad train\_a \neq train\_b \Rightarrow uid\_Train(train\_a) \neq uid\_Train(train\_b)$

552.  $uis \cap tris = \{ \}$



**F.1.2.1.4 Retrieve Net Units**

553. Given a net unit unique identifier and a railway net one can retrieve the net unit with that identifier.

**value**

553.  $\text{retr\_NU}: \text{UI} \rightarrow \text{RS} \xrightarrow{\sim} \text{NU}$

553.  $\text{retr\_NU}(ui)(rs) \equiv \text{let } nu:\text{NU} \bullet nu \in \text{xtr\_NUs}(rs) \wedge \text{uid\_NU}(nu)=ui \text{ in } nu \text{ end}$

553. **pre:**  $ui \in \text{xtr\_UIs}(rs)$

**F.1.2.2 Mereology**

**F.1.2.2.1 Rail Units** The mereology of a rail net unit expresses its topological relation to other rail net units and trains.

- 554. Every rail unit is conceptually related to every train.
- 555. A linear rail unit is connected to exactly two distinct other rail net units of any given rail net.
- 556. A point unit is connected to exactly three distinct other rail net units of any given rail net.
- 557. A rigid crossing unit is connected to exactly four distinct other rail net units of any given rail net.
- 558. A single and a double slip unit is connected to exactly four distinct other rail net units of any given rail net.
- 559. A terminal unit is connected to exactly one distinct other rail net unit of any given rail net.
- 560. So we model the mereology of a railway net unit as a pair of sets of rail net unit unique identifiers distinct from that of the rail net unit.
- 561. Trains can run on every rail unit of any rail system.

|                                                                          |                                                                                |                                                                                      |                                                                                      |
|--------------------------------------------------------------------------|--------------------------------------------------------------------------------|--------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------|
| <p><b>Linear</b></p>                                                     | <p><b>Point</b></p>                                                            | <p><b>Rigid Crossing</b></p>                                                         | <p><b>Double Slip</b></p>                                                            |
| <p><math>\{\{ua\},\{ux\}\}</math><br/><math>\{\{ux\},\{ua\}\}</math></p> | <p><math>\{\{ua\},\{ux,uy\}\}</math><br/><math>\{\{ux,uy\},\{ua\}\}</math></p> | <p><math>\{\{ua,ub\},\{ux,uy\}\}</math><br/><math>\{\{ux,uy\},\{ua,ub\}\}</math></p> | <p><math>\{\{ua,ub\},\{ux,uy\}\}</math><br/><math>\{\{ux,uy\},\{ua,ub\}\}</math></p> |

Figure F.2: Four Symmetric Mereologies

**type**

560.  $\text{Unit\_Mereo} = (\text{UI-set} \times \text{UI-set}) \times \text{TI-set}$

**value**

560.  $\text{mereo\_NU}: \text{NU} \rightarrow \text{Unit\_Mereo}$

**axiom**

560.  $\forall nu:\text{NU} \bullet$

560. **let**  $((uis_i,uis_o),tris)=\text{mereo\_NU}(nu)$  **in**

554.  $tris = tris \wedge$

560. **case**  $(\text{card } uis_i, \text{card } uis_o) =$

555.  $(is\_LU(nu) \rightarrow (1,1),$

```

556. is_PU(nu) → (1,2) ∨ (2,1),
557. is_RU(nu) → (2,2),
558. is_SU(nu) → (2,2), is_DU(nu) → (2,2),
559. is_TU(nu) → (1,0) ∨ (0,1),
560. __ → chaos end
560. ∧ uis_i ∩ uis_o = {}
560. ∧ uid_NU(nu) ∉ (uis_i ∪ uis_o)
560. end

```

#### F.1.2.2.2 Well-formed Mereologies

562. The unique identifiers of any rail unit mereology of a rail net must be of rail units of that net and
563. the set of train identifiers of any rail unit mereology of a rail net must be the set of all train identifiers of that railway system.

#### value

```

562. wf_Mereology: RS → Bool
562. wf_Mereology(rs) ≡
562. let (nus,uis) = (xtr_NUs,xtr_UIs)(rs) in
562. ∃ nu:NU • nu ∈ nus •
562. let ui = uid_NU(nu), ((iuis,ouis),tris) = mereo_NU(nu) in
562. ui ∉ iuis ∪ ouis ∧ iuis ∩ ouis = {} ∧ iuis ∪ ouis ⊆ uis
563. ∧ tris = tris
562. end end

```

#### F.1.2.2.3 Trains

564. Trains can run on every rail unit of any rail system.

We omit consideration of trains communicating with other trains as well as with net management. We leave such “completions” to the reader.

#### type

564. Train\_Mereo = UI-set

#### value

564. mereo\_Train\_Mereo: Train → Train\_Mereo

#### axiom

564. ∃ rs:RS • ∃ train:Train

564. ∃ train:Train • train ∈ obs\_TS(rs) ⇒ mereo\_Train\_Mereo(train) = retr\_UIs(rs)

**F.1.2.2.4 Routes** We decompose the analysis into several preparatory steps.

#### F.1.2.2.4.1 Route Types

565. A route is a finite or infinite sequence of one or more route elements.
566. A route element is a [route] triple of three distinct net unit identifiers, the net unit identifier of an immediately preceding rail unit, the net unit identifier of the present rail unit, the net unit identifier of an immediately succeeding rail unit, irrespective of whether the preceding and succeeding units are actually in the route as analysed.

**type**565.  $R = TUI^\omega$ 566.  $TUI = UI \times UI \times UI$ **axiom**566.  $\forall (pui, ui, sui): TUI \bullet \text{card}\{pui, ui, sui\} = 3$ 565.  $\forall r: R \bullet \forall i: \mathbf{Nat} \bullet \{i, i+1\} \subseteq \text{inds } r \Rightarrow$ 565. **let**  $(pui, ui, sui) = r[i]$ ,  $(pui', ui', sui') = r[i+1]$  **in**565.  $sui = pui' \wedge ui \neq ui' \wedge pui \neq \dots$  **end****F.1.2.2.4.2 Initial Routes**

567. We define an auxiliary function which, for any given railway system, calculates the finite set of all its initial routes – where an initial route is a one element route triplet of a non-terminal net unit.

**value**567.  $\text{initial\_routes}: RS \rightarrow R\text{-set}$ 567.  $\text{initial\_routes}(rs) \equiv$ 567. **let**  $(nus, uis) = (\text{retr\_NUs}, \text{retr\_UIs})(rs)$  **in**567.  $\{ \langle (pui, ui, sui) \rangle, \langle (sui, ui, pui) \rangle$ 567.  $\mid nu: NU \bullet nu \in nus \wedge \sim is\_TU(nu) \wedge$ 567.  $\text{let } (ui, (puis, suis)) = (\text{uid\_NU}, \text{mereo\_NU})(nu)$  **in**567.  $pui \in puis \wedge sui \in suis$  **end** }567. **assert:** [ there are up to eight triplets in the above set ]567. **end****F.1.2.2.4.3 Next Route Elements**

568. Give a route element, i.e., a triplet  $(pui, ui, sui)$ , one can calculate the set of one or two next route triplet designating the net unit with identifier  $sui$ .

**value**568.  $\text{next\_route\_elements}: TUI \rightarrow RS \rightarrow R\text{-set}$ 568.  $\text{next\_route\_elements}(\_, ui, sui)(rs) \equiv$ 568. **let**  $(puis \cup \{ui\}, suis) = \text{mereo\_NU}(\text{retr\_NU}(sui)(rs))$  **in**568.  $\{ \langle (pui, \text{uid\_NU}(\text{retr\_NU}(sui)(rs)), sui') \rangle \mid pui: UI \bullet pui \in puis \wedge sui' \in suis \}$ 568. **assert:** [ there are either one or two triplets in the set above. ]568. **end****F.1.2.2.4.4 Previous Route Elements**

569. Give a route element, i.e., a triplet  $(pui, ui, sui)$ , one can calculate the set of one or two previous route triplet designating the net unit with identifier  $sui$ .

**value**569.  $\text{previous\_route\_elements}: TUI \rightarrow RS \rightarrow R\text{-set}$ 569.  $\text{previous\_route\_elements}(pui, ui, \_)(rs) \equiv$ 569. **let**  $(puis, suis \cup \{ui\}) = \text{mereo\_NU}(\text{retr\_NU}(pui)(rs))$  **in**569.  $\{ \langle (pui', \text{uid\_NU}(\text{retr\_NU}(pui)(rs)), pui) \rangle \mid pui' \in puis \cup suis \}$ 569. **assert:** [ there are either one or two triplets in the set above ]569. **end**

**F.1.2.2.4.5 All Routes**

570. A route is a finite or infinite sequence of triplets.
571. The analysis function `routes` calculates a potentially infinite set of routes.
572. The set `rs` is recursively defined.  
It is the smallest set, i.e., fix-point, satisfying the equation.  
`rs` is initialised, i.e., the base step, with the set of initial routes of the railway system.
573. The induction step (573–576) ”adds”
574. `next`, `nr`, and
575. `previous`, `pr`, triplets
576. to an arbitrarily selected route (so far calculated).
577. The  $\widehat{pr} \widehat{udr} \widehat{nr}$  element of formula line 573 need not be included as it will be calculated in some subsequent recursion.

**value**

571. `routes`:  $RS \rightarrow R\text{-inset}$
571. `routes(rs)  $\equiv$`
572.     **let** `all_routes` = `irs`  $\cup$
573.          $\{ \widehat{udr} \widehat{nr}, \widehat{pr} \widehat{udr}, \widehat{pr} \widehat{udr} \widehat{nr}$
574.              $| nr \in \text{next\_route\_elements}(\widehat{udr}[\text{len } \widehat{udr}])(rs) \wedge$
575.              $pr \in \text{previous\_route\_elements}(\widehat{udr}[1])(rs) \wedge$
576.              $\widehat{udr}:R \bullet \widehat{udr} \in \text{all\_routes} \}$  **end**

**F.1.2.2.4.6 Isolated Rail Net Units** We wish to analyse a rail net for the following property: can one reach every rail unit from any given rail unit? The analysis function `isolated` decides on that!

578. Given two distinct net unit identifiers,  $ui'$  and  $ui''$ , of a railway net,  $ui''$  is isolated from  $ui'$  if there is no route in the railway net from  $ui'$  to  $ui''$ .

**value**

578. `isolated`:  $UI \times UI \rightarrow RS \rightarrow \mathbf{Bool}$
578. `isolated(uif,uit)(rs)  $\equiv$`
578.     **let** `all_routes` = `routes(rs)` **in**
578.      $\sim \exists r:\text{Route} \bullet r \in \text{all\_routes} \Rightarrow \exists i,j:\mathbf{Nat} \bullet \{i,j\} \subseteq \text{inds } r \wedge i < j \wedge r(i) = (\_, uif, \_) \wedge r(j) = (\_, uit, \_)$  **end**
578.     **pre**  $\{uif, uit\} \subseteq \text{xtr\_UIs}(rs)$

**F.1.2.2.4.7 A Delineation: Train Stations** In preparation for our later introduction of a notion of trains we shall attempt to delineate a notion of train station. By a train station we shall understand a largest set of connected rail units all designated as being in that station.

579. We shall therefore, presently, introduce a predicate: `in_station` that applies to a rail unit and yields **true** if it is a designated train station, **false** otherwise.
580. Based on a rail unit,  $nu$ , that satisfies `in_station`, i.e., `in_station(nu)` and on the mereology of stations, i.e., the connected rail units, beginning with  $nu$ , we define an analysis function which calculates the “full” station from  $nu$ .

581. Finally we define an analysis function `station` which, given a station rail unit calculates the largest set of rail units belonging to the same station.

**type**

581. `Station = NU-set`

**value**

579. `in_station: NU → Bool`

**axiom**

581.  $\forall st:Station, \forall nu:NU \bullet nu \in st \Rightarrow in\_station(nu)$

**value**

580. `station: NU → RS → NU-set`

580. `station(inu)(rs) ≡`

580.     **let** `st = {inu} ∪`

580.         `{ nu |`

580.             `stnu:NU • stnu ∈ st ∧`

580.             **let** `(iuis,ouis) = mereo_NU(stnu) in`

580.             **let** `cnus = { get_NU(ui)(rs) | ui:UI • ui ∈ iuis ∪ ouis } in`

580.             `nu ∈ cnus ∧ in_station(nu) end end }`

580.     **in st end**

580.     **pre:** `in_station(nu)`

How we may determine whether a rail unit is a station is left undefined. That is, we refrain from any (speculation) as to whether stations can be characterised by certain topological features of rail unit connections.

#### F.1.2.2.4.8 All Stations of a Railway System

582. We define an analysis function, `all_stations`, which calculates, from a railway system its set of two or more stations.

583. We calculate, `snus`, the set of all station rail units.

584. For each of these we calculate the station to which these station rail units belong.

**value**

582. `all_stations: RS → Station-set`

582. `all_stations(rs) ≡`

583.     **let** `snus = { nu | nu:NU • nu ∈ xtr_NUs(rs) ∧ in_station(nu) } in`

584.     `{ station(nu)(rs) | nu:NU • nu ∈ snus } end`

**axiom**

582. **card** `all_stations(rs) ≥ 2`

Two or more rail units, `nu`, of line 584 may calculate the same station.

#### F.1.2.2.4.9 Rail Lines

585. By a trail line we mean a route that connects two neighbouring stations.

586. `is_connected_stations`: Given two stations it may be that there are no routes connecting them.

587. `connecting_line`: We can calculate a line, `ln`, that does connect two connected stations.

Given two stations that are connected there will be a number of rail units in both stations that can serve as end points of their connecting rail line. We would then say that these end point rail units designate respective station platforms from and to where trains depart, respectively arrive.

588. `is_immediately_connecting_line`: We can inquire as to whether there is an immediately connecting line between two given stations of a railway system.

**type**

585. `LN = R`

**axiom**

585.  $\forall rs:RS \bullet \forall ln:LN \bullet ln \in routes(rs) \Rightarrow$   
 585.     **let**  $(\_,1ui,\_) = hd\ ln, (\_,nui,\_) = ln[ len\ ln ]$  **in**  
 585.     **let**  $1nu = get\_NU(1ui)(rs), nnu = get\_NU(nnu)(rs)$  **in**  
 585.      $in\_station(1nu) \wedge in\_station(nni)$  **end end**

**value**

586. `is_connected_stations`: `Station`  $\times$  `Station`  $\rightarrow$  `RS`  $\rightarrow$  **Bool**  
 586. `is_connected_stations(fs,ts)(rs)  $\equiv$`   
 586.     **let** `all_routes = routes(rs)` **in**  
 586.      $\exists ln:R \bullet ln \in all\_routes \bullet$   
 585.         **let**  $(\_,1ui,\_) = hd\ ln, (\_,nui,\_) = ln[ len\ ln ]$  **in**  
 585.         **let**  $1nu = get\_NU(1ui)(rs), nnu = get\_NU(nnu)(rs)$  **in**  
 585.          $fs = 1nu \wedge ts = nnu$  **end end**  
 586.     **end**  
 586.     **pre**:  $\{fs,ts\} \subseteq all\_stations(rs)$

587. `connecting_line`: `Station`  $\times$  `Station`  $\rightarrow$  `RS`  $\rightarrow$  `LN`  
 587. `connecting_line(fs,ts)(rs)  $\equiv$`   
 586.     **let** `all_routes = routes(rs)` **in**  
 587.     **let**  $ln:R \bullet ln \in all\_routes \bullet$   
 585.         **let**  $(\_,1ui,\_) = hd\ ln, (\_,nui,\_) = ln[ len\ ln ]$  **in**  
 585.         **let**  $1nu = get\_NU(1ui)(rs), nnu = get\_NU(nnu)(rs)$  **in**  
 585.          $fs = 1nu \wedge ts = nnu$  **end end**  
 587.     **in end end**  
 587.     **pre**: `is_connected_stations(fs,ts)(rs)`

588. `is_immediately_connecting_line`: `Station`  $\times$  `Station`  $\rightarrow$  `RS`  $\rightarrow$  **Bool**  
 588. `is_immediately_connecting_line(fs,ts)(rs)  $\equiv$`   
 588.     **let** `ln = connecting_line(fs,ts)(rs)` **in**  
 588.      $\forall (\_,ui,\_):TUI \bullet (\_,ui,\_) \in inds\ ln \Rightarrow$   
 588.         **let**  $s = get\_RU(ui)(rs)$  **in**  
 588.          $s \in fs \cup ts \vee \sim in\_station(s)$  **end end**  
 588.     **pre**: `is_connected_stations(fs,ts)(rs)`

We leave it to the reader to define analysis functions that yield the set of all [immediately] connecting lines between two stations of a railway system.

### F.1.2.3 Attributes

Attributes are either static, or monitorable, or programmable.

#### F.1.2.3.1 Rail Nets

We treat attributes of rail units.

589. A rail unit is either in a station or is not, `STA`.

590. A rail unit is in some state – where a state is a possibly empty set of pairs of unique identifiers of connected rail units – with these being in respective set of the pair of sets making up the mereology of the rail unit, `PRG`.

Figure F.3 shows the twelve possible state of a point.

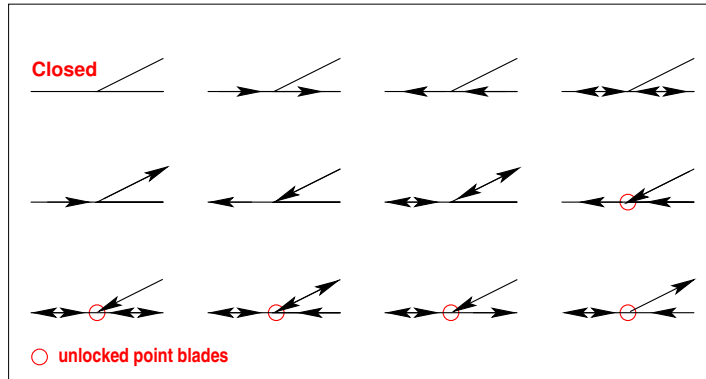


Figure F.3: The 12 Possible States of a Turnout Point

If a switch is unlocked, a train coming from either of the converging directions will pass through the points onto the narrow end, regardless of the position of the points, as the vehicle's wheels will force the points to move. Passage through a switch in this direction is known as a trailing-point movement.

**axiom**

```

 \forall pu:PU • pu \in xtr_NUs(ps) \Rightarrow let ({i},{o1,o2})=mereo_RU(pu), ω =attr_RU Ω (pu) in
 ω = {{}, {{(i,o1)}}, {(o1,i)}, {{(i,o1),(o1,i)}},
 {{(i,o2)}}, {(o2,i)}, {(i,o2),(o2,i)}, {(i,o2),(o2,i),(o1,i)},
 {{(i,o1),(o1,i),(o2,i)}}, {{(i,o2),(o2,i),(o1,i)}}, {{(i,o1),(o2,i)}}, {{(i,o2),(o2,i)}}}
end

```

- 591. A rail unit has a state space – consisting of all the states that a rail unit may attain, STA.
- 592. A point or a slip is either un-locked or locked, that is, its blades can be pressed to move, or cannot.
- 593. A rail unit has a length, STA.
- 594. A rail unit is either occupied by (a section of) an identified train or is not, PRG.
- 595. Et cetera.

**type**

```

589. RU_In_St = Bool static
590. RU Σ = (UI \times UI)-set programmable
591. RU Ω = R Σ -set static
591. Lock_Status = "un-locked" | "locked" programmable
593. RU_Len static
594. RU_Train == TI | ``nil`` programmable
595. ...
value
589. attr_RU_In_st: RU \rightarrow RU_In_St
590. attr_RU Σ : RU \rightarrow RU Σ
591. attr_RU Ω : RU \rightarrow RU Ω
591. attr_(PU|RU|SU|DU)_Lock_Status: (PU|RU|SU|DU) \rightarrow Lock_Status
593. attr_RU_Len: RU \rightarrow RU_Len
594. attr_Train: RU \rightarrow RU_Train

```

**axiom**

```

590. $\forall rs:RS, ru:RU \bullet ru \in retr_NUs(rs) \Rightarrow$
590. let uis = retr_UIs(rs), (iuis,ouis) = mereo_RU(ru), $\sigma = attr_Sigma(ru)$ in
590. $\forall (iui,oui):(UI \times UI) \bullet (iui,oui) \in \sigma \Rightarrow iui \in iuis \wedge oui \in ouis \wedge \{iui,oui\} \subseteq uis$
591. $\wedge \sigma \in attr_Omega(ru)$ end
594. $\wedge (attr_Train(ru) \in tris \vee attr_Train(ru) = \text{"nil"})$
595. ...

```

For any given switch the state space may be a proper subset of the set of all possible states.

### F.1.2.3.2 Open Routes

596. A route is said to be open if all pairs of the first and last element of route triplets are in the current state of the rail unit designated by the second element of these route triplets.

**value**

```

596. is_open_route: R \rightarrow RS \rightarrow Bool
596. is_open_route(r)(rs)
596. $\forall (iu,ui,ou):TUI \bullet (iu,ui,ou) \in elems\ r \Rightarrow$
596. let ru = get_RU(ui)(rs) in let $\sigma = attr_RU\Sigma(ru)$ in $(iu,ou) \in \sigma$ end end
596. pre: $r \in routes(rs)$

```

### F.1.2.3.3 Station Names

597. All rail units of a station has the same station name.

598. No two distinct stations have the same name.

**value**

```

598. station_name: Station \rightarrow Station_Name
598. station_name(st) \equiv let ru:RU \bullet ru \in st in attr_Name(ru) end

```

**axiom**

```

597. $\forall rs:RS \bullet$ let rn = obs_RN(rs) in
597. $\forall st,st':Station \bullet \{st,st'\} \subseteq stations(rn) \Rightarrow$
597. $\forall ru,ru':RU \bullet \{ru,ru'\} \subseteq st \Rightarrow attr_Name(ru) = attr_Name(ru')$
598. $st \neq st' \Rightarrow station_name(st) \neq station_name(st')$
598. end

```

### F.1.2.3.4 Trains

599. Trains have length with those of a given name having not necessarily the same length.

600. Trains [are expected to] follow a route, Train\_Route, and to be, at any time, at a Train\_Position.

A Train\_Route is a sequence of zero, one or more timed triplets, TUIT, of rail unit identifiers. A Train\_Position is a train attribute. It consists of three elements. Two train routes, ptr (past train route) and ntr (next train route), and a [current] timed triplet, TUIT, of rail unit identifiers. The meaning of a Train\_Position is that the train has passed the past route, is at the current timed triplet, and can next enter the next route.

601. No two distinct trains occupy overlapping routes on the net.

602. Trains have a speed and acceleration (or deceleration).

603. ...



**type**

```

599. Train_Length static
600. TUIT = TUI × TIME
600. Train_Route = TUIT*
600. Train_Position = ptr:Train_Route × TUIT × ntr:Train_Route programmable
602. Train_Speed, Train_Acceleration, Train_Deceleration monitorable
603. ...

```

**value**

```

599. attr_Train_Length: Train → Train_Length
600. attr_Train_Position: Train → Train_Position
602. attr_Train_Speed: Train → Train_Speed
602. attr_Train_Acceleration: Train → Train_Acceleration
602. attr_Train_Deceleration: Train → Train_Deceleration
603. ...

```

**axiom**

```

600. ∀ rs:RS •
600. ∀ tr,tr':Train • {tr,tr'} ⊆ obs_TS(rs) ∧ tr ≠ tr'
600. ⇒ is_open_route(attr_Train_Position(train))(rs)
600. ∧ let (trp,trp') = attr_Train_Position(tr,tr') in
600. {rui|(_,rui,_,_):TTUIT•(_,rui,_,_) ∈ elemens trr}
600. ∩
600. {rui|(_,rui',_,_):TUI•(_,rui',_,_) ∈ elemens trr'} = {}
600. end

```

**F.1.2.3.5 An Intentional Pull**

604. For every railway system it is the case that
605. for every rail unit in that system which “records”, as an attribute, a train, there is exactly one train that in its route position records exactly that rail unit,
606. and vice versa.

**axiom**

```

604. ∀ rs:RS •
605. ∀ ru:RU • ru ∈ retr_NUs(rs) ⇒
605. if attr_RU_Train(ru) ≠ “nil” ⇒
605. ∃! tr:Train • tr ∈ trains(rs) ∧
605. uid_NU(ru) ∈ {ui|(_,ui,_,_):TUI • (_,ui,_,_) ∈ elems attr_Train_Position(tr)}
604. ∧
606. ∀ tr:Train • tr ∈ trains(rs) ⇒
606. ∀ (_,ui,_,_):TUI • (_,ui,_,_) ∈ elems attr_Train_Position(tr) ⇒
606. attr_RU_Train(get_NU(ui)(rs)) = uid_Train(tr)
604. end

```

**F.1.2.3.6 History Attributes** The attributes and axioms over them – covered above do not relate to time; they are time-independent. We now treat time-dependent attributes and axioms over them. By **TIME** we mean absolute times, like *March 12, 2024: 10:48 am*, and by **TI** we mean time intervals, like *two hours, three minutes and five seconds*. We shall here consider **TIME** to span a definite “period” of time, say from *January 1, 2020, 00:00am* to *December 31, 2020, 24:00*.

607. Of a road unit we can speak of its history as a time-decreasing, ordered sequence of time-stamped train identifiers.

608. Of a train we can speak of its history as a time-decreasing, ordered sequence of time-stamped rail unit identifiers.

We could have considered other properties to form or be included in event histories, but abstain.

**type**

607.  $\text{RU\_Hist} = (\text{TIME} \times \text{TI})^* \text{programmable}$

608.  $\text{TR\_Hist} = (\text{TIME} \times \text{UI})^* \text{programmable}$

**value**

607.  $\text{attr\_RU\_Hist}: \text{RU} \rightarrow \text{RU\_Hist}$

608.  $\text{attr\_TR\_Hist}: \text{Train} \rightarrow \text{TR\_Hist}$

**axiom**

607. [ descending times in rail unit history ]

608. [ descending times in train history ]

### F.1.2.3.7 The Intentional Pull Revisited

609. For every railway system it is the case that
610. for every rail unit,
611. if at any time it records a train,
612. then that train's event history records that rail unit in the route it is occupying at that time,  
and
613. for every train, if at any time it records a route
614. then exactly the rail units of that route record that train.

... below function has to be redefined ...

**axiom**

609.  $\forall rs:RS \bullet$

610.  $\forall ru:RU \bullet ru \in \text{retr\_NUs}(rs) \Rightarrow$

610. **let**  $\text{ruh} = \text{attr\_RU\_Hist}(ru)$  **in**

611.  $\forall \text{time}:\text{dom } \text{ruh} \bullet \text{ruh}(\text{time}) \neq \{\} \Rightarrow$

612. **let**  $\{ti\} = \text{ruh}(\text{time})$  **in**

612. **let**  $\text{trh} = \text{attr\_TR\_Hist}(\text{get\_Train}(ti)(rs))$  **in**

612.  $\text{trh}(\text{time}) \neq \{\} \wedge$

612. **let**  $\{r\} = \text{trh}(\text{time})$  **in**

612.  $\exists (\_ , \text{ui}, \_):\text{TUI} \bullet (\_ , \text{ui}, \_) \in \text{elems}(r) \Rightarrow \text{ruh} = \text{get\_RU}(\text{ui})(rs)$

612. **end end end end**

613. et cetera

614. et cetera

## F.2 Transcendental Deduction

### F.2.1 General

By a transcendental deduction parts can be “morphed” into behaviours. We consider the following railway system parts:

- all the railway net units and
- all the trains.

That is, we shall not here consider the railway net management, the train operator, the passenger and [freight] shipper parts as behaviours.

**F.2.2 A Note on TIME**

615. We shall consider `TIME` to stand for a time in a definite interval of times, for example from *January 1, 2020, 00:00 am* to *December 31, 2020, 23:59:59*.
616. That is, `TIME-interval`, is the set of all the designated times in the interval.
617. The operators  $\mathcal{F}$ [*irst*] and  $\mathcal{L}$ [*ast*] applied to the `TIME-interval` interval yields the first and last times of the interval `TIME-interval`.
618. We shall introduce a time interval quantity,  $\delta\tau:\text{TI}$  – and shall consider  $\delta\tau$  to be, if not infinitesimal small, then at least “small”, say, in the context of train traffic, *1 second!*
619. We shall, loosely, introduce the operator  $\mathcal{D}$ , applied to the interval `TIMEinterval`, to yield the definite set of times such that if  $\tau$  is in `TIME-interval` and  $\tau$  is not  $\mathcal{L}(\text{TIME-interval})$  then the next time in `TIMEinterval` is  $\tau + \delta\tau$ .

**type**

615. `TIME`  
 616. `TIME-interval`  
 617.  $\mathcal{F}: \text{TIME-interval} \rightarrow \text{TIME}$   
 617.  $\mathcal{L}: \text{TIME-interval} \rightarrow \text{TIME}$

**value**

618.  $\delta\tau:\text{TI}$  [ say 1 second ]  
 619.  $\mathcal{D}: \text{TIME-interval} \rightarrow \text{TIME-set}$

**F.2.3 Train Traffic**

620. By train traffic we shall understand a discrete function, in RSL [100] expressed as a map, over a closed interval of time from time to trains and their route position.

We model this as shown in formula line 620.

Here we have taken the liberty of modeling the traffic as being discrete over infinitesimal small time intervals  $\delta\tau$ .

**type**

620.  $\text{TrainTraffic} = \text{TI} \xrightarrow{\text{map}} (\text{TIME} \xrightarrow{\text{map}} \text{R})$

**F.2.3.1 Well-formed Train Traffics**

621. For every railway system a train traffic is well-formed

- (a) if all trains cover the same time period;
- (b) if all train traffics occur on routes of the railway system;
- (c) if two or more trains do not have overlapping routes at any time; and
- (d) if each train traffic progresses monotonically.

**axiom**

621.  $\forall rs:\text{RS} \bullet$   
 621.  $\forall \text{trtr}:\text{TrainTraffic} \bullet$   
 621a.  $\text{same\_time\_period}(\text{trtr})$   
 621b.  $\wedge \text{routes\_of\_rs}(\text{trtr})(rs)$   
 621c.  $\wedge \text{disjoint\_routes}(\text{trtr})(rs)$   
 621d.  $\wedge \text{monotonic}(\text{trtr})(rs)$

621a. same\_time\_period: TrainTraffic  $\rightarrow$  **Bool**  
 621a. same\_time\_period(trtr)  $\equiv \forall \text{time, time}': \text{TIME} \bullet \text{DOMAIN}(\text{time}) = \text{DOMAIN}(\text{time}')$

**value**

621b. routes\_of\_rs: TrainTraffic  $\rightarrow$  RS  $\rightarrow$  **Bool**  
 621b. routes\_of\_rs(trtr)(rs)  
 621b.  $\forall \text{ti:TI} \bullet \text{ti} \in \text{dom trtr} \Rightarrow$   
 621b.  $\forall \text{time:TIME} \bullet \text{time} \in \text{dom ti}$   
 621b. route\_of((trtr(ti))(time))(rs)  
 621b. route\_of: R  $\rightarrow$  RS  $\rightarrow$  **Bool**  
 621b. route\_of(r)(rs)  $\equiv r \in \text{routes}(\text{rs})$

**value**

621c. disjoint\_routes: TrainTraffic  $\rightarrow$  RS  $\rightarrow$  **Bool**  
 621c. disjoint\_routes(trtr)(rs)  $\equiv$   
 621c.  $\forall \text{ti, ti}': \text{TI} \bullet \{\text{ti, ti}'\} \leq \text{dom trtr} \wedge \text{ti} \neq \text{ti}' \Rightarrow$   
 621c.  $\forall \text{time:TIME} \bullet \text{time} \in \text{dom ti} \Rightarrow$   
 621c. disjoint\_routes((trtr(ti))(time), (trtr(ti'))(time))  
 621c. disjoint\_routes: R  $\times$  R  $\rightarrow$  **Bool**  
 621c. disjoint\_routes(r, r')  $\equiv$   
 621c.  $\{\text{ui}(\_, \text{ui}, \_) : \text{TUI} \bullet (\_, \text{ui}, \_) \in \text{elems } r\} \cap \{\text{ui}(\_, \text{ui}, \_) : \text{TUI} \bullet (\_, \text{ui}, \_) \in \text{elems } r'\} = \{\}$

For a traffic to be monotonic it must be the case that

- 622. for all trains
- 623. for two “closely adjacent” times in the domain of that train’s traffic
- 624. the route positions,  $r, r'$  of any train (at these times) must
- 625. either be the same. i.e.  $r = r'$ ,
- 626. or truncated by at most the first element, i.e.  $r' = \mathbf{tl} r$  (being a route of the system),
- 627. or amended by at most one element, i.e.,  $r' = r \hat{\langle \text{tui} \rangle}$  (being a route of the system),
- 628. or both, i.e.,  $r' = \mathbf{tl} r \hat{\langle \text{tui} \rangle}$  (being a route of the system).

**value**

621d. monotonic: TrainTraffic  $\rightarrow$  RS  $\rightarrow$  **Bool**  
 621d. monotonic(trtr)(rs)  $\equiv$   
 623.  $\forall \text{ti:TI} \bullet \text{ti} \in \text{dom trtr} \Rightarrow \mathbf{in}$   
 623.  $\forall \text{time, time}': \text{TIME} \bullet$   
 623.  $\{\text{time, time}'\} \subseteq \text{DOMAIN}(\text{trtr}(\text{ti}))$   
 623.  $\text{time}' > \text{time} \wedge \text{time}' - \text{time} = \delta\tau \wedge$   
 624. **let**  $(r, r') = ((\text{trtr}(\text{ti}))(\text{time}), (\text{trtr}(\text{ti}))(\text{time}'))$  **in**  
 625.  $(r' = r) \vee$   
 626.  $(r' = \mathbf{tl} r \wedge \mathbf{tl} r \in \text{routes}(\text{rs})) \vee$   
 627.  $(r' = r \hat{\langle \text{tui} \rangle} \wedge r \hat{\langle \text{tui} \rangle} \in \text{routes}(\text{rs})) \vee$   
 628.  $(r' = \mathbf{tl} r \hat{\langle \text{tui} \rangle} \wedge \mathbf{tl} r \hat{\langle \text{tui} \rangle} \in \text{routes}(\text{rs}))$   
 624. **end**

### F.3 Perdurants

To every part, that is,

- |                      |                         |            |
|----------------------|-------------------------|------------|
| 629. linear unit,    | 632. slip (crossing),   | 635. train |
| 630. turn out,       | 633. double (crossing), |            |
| 631. rigid crossing, | 634. terminal unit, and |            |

we associate, by a transcendental deduction, a never ending train behaviour which, as a function, takes some arguments  $\dots \rightarrow \dots$  and otherwise goes on forever (**Unit**).

|                                                         |                                                      |
|---------------------------------------------------------|------------------------------------------------------|
| <b>value</b>                                            | 632. slip: $\dots \rightarrow \dots$ <b>Unit</b>     |
| 629. linear_unit: $\dots \rightarrow \dots$ <b>Unit</b> | 633. double: $\dots \rightarrow \dots$ <b>Unit</b>   |
| 630. turn_out: $\dots \rightarrow \dots$ <b>Unit</b>    | 634. terminal: $\dots \rightarrow \dots$ <b>Unit</b> |
| 631. rigid: $\dots \rightarrow \dots$ <b>Unit</b>       | 635. train: $\dots \rightarrow \dots$ <b>Unit</b>    |

The **Unit** does not refer to the railway units of the domain, but is an RSL ... in effect designating never ending processes.

### F.3.1 Channels

636. Trains and rail net units exchange messages, NT\_Msg.  
These message will eventually be further defined.
637. Trains potentially communicate with all rail net units.  
Rail net units potentially communicate with all trains.

#### type

636. NT\_Msg

#### channel

637.  $\{ \text{ch}[\{ui,tri\}]:\text{NT\_Msg} \mid ui:U1, tri:TR1 \bullet ui \in uis \wedge tri \in trus \}$

In a more realistic railway system domain description a rail net management would monitor trains and control (set) switches etc.

### F.3.2 Behaviour Signatures

We continue sketching some of the railway system behaviour signatures. Rail net unit and train identifiers become [first] parameters; mereology attributes become [second set of] parameters; static attributes become [third set of] parameters; programmable attributes become [fourth] parameters; and channel references become “last” parameters.

#### value

- |                                                                                                                                                                                                                                                     |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 629. linear_unit: $ui:U1 \times (\_,tris):Unit\_Mereo \times (RU\Omega \times RU\_Len) \rightarrow (RU\Sigma \times RU\_Hist)$<br>$\rightarrow \text{in,out} \{ \text{ch}[\{ui,ti\}]   ti:TI \bullet ti \in tris \}$ <b>Unit</b>                    |
| 630. turn_out: $ui:U1 \times (\_,tris):Unit\_Mereo \times (RU\Omega \times RU\_Len) \rightarrow (RU\Sigma \times RU\_Hist)$<br>$\rightarrow \text{in,out} \{ \text{ch}[\{ui,ti\}]   ti:TI \bullet ti \in tris \}$ <b>Unit</b>                       |
| 631. rigid: $ui:U1 \times (\_,tris):Unit\_Mereo \times (RU\Omega \times RU\_Len) \rightarrow (RU\Sigma \times RU\_Hist)$<br>$\rightarrow \text{in,out} \{ \text{ch}[\{ui,ti\}]   ti:TI \bullet ti \in tris \}$ <b>Unit</b>                          |
| 632. slip: $ui:U1 \times (\_,tris):Unit\_Mereo \times (RU\Omega \times RU\_Len) \rightarrow (RU\Sigma \times RU\_Hist)$<br>$\rightarrow \text{in,out} \{ \text{ch}[\{ui,ti\}]   ti:TI \bullet ti \in tris \}$ <b>Unit</b>                           |
| 633. double: $ui:U1 \times (\_,tris):Unit\_Mereo \times (R^U\Omega \times RU\_Le) \rightarrow (RU\Sigma \times RU\_Hist)$<br>$\rightarrow \text{in,out} \{ \text{ch}[\{ui,ti\}]   ti:TI \bullet ti \in tris \}$ <b>Unit</b>                         |
| 634. terminal: $ui:U1 \times (\_,tris):Unit\_Mereo \times (RU\Omega \times RU\_Len) \rightarrow (RU\Sigma \times RU\_Hist)$<br>$\rightarrow \text{in,out} \{ \text{ch}[\{ui,ti\}]   ti:TI \bullet ti \in tris \}$ <b>Unit</b>                       |
| 635. train: $ti:TI \times uis:Train\_Mereo \times (TR\Omega \times Train\_Length) \rightarrow (Train\_Position \times (TR\Sigma \times TR\_Hist))$<br>$\rightarrow \text{in,out} \{ \text{ch}[\{ui,ti\}]   ui:U1 \bullet ii \in uis \}$ <b>Unit</b> |

### F.3.3 Behaviour Definitions

We shall illustrate only a narrow aspect of trains on rails. Namely that of the “simulation” of train traffic as per pre-planned routes. That is we shall not model actual train traffic as per set time tables – that would entail numerous more formulas than we now show. So it is only an illustration of how rail and train behaviours might look.

#### F.3.3.1 Rail Unit Behaviours

We shall only exemplify linear rail unit behaviours.

- 638. Rail unit behaviours all have in common what we now model as the linear rail unit behaviour.
- 639. Non-deterministically, external choice, the rail units offers to accept communication from passing trains,  $ti$ , as to the time they are passing by –
- 640. with this information being added to the rail unit history as the rail unit behaviour resumes.

**value**

```

638. linear_unit(ui,(ruω,...),(_,tris))(ruσ,ruh) ≡
639. let Msg_TR_RU(time,ti) = [] {ch[{ui,ti}] ? | ti:TI • ti ∈ tris} in
640. linear_unit(ui,(ruω,...),(_,tris))(ruσ,⟨(time,ti)⟩^ruh)
638. end
638. pre: ruσ ∈ ruω

```

#### F.3.3.2 Train Behaviour

We focus, in our description of train behaviours sôlely on the un-aided movement of trains and, further, on an “idealised” description.

- 641. There are two train positions of interest when describing train movement:
  - (a) the general situation where the train has not yet reached its final destination, and
  - (b) the special situation where the train has indeed reached its final destination
- 642. In the former (Item 641a.) the train position, at time  $\tau$ , is at rail unit  $ui$ , with the first next unit being  $ui'$  (and where  $aui=aui'$ ).
- 643. If elapsed time is less than planned time  $\tau$ ,
- 644. then the train informs the rail unit behaviour designated by  $ui$  that it is currently passing it.
- 645. and moves on within the current unit  $ui$ , having updated its history;
- 646. else, when elapsed time is up, i.e., equals planned time  $\tau$ , the train informs the rail unit it is now entering that it is so,
- 647. updates its history accordingly and moves on to the next unit,  $ui'$

**value**

```

642. train(ti,sta,uis)(pr,⟨(bui,ui,aui),τ⟩,⟨((aui',ui',nui),τ')⟩^nr),(trσ,trh) ≡
642. let time = record_TIME in
643. if time < τ
644. then ch[{ui,ti}] ! Msg_TR_RU(time,ti) ;
645. train(ti,sta,uis)(pr,⟨(bui,ui,aui),τ⟩,⟨((aui',ui',nui),τ')⟩^nr),(trσ,⟨(time,ui)⟩^trh)
646. else ch[{aui',ti}] ! Msg_TR_RU(time,ti) ; assert: time = τ
647. train(ti,sta,uis)(tp^⟨(τ,(bui,ui,aui))⟩,⟨((aui',ui',nui),τ')⟩,nr),(trσ,⟨(time,ui')⟩^trh)
642. end end
642. pre: trσ ∈ trω ∧ aui=aui' ∧ τ < τ'

```

648. In the other position (Item 641b.) the train, at time  $\tau$ , is at rail unit  $ui$ , with their being no next units to enter.
649. If elapsed time is less than planned time,  $\tau$ ,
650. then the train informs the rail unit behaviour designated by  $ui$  that it is currently passing it
651. and moves on within the current unit  $ui$ , having updated its history;
652. else the train journey has ended and the train behaviour “stops”, i.e., ceases to exist!

```

648. train(ti,sta,uis)((pr,((bui,ui,ai), τ), $\langle \rangle$),(tr σ ,trh)) \equiv
648. let time = record_TIME in
649. if time < τ
650. then ch[{ui,ti}] ! Msg-TR-RU(time,ti) ;
651. train(ti,sta,uis)((pr,((bui,ui,ai), τ), $\langle \rangle$),(tr σ ,((time,ui))^trh))
652. else skip assert: time = τ
648. end end
648. pre: tr σ \in tr ω

```

## F.4 Closing

We end our example here. To analyse & describe a proper railway system we would have to introduce some rail net and train management. Rail net management would monitor the rails, and, according to train time tables issued by train management, set switches. Train management would establish train time tables, pass these onto rail net management, and would monitor and control trains. We have given, we think, enough clues as how to analyse & describe such railway systems.





# Appendix G

## Simple Credit Card Systems

### Contents

---

|            |                        |     |
|------------|------------------------|-----|
| <b>G.1</b> | <b>Introduction</b>    | 211 |
| <b>G.2</b> | <b>Endurants</b>       | 212 |
| G.2.1      | External Qualities     | 212 |
| G.2.2      | Internal Qualities     | 213 |
| G.2.2.1    | Unique Identification  | 213 |
| G.2.2.2    | Mereologies            | 213 |
| G.2.2.3    | Mereologies            | 214 |
| G.2.2.3.1  | Banks:                 | 214 |
| G.2.2.3.2  | Shops:                 | 215 |
| G.2.2.4    | Attributes             | 215 |
| <b>G.3</b> | <b>Perdurants</b>      | 215 |
| G.3.1      | Behaviours             | 215 |
| G.3.2      | Channels               | 216 |
| G.3.3      | Behaviour Interactions | 216 |
| G.3.4      | Credit Card            | 218 |
| G.3.5      | Banks                  | 219 |
| G.3.6      | Shops                  | 221 |

---

We present an attempt at a model of a simple credit card system of credit card holders, shops and banks.<sup>1</sup>

No Model of Temporal History

This model, and we apologise profusely, does not model the time-stamped history of transactions.  
Left as a suitable exercise for the reader!

### G.1 Introduction

We present a domain description of an abstracted credit card system. The narrative part of the description is terse, perhaps a bit too terse.

Credit cards are moving from simple plastic cards to smart phones. Uses of credit cards move from their mechanical insertion in credit card terminals to being swiped. Authentication (hence not modelled) moves from keying in security codes to eye iris “prints”, and/or finger prints or voice prints or combinations thereof.

---

<sup>1</sup>This model evolved during a PhD course at the University of Uppsala, Sweden.

This document abstracts from all that in order to understand a bare, minimum essence of credit cards and their uses. Based on a model, such as presented here, the reader should be able to extend/refine the model into any future technology – for requirements purposes.

## G.2 Endurants

### G.2.1 External Qualities

653. Credit card systems,  $ccs:CCS$ , <sup>2</sup>consists of three kinds of parts:

654. an assembly,  $cs:CS$ , of credit cards<sup>4</sup>,

655. an assembly,  $bs:BS$ , of banks, and

656. an assembly,  $ss:SS$ , of shops.

#### type

653 CCS

654 CS

655 BS

656 SS

#### value

654  $obs\_CS: CCS \rightarrow CS$

655  $obs\_BS: CCS \rightarrow BS$

656  $obs\_SS: CCS \rightarrow SS$

657. There are credit cards,  $c:C$ , banks  $b:B$ , and shops  $s:S$ .

658. The credit card part,  $cs:CS$ , abstracts a set,  $soc:Cs$ , of card.

659. The bank part,  $bs:BS$ , abstracts a set,  $sob:Bs$ , of banks.

660. The shop part,  $ss:SS$ , abstracts a set,  $sos:Ss$ , of shops.

#### type

657 C, B, S

658  $Cs = C\text{-set}$

659  $Bs = B\text{-set}$

660  $Ss = S\text{-set}$

#### value

658  $obs\_CS: CS \rightarrow Cs$ ,  $obs\_Cs: CS \rightarrow Cs$

659  $obs\_BS: BS \rightarrow Bs$ ,  $obs\_Bs: BS \rightarrow Bs$

660  $obs\_SS: SS \rightarrow Ss$ ,  $obs\_Ss: SS \rightarrow Ss$

---

<sup>2</sup>The composite part  $CS$  can be thought of as a credit card company, say VISA<sup>3</sup>. The composite part  $BS$  can be thought of as a bank society, say BBA: British Banking Association. The composite part  $SS$  can be thought of as the association of retailers, say bira: British Independent Retailers Association. The model does not prevent “shops” from being airlines, or car rental agencies, or dentists, or consultancy firms. In this case  $SS$  would be some appropriate association.

<sup>4</sup>We “equate” credit cards with their holders.

**G.2.2 Internal Qualities****G.2.2.1 Unique Identification**

661. Assemblers of credit cards, banks and shops have unique identifiers,  $csi:CSI$ ,  $bsi:BSI$ , and  $ssi:SSI$ .

662. Credit cards, banks and shops have unique identifiers,  $ci:CI$ ,  $bi:BI$ , and  $si:SI$ .

663. One can define functions which extract all the

664. unique credit card,

665. bank and

666. shop identifiers from a credit card system.

661  $CSI, BSI, SSI$

662  $CI, BI, SI$

**value**

661  $uid\_CS: CS \rightarrow CSI, uid\_BS: BS \rightarrow BSI, uid\_SS: SS \rightarrow SSI,$

662  $uid\_C: C \rightarrow CI, uid\_B: B \rightarrow BI, uid\_S: S \rightarrow SI,$

664  $xtr\_CIs: CCS \rightarrow CI\text{-set}$

664  $xtr\_CIs(ccs) \equiv \{uid\_C(c) | c:C \bullet c \in obs\_Cs(obs\_CS(ccs))\}$

665  $xtr\_BIs: CCS \rightarrow BI\text{-set}$

665  $xtr\_BIs(ccs) \equiv \{uid\_B(s) | b:B \bullet b \in obs\_Bs(obs\_BS(ccs))\}$

666  $xtr\_SIs: CCS \rightarrow SI\text{-set}$

666  $xtr\_SIs(ccs) \equiv \{uid\_S(s) | s:S \bullet s \in obs\_Ss(obs\_SS(ccs))\}$

667. For all credit card systems it is the case that

668. all credit card identifiers are distinct from bank identifiers,

669. all credit card identifiers are distinct from shop identifiers,

670. all shop identifiers are distinct from bank identifiers,

**axiom**

667  $\forall ccs:CCS \bullet$

667 **let**  $cis=xtr\_CIs(ccs), bis=xtr\_BIs(ccs), sis = xtr\_SIs(ccs)$  **in**

668  $cis \cap bis = \{\}$

669  $\wedge cis \cap sis = \{\}$

670  $\wedge sis \cap bis = \{\}$  **end**

**G.2.2.2 Mereologies**

671. A credit card has a mereology which “connects” it to any of the shops of the system and to exactly one bank of the system,

672. and some attributes — which we shall presently disregard.

673. The wellformedness of a credit card system includes the wellformedness of credit card mereologies with respect to the system of banks and shops:

674. The unique shop identifiers of a credit card mereology must be those of the shops of the credit card system; and

675. the unique bank identifier of a credit card mereology must be of one of the banks of the credit card system.

**type**

671.  $CM = SI\text{-set} \times BI$

**value**

671.  $mereo\_CM: C \rightarrow CM$

673  $wf\_CM\_of\_C: CCS \rightarrow \mathbf{Bool}$

673  $wf\_CM\_of\_C(ccs) \equiv$

671 **let**  $bis = xtr\_BIs(ccs)$ ,  $sis = xtr\_SIs(ccs)$  **in**

671  $\forall c: C \bullet c \in obs\_Cs(obs\_CS(ccs)) \Rightarrow$

671 **let**  $(ccsis, bi) = mereo\_CM(c)$  **in**

674  $ccsis \subseteq sis$

675  $\wedge bi \in bis$

671 **end end**

**G.2.2.3 Mereologies**

**G.2.2.3.1 Banks:** Our model of banks is (also) very limited.

676. A bank has a mereology which “connects” it to a subset of all credit cards and a subset of all shops,

677. and, as attributes:

678. a cash register, and

679. a ledger.

680. The ledger records for every card, by unique credit card identifier,

681. the current balance: how much money, credit or debit, i.e., plus or minus, that customer is owed, respectively has borrowed from the bank,

682. the dates-of-issue and -expiry of the credit card, and

683. the name, address, and other information about the credit card holder.

684. The wellformedness of the credit card system includes the wellformedness of the banks with respect to the credit cards and shops:

685. the bank mereology’s

686. must list a subset of the credit card identifiers and a subset of the shop identifiers.

**type**

676  $BM = CI\text{-set} \times SI\text{-set}$

678  $CR = Bal$

679  $LG = CI \xrightarrow{m} (Bal \times Dol \times DoE \times \dots)$

681  $Bal = \mathbf{Int}$

**value**

676  $mereo\_B: B \rightarrow BM$

678  $attr\_CR: B \rightarrow CR$

679  $attr\_LG: B \rightarrow LG$

684  $wf\_BM\_B: CCS \rightarrow \mathbf{Bool}$

684  $wf\_BM\_B(ccs) \equiv$

684 **let**  $allcis = xtr\_CIs(ccs)$ ,  $allsis = xtr\_SIs(ccs)$  **in**

```

684 $\forall b:B \bullet b \in \text{obs_Bs}(\text{obs_BS}(\text{ccs}))$ in
685 let (cis, sis) = mereo_B(b) in
686 cis $\subseteq \forall$ cis \wedge sis \subseteq allsis end end

```

### G.2.2.3.2 Shops:

687. The mereology of a shop is a pair: a unique bank identifiers, and a set of unique credit card identifiers.

688. The mereology of a shop

689. must list a bank of the credit card system,

690. band a subset (or all) of the unique credit identifiers.

We omit treatment of shop attributes.

#### type

```
687 SM = CI-set \times BI
```

#### value

```

687 mereo_S: S \rightarrow SM
688 wf_SM_S: CCS \rightarrow Bool
688 wf_SM_S(ccs) \equiv
688 let allcis = xtr_CIs(ccs), allbis = xtr_BIs(ccs) in
688 $\forall s:S \bullet s \in \text{obs_Ss}(\text{obs_SS}(\text{ccs})) \Rightarrow$
688 let (cis, bi) mereo_S(s) in
689 bi \in allbis
690 \wedge cis \subseteq allcis
688 end end

```

### G.2.2.4 Attributes

|               |
|---------------|
| TO BE WRITTEN |
|---------------|

## G.3 Perdurants

### G.3.1 Behaviours

691. We ignore the behaviours related to the *CCS*, *CS*, *BS* and *SS* parts.

692. We therefore only consider the behaviours related to the *Cs*, *Bs* and *Ss* parts.

693. And we therefore compile the credit card system into the parallel composition of the parallel compositions of all the credit card, *crd*, all the bank, *bnk*, and all the shop, *shp*, behaviours.

#### value

```

691 ccs:CCS
691 cs:CS = obs_CS(ccs),
691 uics:CSI = uid_CS(cs),
691 bs:BS = obs_BS(ccs),
691 uibs:BSI = uid_BS(bs),
691 ss:SS = obs_SS(ccs),
691 uiss:SSI = uid_SS(ss),
692 socs:Cs = obs_Cs(cs),
692 sobbs:Bs = obs_Bs(bs),
692 soss:Ss = obs_Ss(ss),

```

**value**

```

693 sys: Unit → Unit,
691 sys() ≡
693 cardsuics(mereo_CS(cs),...)
693 || || {crduid_C(c)(mereo_C(c))|c:C•c ∈ socs}
693 || banksuibs(mereo_BS(bs),...)
693 || || {bnkuid_B(b)(mereo_B(b))|b:B•b ∈ sobS}
693 || shopsuiss(mereo_SS(ss),...)
693 || || {shpuid_S(s)(mereo_S(s))|s:S•s ∈ soss},
691 cardsuics(...) ≡ skip,
691 banksuibs(...) ≡ skip,
691 shopsuiss(...) ≡ skip

```

**axiom** **skip** || behaviour(...) ≡ behaviour(...)

**G.3.2 Channels**

- 694. Credit card behaviours interact with bank (each with one) and many shop behaviours.
- 695. Shop behaviours interact with bank (each with one) and many credit card behaviours.
- 696. Bank behaviours interact with many credit card and many shop behaviours.

The inter-behaviour interactions concern:

- 697. between credit cards and banks: withdrawal requests as to a sufficient,  $\text{mk\_Wdr}(\text{am})$ , balance on the credit card account for buying  $\text{am}:\text{AM}$  amounts of goods or services, with the bank response of either  $\text{is\_OK}()$  or  $\text{is\_NOK}()$ , or the revoke of a card;
- 698. between credit cards and shops: the buying, for an amount,  $\text{am}:\text{AM}$ , of goods or services:  $\text{mk\_Buy}(\text{am})$ , or the refund of an amount;
- 699. between shops and banks: the deposit of an amount,  $\text{am}:\text{AM}$ , in the shops' bank account:  $\text{mk\_Depost}(\text{ui},\text{am})$  or the removal of an amount,  $\text{am}:\text{AM}$ , from the shops' bank account:  $\text{mk\_Removl}(\text{bi},\text{si},\text{am})$

**channel**

```

694 {ch_cb[ci,bi]|ci:CI,bi:BI•ci ∈ cis ∧ bi ∈ bis}:CB_Msg
695 {ch_cs[ci,si]|ci:CI,si:SI•ci ∈ cis ∧ si ∈ sis}:CS_Msg
696 {ch_sb[si,bi]|si:SI,bi:BI•si ∈ sis ∧ bi ∈ bis}:SB_Msg
697 CB_Msg == mk_Wdrw(am:aM) | is_OK() | is_NOK() | ...
698 CS_Msg == mk_Buy(am:aM) | mk_Ref(am:aM) | ...
699 SB_Msg == Depost | Removl | ...
699 Depost == mk_Dep((ci:CI|si:SI),am:aM) |
699 Removl == mk_Rem(bi:BI,si:SI,am:aM)

```

**G.3.3 Behaviour Interactions**

- 700. The credit card initiates

(a) buy transactions

- i. [1.Buy] by enquiring with its bank as to sufficient purchase funds ( $\text{am}:\text{AM}$ );
- ii. [2.Buy] if NOK then there are presently no further actions; if OK

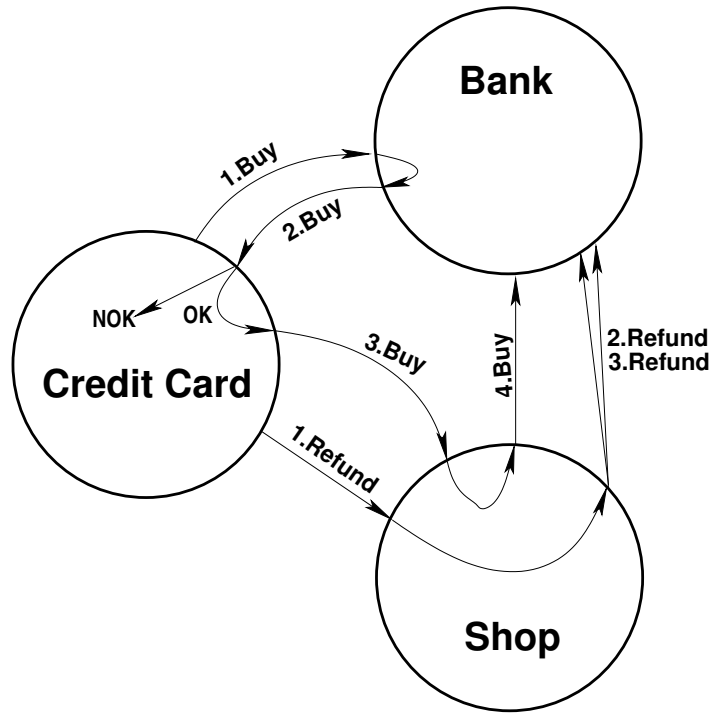


Figure G.1: Credit Card, Bank and Shop Behaviours

- iii. [3.Buy] the credit card requests the purchase from the shop – handing it an appropriate amount;
- iv. [4.Buy] finally the shop requests its bank to deposit the purchase amount into its bank account.

## (b) refund transactions

- i. [1.Refund] by requesting such refunds, in the amount of  $am:aM$ , from a[ny] shop; whereupon
- ii. [2.Refund] the shop requests its bank to move the amount  $am:aM$  from the shop's bank account
- iii. [3.Refund] to the credit card's account.

Thus the three sets of behaviours, *crd*, *bnk* and *shp* interact as sketched in Fig. G.1.

|            |                                           |              |                                                                                                                                                                   |
|------------|-------------------------------------------|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| [1.Buy]    | Item 706, Pg.218<br>Item 715, Pg.219      | card<br>bank | $ch\_cb[ci,bi]!mk\_Wdrw(am)$ (shown as ... three lines down) and<br>$mk\_Wdrw(ci,am)=\prod\{ch\_cb[bi,bi]? ci:CI\bullet ci \in cis\}$ .                           |
| [2.Buy]    | Items 708-709, Pg.218<br>Item 706, Pg.218 | bank<br>shop | $ch\_cb[ci,bi]!is\_N]OK()$ and<br>$(\dots;ch\_cb[ci,bi]?)$ .                                                                                                      |
| [3.Buy]    | Item 708, Pg.218<br>Item 730, Pg.221      | card<br>shop | $ch\_cs[ci,si]!mk\_Buy(am)$ and<br>$mk\_Buy(am)=\prod\{ch\_cs[ci,si]? ci:CI\bullet ci \in cis\}$ .                                                                |
| [4.Buy]    | Item 731, Pg.221<br>Item 720, Pg.220      | shop<br>bank | $ch\_sb[si,bi]!mk\_Dep(si,am)$ and<br>$mk\_Dep(si,am)=\prod\{ch\_cs[ci,si]? si:SI\bullet si \in sis\}$ .                                                          |
| [1.Refund] | Item 712, Pg.219<br>Item 731, Pg.221      | card<br>shop | $ch\_cs[ci,si]!mk\_Ref((ci,si),am)$ and<br>$(si,mk\_Ref(ci,am))=\prod\{si',ch\_sb[si,bi]? si,si':SI\bullet\{si,si'\}\subseteq sis \wedge si=si'\}$ .              |
| [2.Refund] | Item 735, Pg.221<br>Item 724, Pg.220      | shop<br>bank | $ch\_sb[si,cbi]!mk\_Ref(cbi,(ci,si),am)$ and<br>$(si,mk\_Ref(cbi,(ci,am)))=\prod\{(si',ch\_sb[si,bi]?) si,si':SI\bullet\{si,si'\}\subseteq sis \wedge si=si'\}$ . |
| [3.Refund] | Item 736, Pg.221<br>Item 725, Pg.220      | shop<br>bank | $ch\_sb[si,sbi]!mk\_Wdr(si,am)$ <b>end</b> and<br>$(si,mk\_Wdr(ci,am))=\prod\{(si',ch\_sb[si,bi]?) si,si':SI\bullet\{si,si'\}\subseteq sis \wedge si=si'\}$       |

### G.3.4 Credit Card

701. The credit card behaviour, `crd`, takes the credit card unique identifier, the credit card mereology, and attribute arguments (omitted). The credit card behaviour, `crd`, accepts inputs from and offers outputs to the bank, `bi`, and any of the shops, `si`  $\in$  `sis`.
702. The credit card behaviour, `crd`, non-deterministically, internally “cycles” between buying and getting refunds.

#### value

701  $crd_{ci:CI}: (bi, sis):CM \rightarrow \mathbf{in, out} \ ch\_cb[ci, bi], \{ch\_cs[ci, si]|si:SI\bullet si \in sis\}$  **Unit**  
 701  $crd_{ci}(bi, sis) \equiv (buy(ci, (bi, sis)) \prod ref(ci, (bi, sis))) ; crd_{ci}(ci, (bi, sis))$

703. By `am:AM` we mean an amount of money, and by `si:SI` we refer to a shop in which we have selected a number or goods or services (not detailed) costing `am:AM`.
704. The buyer action is simple.
705. The amount for which to buy and the shop from which to buy are selected (arbitrarily).
706. The credit card (holder) withdraws `am:AM` from the bank, if sufficient funds are available<sup>5</sup>.
707. The response from the bank
708. is either OK and the credit card [holder] completes the purchase by buying the goods or services offered by the selected shop,
709. or the response is “not OK”, and the transaction is skipped.

#### type

703 `AM = Int`

#### value

704 `buy: ci:CI  $\times$  (bi, sis):CM  $\rightarrow$`   
 704 **in, out** `ch_cb[ci, bi] out {ch_cs[ci, si]|si:SI $\bullet$ si  $\in$  sis}` **Unit**  
 704 `buy(ci, (bi, sis))  $\equiv$`   
 705 **let** `am:aM  $\bullet$  am > 0, si:SI  $\bullet$  si  $\in$  sis` **in**  
 706 **let** `msg = (ch_cb[ci, bi]!mk_Wdrw(am); ch_cb[ci, bi]?)` **in**  
 707 **case** `msg of`

<sup>5</sup>First the credit card [holder] requests a withdrawal. If sufficient funds are available, then the withdrawal takes place, otherwise not – and the credit card holder is informed accordingly.



```

708 is_OK() → ch_cs[ci,si]!mk_Buy(am),
709 is_NOK() → skip
704 end end end

```

710. The refund action is simple.

711. The credit card [handler] requests a refund  $am:AM$

712. from shop  $si:SI$ .

This request is handled by the shop behaviour's sub-action *ref*, see lines 728.–737. page 221.

**value**

```

710 rfu: ci:CI × (bi,si):CM → out {ch_cs[ci,si]|si:SI•si ∈ sis} Unit
710 rfu(ci,(bi,si)) ≡
711 let am:AM • am>0, si:SI • si ∈ sis in
712 ch_cs[ci,si]!mk_Ref(bi,(ci,si),am)
710 end

```

### G.3.5 Banks

713. The bank behaviour, *bnk*, takes the bank's unique identifier, the bank mereology, and the programmable attribute arguments: the ledger and the cash register. The bank behaviour, *bnk*, accepts inputs from and offers outputs to the any of the credit cards,  $ci \in cis$ , and any of the shops,  $si \in sis$ .

714. The bank behaviour non-deterministically externally chooses to accept either 'withdraw' requests from credit cards or 'deposit' requests from shops or 'refund' requests from credit cards.

**value**

```

713 bnkbi:BI: (cis,si):BM → (LG×CR) →
713 in,out {ch_cb[ci,bi]|ci:CI•ci ∈ cis} {ch_sb[si,bi]|si:SI•si ∈ sis} Unit
713 bnkbi((cis,si))(lg:(bal,doi,doe,...),cr) ≡
714 wdrw(bi,(cis,si))(lg,cr)
714 [] depo(bi,(cis,si))(lg,cr)
714 [] refu(bi,(cis,si))(lg,cr)

```

715. The 'withdraw' request, *wdrw*, (an action) non-deterministically, externally offers to accept input from a credit card behaviour and marks the only possible form of input from credit cards,  $mk\_Wdrw(ci,am)$ , with the identity of the credit card.

716. If the requested amount (to be withdrawn) is not within balance on the account

717. then we, at present, refrain from defining an outcome (**chaos**), whereupon the bank behaviour is resumed with no changes to the ledger and cash register;

718. otherwise the bank behaviour informs the credit card behaviour that the amount can be withdrawn; whereupon the bank behaviour is resumed notifying a lower balance and 'withdraws' the monies from the cash register.

**value**

```

714 wdrw: bi:BI × (cis, sis):BM → (LG × CR) → in, out {ch_cb[bi, ci] | ci:CI • ci ∈ cis} Unit
714 wdrw(bi, (cis, sis))(lg, cr) ≡
715 let mk_Wdrw(ci, am) = [] {ch_cb[ci, bi]? | ci:CI • ci ∈ cis} in
714 let (bal, doi, doe) = lg(ci) in
716 if am > bal
717 then (ch_cb[ci, bi]!is_NOK(); bnkbi(cis, sis)(lg, cr))
718 else (ch_cb[ci, bi]!is_OK(); bnkbi(cis, sis)(lg†[ci → (bal - am, doi, doe)], cr - am)) end
713 end end

```

The ledger and cash register attributes,  $lg, cr$ , are programmable attributes. Hence they are modeled as separate function arguments.

719. The **deposit** action is invoked, either by a shop behaviour, when a credit card [holder] buy's for a certain amount,  $am:AM$ , or requests a refund of that amount. The deposit is made by shop behaviours, either on behalf of themselves, hence  $am:AM$ , is to be inserted into the shops' bank account,  $si:SI$ , or on behalf of a credit card [i.e., a customer], hence  $am:AM$ , is to be inserted into the credit card holder's bank account,  $si:SI$ .
720. The message,  $ch\_cs[ci, si]?$ , received from a credit card behaviour is either concerning a **buy** [in which case  $i$  is a  $ci:CI$ , hence **sale**, or a **refund order** [in which case  $i$  is a  $si:SI$ ].
721. In either case, the respective bank account is “upped” by  $am:AM$  – and the bank behaviour is resumed.

**value**

```

719 deposit: bi:BI × (cis, sis):BM → (LG × CR) →
720 in, out {ch_sb[bi, si] | si:SI • si ∈ sis} Unit
719 deposit(bi, (cis, sis))(lg, cr) ≡
720 let mk_Dep(si, am) = [] {ch_cs[ci, si]? | si:SI • si ∈ sis} in
719 let (bal, doi, doe) = lg(si) in
721 bnkbi(cis, sis)(lg†[si → (bal + am, doi, doe)], cr + am)
719 end end

```

722. The **refund** action
723. non-deterministically externally offers to either
724. non-deterministically externally accept a  $mk\_Ref(ci, am)$  request from a shop behaviour,  $si$ , or
725. non-deterministically externally accept a  $mk\_Wdr(ci, am)$  request from a shop behaviour,  $si$ .

The bank behaviour is then resumed with the

726. credit card's bank balance and cash register incremented by  $am$  and the
727. shop' bank balance and cash register decremented by that same amount.

**value**

```

722 rfu: bi:BI × (cis, sis):BM → (LG × CR) → in, out {ch_sb[bi, si] | si:SI • si ∈ sis} Unit
722 rfu(bi, (cis, sis))(lg, cr) ≡
724 (let (si, mk_Ref(cbi, (ci, am))) = [] {(si', ch_sb[si, bi]? | si, si':SI • {si, si'} ⊆ sis ∧ si = si'} in
722 let (balc, doic, doec) = lg(ci) in
726 bnkbi(cis, sis)(lg†[ci → (balc + am, doic, doec)], cr + am)
722 end end)

```

```

723 []
724 (let (si,mk_Wdr(ci,am)) = [] {(si',ch_sb[si,bi]?)|si,si':SI•{si,si'}⊆sis∧si=si'} in
725 let (bals,dois,does) = lg(si) in
726 bnkbi(cis,sis)(lg†[si→(bals−am,dois,does)],cr−am)
727 end end)

```

### G.3.6 Shops

728. The shop behaviour, **shp**, takes the shop's unique identifier, the shop mereology, etcetera.

729. The shop behaviour non-deterministically, externally

either

730. offers to accept a Buy request from a credit card behaviour,

731. and instructs the shop's bank to deposit the purchase amount.

732. whereupon the shop behaviour resumes being a shop behaviour;

733. or

734. offers to accept a refund request in this amount, **am**, from a credit card [holder].

735. It then proceeds to inform the shop's bank to withdraw the refund from its ledger and cash register,

736. and the credit card's bank to deposit the refund into its ledger and cash register.

737. Whereupon the shop behaviour resumes being a shop behaviour.

#### value

```

728 shpsi:SI: (CI-set×BI)×...→in,out: {ch_cs[ci,si]|ci:CI•ci∈cis},{ch_sb[si,bi']|bi':BI•bi'isin bis} Unit

```

```

728 shpsi((cis,bi),...) ≡

```

```

730 (sal(si,(bi,cis),...)

```

```

733 []

```

```

734 ref(si,(cis,bi),...)):

```

```

728 sal: SI×(CI-set×BI)×...→in,out: {cs[ci,si]|ci:CI•ci∈cis},sb[si,bi] Unit

```

```

728 sal(si,(cis,bi),...) ≡

```

```

730 let mk_Buy(am) = [] {ch_cs[ci,si]?|ci:CI•ci∈cis} in

```

```

731 ch_sb[si,bi]!mk_Dep(si,am) end ;

```

```

732 shpsi((cis,bi),...)

```

```

728 ref: SI×(CI-set×BI)×...→in,out: {ch_cs[ci,si]|ci:CI•ci∈cis},{ch_sb[si,bi']|bi':BI•bi'isin bis} Unit

```

```

734 ref(si,(cis,sbi),...) ≡

```

```

734 let mk_Ref((ci,cbi,si),am) = [] {ch_cs[ci,si]?|ci:CI•ci∈cis} in

```

```

735 (ch_sb[si,cbi]!mk_Ref(cbi,(ci,si),am)

```

```

736 || ch_sb[si,sbi]!mk_Wdr(si,am)) end ;

```

```

737 shpsi((cis,sbi),...)

```



# Appendix H

## A Simple Retailer System

### Contents

---

|            |                                                                           |            |
|------------|---------------------------------------------------------------------------|------------|
| <b>H.1</b> | <b>Two Approaches to Modeling</b>                                         | <b>224</b> |
| H.1.1      | Domain Science & Engineering                                              | 224        |
| H.1.2      | HERAKLIT: <a href="http://heraklit.dfki.de/">http://heraklit.dfki.de/</a> | 225        |
| <b>H.2</b> | <b>The Retailer Market Case Study</b>                                     | <b>225</b> |
| H.2.1      | Three Rough Sketches                                                      | 225        |
| H.2.1.1    | Identification of “Main Players”                                          | 225        |
| H.2.1.2    | Main Transaction Sequences                                                | 226        |
| H.2.1.3    | Detailed Sketch                                                           | 226        |
| H.2.1.4    | Transitions                                                               | 228        |
| <b>H.3</b> | <b>Endurants: External Qualities</b>                                      | <b>229</b> |
| H.3.1      | Main Decompositions                                                       | 229        |
| H.3.2      | Aggregates as Sets                                                        | 230        |
| H.3.3      | The Retailer                                                              | 230        |
| H.3.3.1    | The HERAKLIT View                                                         | 230        |
| H.3.4      | The Market System State                                                   | 231        |
| <b>H.4</b> | <b>Endurants: Internal Qualities</b>                                      | <b>232</b> |
| H.4.1      | Unique Identifiers                                                        | 232        |
| H.4.2      | Mereology                                                                 | 233        |
| H.4.2.1    | Customer Mereology                                                        | 234        |
| H.4.2.2    | Order Management Mereology                                                | 234        |
| H.4.2.3    | Inventory Mereology                                                       | 234        |
| H.4.2.4    | Warehouse Mereology                                                       | 235        |
| H.4.2.5    | Supplier Mereology                                                        | 235        |
| H.4.2.6    | Courier Service Mereology                                                 | 236        |
| H.4.3      | Attributes                                                                | 236        |
| H.4.3.1    | Transactions                                                              | 236        |
| H.4.3.2    | Customer Attributes                                                       | 237        |
| H.4.3.3    | Order Management Attributes                                               | 238        |
| H.4.3.4    | Inventory Attributes                                                      | 239        |
| H.4.3.5    | Warehouse Attributes                                                      | 240        |
| H.4.3.6    | Supplier Attributes                                                       | 241        |
| H.4.3.7    | Courier Attributes                                                        | 242        |
| <b>H.5</b> | <b>Merchandise</b>                                                        | <b>242</b> |
| H.5.1      | “Unique Identity”                                                         | 243        |

|            |                                                                |            |
|------------|----------------------------------------------------------------|------------|
| H.5.2      | <b>“Mereology”</b>                                             | 243        |
| H.5.3      | <b>“Attributes”</b>                                            | 243        |
| H.5.4      | <b>Representation</b>                                          | 243        |
| <b>H.6</b> | <b>Perdurants</b>                                              | <b>243</b> |
| H.6.1      | <b>Channels</b>                                                | 244        |
| H.6.2      | <b>Behaviours</b>                                              | 244        |
| H.6.2.1    | <b>Customer Behaviour</b>                                      | 244        |
| H.6.2.2    | <b>Order Management Behaviour</b>                              | 245        |
| H.6.2.3    | <b>Inventory Behaviour</b>                                     | 248        |
| H.6.2.4    | <b>Warehouse Behaviour</b>                                     | 251        |
| H.6.2.5    | <b>Supplier Behaviour</b>                                      | 255        |
| H.6.2.6    | <b>Courier Service Behaviour</b>                               | 257        |
| H.6.3      | <b>System Initialisation</b>                                   | 258        |
| <b>H.7</b> | <b>Conclusion</b>                                              | <b>258</b> |
| H.7.1      | <b>Critique of the domain analysis &amp; description Model</b> | 258        |
| H.7.2      | <b>Proofs about Models</b>                                     | 259        |
| H.7.3      | <b>Comparison of Models</b>                                    | 259        |
| H.7.3.1    | <b>“Minor” Discrepancies</b>                                   | 259        |
| H.7.3.2    | <b>Use of Diagrams</b>                                         | 260        |
| H.7.3.3    | <b>Interleave versus “True” Concurrency</b>                    | 260        |
| H.7.4      | <b>What Next ?</b>                                             | 260        |

We report an exercise in modeling a retail system such as outlined in both [21, Bjørner, 2002] and [95, Fettke & Reisig, Dec. 21, 2020]. In the present exercise we try follow [95, Fettke & Reisig] – but we do so slavishly following the *domain analysis & description* method of [58, Domain Science & Engineering, Chapters 3–6, Bjørner 2021]<sup>1</sup> (domain science & engineering).<sup>2</sup>

## H.1 Two Approaches to Modeling

In this chapter we present a model of a customer/retailer/supplier/... market. We do it in the more-or-less classical style which emanated from the denotational-like formal specification of programming languages and lead to VDM [72,73,97] – and from there to RAISE [101]. There are other approaches to modeling discrete systems. One is by means of *symbolic Petri nets* [152–156].

### H.1.1 Domain Science & Engineering

At the center of domain science & engineering stands a *domain analysis & description* method. Domain analysis & description is first outlined in [38,41, Bjørner, 2010]. Domain analysis & description found a more final form in [51,56, Bjørner, 2016-2019]. [56] form the core chapters, Chapters 3–6, of [58]. That forthcoming Springer monograph, [58], covers the domain science & engineering concept: *domain science & engineering*.

In this report we shall *slavishly* follow the doctrines of the *domain analysis & description* method. First we consider endurants<sup>3</sup> and “within” our analysis & description of endurants we first focus on their so-called *external qualities* (“form, but not content”), then on *internal qualities: unique identifiers, mereology* and *attributes*. Then, by *transcendental deduction*, we “morph” some endurants into perdurants<sup>4</sup>, that is, behaviours. Here we first consider the *channels* and the messages sent

<sup>1</sup> [58] is scheduled to be published by Springer in their *EATCS Monographs in Theoretical Computer Science* series, Winter/Spring of 2021. Till such a time you may find an electronic copy at [www.imm.dtu.dk/~dibj/2020/mono/mono.pdf](http://www.imm.dtu.dk/~dibj/2020/mono/mono.pdf). That electronic copy may, from time to time, be updated as I “improve” on its text.

<sup>2</sup>Work on this document started December 28, 2020.

<sup>3</sup>Endurants, colloquially speaking, “end up” as data in the computer.

<sup>4</sup>Perdurants, colloquially speaking, “end up” as processes in the computer.

over channels between behaviours, before we consider these latter. We do so in the style of [RSL’s [100]] CSP [109–111, 161, 164].

It may seem a long beginning before we get to “process-oriented” modeling.

*But a worthwhile thing is worth doing right, hence carefully!*

It seems to this author that the HERAKLIT approach, in keeping with its name, from the first beginning, considers “all as flowing”, that is, as [Petri net-like] processes.

### H.1.2 HERAKLIT: <http://heraklit.dfki.de/>

Based on Petri net ideas [152–156] Wolfgang Reisig has conceived and researched HERAKLIT. In a number of reports and papers, [90–96], Peter Fettke and Wolfgang Reisig has developed the HERAKLIT theory & practice of modeling, what they call *service systems*.

The present report “mimics” [95, HERAKLIT case study: retailer] in providing a DOMAIN ANALYSIS & DESCRIPTION-oriented model of “the same” domain!

The HERAKLIT retailer case study [95] straddles three concerns: presenting the HERAKLIT methodology, its mathematical foundation and the retailer case study. The *domain analysis & description* case study presented in this chapter makes use of RSL, the RAISE Specification Language [100] – and can thus concentrate on the case study. The semantics of the RSL-expressed case study is that derived from the semantics of RSL, notably its CSP [109–111, 161, 164] “subset”.

## H.2 The Retailer Market Case Study

The following case study is based on [95]. It does not, in the present version, follow the domain of [95] strictly. But I am quite sure that any discrepancies can be easily incorporated into the present model.

### H.2.1 Three Rough Sketches

It is good domain modeling development practice to start a domain modeling project with one or more alternative rough sketch informal descriptions. But they are to be just such rough sketches. No formal meaning is to be attached to these rough sketches. They are meant to get the domain modeling project team “aligned”.

We present three, obviously “overlapping”, rough sketches.

#### H.2.1.1 Identification of “Main Players”

We **rough sketch** narrate a description of the domain.

The domain is that of a set of **customers**, a set of **retailers**, a set of **suppliers**, a set of **courier services**. **Retailers** each embody three sub-components: an **order management**, an **inventory**; and a **warehouse**.

See Fig. H.1 on the next page

Customers **order** merchandise from retailers’ order management. They in turn **order** that merchandise from their inventory [management]. If inventory [management] judges that they have the needed quantity in their warehouse, they **acknowledge** the order management. If inventory [management] judges that they do not have the needed quantity in their warehouse, they proceed to **order** a sufficient quantity of the desired merchandise from a supplier. The supplier eventually **deliver** a quantity to the warehouse of the ordering retailer. That warehouse **acknowledges** receipt to its inventory which eventually **acknowledges** that receipt to its order management. The order management **acknowledges** the customer order and notifies its warehouse of a proper **dispatch**.

<sup>5</sup>None of the figures in this report, Figs. H.1 on the following page, H.2 on page 227, H.3 on page 229, H.4 on page 230 and H.5 on page 231, are formal. That is, they do not add to or detract from the meaning of the formulas otherwise shown in this report. They merely “support”, by graphics, the narrative text.

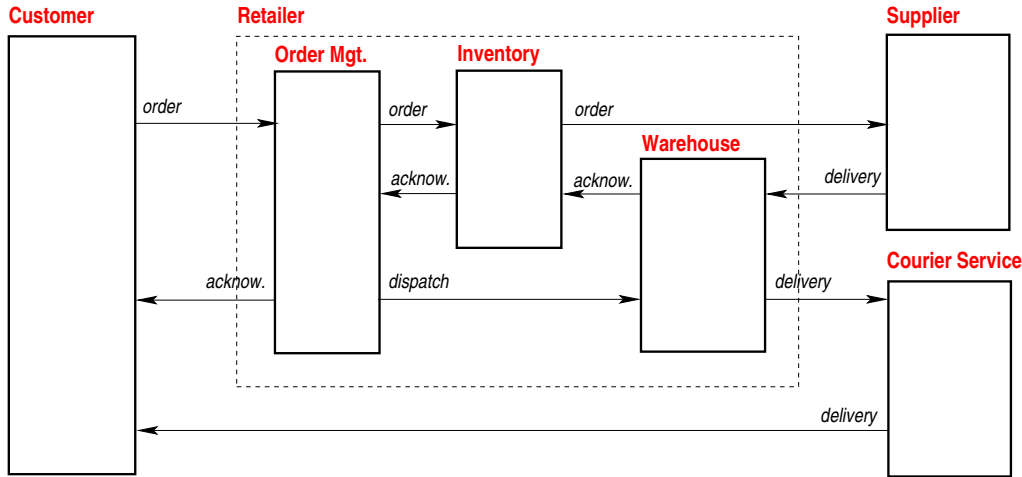


Figure H.1: A Market System

5

The warehouse **delivers** the desired merchandise quantity to a courier service which subsequently **delivers** that desired merchandise quantity to the customer.

It is thus we see that there are essentially three four kinds of transactions between market “players”: **orders**, **acknowledgments**, **dispatches** and **deliveries**.

In the formalisations to follow we shall refer to customers as *c:C*, order managements as *om:OM*, inventories as *iv:IV*, suppliers as *s:S* and courier services as *cs:CS*.<sup>6</sup>

### H.2.1.2 Main Transaction Sequences

**Customers** issue **purchase orders** for **merchandise** from retailers’ **order management**; receive **order acknowledgments** from retailers’ **order management**; and receive **customer delivered merchandise** (via retailers’ warehouses) from **courier services**.

**Retailers’ order management** **inquire** with its **inventory** as to the availability of ordered **merchandise**; await **acknowledgment** of **availability** (of **merchandise**) from its **inventory**; informs **customer** of availability (**order acknowledgment**); and **dispatch order** to **warehouse** when available.

**Retailers’ inventory** issues **acknowledgment** of **merchandise** to **order management**; issues **wholesale orders** for supply of **merchandise**, “when out-of-stock”, from **suppliers**; and receive **acknowledgment** of **supplies** from **suppliers**.

**Retailers’ warehouse** receive **merchandise deliveries** from **suppliers**; informs **inventory** management of **merchandise availability**; accepts **dispatch orders** from **order management**; and **forward merchandise** for such customer **merchandise** dispatches to **courier services**.

Etcetera.

### H.2.1.3 Detailed Sketch

We refer to Fig. H.2 on the next page.

- **A** It all starts with a customer issuing a purchase order. It is date-time stamped with the customers unique identifier.

<sup>6</sup>We shall, corresponding, prefix the transaction names: *C\_OM\_Order*, *OM\_IV\_Order*, *IV\_S\_Order*, *IV\_WH\_Delivery*, *WH\_CS\_Delivery*, *CS\_C\_Delivery*, *WH\_IV\_Ack*, *IV\_OM\_Ack*, *OM\_C\_Ack*, *OM\_WH\_Dispatch*, or some suitable variants thereof.



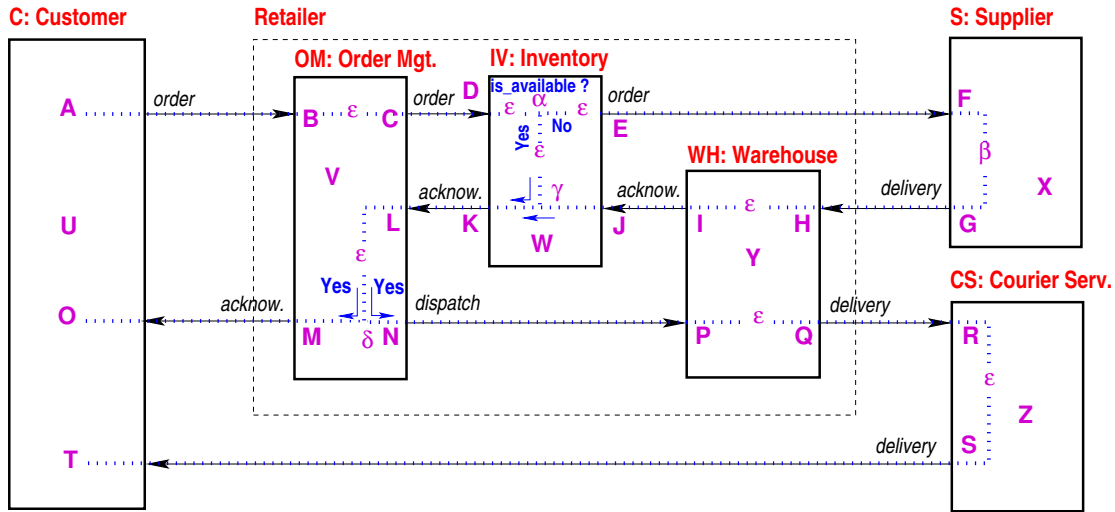


Figure H.2: Transaction Sequences

Since no customer can issue more than one such order at a time, such date-time-customer identification is unique and can serve as the unique customer order identification across the market.

Once the customer has issued the order request it either **O** awaits replies from some retailer's order management or **T** some courier service's delivery (of otherwise ordered products) or **U** resumes other business!

- **B** The customer order is received by some retailer's order management. That order management makes a note of the incoming order and posits that note in a 'work-to-do' dossier.
- $\epsilon_B$  At some time the order management selects an arbitrary "what-to-do-next" note from its dossier. If it is that if an customer order – arising from **B** – then it
- **C** issues an inquiry to its inventory as to the availability of the quantity of the named product of the customer order.
- **D** The inventory receives an inventory inquiry. The inventory makes a note of the incoming order and deposits that note in its 'work-to-do' dossier.
- $\epsilon_D$  At some time the inventory selects an arbitrary "what-to-do-next" note from its dossier. If it is that if an inventory inquiry – arising from **D** – then it
- $\alpha$  examines whether the quantity of the named product of the customer order is "on-hand" (in the retailer's warehouse, as recorded in the inventory).
- **E** If not the inventory issues a wholesale order to a supplier.
- **F** The supplier receives a wholesale order. The supplier makes a note of the wholesale order and deposits that note in its 'work-to-do' dossier.
- $\beta$  It may take same time to respond to the wholesale request. For example, if the supplier first has to manufacture or otherwise get hold of the requested supply.

- **G** Eventually the supplier transfers the requested quantity of named merchandise to the requesting retailer’s warehouse.
- **H** The warehouse receives this delivery and – eventually - stores it –
- **I** while notifying its inventory (management) of availability of the [previously] requested merchandise.
- **J** The inventory receives this notification. It make a note thereof and deposits it in its ‘work-to-do’ dossier.
- $\gamma$  Either inquiry  **$\alpha$**  lead to a positive result, or, as now (**M**) such an inquiry would be positive.
- **K** Eventually the inventory can inform order management of order availability.
- **L** Order management receives positive acknowledgment and deposits notes in its ‘work-to-do’ dossier as to acknowledging the customer of its order and informing the warehouse of its delivery.
- **M** Eventually order management gets around to service this note:
  - ( **$(D, \alpha, \text{Yes})$** ) and ( **$(D, \alpha, \text{No}, E, F, G, J, I, J)$** ) order management informs the customer of upcoming order delivery
  - **N** while also, “at the same time”, issuing an order dispatch to its warehouse.
- **O** The customer receives this information.
- **P** The warehouse receives this dispatch and makes a note thereof in its ‘work-to-do’ dossier.
- **Q** The warehouse eventually issues a delivery order, with ordered merchandise, to a courier service.
- **R** The courier service receives this delivery and makes a note thereof in its ‘work-to-do’ dossier.
- **S** The courier service eventually dispatches the delivery to the customer.
- **T** The customer, finally, receives the ordered quantity of merchandise.

#### H.2.1.4 Transitions

In the technical terms of **Petri nets**, the ten (10) horisontal arrows of Fig. H.2 on the preceding page represent transitions as in **Place-Transition nets**. They are labeled by pairs of upper case alphabetic characters: **A–B**, **C–D**, **E–F**, **G–H**, **I–J**, **K–L**, **M–N**, **O–P**, **Q–R**, and **S–T**. In the technical terms of CS, these ten transitions correspond to pairs of CSP input  $[ch[...]?]$  and output  $[ch[...]!msg]$  clauses. You will find these clauses **highlighted in blue** in Sect. H.6.2:

- **A–B**: Items 834 Page 245 and 849 Page 246
- **C–D**: Items 855 Page 246 and 877 Page 249
- **E–F**: Items 898 Page 251 and 944 Page 256
- **G–H**: Items 913 Page 252 and 952 Page 256
- **I–J**: Items 932 Page 254 and 880 Page 249
- **K–L**: Items 904 Page 251 and 858 Page 247
- **M–O**: Items 865 Page 248 and 836 Page 245
- **N–P**: Items 866 Page 248 and 916 Page 253
- **Q–R**: Items 938 Page 255 and 958 Page 257 and
- **S–T**: Items 963 Page 258 and 838 Page 245.

The pairs of formulas listed in each • above represents the **transition**. The formula text from the behaviour definition parameter line up up to the “transition” line defines the **place**. Thus the RSL/CSP definition that we shall present, in a sense, corresponds to place-transition nets where each transition has exactly two inputs and two outputs. The other way around: Place-transition nets where transitions have different numbers of inputs, respectively outputs, can be likewise “mimicked” by appropriate RSL/CSP definitions.

### H.3 Endurants: External Qualities

We now begin the proper, methodical description of the retailer, i.e., the market system. That description is presented in Sects. H.3–H.6.

We refer to [58, Chapter 3].

#### H.3.1 Main Decompositions

- |                                  |                                   |
|----------------------------------|-----------------------------------|
| <b>Narrative</b>                 | 740. a retailer aggregate,        |
| 738. Our market system comprises | 741. a supplier aggregate and     |
| 739. a customer aggregate,       | 742. a courier service aggregate. |

We consider all these aggregates to be *structures* in the sense of [58, Sect. 4.10].

#### Formalization

**type**

- 738. MKT
- 739. CSTa
- 740. RETa
- 741. SUPa
- 742. CSa

**value**

- 739. obs\_CSTa: MKT → CSTa
- 740. obs\_RETa: MKT → RETa
- 741. obs\_SUPa: MKT → SUPa
- 742. obs\_CSa: MKT → CSa

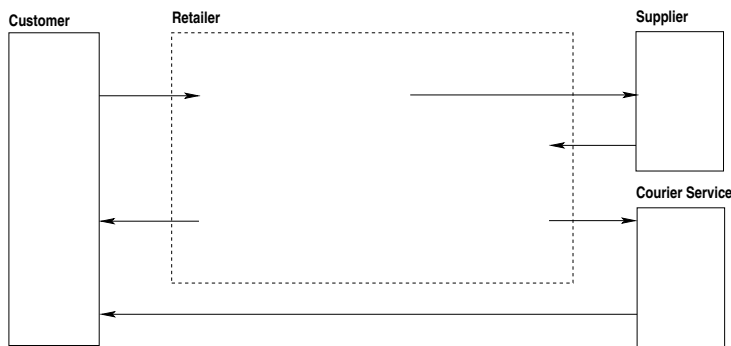


Figure H.3: A Simplified Market System

### H.3.2 Aggregates as Sets

#### Narrative

743. The customer aggregate form a set of one or more customers.  
 744. The retailer aggregate form a set of one or more retailers.  
 745. The supplier aggregate form a set of one or more suppliers.  
 746. The courier service aggregate form a set of one or more courier services.

We consider all these sets to be *structures* and the customers, suppliers and courier services to be *atoms* in the sense of [58, Sects. 4.10 and 4.13].

#### Formalization

##### type

743.  $CSTs = C\text{-set}$ , **axiom**  $\forall csts:CSTs \cdot csts \neq \{\}$   
 744.  $RETs = R\text{-set}$ , **axiom**  $\forall rets:CSTs \cdot rets \neq \{\}$   
 745.  $SUPs = S\text{-set}$ , **axiom**  $\forall sups:CSTs \cdot sups \neq \{\}$   
 746.  $CSs = CS\text{-set}$ , **axiom**  $\forall cts:CSs \cdot trss \neq \{\}$

##### value

743.  $obs\_CSTs: CSTa \rightarrow CSTs$   
 744.  $obs\_RETs: RETa \rightarrow RETs$   
 745.  $obs\_SUPs: SUPa \rightarrow SUPs$   
 746.  $obs\_CSs: CSa \rightarrow CSs$

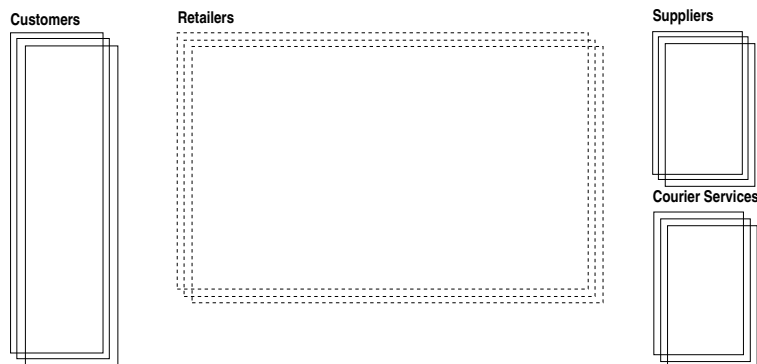


Figure H.4: Aggregates as Sets

### H.3.3 The Retailer

#### H.3.3.1 The HERAKLIT View

We focus on retailers. We treat retailers as *structures*<sup>7,8</sup> of three separately observable parts:

<sup>7</sup>We refer to [95, Sect. 3.10].

<sup>8</sup>We dash the retailer boxes to indicate their “structure”-ness.

**Narrative**

747. an *order management*,

748. an *inventory*<sup>9</sup> and

749. a *warehouse*.

We consider order managements, inventory managements and warehouses to be *atoms* in the sense of [58, Sects. 4.13].

**Formalization****type**

747. OM

748. IV

749. WH

**value**

747.  $\text{obs\_OM}: R \rightarrow \text{OM}$

748.  $\text{obs\_IV}: R \rightarrow \text{IV}$

749.  $\text{obs\_WH}: R \rightarrow \text{WH}$

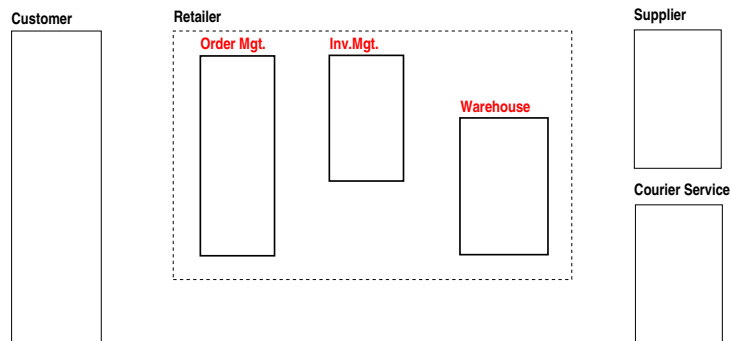


Figure H.5: The Retailer

The domain science & engineering View

Following the DS&E “approach”, i.e., “dogma”, retailers might normally have been decomposed into just two components: The order management and the warehouse. Inventory would then become a programmable attribute of order management.

**H.3.4 The Market System State**

We refer to [58, Sect. 3.18]. We postulate some market system *mkt*. It consists of

**Narrative**

750. the market, *mkt*;

751. all customers *cs*;

752. all retailer order managements *oms*;

<sup>9</sup>We might have modeled a retailer inventory as an attribute of the composite part retailer.

- 753. all retailer inventories *ivs*; and
- 754. all retailer warehouses *whs*;
- 755. all suppliers *ss*; and
- 756. all courier services *css*.

To obtain these we define respective extraction functions.

### Formalization

#### value

750.  $\text{mkt:MKT}$

751.  $\text{xtr\_Cs: MKT} \rightarrow \text{C-set}$

751.  $\text{xtr\_Cs(mkt)} \equiv \text{obs\_CSTs(obs\_CSTa(mkt))}$

751.  $\text{cs:C-set} := \text{xtr\_Cs(mkt)}$

752.  $\text{xtr\_OMs: MKT} \rightarrow \text{OM-sett}$

752.  $\text{xtr\_OMs(mkt)} \equiv \{\text{om|r:RET,om:OM}\bullet\mathbf{r} \in \text{obs\_RETs(obs\_RETa(mkt))} \wedge \text{om}=\text{obs\_OM(r)}\}$

752.  $\text{oms:OM-sett} := \text{xtr\_OMs(mkt)}$

753.  $\text{xtr\_IVS: MKT} \rightarrow \text{IV-set}$

753.  $\text{xtr\_IVS(mkt)} \equiv \{\text{iv|r:RET,iv:IV}\bullet\mathbf{r} \in \text{obs\_RETs(obs\_RETa(mkt))} \wedge \text{iv}=\text{obs\_IV(r)}\}$

753.  $\text{ivs:IV-set} := \text{xtr\_IVS(mkt)}$

754.  $\text{xtr\_WHs: MKT} \rightarrow \text{WH-set}$

754.  $\text{xtr\_WHs(mkt)} \equiv \{\text{wh|r:RET,wh:WH}\bullet\mathbf{r} \in \text{obs\_RETs(obs\_RETa(mkt))} \wedge \text{wh}=\text{obs\_WH(r)}\}$

754.  $\text{whs:WH-set} := \text{xtr\_WHs(mkt)}$

755.  $\text{xtr\_Ss: MKT} \rightarrow \text{S-set}$

755.  $\text{xtr\_Ss(mkt)} \equiv \text{obs\_SUPs(obs\_SUPa(mkt))}$

755.  $\text{ss:S-set} := \text{xtr\_Ss(mkt)}$

756.  $\text{xtr\_CSs: MKT} \rightarrow \text{CS-set}$

756.  $\text{xtr\_CSs(mkt)} \equiv \text{obs\_CSs(obs\_CSa(mkt))}$

756.  $\text{css:CS-set} := \text{xtr\_CSs(mkt)}$

## H.4 Endurants: Internal Qualities

### H.4.1 Unique Identifiers

We refer to [58, Sect. 5.2].

The concept of parts having unique identifiability, that is, that two parts, if they are the same, have the same unique identifier, and if they are not the same, then they have distinct identifiers, that concept is fundamental to our being able to analyse and describe internal qualities of endurants. So we are left with the issue of “sameness” !

#### Narrative

- 757. Customers, retailer order managements, retailer inventories, retailer warehouses, suppliers and courier services all have distinct unique identifiers.
- 758. By UI we designate the sort of all unique identifiers.

759. We define auxiliary functions which observe the unique identifiers of all customers, retailers, suppliers and courier services of a market system.

760. *uis* name the set of all unique identifiers.

### Formalization

#### type

757. C\_UI, OM\_UI, IV\_UI, WH\_UI, S\_UI, CS\_UI

758. UI = C\_UI | OM\_UI | IV\_UI | WH\_UI | S\_UI | CS\_UI

#### value

757. uid\_C: C → C\_UI

757. uid\_OM: OM → OM\_UI

757. uid\_IV: IV → IV\_UI

757. uid\_WH: WH → WH\_UI

757. uid\_S: S → S\_UI

757. uid\_CS: CS → CS\_UI

#### axiom

757.  $\forall c, c': C \bullet \{c, c'\} \subseteq cs \wedge c \neq c' \Rightarrow \text{uid\_C}(c) \neq \text{uid\_C}(c')$ ,

757.  $\forall om, om': OM \bullet \{om, om'\} \subseteq oms \wedge om \neq om' \Rightarrow \text{uid\_OM}(om) \neq \text{uid\_OM}(om')$ ,

757.  $\forall iv, iv': IV \bullet \{iv, iv'\} \subseteq ivs \wedge iv \neq iv' \Rightarrow \text{uid\_IV}(iv) \neq \text{uid\_IV}(iv')$ ,

757.  $\forall wh, wh': WH \bullet \{wh, wh'\} \subseteq whs \wedge wh \neq wh' \Rightarrow \text{uid\_WH}(wh) \neq \text{uid\_WH}(wh')$ ,

757.  $\forall s, s': S \bullet \{s, s'\} \subseteq ss \wedge s \neq s' \Rightarrow \text{uid\_S}(s) \neq \text{uid\_S}(s')$ ,

757.  $\forall cs, cs': CS \bullet \{cs, cs'\} \subseteq css \wedge cs \neq cs' \Rightarrow \text{uid\_CS}(cs) \neq \text{uid\_CS}(cs')$ .

#### value

759. xtr\_C\_UIs: MKT → CI-set

759.  $\text{xtr\_C\_UIs}(\text{mkt}) \equiv \{\text{uid\_C}(c) \mid c: C \bullet c \in cs\}$

759. xtr\_OM\_UIs: MKT → OMI-set

759.  $\text{xtr\_OM\_UIs}(\text{mkt}) \equiv \{\text{uid\_OM}(om) \mid om: OM \bullet om \in oms\}$

759. xtr\_IV\_UIs: MKT → IVI-set

759.  $\text{xtr\_IV\_UIs}(\text{mkt}) \equiv \{\text{uid\_IV}(iv) \mid iv: IV \bullet iv \in ivs\}$

759. xtr\_WH\_UIs: MKT → WHI-set

759.  $\text{xtr\_WH\_UIs}(\text{mkt}) \equiv \{\text{uid\_WH}(wh) \mid wh: WH \bullet wh \in whs\}$

759. xtr\_S\_UIs: MKT → SI-set

759.  $\text{xtr\_S\_UIs}(\text{mkt}) \equiv \{\text{uid\_S}(s) \mid s: S \bullet s \in ss\}$

759. xtr\_CS\_UIs: MKT → CSI-set

759.  $\text{xtr\_CS\_UIs}(\text{mkt}) \equiv \{\text{uid\_CS}(cs) \mid cs: CS \bullet cs \in css\}$

759. *cuis*: CUI-set = xtr\_C\_UIs(mkt)

759. *omuis*: OMUI-set = xtr\_OM\_UIs(mkt)

759. *ivuis*: IVUI-set = xtr\_IV\_UIs(mkt)

759. *whuis*: WHUI-set = xtr\_WH\_UIs(mkt)

759. *suis*: SUI-set = xtr\_S\_UIs(mkt)

759. *csuis*: CSUI-set = xtr\_CS\_UIs(mkt)

760. *uis*: UI-set = *cuis* ∪ *omuis* ∪ *ivuis* ∪ *whuis* ∪ *suis* ∪ *csuis*

#### axiom

759. **card** *cuis* + **card** *omuis* + **card** *ivuis* + **card** *whuis* + **card** *suis* + **card** *csuis* = **card** *uis*

## H.4.2 Mereology

We refer to [58, Sect. 5.3].

Mereology, as a logical/philosophical discipline, can perhaps best be attributed to the Polish mathematician/logician Stanisław Leśniewski [47, 81].

Which are the relations that can be relevant for “endurant-hood”? There are basically two relations: (i) physical ones, and (ii) conceptual ones.

(i) Physically two or more endurants may be topologically either adjacent to one another, like rails of a line, or within an endurant, like links and hubs of a road net, or an atomic part is conjoined to one or more materials, or a material is conjoined to one or more parts. The latter two could also be considered conceptual “adjacencies”.

(ii) Conceptually some parts, like automobiles, “belong” to an embedding endurant, like to an automobile club, or are registered in the local department of vehicles, or are ‘intended’ to drive on roads

#### H.4.2.1 Customer Mereology

##### Narrative

761. The mereology of a customer is a pair:

- the set of all retail order management identifiers and
- the set of all courier service identifiers.

##### Formalization

###### type

761.  $C\_Mer = OM\_UI\text{-set} \times CSU\_I\text{-set}$

###### value

761.  $mereo\_C: C \rightarrow C\_Mer$

761.  $mereo\_C(c) \equiv (omuis, csuis)$

#### H.4.2.2 Order Management Mereology

##### Narrative

762. The mereology of an order management is the triplet of

- the set of all customer identifiers,
- the unique identifier of the retailer’s inventory and
- the unique identifier of the retailer’s warehouse.

##### Formalization

###### type

762.  $OM\_Mer = C\_UI\text{-set} \times IV\_UI \times WH\_UI$

###### value

762.  $mereo\_OM: OM \rightarrow OM\_Mer$

762.  $mereo\_OM(om) \equiv$

762. **let**  $r:R \bullet r \in obs\_RETs(obs\_RETA(mkt)) \wedge om = obs\_OM(r)$  **in**  
 762.  $(cis, uid\_IV(obs\_IV(r)), uid\_WH(obs\_WH(r)))$  **end**

762. **pre:**  $\exists r:R \bullet r \in obs\_RETs(obs\_RETA(mkt)) \wedge om = obs\_OM(r)$

#### H.4.2.3 Inventory Mereology

##### Narrative

763. The mereology of an inventory is a triplet of

- the unique identifier of that inventory’s order management,
- the unique identifier of that inventory’s warehouse and
- the set of all supplier identifiers.



**Formalization****type**763.  $IV\_Mer = OM\_UI \times WH\_UI \times S\_UI\text{-set}$ **value**763. mereo\_IV:  $IV \rightarrow IV\_Mer$ 763. mereo\_IV(iv)  $\equiv$ 763. **let**  $r:R \bullet r \in \text{obs\_RETs}(\text{obs\_RETA}(mkt)) \wedge iv = \text{obs\_IV}(r)$  **in**763.  $(\text{uid\_OM}(\text{obs\_OM}(r)), \text{uid\_WH}(\text{obs\_WH}(r)), \text{suiss})$  **end**763. **pre:**  $\exists r:R \bullet r \in \text{obs\_RETs}(\text{obs\_RETA}(mkt)) \wedge iv = \text{obs\_IV}(r)$ **H.4.2.4 Warehouse Mereology**

narrative

764. The mereology of a warehouse is a quadruplet of

- the warehouse retailer's order management identifier,
- the warehouse retailer's inventory identifier,
- the set of all supplier identifiers, and
- the set of all courier service identifiers,

**Formalization****type**764.  $WH\_Mer = OM\_UI \times IV\_UI \times \_SUI\text{-set} \times CS\_UI\text{-set}$ **value**764. mereo\_WH:  $WH \rightarrow WH\_Mer$ 764. mereo\_WH(wh)  $\equiv$ 764. **let**  $r:R \bullet r \in \text{obs\_RETs}(\text{obs\_RETA}(mkt)) \wedge iv = \text{obs\_WH}(wh)$  **in**764.  $(\text{uid\_OM}(\text{obs\_OM}(r)), \text{uid\_IV}(\text{obs\_IV}(r)), \text{suiss}, \text{csuiss})$  **end**764. **pre:**  $\exists r:R \bullet r \in \text{obs\_RETs}(\text{obs\_RETA}(mkt)) \wedge wh = \text{obs\_WH}(r)$ **H.4.2.5 Supplier Mereology****Narrative**

765. The mereology of a supplier is a pair:

- the set of all inventory identifiers and
- the set of all warehouse identifiers.

**Formalization****type**765.  $S\_Mer = IV\_UI\text{-set} \times WH\_UI\text{-set}$ **value**765. mereo\_S:  $S \rightarrow S\_Mer$ 765. mereo\_S(s)  $\equiv (\{\text{uid\_IV}(iv) \mid iv:IV \bullet iv \in \text{ivuis}\}, \{\text{uid\_WI}(wh) \mid wh:WH \bullet wh \in \text{whs}\})$

#### H.4.2.6 Courier Service Mereology

##### Narrative

766. The mereology of a courier service is a pair

- the set of all warehouse identifiers and
- the set of all customer identifiers.

##### Formalization

###### type

766.  $CS\_Mer = WH\_UI\text{-set} \times CS\_UI\text{-set}$

###### value

766.  $mereo\_CS: CS \rightarrow CSMer$

766.  $mereo\_CS(t) \equiv (\{uid\_WI(w)|wh:WH \bullet wh \in whs\}, cuis)$

#### H.4.3 Attributes

We refer to [58, Sects. 5.4–5.5].

To recall: there are three sets of **internal qualities**: unique identifiers, part mereology and attributes. Unique identifiers and mereology are rather definite kinds of internal enduring qualities; attributes form more “free-wheeling” sets of internal qualities.

Since one can talk about transaction events between the six “players”, i.e., the customers, order managements, inventories, warehouses, suppliers and courier services of the ‘market’ we must, really, consider their transaction histories as [programmable] attributes.

In order to deal with the attributes of these six “players” we really need first consider what they are all focused on: namely the merchandise, i.e., products, they order, store, supply and deliver. For this we refer to Sect. H.5 on page 242.

##### H.4.3.1 Transactions

The ‘market’ is a typical *transaction-oriented* system. By a *transaction* we shall mean an *event* involving two or more “exchanges” of messages between two behaviours. Behaviours will be defined as the result of *transcendental deductions* of part endurants. With part endurants we associate attributes.

Since we can “talk” about events that “occur to parts”, that is, as behaviour properties, we shall attribute some of these events to parts. So parts are attributed the transactions in which their behaviours engage (with other behaviours).

Since we can “talk” about “such-and-such” a transaction having been initiated by a behaviour at such-and-such a time, we shall provide, with each transaction, a prefix of one or more time-stamped unique identifiers of the part/behaviour issuing the transaction.

767. DaTi refers to **TIME**. We refer to [58, Sect. 2.5]. The expression `record_TIME` yields a **TIME**. You should think of **TIMEs**, for example, as of the form `March 12, 2024: 10:48 am and 32 seconds` (day, month, year, hour, minute, second).

768. A transaction prefix is either a pair of a customer identifier and a date-time, or is a pair of a pair of order management, inventory, warehouse, supplier or courier service identifier and a date-time, and a transaction prefix.

The specific details of the pairings of unique identifiers and data-times is given in Items 769–778.

**Formalization****type**

767. DaTi = TIME

768. UI\_Pref = ...

**axiom**

768. [ ... ]

- |                                              |                                  |
|----------------------------------------------|----------------------------------|
| 768. More specifically the prefixes are the: | 774. acknowledge availability,   |
| 769. purchase order,                         | 775. order acknowledgment,       |
| 770. order inquiry,                          | 776. dispatch order,             |
| 771. wholesale order,                        | 777. forward merchandise and the |
| 772. merchandise delivery,                   | 778. customer delivery prefixes. |
| 773. merchandise availability,               |                                  |

**Formalization****type**

768. UI\_Pref = C\_OM\_Pref | OM\_IV\_Pref | IV\_S\_Pref | S\_WH\_Pref | WH\_IV\_Pref | IV\_OM\_Pref

768. | OM\_C\_Pref | OM\_WH\_Pref | WH\_CS\_Pref | CS\_C\_Pref

769. C\_OM\_Pref = (CUI × DaTi)

770. OM\_IV\_Pref = (OMUI × DaTi) × C\_OM\_Pref

771. IV\_S\_Pref = (IVUI × DaTi) × OM\_IV\_Pref

772. S\_WH\_Pref = (SUI × DaTi) × IV\_S\_Pref

773. WH\_IV\_Pref = (WHUI × DaTi) × S\_WH\_Pref

774. IV\_OM\_Pref = (IVUI × DaTi) × (OM\_IV\_Pref | WH\_IV\_Pref)

775. OM\_C\_Pref = (OMUI × DaTi) × IV\_OM\_Pref

776. OM\_WH\_Pref = (OMUI × DaTi) × OM\_C\_Pref

777. WH\_CS\_Pref = (WHUI × DaTi) × OM\_WH\_Pref

778. CS\_C\_Pref = (CSUI × DaTi) × WH\_CS\_Pref

Two customer to order management to inventory etc. transaction prefixes might then schematically be:

**H.4.3.2 Customer Attributes**

In order to go about their business of being customers, customers maintain, somehow or other, in their mind, on paper, or otherwise, a number of notes – which we shall refer to as attributes.

To express some of these attributes we need first introduce some auxiliary types.

**Narrative**

779. Customers, besides unique identity, have further information: customer names, addresses, telephone nos., e-mail addresses, etc.

780. Customers have bank/credit card, i.e., payment refs.

781. An order comprises a product name, a quantity, the total price, and a payment reference.

For simplicity we shall carry this '*order*' information forward in all market transactions.

- 782. Customers transact with retailer order managements and courier Services:
- 783. send purchase order to retailers;
- 784. receive positive acknowledgment on these orders; and
- 818. accept customer deliveries: a set of merchandise.

Transactions sent by customers are time-stamped with customers identity. Transactions received by customers are time-stamped [with a time-ordered, latest transaction first] grouping of handler identifications (ui:UI) – where order managements, inventories, suppliers, warehouses and courier services are the handlers.

### Formalization

#### type

- 779. CustName, CustAddr, CustPhon, CustEmail, ...
- 779. CustInfo = CustName  $\times$  CustAddr  $\times$  CustPhon  $\times$  CustEmail  $\times$  ...
- 780. PayRef
- 781. Order = (ProdNm  $\times$  Quant  $\times$  Price  $\times$  PayRef)
- 782. C-Trans = C\_OM\_Order | OM\_C\_Ack | CS\_C\_Del
- 783. C\_OM\_Order :: C\_OM\_Pref  $\times$  Order
- 784. OM\_C\_Ack :: OM\_C\_Pref  $\times$  Order
- 818. CS\_C\_Del :: CS\_C\_Pref  $\times$  Order  $\times$  (M-set|MI-set)

Now the attributes.

### Narrative

- 785. Customers keep a catalog of merchandise: from whom to order, price, etc. [Simplifying we consider this a static attribute.]
- 786. Customers keep all the merchandise they have acquired. [A programmable attribute.]
- 787. Customers can recall [a programmable attribute] the time-stamped transactions it has taken part in wrt. retailer order managements and courier services.

### Formalization

#### type

- 785. C-Catalog = ...
- 786. C-Merchandise = M-set
- 787. C\_TransHist = C\_Trans\*

#### axiom

- 787.  $\forall$  cth:C\_TransHist • [list is time-ordered]

#### value

- 785. attr\_C\_Catalog: C  $\rightarrow$  C\_Catalog
- 786. attr\_C\_Merchandise: C  $\rightarrow$  C\_Merchandise
- 787. attr\_C\_TransHist: C  $\rightarrow$  CustTransHist

### H.4.3.3 Order Management Attributes

#### Narrative

- 788. Order management partakes in several transactions:
  - 783. accepting customer purchase orders;

789. passing on that order to its inventory;
798. accepting product availability acknowledgment from the inventory;
784. informing customer of product availability; and,
790. when available, directing a dispatch order to its warehouse.
791. Order management makes note of accepted, i.e., incoming messages, (**B** and **L**) by keeping a [programmable attribute] ‘Work-to-do’ “notice board” [a “basket”, a “dossier”].
788.  $OM\_Trans = C\_OM\_Order \mid OM\_IV\_Order \mid IV\_OM\_Ack \mid OM\_C\_Ack \mid OM\_WH\_Dispatch$
783.  $C\_OM\_Order :: C\_OM\_Pref \times Order$
789.  $OM\_IV\_Order :: OM\_IV\_Pref \times Order$
798.  $IV\_OM\_Ack :: IV\_OM\_Pref \times Order$
784.  $OM\_C\_Ack :: OM\_C\_Pref \times Order$
790.  $OM\_WH\_Dispatch :: OM\_WH\_Pref \times Order$

Now the attributes.

### Narrative

792. An order management ‘work-to-do’ dossier keeps a set of zero or more notes: customer orders and inventory acknowledgments.
793. Order management records [a static attribute] which suppliers supply which products.
794. Order management also records the programmable order management transaction history `OMTransHist` attribute records a time-stamped list of all order management transactions, be they vis-a-vis customers, and its retailer’s inventory.

### Formalization

#### type

792.  $OM\_WorkToDo = (C\_OM\_Order \mid IV\_OM\_Ack)\text{-set}$

793.  $OM\_ProdSupp = ProdNm \xrightarrow{m} SUI\text{-set}$

794.  $OM\_TransHist = OM\_Trans^*$

#### value

792.  $attr\_OM\_WorkToDo: OM \rightarrow OrdrMgtWorkToDo$

793.  $attr\_OM\_ProdSupp: OM \rightarrow ProdSupp$

794.  $attr\_OM\_TransHist: OM \rightarrow OM\_TransHist$

#### H.4.3.4 Inventory Attributes

### Narrative

795. Inventories partakes in several transactions:
789. accepting merchandise orders from their order management,
796. issuing wholesale order requests to a designated supplier,
797. accepting order acknowledgments from their warehouse, and
798. issuing merchandise availability messages to their order management.

**Formalization**

795.  $IV\_Trans = OM\_IV\_Order \mid IV\_OM\_Ack \mid IV\_S\_Order \mid WH\_IV\_Ack$   
 796.  $IV\_S\_Order :: IV\_S\_Pref \times Order \times S\_UI$   
 797.  $WH\_IV\_Ack :: WH\_IV\_Pref \times Order \times WH\_UI$   
 798.  $IV\_OM\_Ack :: IV\_OM\_Pref \times Order$

**Narrative**

799. An inventory ‘work-to-do’ dossier (a programmable attribute) keeps a set of zero or more notes: inventory (merchandise availability) inquiry and merchandise availability.  
 800. The inventory (a programmable attribute) records, for every product name, its information (as listed in Items 824–829 Page 243), the name of the supplier, and the stock-in-hand.  
 801. The inventory also records the programmable inventory transaction history  $IVTransHist$  attribute records a time-stamped list of all inventory transactions, be they vis-a-vis order management, its retailer’s warehouse or a supplier.

**Formalization****type**

799.  $IV\_WorkToDo = (IV\_S\_Order \mid IV\_OM\_Ack)\text{-set}$   
 800.  $IV\_Inventory = ProdNm \xrightarrow{m} (WhoSalPrice \times SugRetPrice \times SalPrice \times MInfo \times SupNm \times IV\_Stock)$   
 799.  $IV\_Stock = \mathbf{Nat}$   
 801.  $IV\_TransHist = IVTrans^*$

**value**

799.  $attr\_IV\_WorkToDo: IV \rightarrow IV\_WorkToDo$   
 800.  $attr\_IV\_Inventory: IV \rightarrow Inventory$   
 801.  $attr\_IV\_TransHist: IV \rightarrow IV\_TransHist$

**H.4.3.5 Warehouse Attributes****Narrative**

802. Warehouses partake in four kinds of transactions:  
 803. being delivered sets of a product named merchandise from suppliers,  
 804. informing its inventory of (wholesale) supplier delivery,  
 805. being ordered by its order management, to dispatch merchandise to customers and  
 806. delivering merchandise to couriers (for them to deliver to customers).

**Formalization**

802.  $WH\_Trans = S\_WH\_Del \mid WH\_IV\_Ack \mid OM\_WH\_Del \mid WH\_CS\_Del$   
 803.  $S\_WH\_Del = S\_WH\_Pref \times Order \times M\text{-set}$   
 804.  $WH\_IV\_Ack = WH\_IV\_Pref \times Order$   
 805.  $OM\_WH\_Dispatch = OM\_WH\_Pref \times Order$   
 806.  $WH\_CS\_Del = WH\_CS\_Pref \times Order \times M\text{-set}$

**Narrative**

- 807.
808. The programmable warehouse `Store` attribute reflects, for every product name the zero, one or more merchandise of that name.
809. The programmable warehouse `WHTransHist` attribute records a time-stamped list of all warehouse transactions, be they vis-a-vis suppliers, its retailer’s inventory, its retailer’s order management, and customers.

**Formalization****type**

807.  $WH\_WorkToDo = (S\_WH\_Del | OM\_WH\_Dispatch)\text{-set}$
808.  $WH\_Store = ProdName \xrightarrow{m} M\text{-set}$
809.  $WH\_TransHist = WH\_Trans^*$

**value**

807.  $attr\_WH\_WorkToDo: WH \rightarrow WH\_WH\_WorkToDo$
808.  $attr\_WH\_Store: WH \rightarrow WH\_Store$
809.  $attr\_WH\_TransHist: WH \rightarrow WH\_TransHist$

**H.4.3.6 Supplier Attributes****Narrative**

810. Suppliers, in this model, partake in two transactions:
811. accepting wholesale orders for merchandise from retailers’ inventories, and
812. delivering such merchandise orders to retailers’ warehouses.

**Formalization**

810.  $S\_Trans = IV\_S\_Order | S\_WH\_Del$
811.  $IV\_S\_Order = IV\_S\_Pref \times Order$
812.  $S\_WH\_Del = S\_WH\_Pref \times Order \times M\text{-set}$

**Narrative**

813. The programmable supplier attribute `S_WorkToDo` temporarily contains ‘replicas’ of “incoming” `IV_S_Orders`.
814. The programmable supplier attribute `S_Products` reflects, for every product name a sufficient<sup>10</sup> number of merchandise of that name.
815. The programmable supplier attribute `S_TransHist` records a time-stamped list of all supplier transactions, be they vis-a-vis retailers’ inventory, and retailers’ warehouses.

**Formalization****type**813.  $S\_WorkToDo = IV\_S\_Order\text{-set}$ 814.  $S\_Products = ProdNm \xrightarrow{\mapsto} M\text{-set}$ 815.  $S\_TransHist = S\_Trans^*$ **value**813.  $attr\_S\_WorkToDo: S \rightarrow S\_WorkToDo$ 814.  $attr\_S\_Products: S \rightarrow S\_Products$ 815.  $attr\_S\_TransHist: S \rightarrow S\_TransHist$ **H.4.3.7 Courier Attributes****Narrative**

816. Courier services, in this model, partake in two transactions:

817. accepting merchandise delivery orders to customers from retailers' order management, and

818. delivering merchandise to customers

**Formalization****type**816.  $CS\_Trans = WH\_CS\_Del \mid CS\_C\_Del$ 817.  $WH\_CS\_Del = WH\_CS\_Pref \times Order \times M\text{-set}$ 818.  $CS\_C\_Del = CS\_C\_Pref \times Order \times M\text{-set}$ **Narrative**819. The programmable courier service attribute  $CS\_WorkToDo$  reflects current, "live" deliveries, and820. the programmable attribute  $CS\_TransHist$  the time stamped history of transactions.**Formalization****type**819.  $CS\_WorkToDo = CS\_C\_Del\text{-set}$ 820.  $CS\_TransHist = CS\_Trans^*$ **value**819.  $attr\_CS\_WorkToDo: CS \rightarrow CS\_WtD$ 820.  $attr\_CS\_TransHist: CS \rightarrow CS\_TransHist$ **H.5 Merchandise**

Merchandise (in [95]: Goods) are, using DS&E, modeled as parts. In [95] they are not considered beyond being somehow identified. It is not clear.

821. We shall model merchandise as atomic parts.

**type**821.  $M$



**H.5.1 “Unique Identity”**

822. As parts merchandise have unique identity.

823. Although we shall treat merchandise as behaviours we shall assume that merchandise identities are distinct from any other unique identities of the market.

**type**

822. MI

**value**

822. uid\_M: M → MI

**axiom**

823.  $\forall m:M \cdot \text{uid}_M(m) \notin \text{cuis} \cup \text{omuis} \cup \text{ivuis} \cup \text{whuis} \cup \text{suis} \cup \text{tuis}$

**H.5.2 “Mereology”**

Although merchandise, throughout its lifetime, can be related to suppliers, warehouses, courier services and customers we shall omit modeling the mereology of merchandise.

**H.5.3 “Attributes”**

We suggest the following merchandise attributes:

824. product name;

825. wholesale price;

826. suggested retail price;

827. sales price;

828. actual price;

829. further product information: goods category, weight, packaging measures, volume, manufacturer (with place-of-origin), manufacturing date, sale-by-date, an “how-to-use” guide, guarantee, etc., etc.

**type**

824. ProdNm

825. WhoSalPrice

826. SugRetPrice

827. SalPrice

828. ActPrice

829. ProdInfo

**H.5.4 Representation**

We shall not be concerned with the representation of attributes.

**H.6 Perdurants**

We refer to [58, Chapters 6–7].

By transcendental deduction we now “morph” endurants into perdurants. Parts “morph” into behaviours, here modeled in the style of CSP. Their mereology determine the *channels* between part processes.

### H.6.1 Channels

We refer to [58, Sect. 7.5].

In this report we shall postulate a channel array indexed by pairs (expressed as two-element sets) of unique identifiers. These identifiers are prescribed in the mereology of the relevant parts.

830. So there is a channel whose index sets allow the expression of communication between customers, order management, inventories, warehouses, suppliers and courier services.

831. The type of the messages communicated is the union type of the customer, order management, inventory [management], warehouse, supplier and courier service transactions.

#### channel

830.  $\{\text{ch}[\{\text{ui}, \text{ui}'\}] \mid \text{ui}, \text{ui}' : \text{UI} \bullet \text{ui} \neq \text{ui}' \wedge \{\text{ui}, \text{ui}'\} \subseteq \text{uis}\} : \text{Channel\_Trans}$

#### type

831.  $\text{Channel\_Trans} = \text{C\_Trans} \mid \text{OM\_Trans} \mid \text{IV\_Trans} \mid \text{WH\_Trans} \mid \text{S\_Trans} \mid \text{CS\_Trans}$

### H.6.2 Behaviours

We refer to [58, Sects. 7.6–7.8].

There now follows a sequence of informal narrative and formal specification texts. The formal texts, in a sense, are a culmination of all the previous formal definitions. The formulas involve rather many identifiers. Some are defined locally, some as behaviour function definition parameters, others in previous formal definitions.

#### H.6.2.1 Customer Behaviour

##### Narrative

832. Customers alternate between retailer shopping and otherwise going about their daily life.

Shopping manifests itself in three related events:

- **A** the customer issuing a purchase order;
- **O** the receiving of acknowledgment of upcoming delivery;
- **T** the final acceptance of delivery.

Daily life is “modeled” by **T**. Customers alternate, internal non-deterministically,  $\square$ , between these four events.

**A** When internal non-deterministically choosing to order merchandise, the customer must decide on which retailer, product, how many and at what cost.

833. The customer then assembles a purchase order

834. which it sends to some retailer’s order management.

We refer to [58, Sect. 2.5.3] for understanding the rôle of `record_TIME`.

We presently omit defining `date`.

835. Whereupon the customer resumes being a customer, however with updated transaction history.

836. **O** At some time the customer receives an acknowledgment from a retailer’s order management as to the [positive] acceptance of an order which was purchased some while ago (`omui,dati`).

837. The customer records this in its transaction history while resuming being a customer
838. **T** At some time the customer receives the delivery of previously ordered merchandise.
839. The customer records the identities (as well as the merchandise) and
840. resumes being a customer.
841. **U** Et cetera.<sup>11</sup>

**value**

832. **C**:  $c\_ui:CUI \times c\_mer:(omuis, csuis):C\_Mer \times C\_Catalog \rightarrow (C\_Merchandise \times C\_TransHist)$
832. **in out** {  $ch[\{c\_ui, om\_ui\}] \mid om\_ui:OMUI \bullet om\_ui \in omuis$  }
832. **in** {  $ch[\{c\_ui, cs\_ui\}] \mid cs\_ui:CSUI \bullet cs\_ui \in csuis$  } **Unit**
832. **C\_Beh**( $c\_ui, c\_mer:(omuis, csuis), c\_ctlg$ )( $c\_merch, c\_hist$ )  $\equiv$
832. **A** **let** ( $om\_ui, order$ ) =  $decide\_on\_purchase((custinfo, mertbl), c\_hist)$  **in**
834. **A-B**  $ch[\{c\_ui, om\_ui\}] ! \text{ordr}:C\_OM\_Order(((c\_ui, record\_TIME()), order) ;$
835. **C**( $c\_ui, c\_mer, c\_ctlg$ )( $c\_merch, \langle ordr \rangle^{\wedge} c\_hist$ ) **end**
836.  $\sqcap$  **O** **let** **M-O**  $ack:OM\_C\_Ack(prefix, order) = ch[\{om\_ui, c\_ui\}] ?$  **in**
837. **C**( $cui, cmer, c\_ctlg$ )( $merch, \langle ack \rangle^{\wedge} c\_hist$ ) **end**
838.  $\sqcap$  **T** **let** **S-T**  $del:CS\_C\_Del(prefix, order, ms) = ch[\{cs\_ui, c\_ui\}] ?$  **in**
839. **let**  $ms\_uis = \{uid\_MI(m) \mid m:M \bullet m \in ms\}$  **in**
840. **C**( $c\_ui, c\_mer, c\_ctlg$ )( $merch \cup ms, \langle CS\_C\_Del(prefix, order, ms\_uis) \rangle^{\wedge} c\_hist$ ) **end end**
841.  $\sqcap$  **U** ... **C**( $cui, cmer, ctlg'$ )( $merch', c\_hist$ )

### H.6.2.2 Order Management Behaviour

#### Narrative

842. Being order management, **OM**, manifests itself in six events:
843. **B** accepting customer order,
844. **C** offering inventory order,
845. **L** accepting inventory acknowledgment,
846. **M** offering **OM** acknowledgment acknowledgment to customer, and
- N** offering dispatch order to warehouse, and
847. **V** doing other **OM** business.
848. The **OM** behaviour internal non-deterministically (843., 844., 845., 846. and 847.) alternates between **B, C, L, M, N** and **V**:

842. **OM**:  $om\_ui:OM\_UI \times (ommer:(cuis, ivui, whui)):OM\_Mer \times OM\_ProdSupp \rightarrow$
842.  $(OM\_WorkToDo \times OM\_TransHist)$
842. **in out** {  $ch[\{c\_ui, om\_ui\}] \mid c\_ui:CUI \bullet c\_ui \in cuis$  }
842. **in out**  $ch[\{om\_ui, iv\_ui\}]$  **out**  $ch[\{om\_ui, wh\_ui\}]$  **Unit**
842. **OM**( $om\_ui, om\_mer:(cuis, iv\_ui, wh\_ui), om\_prodsupp$ )( $om\_wtd, om\_hist$ )  $\equiv$
843. **B** **OM.C\_OM\_Order**( $om\_ui, om\_mer, om\_prodsupp$ )( $om\_wtd, om\_hist$ )
844.  $\sqcap$  **C** **OM.OM\_IV\_Order**( $om\_ui, om\_mer, om\_prodsupp$ )( $om\_wtd, om\_hist$ )

<sup>11</sup>We leave it to the reader to be more specific. The “etcetera” could, for example, describe possible updates to the catalog and merchandise repository.

845.  $\square$  **L** **OM.IV\_OM\_Ack**(om\_ui,om\_mer,om\_prodsupp)(om\_wtd,om\_hist)
846.  $\square$  **M,N** **OM.Handle.Input**(om\_ui,om\_mer,om\_prodsupp)(om\_wtd,om\_hist)
847.  $\square$  **V** ... **OM**(om\_ui,om\_mer,om\_prodsupp)(om\_wtd,om\_hist)
849. **B** **OM.C\_OM\_Order** external non-deterministically offers to accept purchase orders from customers.
850. In response, **OM.C\_OM\_Order** makes a note of this request in its work-to-do dossier, that is, of eventually issuing an inventory order.
851. Thereupon **OM.C\_OM\_Order** resumes being ‘order management’ with an appropriately updated work-to-do state.

### Formalization

#### value

843. **COM..OM\_Order**: om\_ui:OM\_UI  $\times$  (om\_mer:(*cuis*,iv\_ui,wh\_ui)):OM\_Mer  $\times$  OM\_ProdSupp  $\rightarrow$   
 843. (OM\_WorkToDo  $\times$  OM\_TransHist)  
 843. **in out** { ch[ {c\_ui,om\_ui} ] | c\_ui:C\_UI • c\_ui  $\in$  *cuis* }  
 843. **in out** ch[ {om\_ui,iv\_ui} ] **out** ch[ {om\_ui,wh\_ui} ] **Unit**  
 843. **COM..OM\_Order**(om\_ui,om\_mer:(*cuis*,iv\_ui,wh\_ui),om\_prodsupp)(om\_wtd,om\_hist)  $\equiv$   
 849. **B**  $\square$  { **let** **A-B** **ordr:C\_OM\_Ord**((c\_ui,dati)),**order**=ch[ {c\_ui,om\_ui} ] ? **in**  
 850. **let** om\_wtd' = om\_wtd  $\cup$  {OM\_IV\_Ord((c\_ui,dati)),order,\_} **in**  
 851. **OM**(om\_ui,om\_mer,om\_prodsupp)(om\_wtd',(ordr) $\wedge$ om\_hist)  
 843. | c\_ui:C\_UI • c\_ui  $\in$  *cuis* **end end** }

### Narrative

852. **C** **OM.OM\_IV\_Order** inquires as to whether order\_management has a ‘work-to-do’ note on ordering a quantity of a named product.
853. If so, it selects that note.
854. It then selects a suitable product supplier and a sufficient quantity of the named product.
855. Finally it offers an inquiry to the inventory.
856. Whereupon it resumes being **OM**.
857. If **OM.OM\_IV\_Order** finds no such note it resumes being **OM**.

### Formalization

#### value

844. **OM.OM\_IV\_Order**: om\_ui:OM\_UI  $\times$  (om\_mer:(*cuis*,iv\_ui,wh\_ui)):OM\_Mer  $\times$  OM\_ProdSupp  $\rightarrow$   
 844. (OM\_WorkToDo  $\times$  OM\_TransHist)  
 844. **in out** { ch[ {c\_ui,om\_ui} ] | c\_ui:C\_UI • c\_ui  $\in$  *cuis* }  
 844. **in out** ch[ {om\_ui,iv\_ui} ] **out** ch[ {om\_ui,wh\_ui} ] **Unit**  
 844. **OM.OM\_IV\_Order**(om\_ui,om\_mer:(*cuis*,iv\_ui,wh\_ui),om\_prodsupp)(om\_wtd,om\_hist)  $\equiv$   
 852. **C** **if** OM\_IV\_Ord((c\_ui,dati)),order,\_  $\in$  wtd  
 853. **then let** ordr:OM\_IV\_Ord((c\_ui,dati)),order,\_ • ordr  $\in$  wtd **in**  
 854. **let** s\_ui:S\_UI • find\_supplier(order)(om\_prodsupp), dati' = record.TIME() **in**  
 855. **C-D** ch[ {om\_ui,iv\_ui} ] ! **ordr':OM\_IV\_Ord**((om\_ui,dati'),(c\_ui,dati)),order) ;  
 856. **OM**(om\_ui,om\_mer,om\_prodsupp)(om\_wtd \ {ordr},(ordr) $\wedge$ om\_hist) **end end**

857. **else OM**(om\_ui,om\_mer,om\_prodsupp)(om\_wtd,om\_hist) **end**

854. find\_supplier: Order  $\times$  OM\_ProdSupp  $\rightarrow$  S\_UI, find\_supplier(order)(om\_prodsupp)  $\equiv$  ...

### Narrative

858. **L** **OM.IV\_OM\_Ack** offers to accept an order acknowledgment from the retailer inventory.
859. It places this acknowledgment in the **OM**'s 'work-to-do' "basket" as a "matching" pair of customer acknowledgment and warehouse order dispatch notes.
860. And resumes being **OM**.

### Formalization

#### value

845. **OM.IV\_OM\_Ack**: OM\_UI  $\times$  OM\_Mer  $\times$  OM\_ProdSupp  $\rightarrow$   
 845. (OM\_WorkToDo  $\times$  OM\_TransHist)  
 845. **in out** { ch[ {c\_ui,om\_ui} ] | c\_ui:C\_UI•c\_ui  $\in$  cuis }  
 845. **in out** ch[ {om\_ui,iv\_ui} ] **out** ch[ {om\_ui,wh\_ui} ] **Unit**  
 845. **OM.IV\_OM\_Ack**(om\_ui,om\_mer:(cuis,iv\_ui,wh\_ui),m\_prodsupp)(om\_wtd,om\_hist)  $\equiv$   
 858. **L** let **K-L** iv\_om\_ack:IV\_OM\_Ack(pref,ordr) = ch[ {om\_ui,iv\_ui} ] ? **in**  
 859. let om\_wtd' = om\_wtd  $\cup$  {OM\_C\_Ack(((om\_ui,\_),pref),ordr),  
 859. OM\_WH\_Dispatch(((om\_ui,\_),pref),ordr)} **in**  
 860. **OM**(om\_ui,om\_mer,om\_prodsupp)(om\_wtd',(iv\_om\_ack)<sup>^</sup>om\_hist) **end end**

### Narrative

861. **M,N** If a suitable, i.e., "matching", pair of customer acknowledgment and warehouse order dispatch notes, can be found in the 'work-to-do' dossier,
862. then time is recorded,
863. the pair of to-do notes identified and
864. that pair removed from the work-to-do basket, whereupon
865. the customer is notified of the acknowledgment, and
866. the warehouse is notified of the order dispatch;
867. an updated order management transaction history is prepared, and
868. the **OM.OM\_C\_Ack\_OM\_WH\_Desp** resumes being **OM\_Beh**;
869. else **OM.OM\_C\_Ack\_OM\_WH\_Desp** resumes being **OM**.

### Formalization

#### value

846. **OM.Handle.Input**: OM\_UI  $\times$  OM\_Mer  $\times$  OM\_ProdSupp  $\rightarrow$  (OM\_WorkToDo  $\times$  OM\_TransHist) **Unit**  
 846. (OM\_WorkToDo  $\times$  OM\_TransHist)  
 846. **in out** { ch[ {c\_ui,om\_ui} ] | c\_ui:C\_UI•c\_ui  $\in$  cuis }  
 846. **in out** ch[ {om\_ui,iv\_ui} ] **out** ch[ {om\_ui,wh\_ui} ] **Unit**  
 846. **OM.Handle.Input**(om\_ui,om\_mer:(cuis,iv\_ui,wh\_ui),om\_prodsupp)(om\_wtd,om\_hist)  $\equiv$   
 861. **M** if  $\exists$  two:{IV\_OM\_Ack(((om\_ui,\_),pref),ordr),OM\_WH\_Dispatch(((om\_ui,\_),pref),ordr)} • two  $\subseteq$  om\_wtd

```

862. then let dati' = record_TIME(),
863. iv_om_ack = OM_C_Ack(((om_ui,_) ,pref),ordr) • iv_om_ack ∈ om_wtd,
863. om_wh_dis = OM_WH_Dispatch(((om_ui,_) ,pref),ordr) • om_wh_dis ∈ wtd,
864. om_wtd' = om_wtd \ {iv_om_ack,om_wh_dis} in
865. { M-O ch[{om_ui,c_ui}] ! om_c_ack':OM_C_Ack(((om_ui,dati') ,pref),ordr) ||
866. N-P ch[{om_ui,wh_ui}] ! om_wh_dis':OM_WH_Dispatch(((om_ui,dati') ,pref),ordr) } ;
867. let om_hist' = (om_c_ack',om_wh_dis')^om_hist in
868. OM(om_ui,om_mer,om_prodsupp)(om_wtd',om_hist') end end
869. else OM(om_ui,om_mer,om_prodsupp)(om_wtd,om_hist) end

```

### Narrative

870. **V** We leave this behaviour further undefined.

### Formalization

870. **V** ...

## H.6.2.3 Inventory Behaviour

### Narrative

871. The **IV** (inventory) behaviour communicates with the order management and the warehouse of the retailer to which it belongs, and with a variety of suppliers.

The **IV** behaviour alters between External non-deterministically offering to accept

872. **D** order input communications from its order management;

873. **J** order acknowledgment input communications from its warehouse; and

874.  $\alpha$  while internal non-deterministically handling incoming orders;

875. **E** internal non-deterministically offering order output communications to a designated supplier; or

876. **K** internal non-deterministically offering acknowledgment communications to its order management.

### Formalization

#### value

```

871. IV: IV_UI × (om_ui,wh_ui,suis):IV_Mer → (IV_WtD×IV_Inventory×IV_TransHist) Unit
871. in out ch[{om_ui,iv_ui}] in ch[{wh_ui,iv_ui}]
871. out { ch[{s_ui,iv_ui}] | s_ui:S_UI•s_ui ∈ suis } Unit
871. IV(iv_ui,iv_mer:(om_ui,wh_ui,suis))(iv_wtd,iv_inv,iv_hist) ≡
872. D IV.OM_IV_Order(iv_ui,iv_mer:(om_ui,wh_ui,suis))(iv_wtd,iv_inv,iv_hist)
873. J ⊑ IV.WH_IV_Ack(iv_ui,iv_mer:(om_ui,wh_ui,suis))(iv_wtd,iv_inv,iv_hist)
874. α ⊑ IV.Handle_Input(iv_ui,iv_mer:(om_ui,wh_ui,suis))(iv_wtd,iv_inv,iv_hist)
875. E ⊑ IV.IV_S_Order(iv_ui,iv_mer:(om_ui,wh_ui,suis))(iv_wtd,iv_inv,iv_hist)
876. K ⊑ IV.IV_OM_Ack(iv_ui,iv_mer:(om_ui,wh_ui,suis))(iv_wtd,iv_inv,iv_hist)
871. pre: iv_ui ∈ iv_uis ∧ om_ui ∈ om_uis ∧ wh_ui ∈ wh_uis
871. ∧ ∃ r:R • r ∈ obs_RETs(obs_RETa(mkt)) ∧ iv_ui=uid_IV(r) ∧ om_ui=uid_OM(r) ∧ wh_ui=uid_WH(r)

```

**Narrative**

877. **D** The **IV.OM\_IV\_Order** behaviour offers to accept an order [input] communication from its order management, which, when received, that order is put in the inventory ‘work-to-do’ basket –
878. to eventually be handled.
879. Whereupon the **IV.OM\_IV\_Order** resumes being the **IV** behaviours.

**Formalization****type**

878.  $\text{Handle\_OM\_IV\_Order} :: \text{Prefix} \times \text{Order}$

**value**

877. **D** **IV.OM\_IV\_order**:  $\text{IV\_UI} \times \text{IV\_Mer} \rightarrow (\text{IV\_WorkToDo} \times \text{IV\_Inventory} \times \text{IV\_TransHist}) \text{ Unit}$
877.     **in out**  $\text{ch}[\{\text{om\_ui, iv\_ui}\}] \text{ in } \text{ch}[\{\text{wh\_ui, iv\_ui}\}]$
877.     **out**  $\{\text{ch}[\{\text{s\_ui, iv\_ui}\}] \mid \text{s\_ui} : \text{S\_UI} \bullet \text{s\_ui} \in \text{suiss}\} \text{ Unit}$
877. **D** **IV.OM\_IV\_order**( $\text{iv\_ui, iv\_mer} : (\text{om\_ui, wh\_ui, suiss})$ )( $\text{iv\_wtd, iv\_inv, iv\_hist}$ )  $\equiv$
877.     **let** **C-D** **OM\_IV\_Order**(**prefix, order**) =  $\text{ch}[\{\text{om\_ui, iv\_ui}\}] ? \text{ in}$
878.     **let**  $\text{iv\_wtd}' = \{\text{Handle\_OM\_IV\_Order}(\text{prefix, order})\} \cup \text{iv\_wtd}$  **in**
879.     **IV**( $\text{iv\_ui, iv\_mer} : (\text{om\_ui, wh\_ui, suiss})$ )( $\text{iv\_wtd}', \text{iv\_inv, iv\_hist}$ ) **end end**

**Narrative**

880. **J** The **IV.WH\_IV\_Ack** behaviour offers to accept a supply availability acknowledgment [input] communication from its warehouse.
881. When received that acknowledgment is put in the inventory ‘work-to-do’ basket.
882. Whereupon the **IV.OM\_IV\_Order** resumes being the **IV** behaviours.

**Formalization****value**

873. **J** **IV.WH\_IV\_Ack**:  $\text{IV\_UI} \times \text{IV\_Mer} \rightarrow (\text{IV\_Inventory} \times \text{IV\_TransHist}) \text{ Unit}$
873.     **in out**  $\text{ch}[\{\text{om\_ui, iv\_ui}\}] \text{ in } \text{ch}[\{\text{wh\_ui, iv\_ui}\}]$
873.     **out**  $\{\text{ch}[\{\text{s\_ui, iv\_ui}\}] \mid \text{s\_ui} : \text{S\_UI} \bullet \text{s\_ui} \in \text{suiss}\} \text{ Unit}$
873. **J** **IV.WH\_IV\_Ack**( $\text{iv\_ui, iv\_mer} : (\text{om\_ui, wh\_ui, suiss})$ )( $\text{iv\_inv, iv\_hist}$ )  $\equiv$
880.     **let** **I-J** **WH\_IV\_ack**(**prefix, order**) =  $\text{ch}[\{\text{wh\_ui, iv\_ui}\}] ? \text{ in}$
881.     **let**  $\text{iv\_wtd}' = \{\}$   $\cup$   $\text{iv\_wtd}$  **in**
882.     **IV**( $\text{iv\_ui, iv\_mer} : (\text{om\_ui, wh\_ui, suiss})$ )( $\text{iv\_wtd}', \text{iv\_inv, iv\_hist}$ ) **end end**

**Narrative**

883. **α** If there exists, in the ‘work-to-do’ basket, a `handle_OM_IV_order`, cf. Item 878.,
884. then observe that order’s product name, `pn`, quantity, `q`, price, `p` and payment reference, `ref`, and
885. observe that product’s entry (its wholesale price, `wp`, suggested retail price, `srp`, sales price, `sp`, a recommended supplier, `s_ui`, and the quantity at hand in the warehouse stock) in the inventory [catalog].
886. If the order quantity is lower than the warehouse stock for that product,

887. then choose a suitable re-order quantity,  $q'$ ,
888. concoct an inventory-to-supplier order,
889. add that to, and remove the handle order from the ‘work-to-do’ basket, and
890. adjust the stock quantity in the inventory catalog,
891. before resuming being the **inventory** behaviour;
892. else update the ‘work-to-do’ basket with an inventory-to-order management acknowledgment to, and remove the handle order from the ‘work-to-do’ basket
893. and resume being the **inventory** behaviour.
894. If there does not exist, in the ‘work-to-do’ basket, a `handle_OM_IV_order`, then resume being the **inventory** behaviour.

### Formalization

#### type

878. `Handle_OM_IV_Order` :: `Prefix`  $\times$  `Order`

#### value

```

874. α IV.Handle_Input: IV_UI \times IV_Mer \rightarrow (IV_WorkToDo \times IV_Inventory \times IV_TransHist)
874. in out ch[{om_ui,iv_ui}] in ch[{wh_ui,iv_ui}]
874. out { ch[{s_ui,iv_ui}] | s_ui:S_UI•s_ui \in suis } Unit
874. α IV.Handle_Input(iv_ui,iv_mer:(om_ui,wh_ui,suis))(iv_wtd,iv_inv,iv_hist) \equiv
883. if \exists ho:Handle_OM_IV_Order(prefix,order) • ho \in iv_wtd
883. then let ho:Handle_OM_IV_Order(prefix,order) • ho \in iv_wtd
884. let (pn,q,p,ref) = ho in axiom pn \in dom iv_inv
885. let (wp,srp,sp,mi,s_ui,stock) = iv_inv(pn) in axiom [p \in {srp,sp}]
886. if q < stock
887. then let q':Nat • q' > q \wedge ... in
888. let iv_s_order = IV_S_Order(prefix,(pn,q',wp,iv_ref)) in
889. let iv_wtd' = {iv_s_order} \cup iv_wtd \setminus {ho},
890. iv_inv' = iv_inv \dagger [pn \mapsto (wp,srp,sp,mi,s_ui,stock - q)] in
891. IV(iv_ui,iv_mer)(iv_wtd',iv_inv',iv_hist) end end end
892. else let iv_wtd' = {IV_OM_Ack(prefix,order)} \cup iv_wtd \setminus {ho} in
893. IV(iv_ui,iv_mer)(iv_wtd',iv_inv,iv_hist) end end
894. else IV(iv_ui,iv_mer)(iv_wtd,iv_inv,iv_hist) end end end end

```

### Narrative

895.  $\mathbf{E}$  If there exists, in the ‘work-to-do’ basket, a `handle_OM_IV_order`,
896. then retrieve that order
897. and remove it from the ‘work-to-do’ basket while
898. communicating the order, updated with a date-timed prefix, to a designated supplier,
899. and resuming being the **inventory** behaviour.
900. If no `handle_OM_IV_order` is in the basket, then resume being “an unchanged” **inventory** behaviour.



**Formalization**

```

875. E IV.IV_S_Order: IV_UI × IV_Mer → (IV_WorkToDo × IV_Inventory × IV_TransHist)
871. in out ch[{om_ui,iv_ui}] in ch[{wh_ui,iv_ui}]
871. out { ch[{s_ui,iv_ui}] | s_ui:S_UI•s_ui ∈ suis } Unit
875. E IV.IV_S_Order(iv_ui,iv_mer:(om_ui,wh_ui,suis))(iv_wtd,iv_inv,iv_hist) ≡
895. if ∃ o:IV_S_Order(prefix,order,s_ui) • o ∈ iv_wtd axiom s_ui ∈ suis
896. then let o:IV_S_Order(prefix,order,s_ui) • o ∈ iv_wtd in
897. let iv_wtd' = iv_wtd \ {o}, dati = record_TIME() in
898. E-F ch[{iv_ui,s_ui}] ! msg:IV_S_Order((iv_ui,dati),prefix,order,wh_ui) ;
899. IV(iv_ui,iv_mer)(iv_wtd',iv_inv,(msg)^iv_hist)
875. end end
899. else IV(iv_ui,iv_mer)(iv_wtd,iv_inv,iv_hist) end

```

**Narrative**

901. **K** If there exists, in the ‘work-to-do’ basket, an IV\_OM\_Ack(prefix,order),
902. then retrieve that order
903. and remove it from the ‘work-to-do’ basket while
904. communicating the order, updated with a date-timed prefix, to a designated supplier,
905. and resuming being the **inventory** behaviour.
906. If no handle\_OM\_IV\_order is in the basket, then resume being “an unchanged” **inventory** behaviour

**Formalization**

```

value
876. IV.IV_OM_Ack: IV_UI × IV_Mer → (IV_Inventory × IV_TransHist)
876. in out ch[{om_ui,iv_ui}] in ch[{wh_ui,iv_ui}]
876. out { ch[{s_ui,iv_ui}] | s_ui:S_UI•s_ui ∈ suis } Unit
876. IV.IV_OM_Ack(iv_ui,iv_mer:(om_ui,wh_ui,suis))(iv_inv,iv_hist)
901. if ∃ a:IV_OM_Ack(prefix,order) • a ∈ iv_wtd
902. then let a:IV_OM_Ack(prefix,order) • a ∈ iv_wtd in
903. let iv_wtd' = iv_wtd \ {a}, dati = record_TIME() in
904. K-L ch[{iv_ui,om_ui}] ! msg:IV_OM_Ack((iv_ui,dati),prefix,order) ;
905. IV(iv_ui,iv_mer)(iv_wtd',iv_inv,(msg)^iv_hist)
875. end end
906. else IV(iv_ui,iv_mer)(iv_wtd,iv_inv,iv_hist) end

```

**H.6.2.4 Warehouse Behaviour****Narrative**

907. The **WH** (warehouse) behaviour accepts supplies from any supplier, provides supply acknowledgments to its inventory, accepts dispatch order from its order management and provides merchandise to any courier service.

Internal non-deterministically the **WH** behaviour alternates between

908. **H** external non-deterministically accepting deliveries from suppliers,

909. **P** accepting order dispatches from its order management,
910. **H,P** handling deferred [but accepted] inputs,
911. **I** offering acknowledgments of supplies to its inventory, and
912. **Q** delivering merchandise (orders) to any one of a number of designated courier services.

### Formalization

#### value

907. **WH**:  $wh\_ui:WH\_UI \times (om\_ui,iv\_ui,suis,csuis):WH\_Mer \rightarrow$   
 (WH\_WorkToDo $\times$ WH\_Store $\times$ WH\_TransHist)  
 907. **in** {  $ch[\{s\_ui,iv\_ui\}] \mid s\_ui:S\_UI \bullet s\_ui \in suis$  }  
 907. **out**  $ch[\{wh\_ui,iv\_ui\}]$  **in**  $ch[\{wh\_ui,om\_ui\}]$   
 907. **out** {  $ch[\{wh\_ui,c\_ui\}] \mid c\_ui:C\_UI \bullet c\_ui \in cuis$  } **Unit**  
 907. **WH**( $wh\_ui,wh\_mer:(wh\_ui,om\_ui,iv\_ui,suis,csuis)$ )( $wh\_wtd,wh\_store,wh\_hist$ )  $\equiv$   
 908. **H** WH.S\_WH\_Del( $wh\_ui,wh\_mer$ )( $wh\_wtd,wh\_store,wh\_hist$ )  
 909. **P**  $\sqcap$  WH.OM\_WH\_Disp( $wh\_ui,wh\_mer$ )( $wh\_wtd,wh\_store,wh\_hist$ )  
 910. **H,P**  $\sqcap$  WH.Handle\_Input( $wh\_ui,wh\_mer$ )( $wh\_wtd,wh\_store,wh\_hist$ )  
 911. **I**  $\sqcap$  WH.WH\_IV\_Ack( $wh\_ui,wh\_mer$ )( $wh\_wtd,wh\_store,wh\_hist$ )  
 912. **Q**  $\sqcap$  WH.WH\_CS\_Deliv( $wh\_ui,wh\_mer$ )( $wh\_wtd,wh\_store,wh\_hist$ )  
 907. **pre**:  $wh\_ui \in whuis \wedge om\_ui \in omuis \wedge iv\_ui \in ivuis$   
 907.  $\wedge \exists r:R \bullet r \in obs\_RETs(obs\_RETA(mkt)) \wedge wh\_ui=uid\_WH(r) \wedge om\_ui=uid\_OM(r) \wedge iv\_ui=uid\_IV(r)$

### Narrative

913. **H** The **WH.S\_WH\_Del** behaviour external non-deterministically offers to accept a supplier to warehouse delivery message.
914. When received the **WH.S\_WH\_Del** behaviour deposits this message in its ‘work-to-do’ basket.
915. It then resumes being the **WH** behaviour.

### Formalization

#### value

908. **H** **WH.S\_WH\_Del**:  $wh\_ui:WH\_UI \times (\_,\_,suis,\_):WH\_Mer \rightarrow$   
 (WH\_WorkToDo $\times$ WH\_Store $\times$ WH\_TransHist)  
 908. **in** {  $ch[\{s\_ui,wh\_ui\}] \mid s\_ui:S\_UI \bullet s\_ui \in suis$  } **Unit**  
 908. **H** **WH.S\_WH\_Del**( $wh\_ui,wh\_mer:(\_,\_,suis,\_)$ )( $wh\_wtd,wh\_store,wh\_hist$ )  $\equiv$   
 913. { **let** **G–H delivery**:**S\_WH\_Del**(**prefix,order,ms**) =  $ch[\{s\_ui,wh\_ui\}]$  ? **in**  
 914. **let**  $wh\_wtd' = wh\_wtd \cup \{delivery\}$  **in**  
 915. **WH**( $wh\_ui,wh\_mer$ )( $wh\_wtd',wh\_store,wh\_hist$ )  
 913. **end end** |  $s\_ui:S\_UI \bullet s\_ui \in suis$  }  
 908. **pre**:  $wh\_ui \in whuis \wedge \exists r:R \bullet r \in obs\_RETs(obs\_RETA(mkt)) \wedge wh\_ui=uid\_WH(r)$

**Narrative**

916. **P** The **WH.OM\_WH\_Dispatch** behaviour offers to accept an order management to warehouse [order] dispatch message.
917. When received the **WH.OM\_WH\_Dispatch** behaviour deposits this message in its ‘work-to-do’ basket.
918. It then resumes being the **WH** behaviour.

**Formalization****value**

909. **P** **WH.OM\_WH\_Dispatch**:  $wh\_ui:WH\_UI \times (om\_ui, \_, \_, \_):WH\_Mer \rightarrow$   
 909.  $(WH\_WorkToDo \times WH\_Store \times WH\_TransHist)$   
 909. **in**  $ch\{om\_ui, wh\_ui\}$  **Unit**
909. **P** **WH.OM\_WH\_Dispatch**( $wh\_ui, wh\_mer:(om\_ui, \_, \_, \_)$ )( $wh\_wtd, wh\_store, wh\_hist$ )  $\equiv$   
 916. **let** **N-P** **dispatch:OM\_WH\_Dis(prefix,order) = ch\{om\\_ui, wh\\_ui\} ? in**  
 917. **let**  $wh\_wtd' = wh\_wtd \cup \{dispatch\}$  **in**  
 918. **WH**( $wh\_ui, wh\_mer$ )( $wh\_wtd', wh\_store, wh\_hist$ )  
 916. **end end**
907. **pre**:  $wh\_ui \in whuis \wedge om\_ui \in omuis \wedge iv\_ui \in ivuis$   
 907.  $\wedge \exists r:R \bullet r \in obs\_RETS(obs\_RETA(mkt)) \wedge wh\_ui=uid\_WH(r) \wedge om\_ui=uid\_OM(r) \wedge iv\_ui=uid\_IV(r)$

**Narrative**

919. **H,P** The rôle of the **WH.Handle\_Input** behaviour is to service either of the two kinds of inputs received by the warehouse, from suppliers, **S**, and from its order management, **OM**.
920. There are two possible kinds of “deferred” messages.
921. If there is a supplier-to-warehouse, **S\_WH\_Del(prefix,order,ms)**, delivery message,
922. then that message is “converted” into a warehouse-to-inventory delivery acknowledgment message in, the ‘work-to-do’ basket,
923. and the **WH.Handle\_Input** behaviour reverts to being the **WH** behaviour;
924. else if there is an order management-to-warehouse, **OM\_WH\_Dis(prefix,order)**, message,
925. then that message is “converted” into a warehouse-to-courier service delivery message, **WH\_CS\_Del(prefix,order)**, in the ‘work-to-do’ basket,
926. and the **WH.Handle\_Input** behaviour reverts to being the **WH** behaviour;
927. if there are no messages in the basket then the **WH.Handle\_Input** behaviour reverts to being the **WH** behaviour.
- 928.

### Formalization

value

```

919. H,P WH.Handle_Input: wh_ui:WH_UI × (__,iv_ui,__,__):WH_Mer →
919. (WH_WorkToDo×WH_Store×WH_TransHist)
919. out ch[{wh_ui,iv_ui}] Unit
919. H,P WH.Handle_Input(wh_ui,wh_mer:(__,iv_ui,__,__))(wh_wtd,wh_store,wh_hist) ≡
920. case wh_wtd of
921. {S_WH_Del(prefix,order,ms)} ∪ wh_wtd' →
922. let wh_wtd'' = wh_wtd' ∪ {WH_IV_Ack(prefix,order,ms)} in
923. WH(wh_ui,wh_mer)(wh_wtd'',wh_store,wh_hist) end
924. {OM_WH_Dis(prefix,order)} ∪ wh_wtd' →
925. let wh_wtd'' = wh_wtd' ∪ {WH_CS_Del(prefix,order)} in
926. WH(wh_ui,wh_mer)(wh_wtd'',wh_store,wh_hist) end
927. _ → WH(wh_ui,wh_mer)(wh_wtd,wh_store,wh_hist)
928. end
929. pre: wh_ui ∈ whuis ∧ iv_ui ∈ ivuis
929. ∧ ∃ r:R • r ∈ obs_RETs(obs_RETa(mkt)) ∧ wh_ui=uid.WH(r) ∧ iv_ui=uid.IV(r)
929. axiom: [there can only at most be the two kinds of messages as 'cased' in the wtd basket.]

```

### Narrative

929. **I** If there exists a warehouse-to-inventory [supplier] delivery acknowledgment message in the 'work-to-do' basket,
930. then select and remove that message from the basket,
931. record the current time, and
932. communicate the acknowledgment message to the inventory,
933. and resume being the appropriately updated **WH** behaviour;
934. else resume being the otherwise unchanged **WH** behaviour.

### Formalization

value

```

911. I WH.WH_IV_Ack: wh_ui:WH_UI × (__,iv_ui,__,__):WH_Mer →
911. (WH_WorkToDo×WH_Store×WH_TransHist)
911. out ch[{wh_ui,iv_ui}] Unit
911. I WH.WH_IV_Ack(wh_ui,wh_mer)(wh_wtd,wh_store,wh_hist) ≡
929. if ∃ a:WH_IV_Ack(prefix,order,ms) • d ∈ om_wtd
930. then let a:WH_IV_Ack(prefix,order,ms) • a ∈ om_wtd in
930. let wh_wtd' = wh_wtd \ {a},
931. dati = record_TIME() in
932. I-J ch[{wh_ui,iv_ui}] ! ack:WH_IV_Ack(((wh_ui,dati),prefix),order) ;
933. WH(wh_ui,wh_mer)(wh_wtd',wh_store,(ack)^wh_hist) end end
934. else WH(wh_ui,wh_mer)(wh_wtd,wh_store,wh_hist) end
907. pre: wh_ui ∈ whuis ∧ iv_ui ∈ ivuis
907. ∧ ∃ r:R • r ∈ obs_RETs(obs_RETa(mkt)) ∧ wh_ui=uid.WH(r) ∧ iv_ui=uid.IV(r)

```

**Narrative**

935. **Q** If there exists a order management to warehouse [supplier] delivery message in the ‘work-to-do’ basket,
936. then select and remove that message from the basket,
937. record the current time, and
938. communicate the acknowledgment message to the inventory,
939. and resume being the appropriately updated **WH** behaviour;
940. else resume being the otherwise unchanged **WH** behaviour.

**Formalization****value**

912. **Q** **WH.WH\_CS\_Deliv**:  $wh\_ui:WH\_UI \times (\_,\_,\_,csuis):WH\_Mer \rightarrow$   
 (WH\_WorkToDo $\times$ WH\_CS\_Dire $\times$ WH\_Store $\times$ WH\_TransHist)  
 912. **out** {  $ch\{wh\_ui,c\_ui\} \mid c\_ui:C\_UI \bullet c\_ui \in csuis$  } **Unit**
912. **Q** **WH.WH\_CS\_Deliv**( $wh\_ui,wh\_mer:(\_,\_,\_,csuis)$ )( $wh\_wtd,wh\_store,wh\_hist$ )  $\equiv$   
 935. **if**  $\exists a:OM\_WH\_Disp(prefix,order,cs\_ui) \bullet a \in om\_wtd$   
 936. **then let**  $d:OM\_WH\_Disp(prefix,order,cs\_ui) \bullet a \in om\_wtd$  **in**  
 936. **let**  $wh\_wtd' = wh\_wtr \setminus \{d\}$ ,  
 937.  $dati = record\_TIME()$ ,  
 937.  $os:M\_set \bullet os \subseteq wh\_store \wedge card\ os = q$  **in**  
 938. **Q-R**  $ch\{wh\_ui,cs\_ui\} ! ack:WH\_CS\_Disp((wh\_ui,dati),prefix),order,os)$  ;  
 939. **WH**( $wh\_ui,wh\_mer$ )( $wh\_wtd',wh\_store \setminus \{os\},(ack)^\sim wh\_hist$ ) **end end**  
 940. **else** **WH**( $wh\_ui,wh\_mer$ )( $wh\_wtd,wh\_store,wh\_hist$ ) **end**  
 907. **pre**:  $wh\_ui \in whuis \wedge \exists r:R \bullet r \in obs\_RETS(obs\_RETA(mkt)) \wedge wh\_ui = uid\_WH(r)$

**H.6.2.5 Supplier Behaviour****Narrative**

941. The **S**upplier behaviour internal non-deterministically “alternates” between
942. **F** accepting orders from any retailers’ inventory, and
943. **G** delivering such orders to retailers’ warehouses.

**Formalization****value**

941. **S**:  $S\_UI \times (ivuis, whuis):S\_Mer \rightarrow (S\_WorkToDo \times S\_Products \times S\_TransHist)$   
 941. **in** {  $ch\{iv\_ui,s\_ui\} \mid iv\_ui:IV\_UI \bullet iv\_ui \in ivuis$  }  
 941. **out** {  $ch\{s\_ui,wh\_ui\} \mid wh\_ui:WH\_UI \bullet wh\_ui \in whuis$  } **Unit**
941. **S**( $s\_ui,s\_mer:(ivuis, whuis)$ )( $s\_wtd,s\_products,s\_hist$ )  $\equiv$   
 942. **F**  $S.IV\_S\_Order(s\_ui,s\_mer:(ivuis, whuis))(s\_wtd,s\_products,s\_hist)$   
 943. **G**  $\sqcap S.S\_WH\_Deliv(s\_ui,s\_mer:(ivuis, whuis))(s\_wtd,s\_products,s\_hist)$   
 941. **pre**:  $s\_ui \in suis \wedge \exists s:S \bullet r \in obs\_Ss(obs\_Sa(mkt)) \wedge s\_ui = uid\_UI(r)$

Please note that we have omitted the “intermediary” behaviour of the **S**upplier *handling* inputs. We suggest that such handling is taken care of directly by the **S.S.WH.Delivery** behaviour. Also note that we do not describe payment aspects.

**Narrative**

944. The **S.IV\_S\_Order** behaviour external non-deterministically offers to accept merchandise orders from any retailer's inventory behaviour.
945. Having received such an order it proceeds to record it in its 'work-to-do' basket –
946. whereupon it resumes being the **S** behaviour (with the updated basket).

**Formalization****value**

```

942. F S.IV_S_Order(s_ui,s_mer:(ivuis,_))(s_wtd,s_products,s_hist) ≡
944. [] { let E-F IV_S_Order(prefix,order) = ch[{iv_ui,s_ui}] ? in
945. let s_wtd' = s_wtd ∪ {IV_S_Order(prefix,order)} in
946. S(s_ui,s_mer)(s_wtd',s_products,s_hist)
944. | iv_ui:IV_UI•iv_ui ∈ ivuis end end }
```

**Narrative**

947. If there exists a IV\_S\_Order(prefix,order) message in the 'work-to-do' basket,
948. then select such a message,
949. examine the order,
950. select the quantified number of merchandise of the ordered product,
951. ascertain the current time,
952. and deliver the message to the warehouse of the requesting retailer;
953. then resume being the **S**upplier behaviour with appropriately updated programmable attributes.
954. If there does not exist a IV\_S\_Order(prefix,order) message in the 'work-to-do' basket, then revert to being the **S**upplier behaviour.

**Formalization****value**

```

943. G S.S-WH_Deliv(s_ui,s_mer:(ivuis, whuis))(s_wtd,s_products,s_hist) ≡
947. if ∃ h:IV_S_Order(prefix,order,wh_ui) • h ∈ s_wtd
948. then let h:IV_S_Order(prefix,order,wh_ui) • h ∈ s_wtd in
949. let (pn,q,cost,payref) = order in
950. let ms:M-set • ms ⊆ s_products(pn)∧card ms = q,
951. dati = record_TIME() in
952. G-H ch[{s_ui,wh_ui}] ! d:S-WH_Deliv((s_ui,dati),pref),order,ms) ;
953. S(s_ui,s_mer)(s_wtd \ {h},s_products†[pn→s_products(pn) \ ms],⟨d⟩^s_hist)
948. end end end
954. else S(s_ui,s_mer)(s_wtd,s_products,s_hist) end
```

## H.6.2.6 Courier Service Behaviour

## Narrative

955. The **CS**, courier service, behaviour internal non-deterministically alternates between
956. **R** offering to accept a warehouse to [customer directed] courier service delivery of merchandise and
957. **S** offering a courier service to customer delivery.

## Formalization

## value

955. **CS**:  $CS\_UI \times CS\_Mer \rightarrow (CS\_WorkToDo \times CS\_TransHist)$  **Unit**
955. **CS**( $cs\_ui, cs\_mer:(whis, cuis)$ )( $cs\_wtd, cs\_hist$ )  $\equiv$
956. **R**  $CS.WH\_CS\_Deliv(cs\_ui, cs\_mer:(whis, cuis))(cs\_wtd, cs\_hist)$
957. **S**  $\sqcap CS.CS\_C\_Deliv(cs\_ui, cs\_mer:(whis, cuis))(cs\_wtd, cs\_hist)$
955. **pre**:  $cs\_ui \in csuis$

## Narrative

958. **R** The **CS.WH\_CS\_Deliv** behaviour external non-deterministically offers to accept a customer directed delivery request from any retailer's warehouse.
959. Receiving such a request it updates its 'work-to-do' basket accordingly,
960. and reverts to being the courier service **CS**.

## Formalization

## value

956. **R** **CS.WH\_CS\_Deliv**:  $CS\_UI \times CS\_Mer \rightarrow (CS\_WorkToDo \times CS\_TransHist)$  **Unit**
956. **R** **CS.WH\_CS\_Deliv**( $cs\_ui, cs\_mer:(whis, \_)$ )( $cs\_wtd, cs\_hist$ )  $\equiv$
958.  $\sqcap \{ \text{let } \mathbf{Q-R} \ r:WH\_CS\_Deliv(((wh\_ui, dati), prefix), order, os) = ch[\{wh\_ui, s\_ui\}] ? \text{ in}$
959.  $\quad \text{let } cs\_wtd' = cs\_wtd \cup \{r\} \text{ in}$
960.  $\quad \mathbf{CS}(cs\_ui, cs\_mer)(cs\_wtd', cs\_hist)$
958.  $\quad | wh\_ui:WH\_UI \bullet wh\_ui \in whuis \text{ end end } \}$
956. **pre**:  $cs\_ui \in csuis$

961. **S** If there exists a **WH\_CS\_Del**( $prefix, order, ms, c\_ui$ ) dispatch in the 'work-to-do' basket of a courier service
962. then retrieve this dispatch
963. pass it on to the designated customer
964. and revert to being the courier service, **CS**, behaviour with appropriately updated arguments.
965. Otherwise continue being the **CS** behaviour.

## Formalization

value

```

957. S CS.CS_C_Deliv: CS_UI × CS_Mer → (CS_WorkToDo × CS_TransHist) Unit
957. S CS.CS_C_Deliv(cs_ui,cs_mer)(cs_wtd,cs_hist) ≡
961. if ∃ whd:WH_CS_Del(prefix,order,ms,c_ui) • whd ∈ cs_wtd
962. then let d:WH_CS_Del(prefix,order,ms,c_ui) • whd ∈ cs_wtd in
963. S-T ch[{cs_ui,c_ui}] ! cd:CS_C_Del((cs_ui,record_TIME),prefix),order,ms) ;
964. CS(cs_ui,cs_mer)(cs_wtd \ {d},{cd}^cs_hist) end
965. else CS(cs_ui,cs_mer)(cs_wtd,cs_hist) end
962. pre: cs_ui ∈ csuis

```

### H.6.3 System Initialisation

We refer to [58, Sect. 7.8].

966. Given a market, cf., mkt Item 750 on page 231, we can “synthesize” an RSL clause that stands for the total behaviour of this market.

We refer to the system state as “generated” in Sect. H.3.4 on page 231.

967. The market behaviour is the parallel composition of

968. the distributed parallel compositions of all customers,

969. the distributed parallel compositions of all order managements,

970. the distributed parallel compositions of all inventories,

971. the distributed parallel compositions of all warehouses,

972. the distributed parallel compositions of all suppliers and

973. the distributed parallel compositions of all courier services.

value

966. mkt, cs, oms, ivs, whs, ss and css.

968. || {**C**(uid\_C(c),mereo\_C(c),attr\_C\_Catalog(c))(attr\_C\_Merchandise(c),⟨⟩)|c:C•c∈cs}

967. ||

969. || {**OM**(uid\_OM(om),mereo\_OM(c),attr\_OM\_ProdSupp(om))(attr\_OM\_WorkToDo(om),⟨⟩)|om:OM•om∈oms}

967. ||

970. || {**IV**(uid\_IV(iv),mereo\_IV(iv))(attr\_IV\_WorkToDo(iv),attr\_IV\_Inventory(iv),⟨⟩)|iv:IV•iv∈ivs}

967. ||

971. || {**WH**(uid\_WH(wh),mereo\_WH(wh))(attr\_WH\_WorkToDo(wh),attr\_WH\_Store(wh),⟨⟩)|wh:WH•wh∈whs}

967. ||

972. || {**S**(uid\_S(s),mereo\_S(s))(attr\_S\_WorkToDo(s),attr\_S\_Products(s),⟨⟩)|s:S•s∈ss}

967. ||

973. || {**CS**(uid\_CS(cs),mereo\_CS(cs))(attr\_CS\_WorkToDo(cs),⟨⟩)|cs:CS•cs∈css}

## H.7 Conclusion

### H.7.1 Critique of the domain analysis & description Model

I am, today, approx. Spring 2021, not quite happy with my description.



- It was developed too quickly. I started on this model on Dec. 28, 2020. I was the only one to develop this model.
- Along the “road” I did not take time to carefully consider the naming of types, values, functions and behaviours.
- Also: the individual definitions of order management, inventory, warehouse, supplier and courier service behaviours (**OM, II, WH, S, CS**) into their , as of Jan. 21, 2021, is/was uneven.
  - In the **C** (customer) behaviour description (Items 832 on page 244.– 841 on page 245.) “all” is expressed in that one behaviour description, whereas in the **OM, IV, WH, S** and **CS** behaviour descriptions the descriptions are decomposed into separate internal non-deterministic behaviours, but not quite consistently.
  - I have yet to check that the mereologies and attributes of parts are consistently used in respective behaviour definitions.
  - And I have yet to check that the indexing of all defined types, sorts, unique identifiers, mereologies, attributes, channel and behaviours is consistent.
- But, on the whole, the model gives a reasonably adequate picture of how a model in the domain science & engineering style would express the HERAKLIT *retailer* “challenge”.
- All I can say, not in any defense, is: *“I am too<sup>12</sup> old for this game these days!”*

### H.7.2 Proofs about Models

Models are developed, carefully, and honed, “perpetually, for several reasons. One is to be able to prove properties of the domain being modeled but where these properties are not explicitly stated. We speculate on a few – with more to come!

- *“The sum total of merchandise, in the market as modeled, is constant: no merchandise “arise out of the blue” (for example at suppliers), no merchandise “disappears mysteriously” (for example in warehouses, courier services or at customers).”*
- *“Product quantity\_on\_hands in a retailer’s inventory (catalog) is always less than or equal to that retailer’s corresponding quantities\_at\_hand in its warehouse.”*
- With the ideal assumption that suppliers can always deliver requested numbers of any product: *“Customers are eventually delivered their ordered merchandise.”*
- Etcetera!

We do not show any such proofs in this technical report.

### H.7.3 Comparison of Models

We compare our model to that of [95].

#### H.7.3.1 “Minor” Discrepancies

- It seems, but this has to be checked, that orders, in the *domain analysis & description* model, are for any number of one particular merchandise product, whereas the HERAKLIT model allows the mixing of several products and of different quantities of these.
- It also seems that ...

MORE TO COME

---

<sup>12</sup>I was born Oct. 4, 1937

### H.7.3.2 Use of Diagrams

- Somewhere, in Footnote 5 on page 225, it is said that none of the figures in this report play any rôle in the formal aspects of the ‘retailer market’ description.
  - That is intended to be so.
  - But is it really true?
    - \* When I first worked as an M.Sc. graduated engineer in designing data communications “gear” and computers for IBM (1962–1965, 1969) we all drew diagrams!
    - \* When I then studied computer science (1965-1968) diagrams of software systems (except for “trivial” program flowcharts) were frowned upon.
    - \* **Petri nets** (around 1962–1963) are based almost exclusively on two-dimensional diagrams. They are easy to grasp,
    - \* The diagrams of **HERAKLIT** are likewise appealing.
  - Figure H.2 on page 227 of this report is rather crucial, I found, in keeping track, while I was developing the description, of all the various segments of that description – in particular in making sure that the interfaces between behaviours “fitted”.
  - I can imagine that some readers will find, especially Fig. H.2 on page 227 useful when reading the description.
- So, perhaps, diagrams, of the kind that Fig. H.2 on page 227 represents, ought be “woven” into the *domain analysis & description* analysis & description, into its *principles, techniques* and *tools*.

### H.7.3.3 Interleave versus “True” Concurrency

The reader is assumed to be quite familiar with these two kinds of semantic terms and their meaning.

- As such, the **HERAKLIT**-based model, appears to be at an advantage – in that it expresses “true concurrency”.
- But, before you get all too excited, the domain science & engineering/*domain analysis & description* model, as its behaviours are defined, “do not lack far behind”, if-at-all!
  - On one hand you can read the basically **CSP** clauses as if actions in separate behaviours do indeed occur “truly concurrent”.
  - On the other hand, by “splitting” up, as in the behaviour definitions of **C**, **OM**, **IV**, **WH**, **S** and **CS**, these into separate actions, such as *input*, *handling*, etc., an as full “measure” of local “true concurrency” seems to be achieved.

### H.7.4 What Next?

- It is my sincere hope that Messrs Fettke and Reisig will comment on the present report.
- I need to know where I have misunderstood the [intentions of the] **HERAKLIT** model [95].
- I need to know where my model fails in modeling what [95] achieves.
- etcetera!

# Appendix I

## Pipelines

### Contents

---

|            |                                       |            |
|------------|---------------------------------------|------------|
| <b>I.1</b> | <b>Endurants: External Qualities</b>  | <b>262</b> |
| I.1.1      | Parts                                 | 262        |
| I.1.2      | An Endurant State                     | 263        |
| <b>I.2</b> | <b>Endurants: Internal Qualities</b>  | <b>263</b> |
| I.2.1      | Unique Identification                 | 263        |
| I.2.2      | Mereology                             | 264        |
| I.2.2.1    | PLS Mereology                         | 264        |
| I.2.2.2    | Unit Mereologies                      | 264        |
| I.2.3      | Pipeline Concepts, I                  | 265        |
| I.2.3.1    | Pipe Routes                           | 265        |
| I.2.3.2    | Well-formed Routes                    | 265        |
| I.2.3.3    | Embedded Routes                       | 266        |
| I.2.3.4    | A Theorem                             | 266        |
| I.2.3.5    | Fluids                                | 267        |
| I.2.4      | Attributes                            | 267        |
| I.2.4.1    | Unit Flow Attributes                  | 267        |
| I.2.4.2    | Unit Metrics                          | 268        |
| I.2.4.3    | Wellformed Unit Metrics               | 269        |
| I.2.4.4    | Summary                               | 269        |
| I.2.4.5    | Fluid Attributes                      | 270        |
| I.2.4.6    | Pipeline System Attributes            | 271        |
| I.2.5      | Pipeline Concepts, II: Flow Laws      | 271        |
| <b>I.3</b> | <b>Perdurants</b>                     | <b>272</b> |
| I.3.1      | State                                 | 272        |
| I.3.2      | Channel                               | 272        |
| I.3.3      | Actions                               | 272        |
| I.3.4      | Behaviours                            | 273        |
| I.3.4.1    | Behaviour Kinds                       | 273        |
| I.3.4.2    | Behaviour Signatures                  | 273        |
| I.3.4.2.1  | Behaviour Definitions                 | 274        |
| I.3.4.2.2  | The Pipeline System Behaviour         | 274        |
| I.3.4.2.3  | The Pump Behaviours                   | 275        |
| I.3.4.2.4  | The Valve Behaviours                  | 275        |
| I.3.4.3    | Sampling Monitorable Attribute Values | 276        |
| I.3.4.4    | System Initialisation                 | 276        |
| <b>I.4</b> | <b>Review</b>                         | <b>276</b> |

---

## I.1 Endurants: External Qualities

### I.1.1 Parts

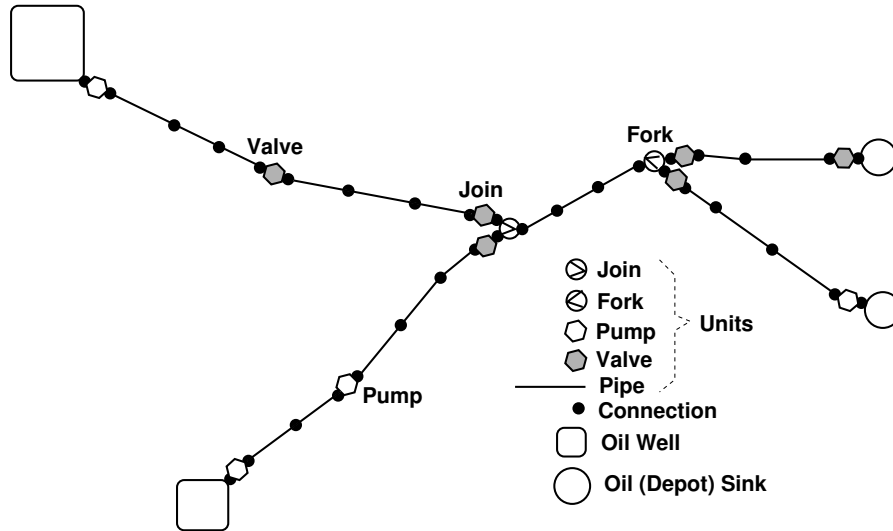


Figure I.1: An example pipeline system

974. A pipeline system contains a set of pipeline units and a pipeline system monitor.
975. The well-formedness of a pipeline system depends on its mereology (cf. Sect. I.2.2) and the routing of its pipes (cf. Sect. I.2.3.2).
976. A pipeline unit is either a well, a pipe, a pump, a valve, a fork, a join, a plate<sup>1</sup>, or a sink unit.
977. We consider all these units to be distinguishable, i.e., the set of wells, the set pipe, etc., the set of sinks, to be disjoint.

#### type

974.  $PLS', U, M$

975.  $PLS = \{ | pls: PLS' \bullet wf\_PLS(pls) | \}$

#### value

975.  $wf\_PLS: PLS \rightarrow \mathbf{Bool}$

975.  $wf\_PLS(pls) \equiv$

975.  $wf\_Mereology(pls) \wedge wf\_Routes(pls) \wedge wf\_Metrics(pls)^2$

974.  $obs\_Us: PLS \rightarrow U\text{-set}$

974.  $obs\_M: PLS \rightarrow M$

#### type

976.  $U = We | Pi | Pu | Va | Fo | Jo | Pl | Si$

977.  $We :: Well$

977.  $Pi :: Pipe$

977.  $Pu :: Pump$

977.  $Va :: Valv$

977.  $Fo :: Fork$

977.  $Jo :: Join$

<sup>1</sup>A *plate* unit is a usually circular, flat steel plate used to “begin” or “end” a pipe segment.

977. Pl :: Plate

977. Si :: Sink

### I.1.2 An Endurant State

978. For a given pipeline system

979. we exemplify an enduring state  $\sigma$

980. composed of the given pipeline system and all its manifest units, i.e., without plates.

#### value

978. pls:PLS

#### variable

979.  $\sigma := \text{collect\_state}(pls)$

#### value

980. collect\_state: PLS

980.  $\text{collect\_state}(pls) \equiv \{pls\} \cup \text{obs\_Us}(pls) \setminus Pl$

## I.2 Endurants: Internal Qualities

### I.2.1 Unique Identification

981. The pipeline system, as such,

982. has a unique identifier, distinct (different) from its pipeline unit identifiers.

983. Each pipeline unit is uniquely distinguished by its unit identifier.

984. There is a state of all unique identifiers.

#### type

982. PLSI

983. UI

#### value

981. pls:PLS

982. uid\_PLS: PLS  $\rightarrow$  PLSI

983. uid\_U: U  $\rightarrow$  UI

#### variable

984.  $\sigma_{uid} := \{ \text{uid\_PLS}(pls) \} \cup \text{xtr\_UIs}(pls)$

#### axiom

983.  $\forall u, u': U \bullet \{u, u'\} \subseteq \text{obs\_Us}(pls) \Rightarrow (u \neq u' \Rightarrow \text{uid\_UI}(u) \neq \text{uid\_UI}(u'))$

983.  $\wedge \text{uid\_PLS}(pls) \notin \{ \text{uid\_UI}(u) \mid u: U \bullet u \in \text{obs\_Us}(pls) \}$

985. From a pipeline system one can observe the set of all unique unit identifiers.

#### value

985. xtr\_UIs: PLS  $\rightarrow$  UI-set

985.  $\text{xtr\_UIs}(pls) \equiv \{ \text{uid\_UI}(u) \mid u: U \bullet u \in \text{obs\_Us}(pls) \}$

---

<sup>2</sup>wf\_Mereology, wf\_Routes and wf\_Metrics will be explained in Sects. I.2.2.2 on the next page, I.2.3.2 on page 266, and I.2.4.3 on page 269.

986. We can prove that the number of unique unit identifiers of a pipeline system equals that of the units of that system.

**theorem:**

986.  $\forall \text{pls:PLS} \bullet \text{card obs\_Us(pl)} = \text{card xtr\_Uls(pls)}$

## I.2.2 Mereology

### I.2.2.1 PLS Mereology

987. The mereology of a pipeline system is the set of unique identifiers of all the units of that system.

**type**

987.  $\text{PLS\_Mer} = \text{UI-set}$

**value**

987.  $\text{mereo\_PLS: PLS} \rightarrow \text{PLS\_Mer}$

**axiom**

987.  $\forall \text{uis:PLS\_Mer} \bullet \text{uis} = \text{card xtr\_Uls(pls)}$

### I.2.2.2 Unit Mereologies

988. Each unit is connected to zero, one or two other existing input units and zero, one or two other existing output units as follows:

- (a) A well unit is connected to exactly one output unit (and, hence, has no “input”).
- (b) A pipe unit is connected to exactly one input unit and one output unit.
- (c) A pump unit is connected to exactly one input unit and one output unit.
- (d) A valve is connected to exactly one input unit and one output unit.
- (e) A fork is connected to exactly one input unit and two distinct output units.
- (f) A join is connected to exactly two distinct input units and one output unit.
- (g) A plate is connected to exactly one unit.
- (h) A sink is connected to exactly one input unit (and, hence, has no “output”).

**type**

988.  $\text{MER} = \text{UI-set} \times \text{UI-set}$

**value**

988.  $\text{mereo\_U: U} \rightarrow \text{MER}$

**axiom**

988.  $\text{wf\_Mereology: PLS} \rightarrow \text{Bool}$

988.  $\text{wf\_Mereology(pls)} \equiv$

988.  $\forall u:\text{U} \bullet u \in \text{obs\_Us(pls)} \Rightarrow$

988. **let**  $(\text{iuis}, \text{ouis}) = \text{mereo\_U}(u)$  **in**  $\text{iuis} \cup \text{ouis} \subseteq \text{xtr\_Uls(pls)} \wedge$

988. **case**  $(u, (\text{card iuis}, \text{card ouis}))$  **of**

988a.  $(\text{mk\_We}(we), (0, 1)) \rightarrow \text{true},$

988b.  $(\text{mk\_Pi}(pi), (1, 1)) \rightarrow \text{true},$

988c.  $(\text{mk\_Pu}(pu), (1, 1)) \rightarrow \text{true},$

988d.  $(\text{mk\_Va}(va), (1, 1)) \rightarrow \text{true},$

988e.  $(\text{mk\_Fo}(fo), (1, 1)) \rightarrow \text{true},$

988f.  $(\text{mk\_Jo}(jo), (1, 1)) \rightarrow \text{true},$

988f.  $(\text{mk\_PI}(pl), (0, 1)) \rightarrow \text{true},$  “begin”

988f.  $(\text{mk\_PI}(pl), (1, 0)) \rightarrow \text{true},$  “end”

988h.  $(\text{mk\_Si}(si), (1, 1)) \rightarrow \text{true},$

988.  $\_ \rightarrow \text{false end end}$

### I.2.3 Pipeline Concepts, I

#### I.2.3.1 Pipe Routes

989. A route (of a pipeline system) is a sequence of connected units (of the pipeline system).

990. A route descriptor is a sequence of unit identifiers and the connected units of a route (of a pipeline system).

##### type

989.  $R' = U^\omega$

989.  $R = \{ | r:Route' \bullet wf\_Route(r) | \}$

990.  $RD = UI^\omega$

##### axiom

990.  $\forall rd:RD \bullet \exists r:R \bullet rd = descriptor(r)$

##### value

990.  $descriptor: R \rightarrow RD$

990.  $descriptor(r) \equiv \langle uid\_UI(r[i]) | i: \mathbf{Nat} \bullet 1 \leq i \leq \mathbf{len} \ r \rangle$

991. Two units are adjacent if the output unit identifiers of one shares a unique unit identifier with the input identifiers of the other.

##### value

991.  $adjacent: U \times U \rightarrow \mathbf{Bool}$

991.  $adjacent(u, u') \equiv \mathbf{let} \ (,ouis) = mereo\_U(u), (,iuis) = mereo\_U(u') \ \mathbf{in} \ ouis \cap \ iuis \neq \ \{\} \ \mathbf{end}$

992. Given a pipeline system, *pls*, one can identify the (possibly infinite) set of (possibly infinite) routes of that pipeline system.

(a) The empty sequence,  $\langle \rangle$ , is a route of *pls*.

(b) Let  $u, u'$  be any units of *pls*, such that an output unit identifier of  $u$  is the same as an input unit identifier of  $u'$  then  $\langle u, u' \rangle$  is a route of *pls*.

(c) If  $r$  and  $r'$  are routes of *pls* such that the last element of  $r$  is the same as the first element of  $r'$ , then  $r \hat{\mathbf{tl}} r'$  is a route of *pls*.

(d) No sequence of units is a route unless it follows from a finite (or an infinite) number of applications of the basis and induction clauses of Items 992a–992c.

##### value

992.  $Routes: PLS \rightarrow RD\text{-infset}$

992.  $Routes(pls) \equiv$

992a.  $\mathbf{let} \ rs = \langle \rangle \cup$

992b.  $\{ \langle uid\_UI(u), uid\_UI(u') \rangle | u, u': U \bullet \{u, u'\} \subseteq obs\_Us(pls) \wedge adjacent(u, u') \}$

992c.  $\cup \{ r \hat{\mathbf{tl}} r' | r, r': R \bullet \{r, r'\} \subseteq rs \}$

992d.  $\mathbf{in} \ rs \ \mathbf{end}$

#### I.2.3.2 Well-formed Routes

993. A route is acyclic if no two route positions reveal the same unique unit identifier.

##### value

993.  $is\_acyclic\_Route: R \rightarrow \mathbf{Bool}$

993.  $is\_acyclic\_Route(r) \equiv \sim \exists \ i, j: \mathbf{Nat} \bullet \{i, j\} \subseteq \mathbf{inds} \ r \wedge i \neq j \wedge r[i] = r[j]$

994. A pipeline system is well-formed if none of its routes are circular (and all of its routes embedded in well-to-sink routes).

**value**

994.  $\text{wf\_Routes}: \text{PLS} \rightarrow \mathbf{Bool}$   
 994.  $\text{wf\_Routes}(\text{pls}) \equiv$   
 994.  $\text{non\_circular}(\text{pls}) \wedge \text{are\_embedded\_Routes}(\text{pls})$

994.  $\text{is\_non\_circular\_PLS}: \text{PLS} \rightarrow \mathbf{Bool}$   
 994.  $\text{is\_non\_circular\_PLS}(\text{pls}) \equiv$   
 994.  $\forall r:R \bullet r \in \text{routes}(\text{p}) \wedge \text{acyclic\_Route}(r)$

995. We define well-formedness in terms of well-to-sink routes, i.e., routes which start with a well unit and end with a sink unit.

**value**

995.  $\text{well\_to\_sink\_Routes}: \text{PLS} \rightarrow \mathbf{R\text{-set}}$   
 995.  $\text{well\_to\_sink\_Routes}(\text{pls}) \equiv$   
 995. **let**  $\text{rs} = \text{Routes}(\text{pls})$  **in**  
 995.  $\{r|r:R \bullet r \in \text{rs} \wedge \text{is\_We}(r[1]) \wedge \text{is\_Si}(r[\text{len } r])\}$  **end**

996. A pipeline system is well-formed if all of its routes are embedded in well-to-sink routes.

996.  $\text{are\_embedded\_Routes}: \text{PLS} \rightarrow \mathbf{Bool}$   
 996.  $\text{are\_embedded\_Routes}(\text{pls}) \equiv$   
 996. **let**  $\text{wsrs} = \text{well\_to\_sink\_Routes}(\text{pls})$  **in**  
 996.  $\forall r:R \bullet r \in \text{Routes}(\text{pls}) \Rightarrow$   
 996.  $\exists r':R, i, j: \mathbf{Nat} \bullet$   
 996.  $r' \in \text{wsrs}$   
 996.  $\wedge \{i, j\} \subseteq \mathbf{inds } r' \wedge i \leq j$   
 996.  $\wedge r = \langle r'[k] | k: \mathbf{Nat} \bullet i \leq k \leq j \rangle$  **end**

### I.2.3.3 Embedded Routes

997. For every route we can define the set of all its embedded routes.

**value**

997.  $\text{embedded\_Routes}: R \rightarrow \mathbf{R\text{-set}}$   
 997.  $\text{embedded\_Routes}(r) \equiv \{\langle r[k] | k: \mathbf{Nat} \bullet i \leq k \leq j \rangle \mid i, j: \mathbf{Nat} \bullet i \{i, j\} \subseteq \mathbf{inds}(r) \wedge i \leq j\}$

### I.2.3.4 A Theorem

998. The following theorem is conjectured:

- (a) the set of all routes (of the pipeline system)
- (b) is the set of all well-to-sink routes (of a pipeline system) and
- (c) all their embedded routes

**theorem:**

998.  $\forall \text{pls}: \text{PLS} \bullet$   
 998. **let**  $\text{rs} = \text{Routes}(\text{pls})$ ,  
 998.  $\text{wsrs} = \text{well\_to\_sink\_Routes}(\text{pls})$  **in**



998a.  $rs =$   
 998b.  $wrs \cup$   
 998c.  $\cup \{ \{r'|r':R \cdot r' \in \text{is\_embedded\_Routes}(r'')\} \mid r'':R \cdot r'' \in wrs \}$   
 997. **end**

### I.2.3.5 Fluids

999. The only fluid of concern to pipelines is the gas<sup>3</sup> or liquid<sup>4</sup> which the pipes transport<sup>5</sup>.

**type**  
 999.  $\text{GoL} [ = M ]$   
**value**  
 999.  $\text{obs\_GoL}: U \rightarrow \text{GoL}$

## I.2.4 Attributes

### I.2.4.1 Unit Flow Attributes

1000. A number of attribute types characterise units:

- (a) estimated current well capacity (barrels of oil, etc.),
- (b) pump height (a static attribute),
- (c) current pump status (not pumping, pumping; a programmable attribute),
- (d) current valve status (closed, open; a programmable attribute) and
- (e) flow (barrels/second, a biddable attribute).

**type**  
 1000a.  $\text{WellCap}$   
 1000b.  $\text{Pump\_Height}$   
 1000c.  $\text{Pump\_State} == \{ \text{not\_pumping, pumping} \}$   
 1000d.  $\text{Valve\_State} == \{ \text{closed, open} \}$   
 1000e.  $\text{Flow}$

1001. Flows can be added and subtracted,

1002. added distributively and

1003. flows can be compared.

**value**  
 1001.  $\oplus, \ominus: \text{Flow} \times \text{Flow} \rightarrow \text{Flow}$   
 1002.  $\oplus: \text{Flow-set} \rightarrow \text{Flow}$   
 1003.  $<, \leq, =, \neq, \geq, >: \text{Flow} \times \text{Flow} \rightarrow \text{Bool}$

1004. Properties of pipeline units include

- (a) estimated current well capacity (barrels of oil, etc.) [a biddable attribute],
- (b) pipe length [a static attribute],

<sup>3</sup>Gaseous materials include: air, gas, etc.

<sup>4</sup>Liquid materials include water, oil, etc.

<sup>5</sup>The description of this document is relevant only to gas or oil pipelines.

- (c) current pump height [a biddable attribute],
- (d) current valve open/close status [a programmable attribute],
- (e) current [ $\mathcal{L}$ aminar] in-flow at unit input [a monitorable attribute],
- (f) current [ $\mathcal{L}$ aminar] in-flow leak at unit input [a monitorable attribute],
- (g) maximum [ $\mathcal{L}$ aminar] guaranteed in-flow leak at unit input [a static attribute],
- (h) current [ $\mathcal{L}$ aminar] leak unit interior [a monitorable attribute],
- (i) current [ $\mathcal{L}$ aminar] flow in unit interior [a monitorable attribute],
- (j) maximum [ $\mathcal{L}$ aminar] guaranteed flow in unit interior [a monitorable attribute],
- (k) current [ $\mathcal{L}$ aminar] out-flow at unit output [a monitorable attribute],
- (l) current [ $\mathcal{L}$ aminar] out-flow leak at unit output [a monitorable attribute] and
- (m) maximum guaranteed [ $\mathcal{L}$ aminar] out-flow leak at unit output [a static attribute].

|                                              |       |                                                        |
|----------------------------------------------|-------|--------------------------------------------------------|
| <b>type</b>                                  | 1004b | attr_LEN: $Pi \rightarrow LEN$                         |
| 1004e In_Flow = Flow                         | 1004c | attr_Height: $Pu \rightarrow Height$                   |
| 1004f In_Leak = Flow                         | 1004d | attr_ValSta: $Va \rightarrow VaSta$                    |
| 1004g Max_In_Leak = Flow                     | 1004e | attr_In_Flow: $U \rightarrow UI \rightarrow Flow$      |
| 1004h Body_Flow = Flow                       | 1004f | attr_In_Leak: $U \rightarrow UI \rightarrow Flow$      |
| 1004i Body_Leak = Flow                       | 1004g | attr_Max_In_Leak: $U \rightarrow UI \rightarrow Flow$  |
| 1004j Max_Flow = Flow                        | 1004h | attr_Body_Flow: $U \rightarrow Flow$                   |
| 1004k Out_Flow = Flow                        | 1004i | attr_Body_Leak: $U \rightarrow Flow$                   |
| 1004l Out_Leak = Flow                        | 1004j | attr_Max_Flow: $U \rightarrow Flow$                    |
| 1004m Max_Out_Leak = Flow                    | 1004k | attr_Out_Flow: $U \rightarrow UI \rightarrow Flow$     |
| <b>value</b>                                 | 1004l | attr_Out_Leak: $U \rightarrow UI \rightarrow Flow$     |
| 1004a attr_WellCap: $We \rightarrow WellCap$ | 1004m | attr_Max_Out_Leak: $U \rightarrow UI \rightarrow Flow$ |

1005. Summarising we can define a two notions of flow:

- (a) static and
- (b) monitorable.

**type**

- 1005a Sta\_Flows =  $Max\_In\_Leak \times In\_Max\_Flow > Max\_Out\_Leak$
- 1005b Mon\_Flows =  $In\_Flow \times In\_Leak \times Body\_Flow \times Body\_Leak \times Out\_Flow \times Out\_Leak$

### I.2.4.2 Unit Metrics

Pipelines are laid out in the terrain. Units have length and diameters. Units are positioned in space: have altitude, longitude and latitude positions of its one, two or three connection PoinTs<sup>6</sup>.

- 1006. length (a static attribute),
- 1007. diameter (a static attribute) and
- 1008. position (a static attribute).

**type**

- 1006. LEN
- 1007.  $\bigcirc$
- 1008. POS ==  $mk\_One(pt:PT) \mid mk\_Two(ipt:PT,opt:PT)$
- 1008.  $\mid mk\_OneTwo(ipt:PT,opts:(lpt:PT,rpt:PT))$

<sup>6</sup>1 for *wells*, *plates* and *sinks*; 2 for *pipes*, *pumps* and *valves*; 1+2 for *forks*, 2+1 for *joints*.

1008.  $\text{mk\_TwoOne}(\text{ipts}:(\text{lpt}:\text{PT},\text{rpt}:\text{PT}),\text{opt}:\text{PT})$   
 1008.  $\text{PT} = \text{Alt} \times \text{Lon} \times \text{Lat}$   
 1008.  $\text{Alt}, \text{Lon}, \text{Lat} = \dots$   
**value**  
 1006.  $\text{attr\_LEN}: \text{U} \rightarrow \text{LEN}$   
 1007.  $\text{attr\_}\bigcirc\text{: U} \rightarrow \bigcirc$   
 1008.  $\text{attr\_POS}: \text{U} \rightarrow \text{POS}$

We can summarise the metric attributes:

1009. Units are subject to either of four (mutually exclusive) metrics:

- (a) Length, diameter and a one point position.
- (b) Length, diameter and a two points position.
- (c) Length, diameter and a one+two points position.
- (d) Length, diameter and a two+one points position.

**type**

1009.  $\text{Unit\_Sta} = \text{Sta1\_Metric} \mid \text{Sta2\_Metric} \mid \text{Sta12\_Metric} \mid \text{Sta21\_Metric}$   
 1009a  $\text{Sta1\_Metric} = \text{LEN} \times \emptyset \times \text{mk\_One}(\text{pt}:\text{PT})$   
 1009b  $\text{Sta2\_Metric} = \text{LEN} \times \emptyset \times \text{mk\_Two}(\text{ipt}:\text{PT},\text{opt}:\text{PT})$   
 1009c  $\text{Sta12\_Metric} = \text{LEN} \times \emptyset \times \text{mk\_OneTwo}(\text{ipt}:\text{PT},\text{opts}:(\text{lpt}:\text{PT},\text{rpt}:\text{PT}))$   
 1009d  $\text{Sta21\_Metric} = \text{LEN} \times \emptyset \times \text{mk\_TwpOne}(\text{ipts}:(\text{lpt}:\text{PT},\text{rpt}:\text{PT}),\text{opt}:\text{PT})$

### I.2.4.3 Wellformed Unit Metrics

The points positions of neighbouring units must “fit” one-another.

1010. Without going into details we can define a predicate,  $\text{wf\_Metrics}$ , that applies to a pipeline system and yields **true** iff neighbouring units must “fit” one-another.

**value**

1010.  $\text{wf\_Metrics}: \text{PLS} \rightarrow \text{Bool}$   
 1010.  $\text{wf\_Metrics}(\text{pls}) \equiv \dots$

### I.2.4.4 Summary

We summarise the static, monitorable and programmable attributes for each manifest part of the pipeline system:

**type**

$\text{PLS\_Sta} = \text{PLS\_net} \times \dots$   
 $\text{PLS\_Mon} = \dots$   
 $\text{PLS\_Prg} = \text{PLS\_}\Sigma \times \dots$   
 $\text{Well\_Sta} = \text{Sta1\_Metric} \times \text{Sta\_Flows} \times \text{Orig\_Cap} \times \dots$   
 $\text{Well\_Mon} = \text{Mon\_Flows} \times \text{Well\_Cap} \times \dots$   
 $\text{Well\_Prg} = \dots$   
 $\text{Pipe\_Sta} = \text{Sta2\_Metric} \times \text{Sta\_Flows} \times \text{LEN} \times \dots$   
 $\text{Pipe\_Mon} = \text{Mon\_Flows} \times \text{In\_Temp} \times \text{Out\_Temp} \times \dots$   
 $\text{Pipe\_Prg} = \dots$   
 $\text{Pump\_Sta} = \text{Sta2\_Metric} \times \text{Sta\_Flows} \times \text{Pump\_Height} \times \dots$   
 $\text{Pump\_Mon} = \text{Mon\_Flows} \times \dots$   
 $\text{Pump\_Prg} = \text{Pump\_State} \times \dots$

Valve\_Sta = Sta2\_Metric×Sta\_Flows×...  
 Valve\_Mon = Mon\_Flows×In\_Temp×Out\_Temp×...  
 Valve\_Prg = Valve\_State×...  
 Fork\_Sta = Sta12\_Metric×Sta\_Flows×...  
 Fork\_Mon = Mon\_Flows×In\_Temp×Out\_Temp×...  
 Fork\_Prg = ...  
 Join\_Sta = Sta21\_Metric×Sta\_Flows×...  
 Join\_Mon = Mon\_Flows×In\_Temp×Out\_Temp×...  
 Join\_Prg = ...  
 Sink\_Sta = Sta1\_Metric×Sta\_Flows×Max\_Vol×...  
 Sink\_Mon = Mon\_Flows×Curr\_Vol×In\_Temp×Out\_Temp×...  
 Sink\_Prg = ...

1011. Corresponding to the above three attribute categories we can define “collective” attribute observers:

**value**

1011. sta\_A\_We: We → Sta1\_Metric×Sta\_Flows×Orig\_Cap×...  
 1011. mon\_A\_We: We →  $\eta$ Mon\_Flows× $\eta$ Well\_Cap× $\eta$ In\_Temp× $\eta$ Out\_Temp×...  
 1011. prg\_A\_We: We → ...  
 1011. sta\_A\_Pi: Pi → Sta2\_Metric×Sta\_Flows×LEN×...  
 1011. mon\_A\_Pi: Pi →  $\mathcal{N}$ Mon\_Flows× $\eta$ In\_Temp× $\eta$ Out\_Temp×...  
 1011. prg\_A\_Pi: Pi → ...  
 1011. sta\_A\_Pu: Pu → Sta2\_Metric×Sta\_Flows×LEN×...  
 1011. mon\_A\_Pu: Pu →  $\mathcal{N}$ Mon\_Flows× $\eta$ In\_Temp× $\eta$ Out\_Temp×...  
 1011. prg\_A\_Pu: Pu → Pump\_State×...  
 1011. sta\_A\_Va: Va → Sta2\_Metric×Sta\_Flows×LEN×...  
 1011. mon\_A\_Va: Va →  $\mathcal{N}$ Mon\_Flows× $\eta$ In\_Temp× $\eta$ Out\_Temp×...  
 1011. prg\_A\_Va: Va → Valve\_State×...  
 1011. sta\_A\_Fo: Fo → Sta12\_Metric×Sta\_Flows×...  
 1011. mon\_A\_Fo: Fo →  $\mathcal{N}$ Mon\_Flows× $\eta$ In\_Temp× $\eta$ Out\_Temp×...  
 1011. prg\_A\_Fo: Fo → ...  
 1011. sta\_A\_Jo: Jo → Sta21\_Metric×Sta\_Flows×...  
 1011. mon\_A\_Jo: Jo → Mon\_Flows× $\eta$ In\_Temp× $\eta$ Out\_Temp×...  
 1011. prg\_A\_Jo: Jo → ...  
 1011. sta\_A\_Si: Si → Sta1\_Metric×Sta\_Flows×Max\_Vol×...  
 1011. mon\_A\_Si: Si →  $\mathcal{N}$ Mon\_Flows× $\eta$ In\_Temp× $\eta$ Out\_Temp×...  
 1011. prg\_A\_Si: Si → ...  
 1011.  $\mathcal{N}$ Mon\_Flows  $\equiv$  ( $\eta$ In\_Flow, $\eta$ In\_Leak, $\eta$ Body\_Flow, $\eta$ Body\_Leak, $\eta$ Out\_Flow, $\eta$ Out\_Leak)

Monitored flow attributes are [to be] passed as arguments to behaviours *by reference* so that their monitorable attribute values can be sampled.

#### I.2.4.5 Fluid Attributes

Fluids, we here assume, oil, as it appears in the pipeline units have no unique identity, have not mereology, but does have attributes: hydrocarbons consisting predominantly of aliphatic, alicyclic and aromatic hydrocarbons. It may also contain small amounts of nitrogen, oxygen, and sulfur compounds

1012. We shall simplify, just for illustration, crude oil fluid of units to have these attributes:

- (a) volume,

- (b) viscosity,
- (c) temperature,
- (d) paraffin content (%age),
- (e) naphthenes content (%age),

| <b>type</b>     | <b>value</b>                                     |
|-----------------|--------------------------------------------------|
| 1012. Oil       | 1012b. obs_Oil: $U \rightarrow Oil$              |
| 1012a. Vol      | 1012a. attr_Vol: $Oil \rightarrow Vol$           |
| 1012b. Visc     | 1012b. attr_Visc: $Oil \rightarrow Visc$         |
| 1012c. Temp     | 1012c. attr_Temp: $Oil \rightarrow Temp$         |
| 1012d. Paraffin | 1012d. attr_Paraffin: $Oil \rightarrow Paraffin$ |
| 1012e. Naphtene | 1012e. attr_Naphtene: $Oil \rightarrow Naphtene$ |

#### I.2.4.6 Pipeline System Attributes

The “root” pipeline system is a compound. In its transcendently deduced behavioral form it is, amongst other “tasks”, entrusted with the monitoring and control of all its units. To do so it must, as a basically static attribute possess awareness, say in the form of a net diagram of how these units are interconnected, together with all their internal qualities, by type and by value. Next we shall give a very simplified account of the possible pipeline system attribute.

1013. We shall make use, in this example, of just a simple pipeline state,  $pls_{\omega}$ .

The pipeline state,  $pls_{\omega}$ , embodies all the information that is relevant to the monitoring and control of an entire pipeline system, whether static or dynamic.

**type**  
1013. PLS\_Ω

#### I.2.5 Pipeline Concepts, II: Flow Laws

1014. “What flows in, flows out !”. For  $\mathcal{L}$ aminar flows: for any non-well and non-sink unit the sums of input leaks and in-flows equals the sums of unit and output leaks and out-flows.

**Law:**

1014.  $\forall u:U \setminus We \setminus Si \bullet$   
 1014.  $sum\_in\_leaks(u) \oplus sum\_in\_flows(u) =$   
 1014.  $attr\_body\_Leak_{\mathcal{L}}(u) \oplus$   
 1014.  $sum\_out\_leaks(u) \oplus sum\_out\_flows(u)$

**value**  
 $sum\_in\_leaks: U \rightarrow Flow$   
 $sum\_in\_leaks(u) \equiv \mathbf{let} (iuis, ) = mereo\_U(u) \mathbf{in} \oplus \{attr\_In\_Leak_{\mathcal{L}}(u)(ui) | ui:U \bullet ui \in iuis\} \mathbf{end}$   
 $sum\_in\_flows: U \rightarrow Flow$   
 $sum\_in\_flows(u) \equiv \mathbf{let} (iuis, ) = mereo\_U(u) \mathbf{in} \oplus \{attr\_In\_Flow_{\mathcal{L}}(u)(ui) | ui:U \bullet ui \in iuis\} \mathbf{end}$   
 $sum\_out\_leaks: U \rightarrow Flow$   
 $sum\_out\_leaks(u) \equiv \mathbf{let} (,ouis) = mereo\_U(u) \mathbf{in} \oplus \{attr\_Out\_Leak_{\mathcal{L}}(u)(ui) | ui:U \bullet ui \in ouis\} \mathbf{end}$   
 $sum\_out\_flows: U \rightarrow Flow$   
 $sum\_out\_flows(u) \equiv \mathbf{let} (,ouis) = mereo\_U(u) \mathbf{in} \oplus \{attr\_Out\_Leak_{\mathcal{L}}(u)(ui) | ui:U \bullet ui \in ouis\} \mathbf{end}$

1015. “What flows out, flows in !”. For  $\mathcal{L}$ aminar flows: for any adjacent pairs of units the output flow at one unit connection equals the sum of adjacent unit leak and in-flow at that connection.

**Law:**

1015.  $\forall u, u': U \bullet \text{adjacent}(u, u') \Rightarrow$   
 1015. **let** ( $\text{ouis}$ )= $\text{mereo}_U(u)$ , ( $\text{iuis}'$ )= $\text{mereo}_U(u')$  **in**  
 1015. **assert:**  $\text{uid}_U(u') \in \text{ouis} \wedge \text{uid}_U(u) \in \text{iuis}'$   
 1015.  $\text{attr\_Out\_Flow}_{\mathcal{L}}(u)(\text{uid}_U(u')) =$   
 1015.  $\text{attr\_In\_Leak}_{\mathcal{L}}(u)(\text{uid}_U(u)) \oplus \text{attr\_In\_Flow}_{\mathcal{L}}(u')(\text{uid}_U(u))$  **end**

These “laws” should hold for a pipeline system without plates.

### I.3 Perdurants

We follow the ontology of Fig. 2.1 on page 12, the right-hand dashed box labeled *Perdurants* and the right-hand vertical and horizontal lines.

#### I.3.1 State

We introduce concepts of *manifest* and *structure* endurants. The former are such compound endurants (Cartesians of sets) to which we ascribe internal qualities; the latter are such compound endurants (Cartesians of sets) to which we **do not** ascribe internal qualities. The distinction is pragmatic.

1016. For any given pipeline system we suggest the state to consist of the manifest endurants of all its non-plate units.

**value**

1016.  $\sigma = \text{obs}_U(\text{pls})$

#### I.3.2 Channel

1017. There is a [global] array channel indexed by a “set pair” of distinct manifest endurant part identifiers – signifying the possibility of the synchronisation and communication between any pair of pipeline units and between these and the pipeline system.

**channel**

1017.  $\{ \text{ch}[\{i, j\}] \mid \{i, j\} : (\text{PLSI} \mid \text{UI}) \bullet \{i, j\} \subseteq \sigma_{id} \}$

#### I.3.3 Actions

These are, informally, some of the actions of a pipeline system:

1018. **start pumping:** from a state of not pumping to a state of pumping “at full blast!”.<sup>7</sup>  
 1019. **stop pumping:** from a state of (full) pumping to a state of no pumping at all.  
 1020. **open valve:** from a state of a fully closed valve to a state of fully open valve.<sup>8</sup>  
 1021. **close valve:** from a state of a fully opened valve to a state of fully closed valve.

We shall not define these actions in this paper. But they will be referred to in the *pipeline\_system* (Items 1040a, 1040b, 1040c), the *pump* (Items 1043a, 1043b) and the *valve* (Items 1046a, 1046b) behaviours.

<sup>7</sup>– that is, we simplify, just for the sake of illustration, and do not consider “intermediate” states of pumping.

<sup>8</sup>– cf. Footnote 7.

### I.3.4 Behaviours

#### I.3.4.1 Behaviour Kinds

There are eight kinds of behaviours:

- |                                                   |                                      |
|---------------------------------------------------|--------------------------------------|
| 1022. the pipeline system behaviour; <sup>9</sup> | 1026. the [generic] valve behaviour, |
| 1023. the [generic] well behaviour,               | 1027. the [generic] fork behaviour,  |
| 1024. the [generic] pipe behaviour,               | 1028. the [generic] join behaviour,  |
| 1025. the [generic] pump behaviour,               | 1029. the [generic] sink behaviour.  |

#### I.3.4.2 Behaviour Signatures

1030. The *pipeline\_system* behaviour, *pls*,
1031. The *well* behaviour signature lists the unique well identifier, the well mereology, the static well attributes, the monitorable well attributes, the programmable well attributes and the channels over which the well [may] interact with the pipeline system and a pipeline unit.
1032. The *pipe* behaviour signature lists the unique pipe identifier, the pipe mereology, the static pipe attributes, the monitorable pipe attributes, the programmable pipe attributes and the channels over which the pipe [may] interact with the pipeline system and its two neighbouring pipeline units.
1033. The *pump* behaviour signature lists the unique pump identifier, the pump mereology, the static pump attributes, the monitorable pump attributes, the programmable pump attributes and the channels over which the pump [may] interact with the pipeline system and its two neighbouring pipeline units.
1034. The *valve* behaviour signature lists the unique valve identifier, the valve mereology, the static valve attributes, the monitorable valve attributes, the programmable valve attributes and the channels over which the valve [may] interact with the pipeline system and its two neighbouring pipeline units.
1035. The *fork* behaviour signature lists the unique fork identifier, the fork mereology, the static fork attributes, the monitorable fork attributes, the programmable fork attributes and the channels over which the fork [may] interact with the pipeline system and its three neighbouring pipeline units.
1036. The *join* behaviour signature lists the unique join identifier, the join mereology, the static join attributes, the monitorable join attributes, the programmable join attributes and the channels over which the join [may] interact with the pipeline system and its three neighbouring pipeline units.
1037. The *sink* behaviour signature lists the unique sink identifier, the sink mereology, the static sing attributes, the monitorable sing attributes, the programmable sink attributes and the channels over which the sink [may] interact with the pipeline system and its one or more pipeline units.

#### value

1030.  $pls: pls_o:PLSI \rightarrow pls\_mer:PLS\_Mer \rightarrow PLS\_Sta \rightarrow PLS\_Mon \rightarrow$   
 1030.  $PLS\_Prg \rightarrow \{ ch[\{plsi,ui\}] \mid ui:UI \bullet ui \in \sigma_{ui} \}$  **Unit**

<sup>9</sup>This “PLS” behaviour summarises the either global, i.e., SCADA<sup>10</sup>-like behaviour, or the fully distributed, for example, manual, human-operated behaviour of the monitoring and control of the entire pipeline system.

<sup>10</sup>Supervisory Control And Data Acquisition

```

1031. well: wid:WI → well_mer:MER → Well_Sta → Well_mon →
1031. Well_Prgr → { ch[{plsi,ui}] | wi:WI • ui ∈ σui } Unit
1032. pipe: UI → pipe_mer:MER → Pipe_Sta → Pipe_mon →
1032. Pipe_Prgr → { ch[{plsi,ui}] | ui:UI • ui ∈ σui } Unit
1033. pump: pi:UI → pump_mer:MER → Pump_Sta → Pump_Mon →
1033. Pump_Prgr → { ch[{plsi,ui}] | ui:UI • ui ∈ σui } Unit
1034. valve: vi:UI → valve_mer:MER → Valve_Sta → Valve_Mon →
1034. Valve_Prgr → { ch[{plsi,ui}] | ui:UI • ui ∈ σui } Unit
1035. fork: fi:FI → fork_mer:MER → Fork_Sta → Fork_Mon →
1035. Fork_Prgr → { ch[{plsi,ui}] | ui:UI • ui ∈ σui } Unit
1036. join: ji:JI → join_mer:MER → Join_Sta → Join_Mon →
1036. Join_Prgr → { ch[{plsi,ui}] | ui:UI • ui ∈ σui } Unit
1037. sink: si:SI → sink_mer:MER → Sink_Sta → Sink_Mon →
1037. Sink_Prgr → { ch[{plsi,ui}] | ui:UI • ui ∈ σui } Unit

```

**I.3.4.2.1 Behaviour Definitions** We show the definition of only three behaviours:

- the **pipe\_line\_system** behaviour,
- the **pump** behaviour and
- the **valve** behaviour.

#### **I.3.4.2.2 The Pipeline System Behaviour**

```

1038. The pipeline system behaviour
1039. calculates, based on its programmable state, its next move;
1040. if that move is [to be] an action on a named
 (a) pump, whether to start or stop pumping, then the named pump is so informed, where-
 upon the pipeline system behaviour resumes in the new pipeline state; or
 (b) valve, whether to open or close the valve, then the named valve is so informed, where-
 upon the pipeline system behaviour resumes in the new pipeline state; or
 (c) unit, to collect its monitorable attribute values for monitoring, whereupon the pipeline
 system behaviour resumes in the further updated pipeline state;
 (d) et cetera;

```

**value**

```

1038. pls(plsi)(uis)(pls_msta)(pls_mon)(pls_ω) ≡
1039. let (to_do,pls_ω') = calculate_next_move(plsi,pls_mer,pls_msta,pls_mon,pls_prgr) in
1040. case to_do of
1040a mk_Pump(pi,α) →
1040a ch[{plsi,pi}] ! α assert: α ∈ {stop_pumping,pump};
1040a pls(plsi)(pls_mer)(pls_msta)(pls_mon)(pls_ω'),
1040b mk_Valve(vi,α) →
1040b ch[{plsi,vi}] ! α assert: α ∈ {open_valve,close_valve};
1040b pls(plsi)(pls_mer)(pls_msta)(pls_mon)(pls_ω'),
1040c mk_Unit(ui,monitor) →
1040c ch[{plsi,ui}] ! monitor;
1040c pls(plsi)(pls_mer)(pls_msta)(pls_mon)(update_pls_ω(ch[{plsi,ui}] ?,ui)(pls_ω')),
1040d ... end
1038 end

```

We leave it to the reader to define the `calculate_next_move` function!



**I.3.4.2.3 The Pump Behaviours**

1041. The [generic] pump behaviour internal non-deterministically alternates between

1042. doing own work (...), or

1043. accepting pump directives from the pipeline behaviour.

- (a) If the directive is either to start or stop pumping, then that is what happens – whereupon the pump behaviour resumes in the new pumping state.
- (b) If the directive requests the values of all monitorable attributes, then these are *gathered*, communicated to the pipeline system behaviour – whereupon the pump behaviour resumes in the “old” state.

**value**

```

1041. pump(π)(pump_mer)(pump_sta)(pump_mon)(pump_prgr) \equiv
1042. ...
1043. \square let $\alpha = \text{ch}[\{\text{plsi}, \pi\}] ?$ in
1043. case α of
1043a. stop_pumping \vee pump
1043a. \rightarrow pump(π)(pump_mer)(pump_sta)(pump_mon)(α)11end,
1043b. monitor
1043b. \rightarrow let mvs = gather_monitorable_values(π , pump_mon) in
1043b. ch[\{\text{plsi}, \pi\}] ! mvs;
1043b. pump(π)(pump_mer)(pump_sta)(pump_mon)(pump_prgr) end
1043. end

```

We leave it to the reader to defined the `gather_monitorable_values` function.

**I.3.4.2.4 The Valve Behaviours**

1044. The [generic] valve behaviour internal non-deterministically alternates between

1045. doing own work (...), or

1046. accepting valve directives from the pipeline system.

- (a) If the directive is either to open or close the valve, then that is what happens – whereupon the pump behaviour resumes in the new valve state.
- (b) If the directive requests the values of all monitorable attributes, then these are *gathered*, communicated to the pipeline system behaviour – whereupon the valve behaviour resumes in the “old” state.

**value**

```

1044. valve(ν)(valv_mer)(valv_sta)(valv_mon)(valv_prgr) \equiv
1045. ...
1046. \square let $\alpha = \text{ch}[\{\text{plsi}, \pi\}] ?$ in
1046. case α of
1046a. open_valve \vee close_valve
1046a. \rightarrow valve(ν)(val_mer)(val_sta)(val_mon)(α)12end,
1046b. monitor
1046b. \rightarrow let mvs = gather_monitorable_values(ν , val_mon) in
1046b. ch[\{\text{plsi}, \pi\}] ! (ν , mvs);
1046b. valve(ν)(val_mer)(val_sta)(val_mon)(val_prgr) end
1046. end

```

<sup>11</sup>Updating the programmable pump state to either **stop\_pumping** or **pump** shall here be understood to mean that the pump is set to not pump, respectively to pump.

### I.3.4.3 Sampling Monitorable Attribute Values

Static and programmable attributes are, as we have seen, *passed by value* to behaviours. Monitorable attributes “surreptitiously” change their values so, as a technical point, these are *passed by reference* – by *passing attribute type names*.

1047. From the name,  $\eta A$ , of a monitorable attribute and the unique identifier,  $u_i$ , of the part having the named monitorable attribute one can then, “dynamically”, “on-the-fly”, as the part behaviour “moves-on”, retrieve the value of the monitorable attribute. This can be illustrated as follows:
1048. The unique identifier  $u_i$  is used in order to retrieve, from the global parts state,  $\sigma$ , that identified part,  $p$ .
1049. Then  $\text{attr\_A}$  is applied to  $p$ .

#### value

1047.  $\text{retr\_U}: \text{UI} \rightarrow \Sigma \rightarrow \text{U}$   
 1047.  $\text{retr\_U}(ui)(\sigma) \equiv \text{let } u:\text{U} \bullet u \in \sigma \wedge \text{uid\_U}(u)=ui \text{ in } u \text{ end}$   
 1048.  $\text{retr\_AttrVal}: \text{UI} \times \eta A \rightarrow \Sigma \rightarrow A$   
 1049.  $\text{retr\_AttrVal}(ui)(\eta A)(\sigma) \equiv \text{attr\_A}(\text{retr\_U}(ui)(\sigma))$

$\text{retr\_AttrVal}(\dots)(\dots)(\dots)$  can now be applied in the body of the behaviour definitions, for example in  $\text{gather\_monitorable\_values}$ .

### I.3.4.4 System Initialisation

System initialisation means to “morph” all manifest parts into their respective behaviours, initialising them with their respective attribute values.

1050. The *pipeline system* behaviour is initialised and “put” in parallel with the parallel compositions of
1051. all initialised *well*,
1052. all initialised *pipe*,
1053. all initialised *pump*,
1054. all initialised *valve*,
1055. all initialised *fork*,
1056. all initialised *join* and
1057. all initialised *sink* behaviours.<sup>13</sup>

#### value

1050.  $\text{pls}(\text{uid\_PLS}(\text{pls}))(\text{mereo\_PLS}(\text{pls}))((\text{pls}))((\text{pls}))(\text{pls}) \text{ pls-init-1700}$   
 1051.  $\parallel \parallel \{ \text{well}(\text{uid\_U}(\text{we}))(\text{mereo\_U}(\text{we}))(\text{sta\_A\_We}(\text{we}))(\text{mon\_A\_We}(\text{we}))(\text{prg\_A\_We}(\text{we})) \mid \text{we}:\text{Well} \bullet \text{w} \in \sigma \}$   
 1052.  $\parallel \parallel \{ \text{pipe}(\text{uid\_U}(\text{pi}))(\text{mereo\_U}(\text{pi}))(\text{sta\_A\_Pi}(\text{pi}))(\text{mon\_A\_Pi}(\text{pi}))(\text{prg\_A\_Pi}(\text{pi})) \mid \text{pi}:\text{Pi} \bullet \text{pi} \in \sigma \}$   
 1053.  $\parallel \parallel \{ \text{pump}(\text{uid\_U}(\text{pu}))(\text{mereo\_U}(\text{pu}))(\text{sta\_A\_Pu}(\text{pu}))(\text{mon\_A\_Pu}(\text{pu}))(\text{prg\_A\_Pu}(\text{pu})) \mid \text{pu}:\text{Pump} \bullet \text{pu} \in \sigma \}$   
 1054.  $\parallel \parallel \{ \text{valv}(\text{uid\_U}(\text{va}))(\text{mereo\_U}(\text{va}))(\text{sta\_A\_Va}(\text{va}))(\text{mon\_A\_Va}(\text{va}))(\text{prg\_A\_Va}(\text{va})) \mid \text{va}:\text{Well} \bullet \text{va} \in \sigma \}$   
 1055.  $\parallel \parallel \{ \text{fork}(\text{uid\_U}(\text{fo}))(\text{mereo\_U}(\text{fo}))(\text{sta\_A\_Fo}(\text{fo}))(\text{mon\_A\_Fo}(\text{fo}))(\text{prg\_A\_Fo}(\text{fo})) \mid \text{fo}:\text{Fork} \bullet \text{fo} \in \sigma \}$   
 1056.  $\parallel \parallel \{ \text{join}(\text{uid\_U}(\text{jo}))(\text{mereo\_U}(\text{jo}))(\text{sta\_A\_Jo}(\text{jo}))(\text{mon\_A\_J}(\text{jo}))(\text{prg\_A\_J}(\text{jo})) \mid \text{jo}:\text{Join} \bullet \text{jo} \in \sigma \}$   
 1057.  $\parallel \parallel \{ \text{sink}(\text{uid\_U}(\text{si}))(\text{mereo\_U}(\text{si}))(\text{sta\_A\_Si}(\text{si}))(\text{mon\_A\_Si}(\text{si}))(\text{prg\_A\_Si}(\text{si})) \mid \text{si}:\text{Sink} \bullet \text{si} \in \sigma \}$

The  $\text{sta\_...}$ ,  $\text{mon\_...}$ , and  $\text{prg\_A\_...}$  functions are defined in Items 1011 on page 270.

Note:  $\parallel \{ f(u)(\dots) \mid u:\text{U} \bullet u \in \{ \} \} \equiv ()$ .

## I.4 Review

TO BE WRITTEN

<sup>12</sup>Updating the programmable valve state to either **open\_valve** or **close\_valve** shall here be understood to mean that the valve is set to open, respectively to closed position.

<sup>13</sup>Plates are treated as are structures, i.e., not “behaviourised” !

# Appendix J

## Shipping

### Contents

---

|            |                                                 |            |
|------------|-------------------------------------------------|------------|
| <b>J.1</b> | <b>Informal Sketches of the Shipping Domain</b> | <b>279</b> |
| J.1.1      | The Purpose of A Domain Model for Shipping      | 279        |
| J.1.2      | A First Sketch                                  | 279        |
| J.1.3      | A Second Sketch                                 | 280        |
| J.1.3.1    | Strands of Interacting Sets of Behaviours       | 280        |
| J.1.3.2    | Freight                                         | 280        |
| J.1.3.2.1  | Freight as Endurants                            | 280        |
| J.1.3.2.2  | Freight as Behaviours                           | 280        |
| J.1.3.3    | Freight Forwarder Behaviour                     | 281        |
| J.1.3.4    | Shipping Line Behaviour                         | 281        |
| J.1.4      | Some Comments                                   | 282        |
| J.1.4.1    | Caveat Concerning Sketches                      | 282        |
| J.1.4.2    | The Insufficiency of Narrative Descriptions     | 282        |
| J.1.4.3    | What Do Formal Descriptions Contribute?         | 283        |
| J.1.4.4    | Limitations of Domain Models                    | 283        |
| J.1.4.5    | Families of Domain Models                       | 283        |
| J.1.4.6    | There is No “Standard Model”                    | 283        |
| <b>J.2</b> | <b>Endurants: External Qualities</b>            | <b>283</b> |
| J.2.1      | Freight                                         | 283        |
| J.2.2      | Endurant Sorts & Observers                      | 283        |
| J.2.3      | Endurant Values                                 | 285        |
| <b>J.3</b> | <b>Endurants: Internal Qualities</b>            | <b>286</b> |
| J.3.1      | Unique Identifiers                              | 286        |
| J.3.1.1    | Unique Identifier Types and Observers           | 286        |
| J.3.1.2    | Domain Unique Identifiers                       | 287        |
| J.3.1.3    | An Axiom                                        | 288        |
| J.3.1.4    | Retrieve Endurant Values                        | 288        |
| J.3.2      | Mereologies                                     | 288        |
| J.3.2.1    | A Shift in Modeling                             | 288        |
| J.3.2.2    | Mereology Types and Observers                   | 288        |
| J.3.2.2.1  | Harbour Mereology                               | 288        |
| J.3.2.2.2  | Vessel Mereology                                | 289        |
| J.3.2.2.3  | Shipping Line Mereology                         | 290        |
| J.3.2.2.4  | Freight Forwarder Mereology                     | 290        |

|       |                      |                                                  |                                                        |     |
|-------|----------------------|--------------------------------------------------|--------------------------------------------------------|-----|
|       | J.3.2.2.5            | Freight Mereology . . . . .                      | 291                                                    |     |
|       | J.3.2.2.6            | Passenger Mereology . . . . .                    | 291                                                    |     |
|       | J.3.2.2.7            | Waterways Mereology . . . . .                    | 292                                                    |     |
|       | J.3.2.2.8            | Landmass Mereology . . . . .                     | 292                                                    |     |
| J.3.3 | Attributes . . . . . |                                                  | 292                                                    |     |
|       | J.3.3.1              | Attribute Types and Observers . . . . .          | 292                                                    |     |
|       |                      | J.3.3.1.1 Freight Forwarder Attributes . . . . . | 293                                                    |     |
|       |                      | J.3.3.1.2 Shipping Line Attributes . . . . .     | 293                                                    |     |
|       |                      | J.3.3.1.3 Vessel Attributes . . . . .            | 293                                                    |     |
|       |                      | J.3.3.1.4 Harbour Attributes . . . . .           | 294                                                    |     |
|       |                      | J.3.3.1.5 Freight Attributes . . . . .           | 294                                                    |     |
|       | J.3.3.2              | Attribute Wellformedness . . . . .               | 294                                                    |     |
| J.4   | Perdurants . . . . . |                                                  | 294                                                    |     |
|       | J.4.1                | Freight as Endurants and as Behaviours . . . . . | 295                                                    |     |
|       | J.4.2                | Actions, Events and Behaviours . . . . .         | 295                                                    |     |
|       | J.4.3                | Global Freight Variable . . . . .                | 295                                                    |     |
|       | J.4.4                | Channels . . . . .                               | 295                                                    |     |
|       | J.4.5                | Behaviours . . . . .                             | 296                                                    |     |
|       |                      | J.4.5.1 Behaviour Signatures . . . . .           | 296                                                    |     |
|       |                      |                                                  | J.4.5.1.1 Freight Forwarder Signature . . . . .        | 296 |
|       |                      |                                                  | J.4.5.1.2 Shipping Line Signature . . . . .            | 296 |
|       |                      |                                                  | J.4.5.1.3 Vessel Signature . . . . .                   | 296 |
|       |                      |                                                  | J.4.5.1.4 Harbour Signature . . . . .                  | 296 |
|       |                      |                                                  | J.4.5.1.5 Freight Signature . . . . .                  | 296 |
|       |                      | J.4.5.2 Behaviour Definitions . . . . .          | 296                                                    |     |
|       |                      |                                                  | J.4.5.2.1 Freight Forwarder Definition . . . . .       | 296 |
|       |                      |                                                  | J.4.5.2.2 Shipping Line Behaviour Definition . . . . . | 301 |
|       |                      |                                                  | J.4.5.2.3 Vessel Behaviour Definition . . . . .        | 301 |
|       |                      |                                                  | J.4.5.2.4 Harbour Behaviour Definition . . . . .       | 301 |
|       |                      |                                                  | J.4.5.2.5 Freight Behaviour Definition . . . . .       | 302 |
| J.5   | Review . . . . .     |                                                  | 303                                                    |     |

---

This chapter reports on an experiment: that of modeling a domain of shipping lines<sup>1</sup>

The purposes of the experiment are (i) to further test the methodology of domain analysis & description as outlined in [58], and (ii) to add yet an, as we think it, interesting domain model to a growing series of such [59].

The report is currently in the process of being written, that is, the domain is still being studied, analysed and tentatively described. Please expect that later versions of this document may have sections that are removed, renumbered and/or rewritten wrt. the present March 12, 2024: 10:48 am version.

**The author regrets** not having had contact to real professionals of the shipping line industry. This is most obvious in our treatment of freight forwarder and shipping line behaviours. Here we used simple reasoning to come up with plausible behaviours. The behaviours that we define should convince the reader that whichever similarly reasonable, but now actual, real behaviours can be likewise defined.

**The author hopes**, even at his advanced age, today he is 83 years old, to be able, somehow, to learn from such contacts.

---

<sup>1</sup>with the Greenland **Royal Arctic Line**, <https://www.royalarcticline.com>, as a leading inspiration.

## J.1 Informal Sketches of the Shipping Domain

We shall, as a necessary element in the analysis of a domain to be rigorously analysed & described, first, and informally, delineate that domain.

This initial step of a full development is typically iterative. One outlines a first sketch. If one is not quite happy with that one either improves on that sketch or, throws it away and, produces another sketch – until “satisfied”.

### J.1.1 The Purpose of A Domain Model for Shipping

Any undertaking of modeling a specific domain has a purpose. The purpose of the shipping domain model of this report is to understand some of the properties of shipping, say such as those expected by people who have freight transported. What these properties are will evolve as the domain model evolves.

### J.1.2 A First Sketch

We structure the sketch in itemized points.

- The **name of the domain** is *Shipping*.
- The **overall context of the domain** is that
  - there is a continuous “stretch” of **navigable waterways**, an **ocean**;
    - \* on which **vessels** can sail;
  - there is a concept of **landmasses** with coasts onto the waterways;
    - \* with **harbours** at which
    - \* the **vessels** can dock
    - \* to **unload** and **load freight** and/or **passengers**.
  - There are **shipping lines** which operate these vessels;
    - \* with these shipping lines accepting requests for and actual freight and/or passengers to be transported;
  - and there are **freight forwarders** which
    - \* either act as go-between those who wishes freight or passenger transport,
    - \* or are those freight “owners”, respectively passengers,
    - \* and requests and services accepted requests, i.e., order, for transport.
- The **closer details of the domain** [of shipping further] involve that
  - harbours have **management** and **staff** (including **stevedores**) – which is ignored in the present domain model;
  - vessels have **staff** (**captains, mates, engineers** and **seamen**) – which is (are) ignored in the present domain model;
  - freight forwarders and shipping lines have **management** and **staff** – whose education, training, hiring, rostering<sup>2</sup>, laying-off and pensioning which is (are) ignored in the present domain model;
  - harbours, freight forwarders and shipping lines need financial capital in order to establish, maintain, renew and operate – which is ignored in the present domain model; and that
  - all of these have to operate in the context of local, state and international rules & regulations (i.e., laws) – which is ignored in the present domain model

We justify the omissions as they are common to many [other] domains and thus do not specifically characterise the chosen domain.

<sup>2</sup>– assignment, per day, to time-slots and places of work

### J.1.3 A Second Sketch

We assume a notion of **state**. The programmable attributes, typically, of endurants are bases for states. States [thus] have values.

- **Actions** are *intended* phenomena that potentially *changes state* values *instantaneously*.
- **Events** are un-intended phenomena which [thus *surreptitiously*] may instantaneously change a state.
- **Behaviours** are sets of sequences of actions, events and behaviours. Behaviours change states, *one change after another* and several changes potentially “at the same time”, i.e., *concurrently*, in *parallel*.

#### J.1.3.1 Strands of Interacting Sets of Behaviours

We shall focus primarily on the behaviours of two “main players”: those of **freight forwarders** and those of **shipping lines**. We shall consider the behaviours of **freight**, **vessels** and **harbours** to be subsidiary, i.e., subservient, to the main behaviours.

The two sets of behaviours each “operate, i.e., behave, on their own”, concurrently, but **inter-acting**.

**Freight forwarders**, in an interleaved fashion, on behalf of many customers, using many shipping lines, **place orders**, **accept offers** (or **refusals**), **deliver freight** and **passengers** to vessel sides (‘alongside’), **fetch freight** and **passengers** from vessel sides (‘alongside’) and **handle** all related “**paper-work**” (‘bill-of-ladings’) and **finances**.

**Shipping lines**, also in an interleaved fashion, serving many freight forwarders, co-sailing, possibly, with other shipping lines, **accept** or **reject orders**, **issue offers**, sees to it that vessels **fetch freight** (and **passengers**) from vessel sides (‘alongside’), sees to it that vessels **deliver freight** and **passengers** to vessel sides (‘alongside’), and **handle** all related “**paper-work**” (‘bill-of-ladings’) and **finances**.

#### J.1.3.2 Freight

*From Middle English freight, from Middle Dutch vracht, Middle Low German vrecht* (“cost of transport”), ultimately from Proto-Germanic \*fra- (intensive prefix) + Proto-Germanic \*aihtiz (“possession”), from Proto-Indo-European \*h<sub>2</sub>eyk (“to possess”), equivalent to for- + aught. Cognate with Old High German frēht (“earnings”), Old English æht (“owndom”), and a doublet of *fraught* [<https://en.wiktionary.org/wiki/freight>].

**J.1.3.2.1 Freight as Endurants** Freight is both a singular and a plural term. So, by freight we shall understand one or more items of discrete endurants or any amount of and liquid endurant. That is, for example, one or more 20 or 40 feet containers may be considered one freight item, and any amount of bunker oil may be considered one freight item.

**J.1.3.2.2 Freight as Behaviours** Freight are also considered behaviours: **created** as both endurants and as behaviours by the freight forwarder at the instant of a freight forwarder’s first booking inquiry, and **dismantled** by the freight forwarder at the completion of that freight’s transport.<sup>3</sup>

We shall further sketch two behaviours. The servicing of freight transports, seen from a freight forwarder and the servicing of freight transports, seen from a shipping line. Each of these behaviours if expressible as the composition of several actions. Were we here to also sketch a vessel’s freight transport, then we would have to introduce such **events** as the vessel being delayed due to unforeseen weather conditions, the vessel being ship-wrecked ()

<sup>3</sup>We shall, without loss of generality, only model that freight undergo one vessel transport.

### J.1.3.3 Freight Forwarder Behaviour

The freight forwarder behaviour basically consists of the following **freight forwarder actions**.

(i) **[FC] Freight Creation**: The freight is “created” ! We do not [have to] define the circumstances of creation.<sup>4</sup>

(ii) **[FB] Freight Being Booked**: There is the action of **booking space and time for freight between two harbours**. It is directed at a **shipping company** by a **freight forwarder**. How that freight forwarder came to book at that shipping company is left undefined. The shipping line either says *no thanks, another time, perhaps!*, or propose a sailing (i.e., a vessel and times of departure and arrival), costs, etc., i.e., a bill-of-lading. The freight forwarder accepts “refusals”, and either accepts the proposed bill-of-lading, or does not accept proposals.

(iii) **[FD] Freight Delivery**: In due time, if proposed bill-of-lading is accepted, the freight forwarder receives notification from the shipping line that the vessel has arrived at designated harbour of departure and the freight forwarder therefore delivers the freight at that harbour.

(iv) **[FT] Freight Transport**: The freight forwarder can now trace the freight transport.

(v) **[FR] Freight Return**: In due time, the freight forwarder receives notification from the shipping line that the vessel has arrived at designated harbour of arrival and the freight forwarder therefore fetches the freight at that harbour.

(vi) **[FE] Freight Dismantlement**: At some [short] time after the freight has been collected it ceases to exist as freight !

(vii) **[FM] Freight Management**: And all the time the freight forwarder manages “paperwork” and finances.

End of that story !

The above sequence of characteristic freight forwarder actions can therefore be interpreted as actions for one particular item of freight with these actions being interleaved with those for other freight items also being handled by a freight forwarder. To distinguish between different freight handlings freight forwarders naturally uses the unique freight identifier obtained in action **[FC]**.

### J.1.3.4 Shipping Line Behaviour

The shipping line behaviour basically consists of the following **shipping line actions**.

(i) **[SQ] Booking Inquiry**: The shipping line, at any time, accepts inquiries, from freight forwarders, as to freight transport. The inquiries states freight essentials, whether inflammable/explosive, whether a container or otherwise packaged (dimensions, weight, etc.), from and to harbours, desirable shipping dates, etc. The shipping line decides, upon acceptance, whether to respond immediately, or after some processing time, say minutes or hours, to the inquiry.

(ii) **[SQH] Query Handling**: The shipping line responds to inquiries either instantly or after some [other] processing time. Either the line can satisfy, i.e., **accepts**, the request and sends the inquiring freight forwarder a transport proposal, tentatively reserves space and time (i.e., vessel) for the subject freight, and then awaits its acceptance or refusal, or the line cannot satisfy, i.e., must **refuse**, the request and sends the inquiring freight forwarder a polite negative response – and “closes” that inquiry.

(iii) **[SR] Booking Reaffirmation**: The shipping line, at any time, accepts acceptance of accepted orders. It does so, for example, by reaffirming, to the freight forwarder, the now mutually accepted order, while initiating a **physical order handling** “process”, i.e., changes the order from tentative to definite.

(iv) **[SV1] Vessel Co-ordination, 1**: The shipping line, at some time thereafter, informs the vessel of its cargo for specific sailings, while assuring itself of that vessel’s availability.

(v) **[SH1] Harbour Co-ordination, 1**: The shipping line, at some time thereafter, informs designated harbours of its plans for the vessel in question to indeed arrive at, unload and load freight, and depart from that harbour, while ensuring that the harbour in question is indeed prepared for that. [We omit treatment of no or negative response from harbours.]

<sup>4</sup>Technically the freight forwarder behaviour “spawns” off a henceforth concurrently operating freight behaviour.

(vi) **[SFA]** *Freight Acceptance*: At the appointed date and time the shipping line observes, by communication from the vessel that the freight forwarder delivers the designated freight alongside the vessel.

(vii) **[SV2]** *Vessel Co-ordination, 2*: Eventually the shipping line is informed that the vessel departs freight origin harbour.

(viii) **[SV3]** *Vessel Co-ordination, 3*: Eventually the shipping line is informed that the vessel arrives at freight destination harbour.

(ix) **[SH2]** *Harbour Co-ordination, 2*: The shipping line informs that harbour of the imminent arrival of one of its vessels.

(x) **[SF]** *Freight Forwarder Notification*: The shipping line informs the freight forwarder of the arrival of “its” freight.

(xi) **[SFD]** *Freight Delivery*: And the freight forwarder informs the shipping line of its receipt of freight.

(xii) **[SFM]** *Freight Management*: All the while the shipping line keeps track of all the “paperwork” and financial matters, and other freight related matters.

End of that story!

• • •

Shipping lines handle much freight. The above sequence of twelve characteristic shipping line actions can therefore be interpreted as actions for one particular item of freight, sometimes, like **[SV1-2-3,SH1-2,SFM]**, merged with those for “similarly” transported freight. with these actions being interleaved with those for other freight items To distinguish between different freight handlings shipping lines naturally uses the unique freight identifier obtained in action **[SQ]**.

• • •

As the reader will have observed: The “workhorse” of the described domain is the shipping line – as one should indeed expect it to be!

• • •

The stories narrated above are as yet not in their final form. Language, clarification and other improvements will eventually find their way into the above text.

## J.1.4 Some Comments

### J.1.4.1 Caveat Concerning Sketches

In the informal language sketching of a domain there is, however, a serious problem. One way of illustrating the problem is as follows: Replace all domain specific nouns and verbs with  $\alpha, \beta, \gamma, \dots$ , respectively  $x, y, z, \dots$ . Now you see the problem: What does the sketch now “describe”? By using nouns and verbs of a domain that may be known to the reader, and for which the reader may have some understanding, but for which any two readers may usually have different understandings, the readers are being “lured” into a possible trap! Only a proper narrative description that is strongly linked to a formal specification – one where the  $\alpha s, \beta s, \gamma s, \dots$ , respectively  $x s, y s, z s, \dots$  are given mathematical meanings – may be satisfactory – provided, of course, that the mathematics is *consistent and relatively complete*<sup>5</sup>.

### J.1.4.2 The Insufficiency of Narrative Descriptions

Why is it not sufficient with just narrative, i.e., informal, descriptions? The answer is simple. For the shipping domain, just sketched, it might seem sufficient. But assume that you, the reader of this sketch, come from somewhere where there is no “nearby” notion of waterways, hence of

<sup>5</sup>– where ‘consistent and relative complete’ are well-defined notion of mathematical logic



vessels, etc. The fact that our sketches uses many terms from the shipping domain does not make them understandable, in-and-by-themselves. Take another example: You are from some Pacific island. There are no railways there. And you sketch a railway domain. The Pacific islander, really, have no clue as to the meaning of such terms as a railway track, a railway switch, etc. So the meaning of terms – such as presented in the above sketches – are far from clear. The danger in this is that these terms may be understood to possess properties that were not sketched.

#### J.1.4.3 What Do Formal Descriptions Contribute?

Conjoining a narrative, informal text with formal, mathematical text is meant to “fill-the-gap”: to allow the user of domain model to asset properties beyond what has been explicitly described and, based on such formalisations, reason that a postulated domain property holds, or does not hold.

#### J.1.4.4 Limitations of Domain Models

But we cannot possibly, neither informally narrate nor formally specify a “complete domain”, that is, “all” domain properties. Our domain descriptions must necessarily focus on some properties while ignoring other properties. That is, every domain model has a purpose.

#### J.1.4.5 Families of Domain Models

So, for the domain of shipping, we can thus expect a set of domain models. One like the one presented in this report. Another which focus of vessels: their loading and unloading. Yet another which focus on vessel navigation: on the ocean and into and out from harbours. Etetera.

#### J.1.4.6 There is No “Standard Model”

Just like for physics, there is not standard model. But, as for physics, there is now, with [58], a “standard” way of developing and presenting domain models. With Newton’s Classical Mechanics<sup>6</sup> described in terms of differential equations etc. there is a “standard” approach to analysing & describing mechanics (etc.). For every heretofore not described classical mechanics domain problem the physicists and engineers now know how to tack the analysis & description of that domain. Similarly for every human-assisted discrete dynamics and primarily artifact “populated” domain the computer scientists and software engineers now know how to tack the analysis & description of that domain.

## J.2 Endurants: External Qualities

### J.2.1 Freight

Although the notion of ‘freight’ is, indeed, a core concept of this report, it will not “*play center stage*”.

### J.2.2 Endurant Sorts & Observers

1058. We shall consider an **aggregate** of **shipping** in the context of

---

<sup>6</sup>Other contributors to the formal description of Classical Mechanics were Gottfried Wilhelm Leibniz, Joseph-Louis Lagrange, Leonard Euler, etc.

- (a) an **aggregate** of *navigable waterways*, i.e., an ocean, rivers and canals<sup>7</sup> with identification of harbours;
- (b) an **aggregate** of **land masses**, i.e., continents and islands, small and large;
- (c) an **aggregate** of thus identified **harbours**;
- (d) an **aggregate** of **vessels** that can carry freight and/or passengers;
- (e) an **aggregate** of **shipping lines** – which commands (owns or operate) these vessels;
- (f) an **aggregate** of **freight forwarders**<sup>8</sup>;
- (g) an **aggregate** of **freight**; and
- (h) an **aggregate** of **passengers**.

1059. The aggregate of harbours is here seen as a set of harbours –

1060. with harbours to be further defined.

1061. The aggregate of vessels is here seen as a set of vessels –

1062. with vessels to be further defined.

1063. The aggregate of shipping lines is here seen as a set of shipping lines –

1064. with shipping lines to be further defined.

1065. The aggregate of freight forwarders is here seen as a set of freight forwarders –

1066. with freight forwarders to be further defined.

1067. The aggregate of freight is here Sean’s as a set of freight –

1068. with freight to be further defined.

1069. The aggregate of passengers is here seen as a set of passengers –

1070. with passenger to be further defined.

The waterways and land masses are here further undefined. Harbours, vessels, shipping lines, freight forwarders, freight and passengers will be further defined below.

|                         |  |                           |
|-------------------------|--|---------------------------|
| <b>type</b>             |  | 1061. Vs = <b>V-set</b>   |
| 1058. S                 |  | 1062. V                   |
| 1058a. WV               |  | 1063. SLs = <b>SC-set</b> |
| 1058b. LM               |  | 1064. SL                  |
| 1058c. AH               |  | 1065. FFs = <b>FF-set</b> |
| 1058d. AV               |  | 1066. FF                  |
| 1058e. ASL              |  | 1067. Fs = <b>F-set</b>   |
| 1058f. AFF              |  | 1068. F                   |
| 1058g. AF               |  | 1069. Ps = <b>P-set</b>   |
| 1058h. AP               |  | 1070. P                   |
| 1059. Hs = <b>H-set</b> |  | <b>value</b>              |
| 1060. H                 |  | 1058a. obs_WV: S → WV     |

<sup>7</sup>There are but two oceans. [We do not exclude the *Caspian Sea*. Our model covers both that and “the other” ocean, as the only two!] The “other”, the larger ocean is, for pragmatic reasons “divided” up into separately named “oceans”: the *Atlantics*, North and South, the *Pacific*, the *Indian*, the *Arabian Sea*, the *Barents Sea*, the *Arctic Sea*, the *Anarctic Sea* (*Southern Ocean*, *Austral Ocean*), etc. Basically two canals provide short-cuts between two otherwise disperse areas of that one ocean: the *Suez* and the *Panama*.

<sup>8</sup>The borderline between freight forwarders and shipping lines is fuzzy. Some shipping lines offer freight forwarding: the logistics of moving freight between end-customer and vessel, etc.

|        |                  |       |                               |
|--------|------------------|-------|-------------------------------|
| 1058b. | obs_LM: S → LM   | 1059. | obs_Hs: AH → Hs               |
| 1058c. | obs_AH: S → AH   | 1061. | obs_Vs: AV → Vs               |
| 1058d. | obs_AV: S → AV   | 1063. | obs_SLs: ASL → SL- <b>set</b> |
| 1058e. | obs_AS�: S → ASC | 1065. | obs_FFs: AFF → FF- <b>set</b> |
| 1058f. | obs_AFF: S → AFF | 1067. | obs_Fs: AF → F- <b>set</b>    |
| 1058g. | obs_AF: S → AF   | 1069. | obs_Ps: AP → P- <b>set</b>    |
| 1058h. | obs_AP: SS → AP  |       |                               |

The waterways with its harbours define an in[de]finite set of [circular] routes that can be sailed by the vessels. There are vessels other than those owned or commanded by the company. The company is also characterised by a definite set of routes sailed/serviced by its vessels. All this will be clear as we proceed.

### J.2.3 Endurant Values

From an **aggregate** of **shipping** one can extract all its subsidiary endurants – starting with that aggregate:

|       |                                                         |       |                                           |
|-------|---------------------------------------------------------|-------|-------------------------------------------|
| 1071. | the <b>aggregate</b> of <b>shipping</b> ,               | 1074. | <b>set</b> of <b>shipping lines</b> ,     |
|       | (a) its <b>aggregate</b> of <b>waterways</b> ,          | 1075. | <b>set</b> of <b>freight forwarders</b> , |
|       | (b) its <b>aggregate</b> of <b>land masses</b> ,        | 1076. | <b>set</b> of <b>freight</b> ,            |
|       | (c) its <b>aggregate</b> of <b>harbours</b> ,           | 1077. | <b>set</b> of <b>passengers</b> ,         |
|       | (d) its <b>aggregate</b> of <b>vessels</b> ,            | 1078. | <b>harbours</b> ,                         |
|       | (e) its <b>aggregate</b> of <b>shipping lines</b> ,     | 1079. | <b>vessels</b> ,                          |
|       | (f) its <b>aggregate</b> of <b>freight forwarders</b> , | 1080. | <b>shipping lines</b> ,                   |
|       | (g) its <b>aggregate</b> of <b>freight</b> and          | 1081. | <b>freight forwarders</b> ,               |
|       | (h) its <b>aggregate</b> of <b>passengers</b> ;         | 1082. | <b>freight</b> and                        |
|       | and their                                               | 1083. | <b>passengers</b> .                       |
| 1072. | <b>set</b> of <b>harbours</b> ,                         |       |                                           |
| 1073. | <b>set</b> of <b>vessels</b> ,                          |       |                                           |

#### value

|        |                                                                               |
|--------|-------------------------------------------------------------------------------|
| 1071.  | $s_e:SL$                                                                      |
| 1071a. | $wv_e:WV = \text{obs\_WV}(s_e)$                                               |
| 1071b. | $lm_e:LM = \text{obs\_LM}(s_e)$                                               |
| 1071c. | $ah_e:AH = \text{obs\_AH}(s_e)$                                               |
| 1071d. | $av_e:AV = \text{obs\_AV}(s_e)$                                               |
| 1071e. | $asl_e:ASL = \text{obs\_ASL}(s_e)$                                            |
| 1071f. | $aff_e:AFF = \text{obs\_AFF}(s_e)$                                            |
| 1071g. | $af_e:AF = \text{obs\_AF}(s_e)$                                               |
| 1071h. | $ap_e:AP = \text{obs\_AP}(s_e)$                                               |
| 1072.  | $hs_e:Hs = \text{obs\_Hs}(ah_e)$                                              |
| 1073.  | $vs_e:Vs = \text{obs\_Vs}(av_e)$                                              |
| 1074.  | $sls_e:SLs = \text{obs\_SLs}(asl_e)$                                          |
| 1075.  | $ffs_e:FFs = \text{obs\_FFs}(aff_e)$                                          |
| 1076.  | $fs_e:Ffs = \text{obs\_Fs}(af_e)$                                             |
| 1077.  | $ps_e:Ps = \text{obs\_Ps}(ap_e)$                                              |
| 1078.  | $h_e s: H\_UI\text{-set} = \{h h:H \bullet h \in \text{obs\_Hs}(ah_e)\}$      |
| 1079.  | $v_e s: V\_UI\text{-set} = \{v v:V \bullet v \in \text{obs\_Vs}(av_e)\}$      |
| 1080.  | $sl_e s: SC\_UI\text{-set} = \{s s:SL \bullet s \in \text{obs\_SLs}(sls_e)\}$ |

1081.  $ff_{es}:FF\_UI\text{-set} = \{ff|ff:FF \bullet ff \in \text{obs\_FF}(fcs_e)\}$   
 1082.  $f_{es}:F\_UI\text{-set} = \{f|f:F \bullet f \in \text{obs\_Fs}(fs_e)\}$   
 1083.  $p_{es}:P\_UI\text{-set} = \{p|p:P \bullet p \in \text{obs\_Ps}(ps_e)\}$

1084. We can define the set of all endurants.

**value**

1084.  $\text{all\_ends} = \{s_e\} \cup \{wv_e\} \cup \{lm_e\} \cup \{ah_e\} \cup \{av_e\} \cup \{asl_e\} \cup \{aff_e\} \cup \{ap_e\}$   
 1084.  $\cup hs_e \cup vs_e \cup sls_e \cup fs_e \cup ffe_s \cup ps_e \cup h_e s \cup v_e s \cup sl_e s \cup ff_e s \cup p_e s$

## J.3 Endurants: Internal Qualities

### J.3.1 Unique Identifiers

#### J.3.1.1 Unique Identifier Types and Observers

We can associate unique identifiers with:

- |                                                         |                                                     |
|---------------------------------------------------------|-----------------------------------------------------|
| 1085. The <b>aggregate</b> of <b>shipping</b> ;         | 1096. the <b>set</b> of <b>vessels</b> ;            |
| 1086. the <b>aggregate</b> of <b>waterways</b> ;        | 1097. each <b>individual vessel</b> ;               |
| 1087. the <b>aggregate</b> of <b>land masses</b> ;      | 1098. the <b>set</b> of <b>shipping lines</b> ;     |
| 1088. the <b>aggregate</b> of <b>harbours</b> ;         | 1099. each <b>individual shipping line</b> ;        |
| 1089. the <b>aggregate</b> of <b>vessels</b> ;          | 1100. the <b>set</b> of <b>freight forwarders</b> ; |
| 1090. the <b>aggregate</b> of <b>shipping lines</b>     | 1101. each <b>individual freight forwarder</b> ;    |
| 1091. the <b>aggregate</b> of <b>freight forwarders</b> | 1102. the <b>set</b> of <b>freight</b> ;            |
| 1092. the <b>aggregate</b> of <b>freight</b>            | 1103. each <b>individual freight</b> ;              |
| 1093. the <b>aggregate</b> of <b>passengers</b>         | 1104. the <b>set</b> of <b>passengers</b> ;         |
| 1094. the <b>set</b> of <b>harbours</b> ;               | 1105. each <b>individual passenger</b> ;            |
| 1095. each <b>individual harbour</b> ;                  |                                                     |
- 
- |              |                             |
|--------------|-----------------------------|
| <b>type</b>  | 1100. FFs_UI                |
| 1085. S_UI   | 1101. FF_UI                 |
| 1086. WV_UI  | 1102. Fs_UI                 |
| 1087. LM_UI  | 1103. F_UI                  |
| 1088. AH_UI  | 1104. Ps_UI                 |
| 1089. AV_UI  | 1105. P_UI                  |
| 1090. ASC_UI | <b>value</b>                |
| 1091. AFF_UI | 1085. uid_S: S → S_UI       |
| 1092. AF_UI  | 1086. uid_WV: WV → WV_UI    |
| 1093. AP_UI  | 1087. uid_LM: LM → LM_UI    |
| 1094. Hs_UI  | 1088. uid_AH: AH → AH_UI    |
| 1095. H_UI   | 1089. uid_AV: AV → AV_UI    |
| 1096. Vs_UI  | 1090. uid_AS: AS → AS_UI    |
| 1097. V_UI   | 1091. uid_AFF: AFF → AFF_UI |
| 1098. SCs_UI | 1092. uid_AF: AF → AF_UI    |
| 1099. SC_UI  | 1093. uid_AP: AP → AP_UI    |

- |                                            |                                            |
|--------------------------------------------|--------------------------------------------|
| 1094. $uid_{Hs}: Hs \rightarrow Hs\_UI$    | 1100. $uid_{FFs}: FFs \rightarrow FFs\_UI$ |
| 1095. $uid_H: H \rightarrow H\_UI$         | 1101. $uid_{FF}: FF \rightarrow FC\_UI$    |
| 1096. $uid_{Vs}: Vs \rightarrow Vs\_UI$    | 1102. $uid_{Fs}: Fs \rightarrow Fs\_UI$    |
| 1097. $uid_V: V \rightarrow V\_UI$         | 1103. $uid_F: F \rightarrow F\_UI$         |
| 1098. $uid_{SLs}: SLs \rightarrow SLs\_UI$ | 1104. $uid_{Ps}: Ps \rightarrow Ps\_UI$    |
| 1099. $uid_{SL}: SL \rightarrow SL\_UI$    | 1105. $uid_P: P \rightarrow P\_UI$         |

### J.3.1.2 Domain Unique Identifiers

From an **aggregate** of **shipping lines** one can extract all the unique identifiers of its subsidiary endurants – starting with that aggregate:

- |                                                         |                                                 |
|---------------------------------------------------------|-------------------------------------------------|
| 1106. the <b>aggregate</b> of <b>shipping</b> ,         | 1109. <b>set</b> of <b>shipping lines</b> ,     |
| (a) its <b>aggregate</b> of <b>waterways</b> ,          | 1110. <b>set</b> of <b>freight forwarders</b> , |
| (b) its <b>aggregate</b> of <b>land masses</b> ,        | 1111. <b>set</b> of <b>freight</b> ,            |
| (c) its <b>aggregate</b> of <b>harbours</b> ,           | 1112. <b>set</b> of <b>passengers</b> ,         |
| (d) its <b>aggregate</b> of <b>vessels</b> ,            | 1113. <b>harbours</b> ,                         |
| (e) its <b>aggregate</b> of <b>shipping lines</b> ,     | 1114. <b>vessels</b> ,                          |
| (f) its <b>aggregate</b> of <b>freight forwarders</b> , | 1115. <b>shipping lines</b> ,                   |
| (g) its <b>aggregate</b> of <b>freight</b> and          | 1116. <b>freight forwarders</b> ,               |
| (h) its <b>aggregate</b> of <b>passengers</b> ;         | 1117. <b>freight</b> and                        |
| and their                                               | 1118. <b>passengers</b> .                       |
| 1107. <b>set</b> of <b>harbours</b> ,                   |                                                 |
| 1108. <b>set</b> of <b>vessels</b> ,                    |                                                 |

#### value

1106.  $s_{ui}: S\_UI = uid_S(s_e)$   
1106a.  $wv_{ui}: WV\_UI = uid_{WV}(obs_{WV}(s_e))$   
1106b.  $lm_{ui}: LM\_UI = uid_{LM}(obs_{LM}(s_e))$   
1106c.  $ah_{ui}: AH\_UI = uid_{AH}(obs_{AH}(s_e))$   
1106d.  $av_{ui}: AV\_UI = uid_{AV}(obs_{AV}(s_e))$   
1106e.  $asl_{ui}: ASL\_UI = uid_{ASC}(obs_{ASL}(s_e))$   
1106f.  $aff_{ui}: AFF\_UI = uid_{AFF}(obs_{AFF}(s_e))$   
1106g.  $af_{ui}: AF\_UI = uid_{AF}(obs_{AF}(s_e))$   
1106h.  $ap_{ui}: AP\_UI = uid_{AP}(obs_{AP}(s_e))$   
1107.  $hs_{ui}: Hs\_UI = uid_{Hs}(obs_{Hs}(ah_e))$   
1108.  $vs_{ui}: Vs\_UI = uid_{Vs}(obs_{Vs}(av_e))$   
1109.  $sls_{ui}: SLs\_UI = uid_{SLs}(obs_{SLs}(asl_e))$   
1110.  $ffs_{ui}: FFs\_UI = uid_{FFs}(obs_{FFs}(afc_e))$   
1111.  $fs_{ui}: Fs\_UI = uid_{Fs}(obs_{Fs}(afe_e))$   
1112.  $ps_{ui}: Ps\_UI = uid_{Ps}(obs_{Ps}(ape_e))$   
1113.  $h_{ui}s: H\_UI\text{-set} = \{uid_H(h)|h:H \cdot h \in obs_{Hs}(ah_e)\}$   
1114.  $v_{ui}s: V\_UI\text{-set} = \{uid_V(v)|v:V \cdot v \in obs_{Vs}(av_e)\}$   
1115.  $s_{ui}s: SL\_UI\text{-set} = \{uid_{SL}(sl)|sl:SL \cdot sl \in obs_{SLs}(sls_e)\}$   
1116.  $ff_{ui}s: FF\_UI\text{-set} = \{uid_{FF}(ff)|ff:FF \cdot ff \in obs_{FF}(ffs_e)\}$   
1117.  $f_{ui}s: F\_UI\text{-set} = \{uid_F(f)|f:F \cdot f \in obs_{Fs}(fs_e)\}$   
1118.  $p_{ui}s: P\_UI\text{-set} = \{uid_P(p)|p:P \cdot p \in obs_{Ps}(ps_e)\}$

1119. We can define the set of all endurant identifiers.

**value**

1119.  $\text{all\_uids} = \{s_{ui}\} \cup \{wv_{ui}\} \cup \{lm_{ui}\} \cup \{ah_{ui}\} \cup \{av_{ui}\} \cup \{asc_{ui}\} \cup \{aff_{ui}\} \cup \{ap_{ui}\}$   
 1119.  $\cup \{hs_{ui}\} \cup \{vs_{ui}\} \cup \{scs_{ui}\} \cup \{ffs_{ui}\} \cup \{fss_{ui}\} \cup \{ps_{ui}\} \cup \{hs_s\} \cup \{v_{ui}\} \cup \{sc_{ui}\} \cup \{ff_{ui}\} \cup \{f_{ui}\} \cup \{p_{ui}\}$

**J.3.1.3 An Axiom**

1120. Endurants are uniquely identified.

**axiom**

1120.  $\square \text{card all\_ends} = \text{card all\_uids}$

The **always** operator,  $\square$ , expresses that  $\text{card all\_ends} = \text{card all\_uids}$  holds at any time.

**J.3.1.4 Retrieve Endurant Values**

1121. Given a unique identifier,  $ui$ , in  $\text{all\_uids}$  and given the set of all endurants  $\text{all\_ends}$  we can retrieve the endurant,  $e$  of identifier  $ui$ .

**value**

1121.  $\text{get\_E}: UI \rightarrow E$   
 1121.  $\text{get\_E}(ui) \equiv \text{let } e:E \cdot e \in \text{all\_ends} \Rightarrow \text{uid\_E}(e)=ui \text{ in } e \text{ end}$

**J.3.2 Mereologies****J.3.2.1 A Shift in Modeling**

Till now we have modeled the shipping line domain considering all its endurants to be non-structures (cf. [58, Sects. 4.8 and 4.10]). From now on we shall consider all aggregates and sets of endurants as structures. This means that we can dismiss our modeling of the unique identifiers for all aggregates and set of endurants void and nil. Thus we shall only model the mereology of what we basically treat as atomic endurants: *freight forwarders*, *shipping lines*, *vessels*, *harbours*, *freight* and *passengers*.

**J.3.2.2 Mereology Types and Observers**

The **mereology** that we shall promote emphasises both **topological** and **conceptual** properties of shipping line systems. They express topological properties when mandating unique identifiers of spatially close/related endurants, And they express conceptual properties when mandating unique identifiers of endurants with which shipping lines “do business”! Further topological and conceptual properties of shipping line systems will be expressed in Sect.J.3.3 where we treat **attributes** of shipping line systems.

**J.3.2.2.1 Harbour Mereology**

1122. Harbour mereologies are

- the non-empty set of unique identifiers of vessels that may use the harbour,
- the pair of two possibly empty sets of unique identifiers of freight: one identifying freight to be loaded (**todo**), the other having been unloaded (**done**),
- the unique identifier of the waterways and the
- the unique identifier of the landmass.

**type**1122.  $H\_Mer = V\_UI\text{-set} \times (\text{todo:F\_UI-set} \times \text{done:F\_UI-set}) \times WV\_UI \times LM\_UI$ **value**1122.  $\text{mereo\_H}: H \rightarrow H\_Mer$ 

1123. The well-formedness of a harbour mereology entails

- that its set of vessel identifiers is non-empty and included in the set of all vessel identifiers,
- the “to do” and the “done” freight does not “overlap” and are a subset of all freight.
- that its waterways identifier is that of the known waterway[s], and
- that its landmass identifier is that of the known landmass.

**value**1123.  $\text{wf\_H\_Mer}: H\_Mer \rightarrow \text{Bool}$ 1123.  $\text{wf\_H\_Mer}(\text{vuis}, (\text{todo}, \text{done}), \text{wvui}, \text{lmui}) \equiv$ 1123.  $\{\} \neq \text{vuis} \subseteq v_{uis}$ 1123.  $\wedge \text{todo} \cap \text{done} = \{\} \wedge \text{todo} \cup \text{done} \subseteq f_{uis}$ 1123.  $\wedge \text{wvui} = wv_{ui}$ 1123.  $\wedge \text{lmui} = lm_{ui}$ **J.3.2.2.2 Vessel Mereology**

1124. Vessel mereologies are

- the non-empty set of unique identifiers of harbours that it may use,
- the non-empty set of unique identifiers of shipping lines for which it sails, i.e., which share an agreement to operate that vessel, and
- the unique identifier of the waterways.

**type**1124.  $V\_Mer = H\_UI\text{-set} \times SL\_UI\text{-set} \times WV\_UI$ **value**1124.  $\text{mereo\_V}: V \rightarrow V\_Mer$ 

1125. The well-formedness of a vessel mereology entails

- that its set of harbour identifiers is non-empty and included in the set of all harbour identifiers,
- that its set of shipping line identifiers is non-empty and included in the set of all shipping line identifiers,
- and that its waterways identifier is that of the known waterways.

1125.  $\text{wf\_V\_Mer}: V\_Mer \rightarrow \text{Bool}$ 1125.  $\text{wf\_V\_Mer}(\text{huis}, \text{scuis}, \text{wvui}) \equiv$ 1125.  $\{\} \neq \text{huis} \subseteq h_{uis}$ 1125.  $\wedge \{\} \neq \text{scuis} = sc_{uis}$ 1125.  $\wedge \text{wvui} = wv_{ui}$

### J.3.2.2.3 Shipping Line Mereology

1126. Shipping line mereologies are

- the non-empty set of unique identifiers of vessels that it operates,
- the non-empty set of unique identifiers of freight forwarders which it services and
- the non-empty set of identifiers of harbours that it uses,

**type**

1126.  $SL\_Mer = V\_UI\_set \times FF\_UI\_set \times H\_UI\_set$

**value**

1126.  $mereo\_SL: SL \rightarrow SL\_Mer$

1127. The well-formedness of a shipping line mereology entails

- that its set of vessel identifiers is non-empty and included in the set of all vessel identifiers,
- that its set of freight forwarder identifiers is non-empty and included in the set of all freight forwarder identifiers, and that its set of harbour identifiers is non-empty and included in the set of all harbour identifiers.

**value**

1127.  $wf\_SC\_Mer: SC\_Mer \rightarrow \mathbf{Bool}$

1127.  $wf\_SC\_Mer(vuis,fcuis,huis) \equiv$

1127.  $\{\} \neq vuis \subseteq v_{uis}$

1127.  $\wedge \{\} \neq ffuis \subseteq f_{ffuis}$

1127.  $\wedge \{\} \neq huis \subseteq h_{huis}$

Two or more shipping lines may **co-sail** one or more vessels<sup>9</sup>.

### J.3.2.2.4 Freight Forwarder Mereology

1128. Freight forwarder mereologies are

- the non-empty set of unique identifiers of shipping lines that it uses,
- the non-empty set of unique identifiers of harbours to which it delivers and from which it fetches freight, and the possibly empty set of unique identifiers of freight with which it is involved.

**type**

1128.  $FF\_Mer = SL\_UI\_set \times H\_UI\_set \times F\_UI\_set$

**value**

1128.  $mereo\_FF: FF \rightarrow FF\_Mer$

1129. The well-formedness of a freight forwarder mereology entails

- the non-empty set of unique identifiers of known shipping lines that it uses and
- the non-empty set of unique identifiers of known harbours

**value**

1129.  $wf\_FF\_Mer: FF\_Mer \rightarrow \mathbf{Bool}$

1129.  $wf\_FF\_Mer(sluis,huis,_) \equiv$

1129.  $\{\} \neq sluis \subseteq sl_{uis}$

1129.  $\wedge \{\} \neq huis \subseteq h_{huis}$

<sup>9</sup>We shall not model the specifics, i.e., details of co-sailing.



**J.3.2.2.5 Freight Mereology**

1130. Freight mereologies are

- the unique identifier of the freight forwarder,
- the unique identifier of the shipping line which is intended to ship, or which ships that freight, and
- the pair of unique identifiers of the two harbour involved in the freight transport.

**type**

1130.  $F\_Mer = FF\_UI \times SC\_UI \times (H\_UI \times H\_UI)$

**value**

1130.  $mereo\_F: F \rightarrow F\_Mer$

1131. The well-formedness of a freight mereology entails

- that the freight forwarder identifier is known,
- that the shipping line identifier is known and
- that the two known harbours are different.

1131.  $is\_wf\_F\_Mer: F\_Mer \rightarrow Bool$

1131.  $is\_wf\_F\_Mer(ffui,scui,(fhui,thui)) \equiv$

1131.  $ffui \in f\_uis$

1131.  $\wedge scui \in sc\_uis$

1131.  $\wedge fhui \neq thui \wedge \{fhui,thui\} \subseteq h\_uis$

**J.3.2.2.6 Passenger Mereology**

1132. Passenger mereologies are

- the identifier of the vessels with which they have traveled, are traveling or intend to travel, and
- the unique identifier of the shipping lines with whom they have travel-led, are traveling or intend to travel.

**type**

1132.  $P\_Mer = V\_UI\_set \times SC\_UI\_set$

**value**

1132.  $mereo\_P: P \rightarrow P\_Mer$

1133. The well-formedness of a passenger mereology entails

- that the set of vessel identifiers is known,
- that the set of shipping line identifiers is known, and
- that the shipping lines are indeed operating the identified vessels.

**value**

1133.  $wf\_P\_Mer: P\_Mer \rightarrow Bool$

1133.  $wf\_P\_Mer(vuis,scuis) \equiv$

1133.  $vuis \subseteq v\_uis \wedge scuis \subseteq sc\_uis \wedge$

1133.  $\forall v\_ui:V\_UI \bullet v\_ui \in vuis, \exists sc\_ui:SC\_UI \bullet sc\_ui \in scuis \Rightarrow$

1133. **let**  $sc = get\_part(sc\_ui)$  **in** **let**  $(vuis',\_)$   $= mereo\_SC(sc)$  **in**  $v\_ui \in vuis'$  **end end**

**J.3.2.2.7 Waterways Mereology**

1134.

1135.

1136.

1137.

**type**

1134.

1135.

1136.

1137.

**value**

1134.

1135.

1136.

1137.

**J.3.2.2.8 Landmass Mereology**

1138.

1139.

1140.

1141.

**type**

1138.

1139.

1140.

1141.

**value**

1138.

1139.

1140.

1141.

**J.3.3 Attributes****J.3.3.1 Attribute Types and Observers**

We shall illustrate but a very few attributes. Those we choose to illustrate appear to be the ones most relevant for the specific examples of *freight forwarder*, *shipping line*, *vessel*, *harbour* and *freight behaviours*.

**J.3.3.1.1 Freight Forwarder Attributes**

1142. For any one specific freight, the freight forwarder, undergoes a sequence of states. These are sketched in Sect. J.1.3.3 on page 281.  $FFH\Sigma$  models the set of state names for these.
1143. Freight forwarder history is a freight identifier indexed, reverse-ordered chronological sequence of freight state labeled freight information.
1144. We leave  $FFInfo$  further undefined,

**type**

1142.  $FFH\Sigma = "FC" \mid "FBB" \mid "FB" \mid "FD" \mid "FT" \mid "FR" \mid "FE" \mid "FM"$

1143.  $FFHist = F\_UI \xrightarrow{\overline{m}} (TIME \times FFH\Sigma \times FFInfo)^*$

1143.  $FFInfo = \dots$

**value**

1143.  $attr\_FFHist: FF \rightarrow FFHist$

**J.3.3.1.2 Shipping Line Attributes**

1145.

1146.

1147.

1148.

**type**

1145.

1146.

1147.

1148.

**value**

1145.

1146.

1147.

1148.

**J.3.3.1.3 Vessel Attributes**

1149.

1150.

1151.

1152.

**type**

1149.

1150.

1151.

1152.

**value**

1149.

1150.

1151.

1152.

**J.3.3.1.4 Harbour Attributes**

1153.

1154.

1155.

1156.

**type**

1153.

1154.

1155.

1156.

**value**

1153.

1154.

1155.

1156.

**J.3.3.1.5 Freight Attributes**

1157.

1158.

1159.

1160.

**type**

1157.

1158.

1159.

1160.

**value**

1157.

1158.

1159.

1160.

**J.3.3.2 Attribute Wellformedness**

TO BE WRITTEN

**J.4 Perdurants**

By the **transcendental deductions** introduced in [58, *Chapter 6*] we now interpret some enduring parts as behaviours. A behaviour is a set of sequences of actions, events and behaviours. Behaviours interact, here expressed in the style of CSP [109–111, *C.A.R. Hoare*] as embedded in RSL [100].

### J.4.1 Freight as Endurants and as Behaviours

The central entity of the shipping line domain is that of **freight**. Freight have, so far, been considered as atomic endurants. We shall now transcendently deduce freight into behaviours. There is a dynamically varying number of uniquely identified freight. We suggest to model freight as follows: Freight is created by the freight forwarder. At the moment of such creation the freight “receives” its, i.e., a unique identifier, one that has not been used before, and one that will never be used, in the creation of other freight, again. Once a freight has completed a full transport as directed by the freight forwarder and carried out by a shipping line and one of its vessels, that freight ceases to be a freight, that is, as an endurants and as a behaviour. Its unique identifier will never be the identifier of other freight.

### J.4.2 Actions, Events and Behaviours

TO BE WRITTEN

### J.4.3 Global Freight Variable

Freight occurs, appears, and freight disappears. In this model we assume a fixed number of freight forwarders, shipping lines, vessels and harbours<sup>10</sup>. But we must model a varying number of freight. We shall, for simplicity, and without loss of generality, assume that freight becomes so when in the care of freight forwarders, and that freight ceases to be freight, i.e., to exist, one it has been transported.

Although we shall model freight as behaviours we shall introduce, as a technicality,

1161. a global variable `freight_uids` which is initialised to an empty set of unique freight identifiers.

At any time it contains the set of all unique identifiers of freight which have been created as freight, When freight ceases to exist that freight’s unique identifier is not deleted from `freight_uids`.

#### variable

1161. `freight_uids:F_UI-set := {}`

#### value

1162. `get_F_UI: Unit → F_UI`

1162. `get_F_UI() ≡`

1162. `let f_ui:F_UI • f_ui ∉ freight_uids in`

1162. `freight_uids := freight_uids ∪ {f_ui};`

1162. `f_ui end`

1162. `get_F_UI` is a value-returning action.

- It applies to the global state and returns a “new, hitherto unused” unique freight identifier
- while updating the global state variable `freight_uids` with that identifier.

### J.4.4 Channels

In order for CSP-modeled behaviours to **interact**, they must **communicate**, and they do so over the medium of, as here, **channels**.

We shall name the full ensemble of channels over which any of the *shipping company*, *freight forwarder*, *harbour* and *harbour* behaviours communicate

- **channel** `ch[{uii, uij}]`: MSG

where indices *ui<sub>i</sub>* and *ui<sub>j</sub>* are unique identifiers of these behaviours – cum endurant parts, and where MSG is the **type** of the communicated value.

<sup>10</sup>We also assume fixed waterways and land masses.

## J.4.5 Behaviours

### J.4.5.1 Behaviour Signatures

#### J.4.5.1.1 Freight Forwarder Signature

1163. We introduce the notion of “the making of a freight behaviour skeleton” `NewF`:

- either there is not such skeleton, `"nil"`,
- or there are the elements that make up a freight endurant: a unique freight identifier, a freight mereology and the static attributes of a freight. What they are is really of no consequence. The programmable attribute only becomes relevant as soon as the freight endurant, and hence the freight behaviour is created.

1164.

**type**

1163. `NewF = "nil" | F_UI × F_Mer × F_Stat`

**value**

1164. `ff: ffui:FF_UI × (sluis,vuis,fuis):FF_Mer × ffstat:FF_Stat → ffprgr:FF_Prgr`

1164. `→ { ch[ {ffui,ui} ] | ui:SL_UI|F_UI•slui∈sluis∪vuis∪fuis } Unit`

#### J.4.5.1.2 Shipping Line Signature

1165.

**value**

1165. `sl: slui:SL_UI × (vuis,ffuis,huis):SL_Mer × slstat:SL_Stat → slprgr:SL_Prgr`

1165. `→ { ch[ {slui,ui} ] | ui:FF_UI|V_UI|H_UI•ui∈ffuis∪huis } Unit`

#### J.4.5.1.3 Vessel Signature

#### J.4.5.1.4 Harbour Signature

#### J.4.5.1.5 Freight Signature

### J.4.5.2 Behaviour Definitions

#### J.4.5.2.1 Freight Forwarder Definition

1163. We have introduced, cf. Item 1163, the notion of “the making of a freight behaviour skeleton” `NewF`. To repeat:

- either there is not such skeleton, `"nil"`,
- or there are the elements that make up a freight endurant: a unique freight identifier, a freight mereology and the static attributes of a freight, What they are is really of no consequence. The programmable attribute only becomes relevant as soon as the freight endurant, and hence the freight behaviour is created.

1166. The *freight forwarder* behaviour may

1167. [FC] non-deterministically internally, [], choose to [somehow] accept an item of freight, ..., as expressed in the `ffc` behaviour, and, likewise non-deterministically internally, decide to “convert” the skeleton into a behaviour.

1167a.–1167d. Non-deterministically internally the freight forwarder behaviour chooses among the former alternative behaviour, ffc, or the following specific freight related alternatives.

- (a) [FB] The freight forwarder communicates a booking order to a shipping line. The shipping line either accepts this booking with a proposed bill-of-lading, or declines it. The freight forwarder must accept declined bookings and must either accept or decline a proposed bill-of-lading.

*We assume that the time elapsed between the freight forwarder communicating its booking and the shipping line responding to this booking is such that the booking and its response can be modeled as a single behaviour composed from two CSP output/input actions.*

[ffb stands for ‘freight forwarder booking’.]

- (b) [FD] The freight forwarder is informed by the shipping line that the designated vessel is ready to accept the freight for transport.

*We assume that the time elapsed between the freight forwarder receiving this alert and the freight forwarder being able to respond is such that the alert and its response can realistically be modeled as a single behaviour composed from two CSP output/input actions. See next.*

[ffd stands for ‘freight forwarder delivery alert (from shipping line)’.]

- (c) [FR] The freight forwarder is informed by the shipping line that the designated vessel is ready to return the freight it has transported.

*We assume that the time elapsed between the freight forwarder receiving this alert and the freight forwarder being able to respond is such that the alert and its response must most realistically be modeled as two behaviours. See next.*

[ffr stands for ‘freight forwarder freight return (message, from shipping line)’.]

- (d) [FE] The freight forwarder collects the freight and its saga as ‘freight’ is over.

[ffe stands for ‘freight forwarder freight ending’.]

1168. [FM] In-between, before and after these specific freight related actions, the freight forwarder “performs” management actions “of its own” !

[ff stands for ‘freight forwarder management’.]

[ stands for ]

**type**

1163.  $\text{NewF} = \text{"nil"} \mid \text{F\_UI} \times \text{F\_Mer} \times \text{F\_Stat}$

**value**

1164.  $\text{ff} : \text{fui} : \text{FF\_UI} \times (\text{sluis}, \text{fuis}) : \text{FF\_Mer} \times \text{ffstat} : \text{FF\_Stat} \rightarrow \text{ffhist} : \text{FF\_Hist}$

1164.  $\rightarrow \{ \text{ch}[\{ \text{slui}, \text{fui} \}] \mid \text{slui} : \text{SL\_UI} \bullet \text{slui} \in \text{sluis} \}$  **Unit**

1166.  $\text{ff}(\text{ffui}, (\text{sluis}, \text{fuis}), \text{ffstat})(\text{ffhist}) \equiv$

1167. [FC]  $\text{ffc}(\text{ffui}, (\text{sluis}, \text{fuis}), \text{ffstat})(\text{ffhist})$

1167a. [FB]  $\square (\square \text{ffb}(\text{ffui}, (\text{sluis}, \text{fuis}), \text{ffstat})(\text{ffhist}))$

1167b. [FD]  $\square \text{ffd}(\text{ffui}, (\text{sluis}, \text{fuis}), \text{ffstat})(\text{ffhist})$

1167c. [FR]  $\square \text{ffr}(\text{ffui}, (\text{sluis}, \text{fuis}), \text{ffstat})(\text{ffhist})$

1167d. [FE]  $\square \text{ffe}(\text{ffui}, (\text{sluis}, \text{fuis}), \text{ffstat})(\text{ffhist})$

1168. [FM]  $\square \text{ffm}(\text{ffui}, (\text{sluis}, \text{fuis}), \text{ffstat})(\text{ffhist})$

**Freight Creation:**

1169. Freight forwarders

1170. non-deterministically internally, somehow, accept freight. Technically this is modeled by the freight forwarder obtaining a hitherto unused unique identifier,

1171. and, from own attribute values and from the freight “customer”, ”...”, creating a freight endurant,  $\text{mkF}(\text{fui}, \text{fmer}, \text{fstat}) -$

1172. which it transcendently deduces into a freight behaviour
1173. which behaves concurrently,  $\parallel$ ,
1174. with a resumed freight forwarder behaviour with an augmented history that reflects the creation of a freight (endurant and behaviour).

**type**

1163.  $\text{mkF} :: \text{F\_UI} \times \text{F\_Mer} \times \text{F\_Stat}$

**value**

1164.  $\text{ffc} : \text{ffui} : \text{FF\_UI} \times (\text{sluis}, \text{fuis}) : \text{FF\_Mer} \times \text{ffstat} : \text{FF\_Stat} \rightarrow \text{ffprgr} : \text{FF\_Prgr}$

1164.  $\rightarrow \{ \text{ch}[\{ \text{slui}, \text{fui} \}] \mid \text{slui} : \text{SL\_UI} \bullet \text{slui} \in \text{sluis} \}$  **Unit**

1169.  $\text{ffc}(\text{ffui}, (\text{sluis}, \text{fuis}), \text{ffstat})(\text{ffhist}) \equiv$

1170. **let**  $\text{f\_ui} = \text{get\_F\_UI}()$  **in**

1171. **let**  $\text{mkF}(\text{fui}, \text{fmer}, \text{fstat}) = \text{heureka\_Freight}(\text{f\_ui}, \text{ffstat}, \dots)$  **in** [**axiom**  $\text{fui} = \text{f\_ui}$ ]

1172.  $\text{f}(\text{mkF}(\text{fui}, \text{fmer}, \text{fstat}))(\langle \langle \text{record\_TIME}() \rangle \rangle)$  **end**

1173.  $\parallel$

1174.  $\text{ff}(\text{ffui}, (\text{sluis}, \text{fuis}), \text{ffstat})([\text{fui} \rightarrow \langle \langle \text{record\_TIME}(), \text{mkF}(\text{fui}, \text{fmer}, \text{fstat}) \rangle \rangle]) \cup \text{ffhist}$  **end**

1171.  $\text{heureka\_Freight} : \text{F\_UI} \times \text{F\_Stat} \times \dots \rightarrow \text{mkF}$

**Freight Booking:**

1175. For the case that the freight forwarder history, for some freight,  $\text{fui}$ , records a singleton,  $\text{h}$ , which designates the creation of that freight, the freight forwarder offers the following transactions
- with a selected shipping line,  $\text{slui}$ , and for transport between specific harbours:
  - I offers, to that shipping line, a booking request containing the description,  $\text{mkF}(\dots)$ , of the freight, and the from- and to harbours of requested transport.
  - While awaiting a reply from the shipping line,
  - the freight forwarder records the time,  $\tau'$ , and an element,  $\text{h}'$ , of the freight forwarder history.
  - Before resuming being the freight forwarder behaviour,  $\text{ff}$ , the freight forwarder
  - records the time,  $\tau'$ , and an element,  $\text{h}'$ , of the freight forwarder history.
1176. For the case that the freight forwarder history, for some freight,  $\text{fui}$ , does not, for any freight ( $\text{fui}$ ), record a singleton,  $\text{h} : \langle \langle \tau, \text{mkF}(\text{fui}, \text{fmer}, \text{fstat}) \rangle \rangle ] \cup \text{ffhist}$ , which designates the creation of some freight, the freight forwarder does not engage in this alternative of the freight forwarder,  $\text{ff}$ , behaviour.

**type**

1175b.  $\text{mkBooking} :: \text{SL\_UI} \times \text{mkF}(\text{F\_UI}, \text{F\_Mer}, \text{F\_Stat}) \times (\text{H\_UI} \times \text{fd} : \text{TIME}) \times (\text{H\_UI} \times \text{td} : \text{TIME})$

1175b. **axiom**  $\forall \text{mkb} : \text{mkBooking} \bullet \text{fd}(\text{mkb}) < \text{td}(\text{mkb})$

1175c.  $\text{Reply} == \text{mk\_Decline\_Booking\_Request}(\text{SL\_UI}, \text{t} : \text{TIME}, \text{F\_UI})$

1175c.  $\mid \text{mk\_Accept\_Booking\_Request}(\text{SL\_UI}, \text{t} : \text{TIME}, \text{bol} : \text{BoL}, (\text{H\_UI} \times \text{TIME}), (\text{H\_UI} \times \text{TIME}))$

**value**

1164.  $\text{ffb} : \text{ffui} : \text{FF\_UI} \times (\text{sluis}, \text{fuis}) : \text{FF\_Mer} \times \text{ffstat} : \text{FF\_Stat} \rightarrow \text{ffhist} : \text{FF\_Hist}$

1164.  $\rightarrow \{ \text{ch}[\{ \text{slui}, \text{fui} \}] \mid \text{slui} : \text{SL\_UI} \bullet \text{slui} \in \text{sluis} \}$  **Unit**

1175.  $\text{ffb}(\text{ffui}, (\text{sluis}, \text{fuis}), \text{ffstat})(\text{ffhist} : [\text{fui} \rightarrow \text{h} : \langle \langle \tau, \text{mkF}(\text{fui}, \text{fmer}, \text{fstat}) \rangle \rangle ] \cup \text{ffhist}')$   $\equiv$

1175a.  $\text{freight\_booking}(\text{ffui}, (\text{sluis}, \text{fuis}), \text{ffstat})(\text{mkF}(\text{fui}, \text{fmer}, \text{fstat}))(\text{ffhist})$

1175a.  $\text{freight\_booking} : \text{ffui} : \text{FF\_UI} \times (\text{sluis}, \text{fuis}) : \text{FF\_Mer} \times \text{ffstat} : \text{FF\_Stat} \rightarrow \text{mkf} : \text{MkF} \rightarrow \text{ffhist} : \text{FF\_Hist}$

1164.  $\rightarrow \{ \text{ch}[\{ \text{slui}, \text{fui} \}] \mid \text{slui} : \text{SL\_UI} \bullet \text{slui} \in \text{sluis} \}$  **Unit**



```

1175a. freight_booking(ffui,(sluis,fuis),ffstat)(mkf)(ffhist) ≡
1175a. let (slui,(fh,fd),(th,td)) = select_shipping_line_and_time(ffstat,mkf,ffhist) in
1175b. ch[{ffui,slui}] ! mkBooking(slui,mkF(fui,fmer,fstat),(fh,fd),(th,td)) ;
1175d. let τ' = record_TIME(), h'' = ⟨(τ',mkBooking(slui,mkF(fui,fmer,fstat),(fh,fd),(th,td)))⟩ in
1175c. let reply = ch[{ffui,slui}] ? in
1175f. let τ'' = record_TIME(), h''' = ⟨(τ'',reply)⟩ in
1175e. ff(ffui,(sluis,fuis),ffstat)([fui→h'''^h''^h]∪fhist)
1175. end end end end

```

```

1175a. select_shipping_line_and_time: mkF(F_UI,F_Mer,F_Stat) × MkF × FF_Hist
1175a. → SL_UI × (H_UI×fd:TIME) × (H_UI×td:TIME)

```

### Freight Acceptance and Delivery

1177. For the case that the freight forwarder history, for some freight, fui, records a first, i.e., a most recent element which designates the booking acceptance,  $[fui \rightarrow \langle (\tau, \text{mk\_Accept\_Booking\_Request}(slui, t, \text{bol}, (fh, fd), (th, td))) \rangle^h] \cup fhist$ , for a freight, the freight forwarder offers the following transactions:

- (a) initially it offers to accept a designated, previously booked freight delivery to harbour of disembarkment;
- (b) before delivering this freight
- (c) the freight forwarder records the time,  $\tau''$ , and an element,  $h''$ , of the freight forwarder history;
- (d) before resuming being the freight forwarder behaviour, ff,
- (e) and, concurrently informing the freight of its freight forwarder to harbour transfer,
- (f) the freight forwarder records the time,  $\tau'''$ , and an element,  $h'''$ , of the freight forwarder history.

### type

```

1177. BoL [Bill-of-Lading]
1177. mk_Accept_Booking_Request :: TIME × BoL × (H_UI×fd:TIME) × (H_UI×td:TIME)
1177a. mkPIsDelive :: F_UI × H_UI × TIME
1177a. mkDelivery :: F_UI × H_UI × V_UI × TIME

```

### value

```

1164. ffd: ffui:FF_UI × (sluis,fuis):FF_Mer × ffstat:FF_Stat → FF_Hist
1164. → { ch[{slui,fui}] | slui:SL_UI•slui∈sluis } Unit
1177. ffd(ffui,(sluis,fuis),ffstat)
1177. ([fui→h:⟨(τ,mk_Accept_Booking_Request(slui,t,bol,(fh,fd),(th,td)))⟩^h']∪fhist) ≡
1177a. let mkPIsDeliver(slui,fui,hui,vui,τ') = ch[{ffui,slui}] ? in
1177c. let τ'' = record_TIME(), h'' = ⟨(τ'',mkPIsDeliver(slui,fui,hui,τ'))⟩ in
1177b. ch[{ffui,hui}] ! mkDelivery(ffui,fui,hui,vui) ;
1177f. let τ''' = record_TIME(), h''' = ⟨(τ''',mkDelivery(slui,fui,hui,τ'))⟩ in
1177d. ff(ffui,(sluis,fuis),ffstat)([fui→h'''^h''^h]∪fhist)
1177e. || ch[{ffui,fui}] ! mkXferFFtoH(τ''',ffui,hui)
1177. end end end

```

### Freight Declination and Re-booking:

1178. For the case that the freight forwarder history, for some freight, fui, records a first, i.e., a most recent element which designates a booking rejection  $\text{mk\_Decline\_Booking\_Request}(slui, t, fui)$ , the freight forwarder offers the transactions that are similar to those of Items 1175a–1175e  
Page 299.

**value**

1178.  $\text{ffd}(\text{ffui}, (\text{sluis}, \text{fuis}), \text{ffstat})$   
 1178.  $(\text{ffhist} : [\text{fui} \mapsto \text{h} : \langle (\tau, \text{mk\_Decline\_Booking\_Request}(\text{slui}, \text{t}, \text{fui}, \text{mkF}(\text{fui}, \text{fmer}, \text{fstat})) \rangle)^{\wedge} \text{h}' ] \cup \text{ffhist}') \equiv$   
 1178.  $\text{freight\_booking}(\text{ffui}, (\text{sluis}, \text{fuis}), \text{ffstat})(\text{mkF}(\text{fui}, \text{fmer}, \text{fstat}))(\text{ffhist})$

**Freight Recovery:**

1179. For the case that the freight forwarder history, for some freight, fui, records a first, i.e., a most recent element which designates the delivery of freight, in its care:  $\text{mkDelivery}(\text{slui}, \text{fui}, \text{hui}, \tau)$ , the freight forwarder offers the following transaction:

- (a) it offers to accept an alert from the shipping line as to the impending vessel arrival at destination port whereupon it
- (b) informs the freight of its harbour to freight forwarder transfer,
- (c) resumes being the freight forwarder behaviour now suitably updated with that knowledge!

**value**

1164.  $\text{ffr} : \text{ffui} : \text{FF\_UI} \times (\text{sluis}, \text{fuis}) : \text{FF\_Mer} \times \text{ffstat} : \text{FF\_Stat} \rightarrow \text{ffhist} : \text{FF\_Prgr}$   
 1164.  $\rightarrow \{ \text{ch}[\{ \text{slui}, \text{fui} \}] \mid \text{slui} : \text{SL\_UI} \bullet \text{slui} \in \text{sluis} \}$  **Unit**  
 1179.  $\text{ffr}(\text{ffui}, (\text{sluis}, \text{fuis}), \text{ffstat})([\text{fui} \mapsto \text{hist} : \langle (\tau, \text{mkDelivery}(\text{slui}, \text{fui}, \text{hui}, \tau')) \rangle^{\wedge} \text{hist}' ] \cup \text{ffhist}) \equiv$   
 1179a. **let**  $\text{mkReturn}(\text{slui}, \text{fui}, \text{hui}, \text{vui}, \text{dat}) = \text{ch}[\{ \text{slui}, \text{ffui} \}] ?$   
 1179a.  $\tau'' = \text{record\_TIME}()$  **in**  
 1179b.  $\text{ch}[\{ \text{ffui}, \text{fui} \}] ! \text{mkXferHtoFF}(\tau'', \text{ffui}, \text{hui})$   
 1179c.  $\parallel \text{ff}(\text{ffui}, (\text{sluis}, \text{fuis}), \text{ffstat})([\text{fui} \mapsto \langle (\tau'', \text{mkReturn}(\text{slui}, \text{fui}, \text{hui}, \text{vui}, \text{dat})) \rangle^{\wedge} \text{hist}' ] \cup \text{ffhist})$   
 1179. **end**

**Freight Termination:**

1180. For the case that the freight forwarder history, for some freight, fui, records a first, i.e., a most recent element which designates the return of freight, in its care:  $\text{mkReturn}(\text{slui}, \text{fui}, \text{hui}, \text{vui}, \text{dat})$ , the freight forwarder offers the following transaction:

- (a) the freight forwarder inquires with a designated return harbour, hui, as to the designated, returned freight, fui
- (b) and resumes being the freight forwarder behaviour now suitably updated with that knowledge!
- (c) while, at the same time as resumption also informing the freight that it no longer has freight status!

**value**

1164.  $\text{ffe} : \text{ffui} : \text{FF\_UI} \times (\text{sluis}, \text{fuis}) : \text{FF\_Mer} \times \text{ffstat} : \text{FF\_Stat} \rightarrow \text{hist} : \text{FF\_Hist}$   
 1164.  $\rightarrow \{ \text{ch}[\{ \text{slui}, \text{fui} \}] \mid \text{slui} : \text{SL\_UI} \bullet \text{slui} \in \text{sluis} \}$  **Unit**  
 1167d.  $\text{ffe}(\text{ffui}, (\text{sluis}, \text{fuis}), \text{ffstat})([\text{fui} \mapsto \text{hist} : \langle (\tau''', \text{hist} : \text{mkReturn}(\text{slui}, \text{fui}, \text{hui}, \text{vui}, \text{dat})) \rangle^{\wedge} \text{hist}' ] \cup \text{ffhist}) \equiv$   
 1180a. **let**  $\text{mkReturnedFreight}(\text{fui}, \dots) = \text{ch}[\{ \text{hui}, \text{ffui} \}] ?$  **in**  
 1180b.  $\text{ff}(\text{ffui}, (\text{sluis}, \text{fuis}), \text{ffstat})([\text{fui} \mapsto \langle \text{mkReturnedFreight}(\text{fui}, \dots) \rangle^{\wedge} \text{hist}' ] \cup \text{ffhist})$   
 1180c.  $\parallel \text{ch}[\{ \text{ffui}, \text{fui} \}] ! \text{mkTerminateFreight}(\text{ffui}, \dots)$   
 1167d. **end**

**Freight Forwarder Management:**

1181.

1182.

1183.

**value**1164.  $\text{ffm}: \text{ffui}:\text{FF\_UI} \times (\text{sluis}, \text{fuis}):\text{FF\_Mer} \times \text{ffstat}:\text{FF\_Stat} \rightarrow \text{ffprgr}:\text{FF\_Prgr}$ 1164.  $\rightarrow \{ \text{ch}[\{ \text{slui}, \text{fui} \}] \mid \text{slui}:\text{SL\_UI} \bullet \text{slui} \in \text{sluis} \}$  **Unit**1168.  $\text{ffm}(\text{ffui}, (\text{sluis}, \text{fuis}), \text{ffstat})([\text{fui} \mapsto \langle \tau, [\text{FC}] \rangle]) \cup \text{fhist} \equiv$ 

1181.

1182.

1183.

**J.4.5.2.2 Shipping Line Behaviour Definition**

1184.

1185.

1186.

1187.

1188.

1189.

1190.

1191.

1192.

1193.

**value**1165.  $\text{sl}: \text{slui}:\text{SL\_UI} \times (\text{vuis}, \text{ffuis}, \text{huis}):\text{SL\_Mer} \times \text{slstat}:\text{SL\_Stat} \rightarrow \text{slprgr}:\text{SL\_Prgr} \rightarrow \text{newf}:\text{NewF}$ 1165.  $\rightarrow \{ \text{ch}[\{ \text{slui}, \text{ui} \}] \mid \text{ui}:\text{FF\_UI} \mid \text{V\_UI} \mid \text{H\_UI} \bullet \text{ui} \in \text{ffuis} \cup \text{huis} \}$  **Unit**1185.  $\text{sl}(\text{slui}, (\text{vuis}, \text{ffuis}, \text{huis}), \text{slstat})(\text{slprgr})(\text{newf}) \equiv$ 

1185.

1186.

1187.

1188.

1189.

1190.

1191.

1192.

1193.

**J.4.5.2.3 Vessel Behaviour Definition****J.4.5.2.4 Harbour Behaviour Definition**

1194.

1195.

1196.

1197.

302

CONTENTS

1198.

1199.

1200.

1201.

1202.

1203.

**value**

1194. harbour:

1194. harbour(hui,(ffuis,sluis),hstat)(hhist)  $\equiv$

1195.

1196.

1197.

1198.

1199.

1200.

1201.

1202.

1203.

#### J.4.5.2.5 Freight Behaviour Definition

1204.

1205.

1206.

1207.

1208.

1209.

1210.

1211.

1212.

1213.

**value**

1204. freight: fui:F\_UI  $\times$  (ffui,(fhui,thui),vui,slui):F\_Met  $\times$  F\_Stat  $\rightarrow$  F\_Hist  $\rightarrow$

1204. **in** { ch[ {fhui,ui} ] | ui:(FH\_UI|V\_UI|SL\_UI)•ui  $\in$  {ffui,fhui,thui,vui,slui} } **Unit**

1204. freight(fui,(ffui,(fhui,thui),vui,slui),hstat)(fhist)  $\equiv$

1205. **let** i:mkFFtoH(...) = ch[ {ffui,fui} ] ? **in** freight(fui,(ffui,(fhui,thui),vui,slui),hstat)((i)^fhist) **end**

1206. **let** i:mk(...) = ch[ {vui,fui} ] ? **in** freight(fui,(ffui,(fhui,thui),vui,slui),hstat)((i)^fhist) **end**

1207. **let** i:mk(...) = ch[ {thui,fui} ] ? **in** freight(fui,(ffui,(fhui,thui),vui,slui),hstat)((i)^fhist) **end**

1208. **let** i:mk(...) = ch[ {thui,fui} ] ? **in** freight(fui,(ffui,(fhui,thui),vui,slui),hstat)((i)^fhist) **end**

1209. **let** i:mk(...) = ch[ {ffui,fui} ] ? **in** freight(fui,(ffui,(fhui,thui),vui,slui),hstat)((i)^fhist) **end**

1211. **let** i:mk(...) = ch[ {ffui,fui} ] ? **in** freight(fui,(ffui,(fhui,thui),vui,slui),hstat)((i)^fhist) **end**

1212. **let** i:mk(...) = ch[ {ffui,fui} ] ? **in** freight(fui,(ffui,(fhui,thui),vui,slui),hstat)((i)^fhist) **end**

1213. **let** i:mk(...) = ch[ {ffui,fui} ] ? **in** skip **end**

**J.5 Review**

TO BE WRITTEN



# Appendix K

## Container Terminals

### Contents

---

|            |                                                    |            |
|------------|----------------------------------------------------|------------|
| <b>K.1</b> | <b>Introduction</b>                                | <b>308</b> |
| K.1.1      | Reference Literature on Container-related Matters  | 308        |
| <b>K.2</b> | <b>Some Pictures</b>                               | <b>309</b> |
| K.2.1      | Terminal Port Container Stowage Area               | 309        |
| K.2.2      | Container Stowage Area and Quay Cranes             | 309        |
| K.2.3      | Container Vessel Routes                            | 310        |
| K.2.4      | Containers                                         | 310        |
| K.2.4.1    | 40 and 20 Feet Containers                          | 310        |
| K.2.4.2    | Container Markings                                 | 310        |
| K.2.5      | Container Vessels                                  | 311        |
| K.2.6      | Container Stowage Area: Bays Rows, Stacks and Tier | 311        |
| K.2.7      | Stowage Software                                   | 312        |
| K.2.8      | Quay Cranes                                        | 312        |
| K.2.9      | Container Stowage Area and Stack Cranes            | 312        |
| K.2.10     | Container Stowage Area                             | 312        |
| K.2.11     | Quay Trucks                                        | 313        |
| K.2.12     | Map of Shanghai and YangShan                       | 313        |
| <b>K.3</b> | <b>SECT</b>                                        | <b>313</b> |
| <b>K.4</b> | <b>Main Behaviours</b>                             | <b>315</b> |
| K.4.1      | A Diagram                                          | 316        |
| K.4.2      | Terminology - a Caveat                             | 316        |
| K.4.3      | Assumptions                                        | 317        |
| <b>K.5</b> | <b>Endurants</b>                                   | <b>317</b> |
| K.5.1      | Parts                                              | 317        |
| K.5.1.1    | Terminal Ports                                     | 318        |
| K.5.1.2    | Quays                                              | 319        |
| K.5.1.3    | Container Stowage Areas: Bays, Rows and Stacks     | 319        |
| K.5.1.4    | Vessels                                            | 319        |
| K.5.1.5    | Functions Concerning Container Stowage Areas       | 320        |
| K.5.1.6    | Axioms Concerning Container Stowage Areas          | 320        |
| K.5.1.7    | Stacks                                             | 321        |
| K.5.2      | Terminal Port Command Centers                      | 321        |
| K.5.2.1    | Discussion                                         | 321        |
| K.5.2.2    | Justification                                      | 321        |
| K.5.3      | Unique Identifications                             | 322        |

|           |                                                               |     |
|-----------|---------------------------------------------------------------|-----|
| K.5.3.1   | Unique Identifiers: Distinctness of Parts                     | 322 |
| K.5.3.2   | Unique Identifiers: Two Useful Abbreviations                  | 322 |
| K.5.3.3   | Unique Identifiers: Some Useful Index Set Selection Functions | 323 |
| K.5.3.4   | Unique Identifiers: Ordering of Bays, Rows and Stacks         | 323 |
| K.5.4     | States, Global Values and Constraints                         | 323 |
| K.5.4.1   | States                                                        | 323 |
| K.5.4.2   | Unique Identifiers                                            | 324 |
| K.5.4.3   | Some Axioms on Uniqueness                                     | 325 |
| K.5.5     | Mereology                                                     | 325 |
| K.5.5.1   | Physical versus Conceptual Mereology                          | 325 |
| K.5.5.2   | Vessels                                                       | 325 |
| K.5.5.2.1 | Physical Mereology:                                           | 325 |
| K.5.5.2.2 | Conceptual Mereology:                                         | 326 |
| K.5.5.3   | Quay Cranes                                                   | 326 |
| K.5.5.3.1 | Physical Mereology:                                           | 326 |
| K.5.5.3.2 | Conceptual Mereology:                                         | 326 |
| K.5.5.4   | Quay Trucks                                                   | 327 |
| K.5.5.4.1 | Physical Mereology:                                           | 327 |
| K.5.5.4.2 | Conceptual Mereology:                                         | 327 |
| K.5.5.5   | Stack Cranes                                                  | 327 |
| K.5.5.5.1 | Physical Mereology:                                           | 327 |
| K.5.5.5.2 | Conceptual Mereology:                                         | 327 |
| K.5.5.6   | Container Stowage Areas                                       | 328 |
| K.5.5.6.1 | Bays, Rows and Stacks:                                        | 328 |
| K.5.5.7   | Bay Mereology                                                 | 328 |
| K.5.5.7.1 | Physical Vessel Bay Mereology:                                | 328 |
| K.5.5.7.2 | Conceptual Vessel Bay Mereology:                              | 328 |
| K.5.5.7.3 | Physical Terminal Port Bay (cum Stack) Mereology:             | 329 |
| K.5.5.7.4 | Conceptual Terminal Port Bay (cum Stack) Mereology:           | 329 |
| K.5.5.8   | Land Trucks                                                   | 329 |
| K.5.5.8.1 | Physical Mereology:                                           | 329 |
| K.5.5.8.2 | Conceptual Mereology:                                         | 329 |
| K.5.5.9   | Command Center                                                | 329 |
| K.5.5.10  | Conceptual Mereology of Containers                            | 330 |
| K.5.6     | Attributes                                                    | 330 |
| K.5.6.1   | States                                                        | 330 |
| K.5.6.2   | Actions                                                       | 330 |
| K.5.6.3   | Attributes: Quays                                             | 330 |
| K.5.6.4   | Attributes: Vessels                                           | 331 |
| K.5.6.5   | Attributes: Quay Cranes                                       | 331 |
| K.5.6.6   | Attributes: Quay Trucks                                       | 332 |
| K.5.6.7   | Attributes: Terminal Stack Cranes                             | 332 |
| K.5.6.8   | Attributes: Container Stowage Areas                           | 332 |
| K.5.6.9   | Attributes: Land Trucks                                       | 333 |
| K.5.6.10  | Attributes: Command Center                                    | 333 |
| K.5.6.11  | Attributes: Containers                                        | 334 |
| K.6       | Perdurants                                                    | 335 |
| K.6.1     | A Modelling Decision                                          | 335 |
| K.6.2     | Virtual Container Storage Areas                               | 335 |
| K.6.3     | Changes to The Parts Model                                    | 336 |



|            |                                                            |     |
|------------|------------------------------------------------------------|-----|
| K.6.4      | <b>Basic Model Parts</b>                                   | 336 |
| K.6.5      | <b>Actions, Events, Channels and Behaviours</b>            | 337 |
| K.6.6      | <b>Actions</b>                                             | 337 |
| K.6.6.1    | <b>Command Center Actions</b>                              | 337 |
| K.6.6.1.1  | <b>Motivating the Command Center Concept:</b>              | 337 |
| K.6.6.1.2  | <b>Calculate Next Transaction:</b>                         | 337 |
| K.6.6.1.3  | <b>Command Center Action [A]: update_mcc_from_vessel:</b>  | 339 |
| K.6.6.1.4  | <b>Command Center Action [B]: calc-ves-pos:</b>            | 339 |
| K.6.6.1.5  | <b>Command Center Action [C-D-E]: calc-ves_qc</b>          | 339 |
| K.6.6.1.6  | <b>Command Center Action [F-G-H]: calc_qc_qt</b>           | 339 |
| K.6.6.1.7  | <b>Command Center Action [I-J-K]: calc_qt_sc</b>           | 340 |
| K.6.6.1.8  | <b>Command Center Action [L-M-N]: calc_sc_stack</b>        | 340 |
| K.6.6.1.9  | <b>Command Center Action [N-M-L]: calc_stack_sc</b>        | 340 |
| K.6.6.1.10 | <b>Command Center Action [O-P-Q]: calc_sc_lt</b>           | 341 |
| K.6.6.1.11 | <b>Command Center Action [Q-P-O]: calc_lt_sc</b>           | 341 |
| K.6.6.1.12 | <b>Command Center: Further Observations</b>                | 341 |
| K.6.6.2    | <b>Container Storage Area Actions</b>                      | 341 |
| K.6.6.2.1  | <b>The Load Pre-/Post-Conditions</b>                       | 342 |
| K.6.6.2.2  | <b>The Unload Pre-/Post-Conditions</b>                     | 342 |
| K.6.6.3    | <b>Vessel Actions</b>                                      | 343 |
| K.6.6.3.1  | <b>Action [A]: calc_next_port:</b>                         | 343 |
| K.6.6.3.2  | <b>Vessel Action [B]: calc-ves_msg:</b>                    | 344 |
| K.6.6.4    | <b>Land Truck Actions</b>                                  | 344 |
| K.6.6.4.1  | <b>Land Truck Action [R]: calc_truck_delivery:</b>         | 344 |
| K.6.6.4.2  | <b>Land Truck Action [S]: calc_truck_avail:</b>            | 344 |
| K.6.7      | <b>Events</b>                                              | 345 |
| K.6.7.1    | <b>Active Part Initiation Events</b>                       | 345 |
| K.6.7.2    | <b>Active Part Completion Events:</b>                      | 346 |
| K.6.8      | <b>Channels</b>                                            | 346 |
| K.6.8.1    | <b>Channel Declarations</b>                                | 346 |
| K.6.8.2    | <b>Channel Messages</b>                                    | 347 |
| K.6.8.2.1  | <b>A,B,X,Y,C': Vessel Messages</b>                         | 347 |
| K.6.8.2.2  | <b>C,D,E,E': Vessel/Container/Quay Crane Messages</b>      | 347 |
| K.6.8.2.3  | <b>F,G,H,H': Quay Crane/Container/Quay Truck Messages</b>  | 348 |
| K.6.8.2.4  | <b>I,J,K,K': Quay Truck/Container/Stack Crane Messages</b> | 348 |
| K.6.8.2.5  | <b>L,M,N,N': Stack Crane/Container/Stack Messages</b>      | 349 |
| K.6.8.2.6  | <b>O,P,Q,Q': Land Truck/Container/Stack Crane Messages</b> | 349 |
| K.6.8.2.7  | <b>R,S,T,U,Q,V: Land Truck Messages</b>                    | 350 |
| K.6.9      | <b>Behaviours</b>                                          | 350 |
| K.6.9.1    | <b>Terminal Command Center</b>                             | 350 |
| K.6.9.1.1  | <b>The Command Center Behaviour:</b>                       | 351 |
| K.6.9.1.2  | <b>The Command Center Monitor Behaviours:</b>              | 351 |
| K.6.9.1.3  | <b>The Command Center Control Behaviours:</b>              | 352 |
| K.6.9.2    | <b>Vessels</b>                                             | 353 |
| K.6.9.2.1  | <b>Port Approach</b>                                       | 353 |
| K.6.9.2.2  | <b>Port Arrival</b>                                        | 354 |
| K.6.9.2.3  | <b>Unloading of Containers</b>                             | 354 |
| K.6.9.2.4  | <b>Loading of Containers</b>                               | 355 |

|            |                                                  |            |
|------------|--------------------------------------------------|------------|
| K.6.9.2.5  | Port Departure                                   | 355        |
| K.6.9.3    | Quay Cranes                                      | 355        |
| K.6.9.4    | Quay Trucks                                      | 356        |
| K.6.9.5    | Stack Crane                                      | 356        |
| K.6.9.6    | Stacks                                           | 356        |
| K.6.9.7    | Land Trucks                                      | 357        |
| K.6.9.8    | Containers                                       | 358        |
| K.6.10     | Initial System                                   | 358        |
| K.6.10.1   | The Distributed System                           | 358        |
| K.6.10.2   | Initial Vessels                                  | 358        |
| K.6.10.3   | Initial Land Trucks                              | 359        |
| K.6.10.4   | Initial Containers                               | 359        |
| K.6.10.5   | Initial Terminal Ports                           | 359        |
| K.6.10.6   | Initial Quay Cranes                              | 359        |
| K.6.10.7   | Initial Quay Trucks                              | 359        |
| K.6.10.8   | Initial Stack Cranes                             | 360        |
| K.6.10.9   | Initial Stacks                                   | 360        |
| <b>K.7</b> | <b>Conclusion</b>                                | <b>360</b> |
| K.7.1      | An Interpretation of the Behavioural Description | 360        |
| K.7.2      | What Has Been Done                               | 360        |
| K.7.3      | What To Do Next                                  | 360        |
| K.7.4      | Acknowledgements                                 | 360        |

---

We present a recording of stages and steps of a development of a domain analysis & description of an answer to the question: *what, mathematically, is a container terminal?*

This is a report on an experiment. At any stage of development, and the present draft stage is judged 2/3 “completed” it reflects how I view an answer to the question *what is a container terminal port?* mathematically speaking.

## K.1 Introduction

|               |
|---------------|
| TO BE WRITTEN |
|---------------|

### K.1.1 Reference Literature on Container-related Matters

We refer to: [29, A Container Line Industry Domain, 2007], [89, A-Z Dictionary of Export, Trade and Shipping Terms], [145, Portworker Development Programme: PDP Units], [151, An interactive simulation model for the logistics planning of container operations in seaports, 1996], [4, Stowage planning for container ships to reduce the number of shifts, 1998], [180, Container stowage planning: a methodology for generating computerised solutions, 2000], [3, Container ship stowage problem: complexity and connection to the coloring of circle graphs, 2000], [181, Container stowage pre-planning: using search to generate solutions, a case study, 2001], [87, A genetic algorithm with a compact solution encoding for the container ship stowage problem, 2002], [114, Multi-objective ... stowage and load planning for a container ship with container rehandle ..., 2004], [175, Container terminal operation and operations research - a classification and literature review, 2004], [80, On-line rules for container stacking, 2010],

## K.2 Some Pictures

### K.2.1 Terminal Port Container Stowage Area



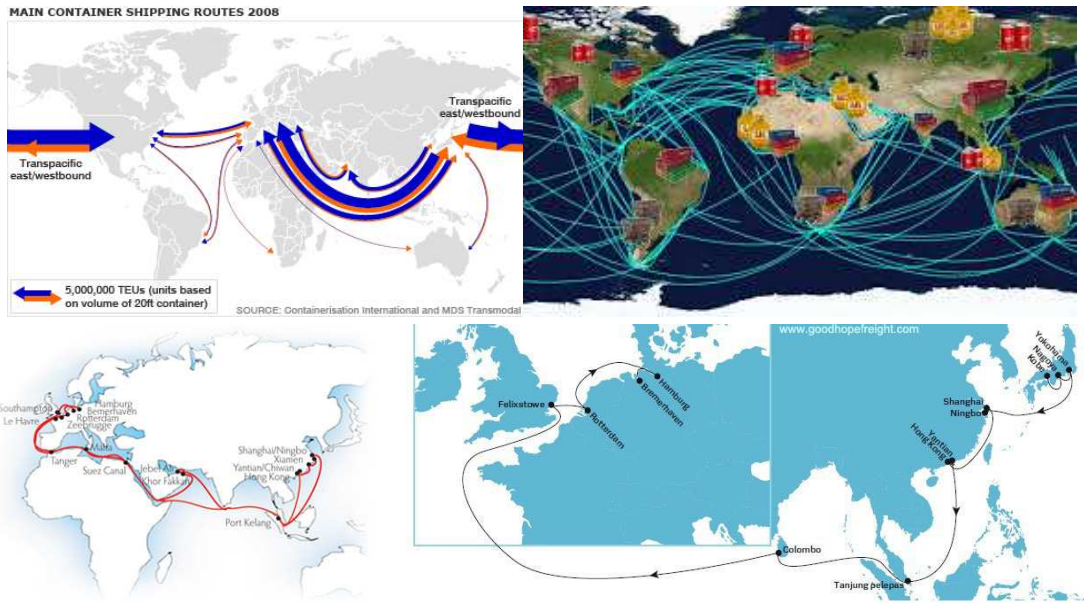
Analysis of the above picture:

- The picture shows a *terminal*.
- At bottom we are hinted (through shadows) at *quay cranes* serving (unshown) *vessels*.
- Most of the picture shows a *container stowage area*, here organized as a series of columns, from one side of the picture to the other side, e.g., left-to-right, sequences (top-to-bottom) of [blue] *bays* with *rows* of *stacks* of *containers*.
- Almost all columns show just one *bay*.
- Three “rightmost” columns show many [non-blue] *bays*.
- Most of the column “tops” and “bottoms” show *stack cranes*.
- The four leftmost columns show *stack cranes* at *bays* “somewhere in the middle” of a column.

### K.2.2 Container Stowage Area and Quay Cranes



### K.2.3 Container Vessel Routes



### K.2.4 Containers

#### K.2.4.1 40 and 20 Feet Containers



#### K.2.4.2 Container Markings





K.2.5 Container Vessels



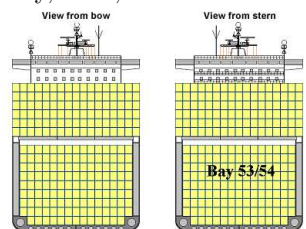
www.alarby.com - E8WD68

Quay cranes and vessel showing row of aft (rear) bay.

K.2.6 Container Storage Area: Bays Rows, Stacks and Tier



Bay, Row, Tier Numbers.



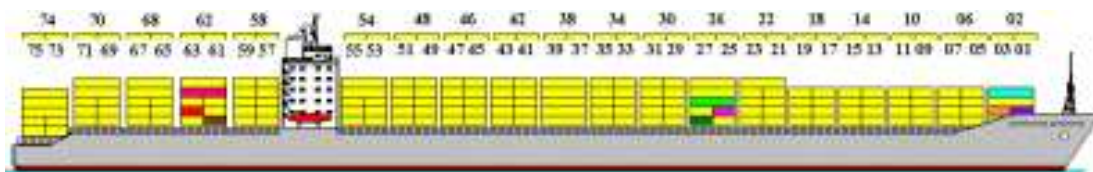
Cross section of a Bay.



Row Numbers

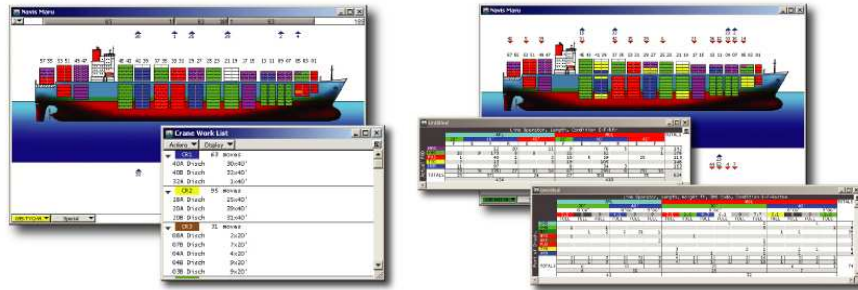


Tier Numbers.



Bay Numbering

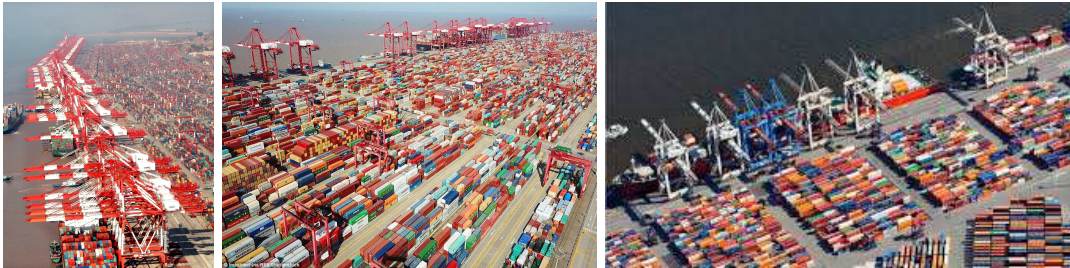
### K.2.7 Stowage Software



### K.2.8 Quay Cranes



### K.2.9 Container Storage Area and Stack Cranes



### K.2.10 Container Storage Area





### K.2.11 Quay Trucks



### K.2.12 Map of Shanghai and YangShan



## K.3 SECT

- *Shanghai East Container Terminal*
  - is the joint venture terminal of
  - *APM Terminals* and
  - *Shanghai International Port Group*
  - in *Wai Gao Qiao* port area of *Shanghai*.
- No.1 Gangjian Road, Pudong New District, Shanghai, China





- Wusong Port Area
- Weigeoqiao Port Area
- Yangshan Deepwater Port Area

| Distance Between Ports and Stations/terminals in Shanghai |                  |                              |
|-----------------------------------------------------------|------------------|------------------------------|
| Shanghai city center                                      | Wusong Port Area | Yangshan Deepwater Port Area |
| Shanghai city center                                      | 24 km (15 mi)    | 24 km (15 mi)                |
| Shanghai Minsheng Station                                 | 24 km (15 mi)    | 24 km (15 mi)                |
| Shanghai South Railway Station                            | 34 km (21 mi)    | 34 km (21 mi)                |
| Shanghai Hongqiao Railway Station                         | 47 km (29 mi)    | 47 km (29 mi)                |
| Shanghai Long Distance Bus Station                        | 32 km (20 mi)    | 32 km (20 mi)                |
| Shanghai Pudong International Airport                     | 12 km (7 mi)     | 12 km (7 mi)                 |
| Shanghai International Airport                            | 34 km (21 mi)    | 34 km (21 mi)                |
|                                                           | Wusong Port Area | Yangshan Deepwater Port Area |
|                                                           | 24 km (15 mi)    | 24 km (15 mi)                |
|                                                           | 24 km (15 mi)    | 24 km (15 mi)                |
|                                                           | 34 km (21 mi)    | 34 km (21 mi)                |
|                                                           | 47 km (29 mi)    | 47 km (29 mi)                |
|                                                           | 32 km (20 mi)    | 32 km (20 mi)                |
|                                                           | 12 km (7 mi)     | 12 km (7 mi)                 |
|                                                           | 34 km (21 mi)    | 34 km (21 mi)                |

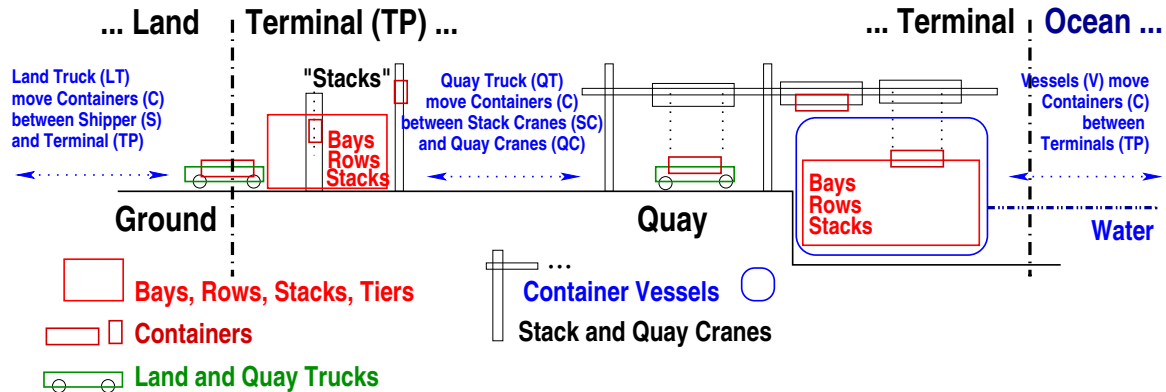


## K.4 Main Behaviours



- From consumer/origin to consumer/final destination:
  - container loads onto land truck;
  - land truck travels to terminal stack;
  - container unloads by means of terminal stack crane from land truck onto terminal stack.
  - Container moves from stack to vessel:
    - \* terminal stack crane moves container from terminal stack to quay truck,
    - \* quay truck moves container from terminal stack to quay,
    - \* quay crane moves container to top of a vessel stack;
  - Container moves on vessel from terminal to terminal:
    - \* Either container is unloaded at a next terminal port to a stack and from there to a container truck
    - \* or: container is unloaded at a next terminal port to a stack and from there to a next container vessel.

### K.4.1 A Diagram



**Fig. 1: Container Terminal Ports, I**  
A “from the side” snapshot of terminal port activities

### K.4.2 Terminology - a Caveat

**Bay**<sup>1</sup>: contains indexed set of *rows* (of stacks of containers).

**Container**: smallest unit of central (i.e., huge) concern !

**Container Stowage Area**: An area of a vessel or a terminal where containers are stored, during voyage, respectively awaiting to be either brought out to shippers or onto vessels.

**Crane**:

**Stack Crane**: moves *containers* between *land* or *terminal trucks* and *terminal stacks*.

**Quay Crane**: moves *containers* between [*land* or] *terminal trucks* and *vessels*.

**Land**: ... as you know it ...

**Ocean**: ... as you know it ...

**Shipper**: arranges shipment of containers with container lines

**Quay**: area of terminal next to vessels (hence water).

**Row**: contains indexed set of *stacks* (of containers).

**Stack**: contains indexed set of *containers*.

We shall also, perhaps confusingly, use the term *stack* referring to the land-based bays of a terminal.

**Terminal**: area of land and water between land and ocean equipped with container stowage area, and stack and quay cranes, etc.

**Truck**:

**Land Truck**: privately operated truck transport *containers* between *shippers* and *stack cranes*.

**Quay Truck**: terminal operated special truck transport *containers* between *stack cranes* and *quay cranes*.

<sup>1</sup>The terms introduced in this section are mine. They are most likely not the correct technical terms of the container shipping and stowage trade. I expect to revise this section, etc.

**Tier** : index of *container* in *stack*.

**Vessel** : contains a *container stowage area*.

### K.4.3 Assumptions

Without loss of generality we can assume that there is exactly one stack crane per land-based terminal stack; quay cranes each serve exactly one bay on a vessel; there are enough quay cranes to serve all bays of any berthed vessel; quay trucks may serve any (quay and stack) crane; land trucks may serve more than one terminal; et cetera.

## K.5 Endurants



**Fig. 2: Container Terminal Ports, II**

A “from above” snapshot of terminal port activities

We refer to [56, Sects. 3., 4., and 5.].

Our model focuses initially on parts, that is, manifest, observable phenomena. Our choice of these is expected to be subject to serious revision once we ... MORE TO COME ...

### K.5.1 Parts

We refer to [56, Sect. 3.3].

Our model has, perhaps arbitrarily, focused on just some of the manifest, i.e., observable parts of a domain of container terminal ports. We shall invariably refer to container terminal ports as either container terminals, or terminal ports,  $tp:TP$ , or just terminals. We expect revisions to the decomposition as shown as we learn more from professional stakeholders, e.g., *APM Terminals/SECT*, Shanghai.

1. In the container line industry, CLI, we can observe
2. a structure,  $TPS$ , of all terminal ports, and from each such structure, an indexed set,  $TPs$ , of two or more container *terminal* ports,  $TP$ ;

3. a structure,  $VS$ , of all container vessels, and from each such structure, an indexed set,  $Vs$ , of one or more container *vessels*,  $V$ ; and
4. a structure,  $LTS$ , of all land trucks, and from each such structure, a non-empty, indexed set,  $LTs$  of land trucks,  $LT$ ;

**type**

- 1 CLI
- 2 STPs, TP<sub>s</sub> = TP-**set**, TP
- 3 SVs, V<sub>s</sub> = V-**set**, V
- 4 SLTs, LT<sub>s</sub> = LT-**set**, LT

**value**

- 2 obs\_STPs: CLI → STPs, obs\_TP<sub>s</sub>: STPs → TP<sub>s</sub>
- 3 obs\_SVs: CLI → SVs, obs\_V<sub>s</sub>: SVs → V<sub>s</sub>
- 4 obs\_SLTs: CLI → SLTs, obs\_LT<sub>s</sub>: SLTs → LT<sub>s</sub>

**axiom**

- 2  $\forall cli:CLI \bullet \mathbf{card} \text{ obs\_TPs}(\text{obs\_STPs}(cli)) \geq 2$
- 3  $\wedge \mathbf{card} \text{ obs\_Vs}(\text{obs\_SVs}(cli)) \geq 1$
- 4  $\wedge \mathbf{card} \text{ obs\_LTs}(\text{obs\_SLTs}(cli)) \geq 1$

**K.5.1.1 Terminal Ports**

In a terminal port,  $tp:TP$ , one can observe

5. a [composite] container stowage area,  $csa:CSA$ ;
6. a structure,  $sqc:SQC$ , of quay cranes, and from that, a non-empty, indexed set,  $qcs:QCs$ , of one or more quay cranes,  $qc:QC$ ;
7. structure,  $sqt:SQT$ , of quay trucks, and from that a non-empty, indexed set,  $qts:QTs$ , of quay trucks,  $qt:QT$ ;
8. a structure,  $scs:SCS$ , of stack cranes, and from that a non-empty, indexed set,  $scs:SCs$ , of one or more stack cranes,  $sc:SC$ ;
9. a[n atomic] quay<sup>2</sup>,  $q:Q^3$ ; and
10. a[n atomic] terminal port monitoring and control center,  $mcc:MCC$ .

**type**

- 5 CSA
- 6 SQC, QC<sub>s</sub> = QC-**set**, QC
- 7 SQT, QT<sub>s</sub> = QT-**set**, QT
- 8 SCS, SC<sub>s</sub> = SC-**set**, SC
- 9 Q
- 10 MCC

**value**

- 5 obs\_CSA: TP → CSA
- 6 obs\_SQC: TP → SQC, obs\_QCs: SQC → QC<sub>s</sub>
- 7 obs\_SQT: TP → SQT, obs\_QTs: SQT → QT<sub>s</sub>
- 8 obs\_SCS: TP → SCS, obs\_SCs: SCS → SC<sub>s</sub>

<sup>2</sup>We can, without loss of generality, describe a terminal as having exactly one quay (!) – just as we, again without any loss of generality, describe it as having exactly one container stowage area.

<sup>3</sup>*Quay*: a long structure, usually built of stone, where boats can be tied up to take on and off their goods.

*Pronunciation*: key.

*Thesaurus*: berth, jetty, key, landing, levy, slip, wharf

```

9 obs_Q: TP → Q
10 obs_MCC: TP → MCC
axiom
6 ∀ sqc:SQC • card obs_QCs(sqc) ≥ 1
7 ∀ sqt:SQT • card obs_QTs(sqt) ≥ 1
8 ∀ scs:SCS • card obs_SCs(scs) ≥ 1

```

### K.5.1.2 Quays

Although container terminal port quays can be modelled as composite parts we have chosen to describe them as atomic. We shall subsequently endow the single terminal port quay with such attributes as quay segments, quay positions and berthing<sup>4</sup>.

### K.5.1.3 Container Stowage Areas: Bays, Rows and Stacks

11. From a container stowage area one can observe a non-empty indexed set of bays,
12. From a bay we can observe a non-empty indexed set of rows.
13. From a row we can observe a non-empty indexed set of stacks.
14. From a stack we can observe a possibly empty indexed set of containers.

#### type

```

11 BAYS, BAYs = BAY-set, BAY
12 ROWS, ROWs = ROW-set, ROW
13 STKS, STKs = STK-set, STK
14 CONS, CONs = CON-set, CON

```

#### value

```

11 obs_BAYS: CSA → BAYS, obs_BAYs: BAYS → BAYs
12 obs_ROWS: BAY → ROWS, obs_ROWs: ROWS → ROWs
13 obs_STKS: ROW → STKS, obs_STKs: STKS → STKs
14 obs_CONS: STK → CONS, obs_CONs: CONS → CONs

```

#### axiom

```

11 ∀ bays:BAYs • card bays > 0
12 ∀ rows:ROWs • card rows > 0
13 ∀ stks:STKs • card stks > 0

```

### K.5.1.4 Vessels

From (or in) a vessel one can observe

15. [5] a container stowage area
16. and some other parts.

#### type

```

5 CSA
16 ...

```

#### value

```

5 obs_CSA: V → CSA
16 ...

```

---

<sup>4</sup>Berth: Sufficient space for a vessel to maneuver; a space for a vessel to dock or anchor; (whether occupied by vessels or not). Berthing: To bring (a vessel) to a berth; to provide with a berth.

### K.5.1.5 Functions Concerning Container Stowage Areas

17. One can calculate
18. the set of all container storage areas:
19. of all terminal ports together with those
20. of all container lines.

#### value

- 17 cont\_stow\_areas: CLI  $\rightarrow$  CSA-set
- 18 cont\_stow\_areas(cli)  $\equiv$
- 19  $\{ \text{obs\_CSA}(tp) \mid tp:TP \bullet tp \in \text{obs\_TPs}(\text{obs\_TPS}(cli)) \}$
- 20  $\cup \{ \text{obs\_CSA}(cl) \mid cl:CL \bullet cl \in \text{obs\_CLs}(\text{obs\_CLS}(cli)) \}$

One can calculate the containers of

21. a stack,
22. a row,
23. a bay, and
24. a container stowage area.

#### value

- 21 extr\_cons\_stack: STK  $\rightarrow$  CONs
- 21 extr\_cons\_stack(stk)  $\equiv \text{obs\_CONs}(\text{obs\_CONS}(stk))$
- 22 extr\_cons\_row: ROW  $\rightarrow$  CONs
- 22 extr\_cons\_row(row)  $\equiv$
- 22  $\{ \text{obs\_CONs}(\text{obs\_CONS}(stk)) \mid stk:STK \bullet stk \in \text{obs\_STKs}(\text{obs\_STKS}(stk)) \}$
- 23 extr\_cons\_bay: BAY  $\rightarrow$  CONs
- 23 extr\_cons\_bay(bay)  $\equiv$
- 23  $\{ \text{obs\_CONs}(\text{obs\_CONS}(row)) \mid row:ROW \bullet row \in \text{obs\_ROWS}(\text{obs\_ROWS}(bay)) \}$
- 24 extr\_cons\_csa: CSA  $\rightarrow$  CONs
- 24 extr\_cons\_csa(csa)  $\equiv$
- 24  $\{ \text{obs\_CONs}(\text{obs\_CONS}(bay)) \mid bay:BAY \bullet bay \in \text{obs\_BAYs}(\text{obs\_BAYS}(csa)) \}$

### K.5.1.6 Axioms Concerning Container Stowage Areas

25. All rows contain different, i.e. distinct containers.
26. All bays contain different, i.e. distinct containers.
27. All container stowage areas contain different, i.e. distinct containers.

#### value

- 25  $\forall cli:CLI \bullet$
- 25  $\forall csa, csa':CSA \bullet \{ csa, csa' \} \subseteq \text{cont\_stow\_areas}(cli) \bullet$
- 25  $\forall row, row':ROW \bullet$
- 25  $\{ row, row' \} \subseteq \text{obs\_ROWS}(\text{obs\_ROWS}(csa)) \cup \text{obs\_ROWS}(\text{obs\_ROWS}(csa')) \Rightarrow$
- 25  $\text{extr\_cons\_row}(row) \cap \text{extr\_cons\_row}(row') = \{ \} \wedge$
- 26  $\forall bay, bay':BAY \bullet$
- 26  $\{ bay, bay' \} \subseteq \text{obs\_ROWS}(\text{obs\_ROWS}(csa)) \cup \text{obs\_ROWS}(\text{obs\_ROWS}(csa')) \Rightarrow$
- 26  $\text{extr\_cons\_bay}(bay) \cap \text{extr\_cons\_bay}(bay') = \{ \} \wedge$
- 27  $\text{extr\_cons\_csa}(csa) \cap \text{extr\_cons\_csa}(csa') = \{ \}$

### K.5.1.7 Stacks

**An aside:** We shall use the term ‘stack’ in two senses: (i) as a component of container storage area bays; and (ii) to refer to the collection of stacks in a bay of a terminal container storage area.

28. Stacks are created empty, and hence stacks can be *empty*.
29. One can *push* a *container* onto a *stack* and obtain a *non-empty stack*.
30. One can *pop* a *container* from a *non-empty stack* and obtain a pair of a *container* and a possibly empty *stack*.

#### value

```
28 empty: () → STK, is_empty: STK → Bool
29 push: CON × STK → STK
30 pop: STK → (CON × STK)
```

#### axiom

```
28 is_empty(empty()), ~is_empty(push(c,stk))
29 pop(push(c,stk)) = (c,stk)
30 pre pop(stk),pop(push(c,stk)): ~is_empty(stk)
30 pop(empty()) = chaos
```

## K.5.2 Terminal Port Command Centers

### K.5.2.1 Discussion

We consider terminal port monitoring & control command centers to be atomic parts. The purpose of a terminal port command center is to monitor and control the allocation and servicing (berthing) of any visiting vessel to quay positions and by quay cranes, the allocation and servicing of vessels by quay cranes, the allocation and servicing of quay cranes by quay trucks, the allocation and servicing of quay trucks to quay cranes, containers and terminal stacks, the allocation and servicing of land trucks to containers and terminal stacks, This implies that there are means for communication between a terminal command center and vessels, quay cranes, stack cranes, quay trucks, land trucks, terminal stacks and containers.

### K.5.2.2 Justification

We shall justify the concept of terminal monitoring & control, i.e., command centers. First, using the *domain analysis & description* approach of [56], we know that we are going, through a transcendental deduction, to model certain parts as behaviours. These parts, we decide, after some analysis that we forego, to be vessels, quay cranes, quay trucks, stack cranes stacks, land trucks, and containers. Behaviours are usually like actors:they can instigate actions. But we decide, in our analysis, that some of these behaviours, quay cranes, quay trucks, stack cranes and stacks, are “*passive*” actors: are behaviourally not endowed with being able to initiate “own” actions. Instead, therefore, of all these behaviours, being able to communicate directly, pairwise, as loosely indicated by the figures of Pages 316 and 317, we model them to communicate *via* their terminal command centers.

*This is how we justify the introduction of the concept of terminal command centers.* They are an abstraction. In “*ye olde days*” you could observe, not one, but, perhaps, a hierarchy of terminal port offices, staffed by people, [each office, each group of staff] with its set of duties: communicating (by radio-phone) with approaching [and departing] vessels; scheduling quay positions, quay cranes and quay trucks; managing the operation of cranes and trucks; and, on a large scale, calculating stowage: on vessels and in terminals. Today, “*an age of ubiquitous computing*”, most of these offices and their staff are replaced by electronics: sensors, actuators, communication and computing, and with massive stowage data processing: where should containers be stowed on board vessels and in terminals so as to near-optimize all operations.

### K.5.3 Unique Identifications

We refer to [56, Sect. 5.1].

- |                                                                                               |                                                              |
|-----------------------------------------------------------------------------------------------|--------------------------------------------------------------|
| 31. Vessels have unique identifiers.                                                          | 37. Terminal port command centers have unique identifiers.   |
| 32. Quay cranes have unique identifiers.                                                      | 38. Containers have unique identifiers.                      |
| 33. Quay trucks have unique identifiers.                                                      | 39. Bays of container stowage areas have unique identifiers. |
| 34. Stack cranes have unique identifiers.                                                     | 40. Rows of a bay have unique identifiers.                   |
| 35. Bays (“Stacks”) of terminal container stowage areas have unique identifiers, cf. Item 39. | 41. Stacks of a row have unique identifiers.                 |
| 36. Land trucks have unique identifiers.                                                      | 42. The part unique identifier types are mutually disjoint.  |

#### type

- 31 VI
- 32 QCI
- 33 QTI
- 34 SCI
- 35 TBI
- 36 LTI
- 37 MCCI
- 38 CI
- 39 BI
- 40 RI
- 41 SI

#### axiom

- 42 VI, QCI, QTI, SCI, TBI, LTI, MCCI, CI, RI and SI mutually disjoint
- 42 TBI  $\subset$  BI

#### value

- 31 uid\_V: V  $\rightarrow$  VI
- 32 uid\_QC: QC  $\rightarrow$  QCI
- 33 uid\_QT: QT  $\rightarrow$  QTI
- 34 uid\_SC: SC  $\rightarrow$  SCI
- 34 uid\_TBI: BAY  $\rightarrow$  TBI
- 35 uid\_LT: LT  $\rightarrow$  LTI
- 37 uid\_MCC: MCC  $\rightarrow$  MCCI
- 37 uid\_CON: CON  $\rightarrow$  CI
- 34 uid\_BAY: BAY  $\rightarrow$  BI
- 35 uid\_ROW: ROW  $\rightarrow$  RI
- 36 uid\_STK: STK  $\rightarrow$  SI

#### K.5.3.1 Unique Identifiers: Distinctness of Parts

- 43. If two containers are different then their unique identifiers must be different.

#### axiom

- 43  $\forall \text{con}, \text{con}' : \text{CON} \cdot \text{con} \neq \text{con}' \Rightarrow \text{uid\_CON}(\text{con}) \neq \text{uid\_CON}(\text{con}')$

The same distinctness criterion applies to stacks, rows, bays, container storage areas, terminal ports, cranes, vessels, etc.

#### K.5.3.2 Unique Identifiers: Two Useful Abbreviations

Container positions within a container stowage area can be represented in two ways:

- 44. by a triple of a bay identifier, a row identifier and a stack identifier, and
- 45. by these three elements and a tier position (i.e., position within a stack).

$$44 \quad \text{BRS} = \text{BI} \times \text{RI} \times \text{SI}$$

$$45 \quad \text{BRSP} = \text{BI} \times \text{RI} \times \text{SI} \times \mathbf{Nat}$$

#### axiom

- 45  $\forall (\text{bu}, \text{ri}, \text{si}, \text{n}) : \text{BRSP} \cdot \text{n} > 0$



**K.5.3.3 Unique Identifiers: Some Useful Index Set Selection Functions**

- 46. From a container stowage area once can observe all bay identifiers.
- 47. From a bay once can observe all row identifiers.
- 48. From a row once can observe all stack identifiers.
- 49. From a virtual container storage area, i.e., an *icsa:iCSA*, one can extract all the unique container identifiers.

**value**

```

46 xtr_BIs: CSA → BI-set
46 xtr_BIs(csa) ≡ {uid_BAY(bay)|bay:BAY•bay ∈ xtr_BAYs(csa)}

46 xtr_RIs: BAY → RI-set
47 xtr_RIs(bay) ≡ {uid_ROW(row)|row:ROW•row ∈ obs_ROWs(bay)}

46 xtr_SIs: ROW → SI-set
48 xtr_SIs(row) ≡ {uid_STK(stk)|stk:STK•stk ∈ obs_STKs(row)}

49 xtr_CIs: iCSA → CI-set
49 xtr_CIs(icsa) ≡
49 ... [to come] ...

```

**K.5.3.4 Unique Identifiers: Ordering of Bays, Rows and Stacks**

The bays of a container stowage area are usually ordered. So are the rows of bays, and stacks of rows. Ordering is here treated as *attributes* of container stowage areas, bays and stacks. We shall treat *attributes* further on.

**K.5.4 States, Global Values and Constraints****K.5.4.1 States**

- 50. We postulate a container line industry *cli:CLI*.

From that we observe, successively, all parts:

- 51. the set, *cs:C-set*, of all containers;
- 52. the set, *tps:TPs*, of all terminal ports;
- 53. the set, *vs:Vs*, of all vessels; and
- 54. the set, *lts:LTS*, of all land trucks.

**value**

```

50 cli:CLI
51 cs:C-set = obs-Cs(obs_CS(cli))
52 tps:TP-set = obs_TPs(obs_TPS(cli))
53 vs:V-set = obs_Vs(obs_VS(cli))
54 lts:LTS = obs_LTS(obs_LTS(cli))

```

We can observe

- 55. *csas:CSA-set*, the set of all terminal port container stowage areas of all terminal ports;

- 56. *bays*:**BAY-set**, the terminal port bays of all terminals;
- 57. the set, *qcs*:**QC-set**, of all quay cranes of all terminals;
- 58. the set, *qts*:**QT-set**, of all quay trucks of all terminal ports; and
- 59. the set, *scs*:**SC-set**, of all terminal (i.e., stack) cranes of all terminal ports.

**value**

- 55 *csas*:**CSA-set** =  $\{\text{obs\_CSA}(tp)|tp:TP \bullet tp \in tps\}$
- 55 *bays*:**BAY-set** =  $\{\text{obs\_BAY}(csa)|csa:CSA \bullet csa \in csas\}$
- 57 *qcs*:**QC-set** =  $\{\text{obs\_QCs}(\text{obs\_QCS}(tp))|tp:TP \bullet tp \in tps\}$
- 58 *qts*:**QT-set** =  $\{\text{obs\_QTS}(\text{obs\_QTS}(tp))|tp:TP \bullet tp \in tps\}$
- 59 *scs*:**SC-set** =  $\{\text{obs\_SCs}(\text{obs\_SCS}(tp))|tp:TP \bullet tp \in tps\}$

**K.5.4.2 Unique Identifiers**

Given the generic parts outlined in Sect. K.5.4.1 we can similarly define generic sets of unique identifiers.

- 60. There is the set, *c\_uis*, of all container identifiers;
- 61. the set, *tp\_uis*, of all terminal port identifiers;
- 62. the set, *mcc\_uis*, of all terminal port command center identifiers;
- 63. the set, *v\_uis*, of all vessel identifiers;
- 64. the set, *qc\_uis*, of quay crane identifiers of all terminal ports;
- 65. the set, *qt\_uis*, of quay truck identifiers of all terminal ports;
- 66. the set, *sc\_uis*, of stack crane identifiers of all terminal ports;
- 67. the set, *stk\_uis*, of stack identifiers of all terminal ports;
- 68. the set, *lt\_uis*, of all land truck identifiers; and
- 69. the set, *uis*, of all vessel, crane and truck identifiers.

**value**

- 60 *c\_uis*:**CI-set** =  $\{\text{uid\_C}(c)|c:C \bullet c \in cs\}$
- 61 *tp\_uis*:**TPI-set** =  $\{\text{uid\_TP}(tp)|tp:TP \bullet tp \in tps\}$
- 62 *mcc\_uis*:**TPI-set** =  $\{\text{uid\_MCC}(\text{obs\_MCC}(tp))|tp:TP \bullet tp \in tps\}$
- 63 *v\_uis*:**VI-set** =  $\{\text{uid\_V}(v)|v:V \bullet v \in vs\}$
- 64 *qc\_uis*:**QCI-set** =  $\{\text{uid\_QC}(qc)|qc:QC \bullet qc \in qcs\}$
- 65 *qt\_uis*:**QTI-set** =  $\{\text{uid\_QT}(qt)|qt:QT \bullet qt \in qts\}$
- 66 *sc\_uis*:**SCI-set** =  $\{\text{uid\_SC}(sc)|sc:SC \bullet sc \in scs\}$
- 67 *stk\_uis*:**BI-set** =  $\{\text{uid\_BAY}(stk)|stk:BAY \bullet stk \in stks\}$
- 68 *lt\_uis*:**LTI-set** =  $\{\text{uid\_LL}(lt)|lt:LT \bullet lt \in lts\}$
- 69 *uis*:**(VI|QCI|QTI|SCI|BI|LTI)-set** =  $v\_uis \cup qc\_uis \cup qt\_uis \cup sc\_uis \cup stk\_uis \cup lt\_uis$

- 70. the map, *tpmcc\_idm*, from terminal port identifiers into the identifiers of respective command centers;
- 71. the map, *mccqc\_idsm*, from command center identifiers into the set of quay crane identifiers of respective ports;

72. the map,  $mccqt\_idsm$ , from command center identifiers into the identifiers of quay trucks of respective ports;
73. the map,  $mccsc\_idsm$ , from command center identifiers into the identifiers of quay trucks of respective ports; and
74. the map,  $mccbays\_idsm$ , from command center identifiers into the set of bay identifiers (i.e., “stacks”) of respective ports;

**value**

```

70 $tpmcc_idm:(TI \xrightarrow{\overline{m}} MCCI) = [uid_TP(tp) \mapsto uid_MCC(obs_MCC(tp)) | tp:TP \bullet tp \in tps]$
71 $mccqc_idsm:(MCCI \xrightarrow{\overline{m}} QCI\text{-set})$
71 = [$tpmcc_uim(uid_TP(tp)) \mapsto \{ uid_QC(qc)$
71 | $qc:QC \bullet qc \in obs_QCs(obs_QCS(tp)) \}$ | $tp:TP \bullet tp \in tps$]
72 $mccqt_idsm:(MCCI \xrightarrow{\overline{m}} QTI\text{-set}) =$
72 = [$tpmcc_uim(uid_TP(tp)) \mapsto \{ uid_QT(qt)$
72 | $qt:QT \bullet qt \in obs_QTs(obs_QTS(tp)) \}$ | $tp:TP \bullet tp \in tps$]
73 $mccsc_idsm:(MCCI \xrightarrow{\overline{m}} SCI\text{-set})$
73 = [$tpmcc_uim(uid_TP(tp)) \mapsto \{ uid_SC(sc)$
73 | $sc:SC \bullet sc \in obs_SCs(obs_SCS(tp)) \}$ | $tp:TP \bullet tp \in tps$]
74 $mccbays_idsm:(MCCI \xrightarrow{\overline{m}} BI\text{-set})$
74 = [$tpmcc_uim(uid_TP(tp)) \mapsto \{ uid_B(b)$
74 | $b:BAY \bullet b \in obs_BAYs(obs_BAYS(obs_CSA(tp))) \}$ | $tp:TP \bullet tp \in tps$]

```

**K.5.4.3 Some Axioms on Uniqueness**

TO BE WRITTEN

**K.5.5 Mereology**

We refer to [56, Sect. 5.2].

**K.5.5.1 Physical versus Conceptual Mereology**

We briefly discuss a distinction that was not made in [56]: whether to base a mereology on *physical connections* or on *functional* or, as we shall call it, *conceptual relations*. We shall, for this domain model, choose the conceptual view. The physical mereology view can be motivated, i.e. justified, from the figures on pages 316 and 317. The conceptual view is chosen on the basis of the justification of the terminal command centers, cf. Sect. K.5.2 on page 321. We shall model physical mereology as attributes.<sup>5</sup>

**K.5.5.2 Vessels****K.5.5.2.1 Physical Mereology:**

75. Vessels are physically “connectable” to quay cranes of any terminal port.

**type**

```
75 Phys_V_Mer = QCI-set
```

**value**

```
75 attr_Phys_V_Mer: V → Phys_V_mer
```

<sup>5</sup>Editorial note: Names of physical and of conceptual mereologies have to be “streamlined”. As now, they are a “mess”!

**K.5.5.2.2 Conceptual Mereology:**

76. Container vessels can potentially visit any container terminal port, hence have as [part of] their mereology, a set of terminal port command center identifiers.

**type**

76  $V\_Mer = MCCI\_set$

**value**

76  $mereo\_V: V \rightarrow V\_Mer$

**axiom**

76  $\forall v:V \bullet v \in vs \Rightarrow mereo\_V(v) \subseteq mcci\_uis$

**K.5.5.3 Quay Cranes**

**K.5.5.3.1 Physical Mereology:** In modelling the physical mereology, though as an attribute, of quay cranes, we need the notion of quay positions.

77. Quay cranes are, at any time, positioned at one or more adjacent quay positions of an identified segment of such.

**type**

77  $Phys\_QC\_Mereo = QPSid \times QP^*$

**value**

77  $attr\_Phys\_QC: QC \rightarrow Phys\_QC\_Mereo$

78. The quay positions,  $qcmereo = (qpsid, qpl): QC\_Mereo$ , must be proper quay positions of the terminal,

79. that is, the segment identifier,  $qpsid$ , must be one of the terminal,

80. and the list,  $qpl$ , must be contiguously contained within the so identifier segment.

**axiom**  $\forall tp:TP,$

78 **let**  $q = obs\_Q(tp), qcs = obs\_QCs(obs\_QCS(tp))$  **in**

79  $\forall q:Q \bullet q \in qcs \Rightarrow$

79 **let**  $(qpsid, qpl) = obs\_Mereo(q), qps = attr\_QPSs(q)$  **in**

79  $qpsid \in \mathbf{dom} \ qps$

80  $\wedge \exists i, j: \mathbf{Nat} \bullet \{i, j\} \in \mathbf{inds} \ qpl \wedge \langle (qps(qpsi))[k] \mid i \leq k \leq j \rangle = qpl$

78 **end end**

**K.5.5.3.2 Conceptual Mereology:** The conceptual mereology is simpler.

81. Quay cranes are conceptually related to the command center of the terminal in which they are located.

**type**

81  $QC\_Mer = MCCI$

**value**

81  $mereo\_QC: QC \rightarrow QC\_Mer$

**K.5.5.4 Quay Trucks****K.5.5.4.1 Physical Mereology:**

82. Quay trucks are physically “connectable” to quay and stack cranes.

**type**

82 Phys\_QT\_Mer = QCI-set  $\times$  QCI-set

**value**

82 attr\_Phys\_QT\_Mer: QT  $\rightarrow$  Phys\_QT\_Mer

**K.5.5.4.2 Conceptual Mereology:**

83. Quay trucks are conceptually connected to the command center of the terminal port of which they are a part.

**type**

83 QT\_Mer = MCCI

**value**

83 mereo\_QT: QT  $\rightarrow$  QT\_Mer

**K.5.5.5 Stack Cranes****K.5.5.5.1 Physical Mereology:**

84. Terminal stack cranes are positioned to serve one or more terminal area bays, one or more quay trucks and one or more land trucks.

85. The terminal stack crane positions are indeed positions of their terminal

86. and no two of them share bays.

**type**

84 Phys\_SCmereo = s\_bis:BI-set  $\times$  s\_qtis:QTI-set  $\times$  s\_ltis:LTI-set

**axiom**

84  $\forall (bis,qtis,ltis):Phys\_SCmereo \bullet bis \neq \{\} \wedge qtis \neq \{\} \wedge ltis \neq \{\}$

**value**

84 Phys\_SCmereo: SC  $\rightarrow$  Phys\_SCmereo

**axiom**

84  $\forall tp:TP \bullet$

84 **let** csa=obs\_CSA(tp), bays=obs\_BAYS(obs\_BAYS(csa)), scs=obs\_SCs(obs\_SCS(tp)) **in**

85  $\forall sc:SC \bullet sc \in scs \Rightarrow Phys\_SCmereo(sc) \subseteq xtr\_BIs(csa)$

86  $\wedge \forall tp',tp'':TP \bullet \{tc',tc''\} \subseteq tcs \wedge tc' \neq tc''$

86  $\Rightarrow s\_bis(Phys\_SCmereo(tc')) \cap s\_bis(Phys\_SCmereo(tc'')) = \{\}$  **end**

**K.5.5.5.2 Conceptual Mereology:** The conceptual stack crane mereology is simple:

87. Each stack is conceptually related to the command center of the terminal at which it is located.

**type**

87 SC\_Mer = MCCI

**value**

87 mereo\_SC: SC  $\rightarrow$  SC\_Mer

### K.5.5.6 Container Stowage Areas

**K.5.5.6.1 Bays, Rows and Stacks:** The following are some comments related to, but not defining a mereology for container stowage areas.

88. A bay of a container stowage area
- (a) has either a predecessor
  - (b) or a successor,
  - (c) or both (and then distinct).
  - (d) No row cannot have neither a predecessor nor a successor.
89. A row of a bay has a predecessor and a successor, the first stack has no predecessor and the last stack has no successor.
90. A stack of a row has a predecessor and a successor, the first stack has no predecessor, and the last stack has no successor.

#### value

- 88 BAY\_Mer: BAY  $\rightarrow$  ( $\{\text{'nil'}\}$ |BI)  $\times$  (BI| $\{\text{'nil'}\}$ )  
 89 ROW\_Mer: ROW  $\rightarrow$  ( $\{\text{'nil'}\}$ |RI)  $\times$  (RI| $\{\text{'nil'}\}$ )  
 90 STK\_Mer: STK  $\rightarrow$  ( $\{\text{'nil'}\}$ |SI)  $\times$  (SI| $\{\text{'nil'}\}$ )

#### axiom

- 88  $\forall$  csa:CSA • **let** bs = obs\_BAYs(obs\_BAYS(csa)) **in**  
 88  $\forall$  b:BAY •  $b \in$  bs  $\Rightarrow$   
 88 **let** (nb,nb') = mereo\_BAY(b) **in**  
 88 **case** (nb,nb') **of**  
 88a ('nil',bi)  $\rightarrow$  bi  $\in$  xtr\_BIs(csa),  
 88b (bi,'nil')  $\rightarrow$  bi  $\in$  xtr\_BIs(csa),  
 88d ('nil','nil')  $\rightarrow$  **chaos**,  
 88c (bi,bi')  $\rightarrow$  {bi,bi'}  $\subseteq$  xtr\_BIs(csa)  $\wedge$  bi  $\neq$  bi'  
 88 **end end end**  
 89 as for rows  
 90 as for stacks

### K.5.5.7 Bay Mereology

#### K.5.5.7.1 Physical Vessel Bay Mereology:

91. A vessel bay is topologically related to the vessel on board of which it is placed and to the set of all quay cranes of all terminal ports.

#### type

- 91 Phys\_VES\_BAY\_Mer = VI  $\times$  QCI-set

#### K.5.5.7.2 Conceptual Vessel Bay Mereology:

92. A vessel bay is conceptually related to the set of all command centers of all terminal ports.

#### type

- 92 V\_BAY\_Mer = MCCI-set

**K.5.5.7.3 Physical Terminal Port Bay (cum Stack) Mereology:**

93. A terminal bay (cum stack) is topologically related to the stack cranes of a given terminal port and all land trucks.

**type**

93 Phys\_STK\_Mer = SCI-set  $\times$  LTI-set

**K.5.5.7.4 Conceptual Terminal Port Bay (cum Stack) Mereology:**

94. A terminal port bay is conceptually related to the command center of its port.

**type**

94 T\_BAY\_Mer = MCCI

**K.5.5.8 Land Trucks****K.5.5.8.1 Physical Mereology:**

95. Land trucks are physically “connectable” to stack cranes – of any port.

**type**

95 Phys\_LT\_Mer = SCI-set

**value**

95 attr\_Phys\_LT\_Mer: LT  $\rightarrow$  Phys\_LT\_Mer

**K.5.5.8.2 Conceptual Mereology:**

96. Land trucks are conceptually connected to the command centers of any terminal port.

**type**

96 LT\_Mer = MCCI-set

**value**

96 mereo\_LT: LT  $\rightarrow$  LT\_Mer

**K.5.5.9 Command Center**

Command centers are basically conceptual quantities. Hence we can expect the physical mereology to be the conceptual mereology.

97. Command centers are physically and conceptually connected to all vessels, all cranes of the terminal port of the command center, all quay trucks of the terminal port of the command center, all stacks (i.e., bays) of the terminal port of the command center, and all land trucks, and all containers.

**type**

97 MCC\_Mer = VI-set  $\times$  QCI-set  $\times$  QTI-set  $\times$  SCI-set  $\times$  BI-set  $\times$  LTI-set  $\times$  CI-set

**value**

97 mereo\_MCC: MCC  $\rightarrow$  MCC\_Mer

**axiom**

97  $\forall$  tp:TP  $\bullet$  tp  $\in$  tps  $\bullet$

97 **let** qcs:QC-set  $\bullet$  qcs = obs\_QCs(obs\_QCS(tp)),

97 **qts**:QT-set  $\bullet$  qts = obs\_QTs(obs\_QTS(tp)),

```

97 scs:SC-set • scs = obs_SCs(obs_SCS(tp)),
97 bs:iBAY-set • bs = obs_Bs(obs_BS(obs_CSA(tp))) in
97 let vis:VI-set • vis = {uid_VI(v)|v:V•v ∈ vs},
97 qcis:QCI-set • qcis = {uid_QCI(qc)|qc:QC•qc ∈ qcs},
97 qtis:QTI-set • qcis = {uid_QTI(qc)|qt:QT•qt ∈ qts},
97 scis:SCI-set • scis = {uid_SCI(sc)|sc:SC•sc ∈ scs},
97 bis:iBAY-set • bis = {uid_BI(b)|b:iBAY•b ∈ bs},
97 ltis:LTI-set • ltis = {uid_LTI(lt)|lt:LT•lt ∈ lts},
97 cis:SCI-set • cis = {uid_CI(c)|c:C•c ∈ cs} in
97 mereo_MCC(obs_MCC(tp)) = (vis,qcis,scis,sis,bis,ltis,cis) end end

```

### K.5.5.10 Conceptual Mereology of Containers

The physical mereology of any container is modelled as a container attribute.

98. The conceptual mereology is modelled by containers being connected to all terminal command centers.

#### type

```
98 C_Mer = MCCI-set
```

#### value

```
98 mereo_C: C → C_Mer
```

#### axiom

```
98 ∀ c:C • mereo_C(c) = mcc_wis
```

## K.5.6 Attributes

We refer to [56, Sect. 5.3].

### K.5.6.1 States

By a state we shall mean one or more parts such that these parts have *dynamic* attributes, in our case typically *programmable* attributes.

### K.5.6.2 Actions

Actions apply to states and yield possibly updated states and, usually, some result values.

We shall in this section, Sect. K.5.6, on attributes, outline a number of *simple* (usually called *primitive*) actions of states. These actions are invoked by some behaviours either at their own volition, or in response to events occurring in other behaviours. The action outcomes are simple enough, but calculations resulting in these outcomes are not. Together the totality of the actions performed by the terminal's monitoring & control of vessels, cranes, trucks and the container stowage area, reflect the complexity of stowage handling.

### K.5.6.3 Attributes: Quays

99. Quays are segmented into one or more quay segments, qs:QS, each with a sequence of one or more crane positions, cp:CP.
100. Quay segments and
101. crane positions are further unspecified.



**type**

```

99 QPOS = QS × CP* axiom $\forall (_,cpl):QPOS \bullet cpl \neq \langle \rangle$
100 QS
101 CP

```

**K.5.6.4 Attributes: Vessels**

102. A vessel is

- (a) either at sea, at some *programmable* geographical location (longitude and latitude),
- (b) or in some *programmable* terminal port – designated by the identifier of its command center and its quay position.

103. We consider the “remainder” of the vessel state as a programmable attribute – which we do not further define. The remainder includes all information about all containers, their bay/row/stack/tier positions, their bill-of-ladings, etc.

104. There may be other vessel attributes.

**type**

```

102 V_Pos == AtSea | InPort
102a Longitude, Latitude
102a AtSea :: Longitude × Latitude
102b InPort :: MCCI × QPOS
103 VΣ
104 ...

```

**value**

```

102 attr_V_Pos: V → V_Pos
104 attr_VΣ: V → VΣ
104 attr_...: V → ...

```

**axiom**

```

102b $\forall \text{mkInPort}(ti):\text{InPort} \bullet ti \in tp_uis$

```

**K.5.6.5 Attributes: Quay Cranes**

105. At any one time a quay crane may *programmably* hold a container or may not. We model the container held by a crane by the container identifier.

106. At any one time a quay crane is *programmably* positioned in a quay position within a quay segment.

107. Quay cranes may have other attributes.

**type**

```

105 QCHold == mkNil('nil') | mkCon(ci:CI)
106 QCPos = QSId × QP
107 ...

```

**value**

```

105 attr_QCHold: QC → QCHold
106 attr_QCPos: QC → QCPos
107 ...

```

**K.5.6.6 Attributes: Quay Trucks**

108. At any one time a land truck may *programmably* hold a container or may not. We model the container held by a quay truck by the container identifier.

109. Quay trucks may have other attributes.

Note that we do not here model the position of quay trucks.

**type**

108 QTHold == mkNil('nil') | mkCon(ci:CI)

109 ...

**value**

108 attr\_QTHold: QT → QTHold

109 ...

**K.5.6.7 Attributes: Terminal Stack Cranes**

110. At any one time a stack crane may *programmably* hold a container or may not. We model the container held by a crane by the container identifier.

111. Stack cranes are *programmably* positioned at a terminal bay.

112. Stack cranes may have other attributes.

**type**

110 SCHold == mkNil('nil') | mkCon(ci:CI)

111 SCPos = BI

111 ...

**value**

110 attr\_SCHold: SC → SCHold

111 attr\_SCPos: SC → SCPos

112 ...

**K.5.6.8 Attributes: Container Stowage Areas**

113. Bays of container storage areas *statically* have total order.

114. Rows of bays *statically* have total order.

115. Stacks of rows *statically* have total order.

We abstract orderings in two ways.

**type**

113 BOm = BI  $\xrightarrow{m}$  Nat, BOI = BI\*

114 ROm = RI  $\xrightarrow{m}$  Nat, ROI = RI\*

115 SOm = SI  $\xrightarrow{m}$  Nat, SOI = SI\*

**axiom**

113  $\forall$  bom:BOm•rng bom={1:card dom bom},  $\forall$  bol:BOI•inds bol={1:len bol}

114  $\forall$  rom:ROm•rng rom={1:card dom rom},  $\forall$  rol:ROI•inds rol={1:len rol}

115  $\forall$  som:SOm•rng som={1:card dom som},  $\forall$  sol:SOI•inds sol={1:len sol}

**value**

113 attr\_BOm: CSA → BOm, attr\_BOI: CSA → BOI

114 attr\_ROm: BAY → ROm, attr\_ROI: BAY → ROI

115 attr\_SOm: ROW → SOm, attr\_SOI: ROW → SOI

CSAs, BAYs, ROWs and STKs have (presently further) *static* descriptions<sup>6</sup> and terminal and vessel container stowage areas have definite numbers

- 116. of bays,
- 117. and any one such bay a definite number of rows,
- 118. and any one such row a definite number of stacks,
- 119. and any one such stack a maximum loading of containers.

**type**

- 116 CASd
- 117 BAYd
- 118 ROWd
- 119 STKd

**value**

- 116 attr\_CSAD: CSA  $\rightarrow$  BI  $\xrightarrow{m}$  CSAd
- 117 attr\_BAYD: BAY  $\rightarrow$  RI  $\xrightarrow{m}$  BAYd
- 118 attr\_ROWd: ROW  $\rightarrow$  SI  $\xrightarrow{m}$  ROWd
- 119 attr\_STKD: STK  $\rightarrow$  (**Nat**  $\times$  STKd)

**K.5.6.9 Attributes: Land Trucks**

- 120. At any one time a land truck may *programmably* hold a container or may not. We model the container held by a land truck by the container identifier.
- 121. Land trucks also possess a further undefined *programmable* land truck state.
- 122. Land trucks may have other attributes.

Note that we do not here model the position of land trucks.

**type**

- 120 LTHold == mkNil('nil') | mkCon(ci:CI)
- 121 LT $\Sigma$
- 122 ...

**value**

- 120 attr\_LTHold: LT  $\rightarrow$  LTHold
- 121 attr\_LT $\Sigma$ : LT  $\rightarrow$  LT $\Sigma$
- 122 ...

**K.5.6.10 Attributes: Command Center**

- 123. The *syntactic description*<sup>7</sup> of the spatial positions of quays, cranes and the container storage area of a terminal, `TopLogDescr`, is a *static* attribute.
- 124. The *syntactic description*<sup>8</sup> of the terminal state, i.e., the actual positions and deployment of vessels at quays, quay and stack cranes, quay and land trucks, and the actual container “contents” of these, `Term $\Sigma$ Descr`, is a *programmable* attribute.

<sup>6</sup>Such descriptions include descriptions of for what kind of containers a container stowage area, a bay, a row and a stack is suitable: flammable, explosives, etc.

<sup>7</sup>A *syntactic description* describes something, i.e., has some *semantics*, from which it is, of course, different.

<sup>8</sup>The *syntactic description* of the terminal state is, of course, not that state, but only its description. The terminal state is the combined states of all cranes, trucks and the container storage area.

**type**

123 TopLogDescr

124 MCCΣDescr

**value**

123 attr\_TopLogDescr: MCC → TopLogDescr

124 attr\_TermΣDescr: MCC → TermΣDescr

**K.5.6.11 Attributes: Containers**125. A Bill-of-Lading<sup>9</sup> is a *static* container attribute.<sup>10</sup>**type**

125 BoL

**value**

125 attr\_BoL: C → BoL

126. At any one time a container is positioned either

- (a) in a stack on a vessel: at sea or in a terminal, or
- (b) on a quay crane in a terminal port, being either unloaded from or loaded onto a vessel, or
- (c) on a quay truck to or from a quay crane, i.e., from or to a stack crane, in a terminal port, or
- (d) on a stack crane in a terminal port, being either unloaded from a quay truck onto a terminal stack or loaded from a terminal stack onto a quay truck, or
- (e) on a stack in a terminal port, or
- (f) on a land truck, or
- (g) idle.

A container position is a *programmable* attribute.

127. There are other container attributes. For convenience we introduce an aggregate attribute: CAttrs for all attributes.

**type**

126 CPos == onV | onQC | onQT | onSC | onStk | onLT | Idle

126a onV :: VI × BRSP × VPos

126a VPos == AtSea | InTer

126a AtSea :: Geo

126a InTer :: QPSid × QP<sup>+</sup>

<sup>9</sup>[https://en.wikipedia.org/wiki/Bill\\_of\\_lading](https://en.wikipedia.org/wiki/Bill_of_lading): A bill of lading (sometimes abbreviated as B/L or BoL) is a document issued by a carrier (or their agent) to acknowledge receipt of cargo for shipment. In British English, the term relates to ship transport only, and in American English, to any type of transportation of goods. A bill of Lading must be transferable, and serves three main functions: it is a conclusive receipt, i.e. an acknowledgment that the goods have been loaded; and it contains or evidences the terms of the contract of carriage; and it serves as a document of title to the goods, subject to the nemo dat rule. Bills of lading are one of three crucial documents used in international trade to ensure that exporters receive payment and importers receive the merchandise. The other two documents are a policy of insurance and an invoice. Whereas a bill of lading is negotiable, both a policy and an invoice are assignable. In international trade outside of the USA, Bills of lading are distinct from waybills in that they are not negotiable and do not confer title. The **nemo dat rule**: that states that the purchase of a possession from someone who has no ownership right to it also denies the purchaser any ownership title.

<sup>10</sup>For waybills see <https://en.wikipedia.org/wiki/Waybill>: A waybill (UIC) is a document issued by a carrier giving details and instructions relating to the shipment of a consignment of goods. Typically it will show the names of the consignor and consignee, the point of origin of the consignment, its destination, and route. Most freight forwarders and trucking companies use an in-house waybill called a house bill. These typically contain "conditions of contract of carriage" terms on the back of the form. These terms cover limits to liability and other terms and conditions

```

126b onQC :: MCCI × QCI
126c onQT :: MCCI × QTI
126d onSC :: MCCI × SCI
126e onStk :: MCCI × BRSP
126f onLT :: MCCI × LTI
126g Idle :: {"idle"}
127 CAttrs
value
126 attr_CPos: C → CPos
127 attr_CAttrs: C → CAttrs

```

## K.6 Perdurants

We refer to [56, Sect. 7].

### K.6.1 A Modelling Decision

In the *transcendental interpretation* of parts into behaviours we make the following modelling decisions: All atomic and all composite parts become separate behaviours. But there is a twist. Vessels and terminal stacks are now treated as “atomic” behaviours. Containers that up till now were parts of container stowage areas on vessels and in terminal stacks are not behaviours embedded in the behaviours of vessels and terminal stacks, but are “factored” out as separate, atomic behaviours.

This modelling decision entails that container stowage areas, CSAs, of vessels and terminal stacks are modelled by replacing the [physical] containers of these CSAs with *virtual container stowage areas*, *vir\_CSAs*. Where there “before” were containers there are now, instead, descriptions of these: their unique identifiers, their mereology, and their attributes.

### K.6.2 Virtual Container Storage Areas

In our transition from endurants to perdurants we shall thus need a notion of container stowage areas which, for want of a better word, we shall call *virtual CSAs*. Instead of stacks embodying containers, they embody

128. container information: their unique identifier, mereology and attributes.

We must secure that no container is referenced more than once across the revised-model;

129. that is, that all *ci:Cls* are distinct.

#### type

```

5' vir_CSA
11' vir_BAY_s = vir_BAY-set, vir_BAY
12' vir_ROW_s = vir_ROW-set, vir_ROW = vir_STK-set
13' vir_STK = vir_STK-set, vir_STK
14' vir_STK = CInfo*
128 CInfo = CI × CMereo × CAttrs

```

#### value

```

5' attr_vir_CSA: TP → vir_CSA
11' attr_vir_BAY_s: vir_CSA → vir_BAY_s, vir_BAY_s = vir_BAY-set, vir_BAY
11' uid_vir_BAY: vir_BAY → BI
12' attr_vir_ROW_s: vir_BAY → vir_ROW_s

```

#### axiom

129 [all *CI*s of all *vir\_CSAs* are distinct]

### K.6.3 Changes to The Parts Model

We revise the parts model of earlier:

#### type

2 STPs, TPs = TP-set, TP

3 SVs, Vs = V-set, V

#### value

2 obs\_STPs: CLI  $\rightarrow$  STPs, obs\_TPs: STPs  $\rightarrow$  TPs

3 obs\_SVs: CLI  $\rightarrow$  SVs, obs\_Vs: SVs  $\rightarrow$  Vs

We treat the former CSAs of terminal ports as a composite, concrete part, `vir_BAY_m` consisting of a set of atomic virtual bays, `vir_BAY`.

#### type

11' `vir_BAY_s` = `vir_BAY-set`, `vir_BAY`

#### value

5 `obs_BAY_s`: TP  $\rightarrow$  `vir_BAY_s`

5 `uid_BAY`: `vir_BAY`  $\rightarrow$  BI

And we treat the former CSAs of vessels as a programmable attribute of vessels:

`attr_vir_CSA`: V  $\rightarrow$  `vir_CSA`

### K.6.4 Basic Model Parts

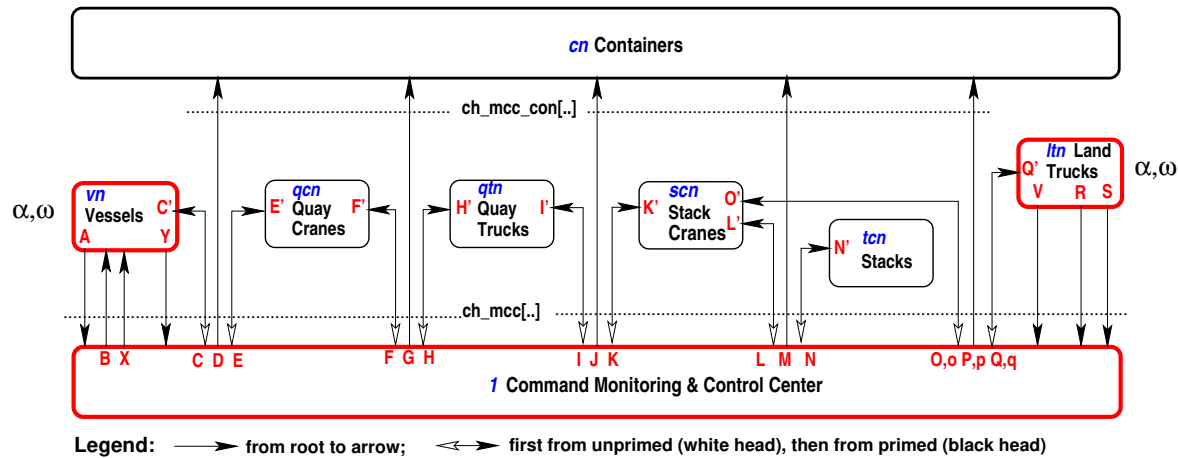


Fig. 3: The Container Terminal Behaviours<sup>11</sup>

There are  $c_n$  container behaviours, where  $c_n$  is the number of all containers of the system we are modelling. For each terminal port there is 1 controller behaviour,  $v_n$  vessel behaviours, where  $v_n$  is the number of vessels visiting that terminal port,  $qcn$  quay crane behaviours, where  $qcn$  is the number of quay cranes of that terminal port,  $qtn$  quay truck behaviours, where  $qtn$  is the number of quay trucks of that terminal port,  $lt_n$  land truck behaviours, where  $lt_n$  is the number of land trucks (of that terminal port), and  $ts_n$  terminal stack behaviours, where  $ts_n$  is the number of terminal bays of that terminal port.

The vessel, the land truck and the terminal monitoring & control [command] center behaviours are *pro-active*: At their own initiative (volition), they may decide to communicate with other behaviours. The crane, quay truck, stack and container behaviours are *passive*: They respond to interactions with other behaviours.

<sup>11</sup>The labeling **A, B, C, D, ..., X, Y** may seem arbitrary, but isn't!

## K.6.5 Actions, Events, Channels and Behaviours

We refer to [56, Sect. 7.1].

In building up to the behavioral analysis & description of the terminal container domain we first analyse the actions and events of that domain. These actions and events are the building blocks of behaviours.

**Actions**, to remind the reader, are explicitly performed by an actor, i.e., a behaviour, calculates some values and, usually, effect a state change.

**Events** “*occur to*” actors (behaviours), that is, are not initiated by these, but usually effect state changes.

## K.6.6 Actions

We refer to [56, Sects. 7.1.5, 7.3.1].

The unloading of containers from and the loading of container onto container stowage areas are modelled by corresponding actions on virtual container stowage areas. Vessels, land trucks and terminal monitoring & control centers, i.e., command centers, are here modelled as the only entities that can *initiate* actions.

### K.6.6.1 Command Center Actions

**K.6.6.1.1 Motivating the Command Center Concept:** We refer to the **[A,B,...,U]** labeled arrows of the figure on Page 336.

Imagine a terminal port. It has several vessels berthed along quays. It also has quay space, i.e., positions, for more vessels to berth. Berthed vessels are being serviced by several, perhaps many quay cranes. The totality of quay cranes are being serviced by [many more] quay trucks. The many quay trucks service several terminal bays, i.e., *stacks*. Land trucks are arriving, attending *stacks* and leaving. Quite a “busy scene”. So is the case for all container terminal ports.

The concept of a *monitoring & control*, i.e., a **command center**, is an abstract one; the figure on Page 336 does not show a part with a ... **center** label. The *actions* of vessels and trucks, and the *events* of cranes, terminal stacks and trucks are either hap-hazard, no-one interferes, they somehow “just happen”, or they are somehow co-ordinated.

Whether “free-wheeling” or “more-or-less coordinated” we can think of a *command center* as somehow *monitoring and controlling actions and events*.

Terminal *monitoring & control centers*, also interchangeably referred to as *command centers*, are thus where the logistics of container handling takes place.

You may think of this command center as receiving notices from vessels and land trucks as to their arrival and with information about their containers; thus building up awareness, i.e., a state, of the containers of all incoming and arrived vessels and land trucks, the layout of the terminal and the state of its container stowage area, the current whereabouts of vessels, cranes and trucks. Quite a formidable “state”.

We shall therefore model the “comings” and “goings” of vessels, trucks, cranes and stacks as if they were monitored and controlled by a command center, In our modelling we are not assuming any form of efficiency; there is, as yet no notion of optimality, nor of freedom from mistakes and errors. Our modelling – along these lines – is “hidden” in action pre- and post-conditions and thus allows for any degree of internal non-determinism.

**K.6.6.1.2 Calculate Next Transaction:** The *core* action of the command center is `calc_nxt_transaction`. We shall define `calc_nxt_transaction` only by its signature and a pair of **pre/post** conditions. In this way we do not have to consider *efficiency, security, safety, etc.*, issues. These, i.e., the efficiency, security, safety, etc., issues can “always” be included in an *requirements engineering* implementation of `calc_nxt_transaction`. Basically the `calc_nxt_transaction` has to consider which of a non-trivially large number of possible actions have to be invoked. They are listed in

Items 131 to 137 below. The `calc_nxt_transaction` occurs in time, and occur repeatedly, endlessly, i.e., “ad-infinitum”, At any time that `calc_nxt_transaction` is invoked the monitoring and control command center (`mcc`) is in some state. That state changes as the result of both monitoring actions and control actions. The `calc_nxt_transaction` therefore non-deterministically-internally chooses one among several possible alternatives. If there is no alternative, then a **skip** action is performed.

The command center, `mcc`, models the following actions and events: **[A]** the update of the `mcc` state, `mccσ`, in response to the vessel action that inform the `mcc` of the vessel arrival.

130. The result of a `calc_nxt_transaction` is an transaction designator, `MCCTrans` and a state change. There are several alternative designators. We mention some:
131. **[B]**: the calculation of vessel positions for [their] arrivals;
132. **[CDE]**: the calculation of vessel to quay crane container transfers;
133. **[FGH]**: the calculation of quay crane to quay truck container transfers;
134. **[IJK]**: the calculation of quay truck to stack crane container transfers;
135. **[LMN]**: the calculation of stack crane to stack container transfers;
136. **[OPQ]**: the calculation of land truck to stack crane container transfers;
137. **[X]**: the calculation that stowage, for a given vessel, has completed; and
138. the calculation that there is no next transaction that can be commenced.
139. The signature of the `calc_nxt_transaction` involves the unique identifier, mereology, static and programmable attributes, i.e., the state of the command center, and indicates that a command center transaction results and a next state “entered”.
140. For this, the perhaps most significant action of the entire container terminal port operation, we “skirt” the definition and leave to a pair pf **pre/post** conditions that of characterising the result and next state.

#### type

```

130 MCCTrans == QuayPos | VSQC_Xfer | QCQT_Xfer | QTSC_Xfer
130 | SCSTK_Xfer | SCLT_Xfer | LT_Dept | VS_Dept | Skip
131 [B]: QuayPos :: VI × QPos
132 [CDE]: VSQC_Xfer :: VI × BRS × CI × QCI
133 [FGH]: QCQT_Xfer :: QCI × CI × QTl
134 [IJK]: QTSC_Xfer :: QTl × CI × SCI
135 [LMN]: SCSTK_Xfer :: SCI × CI × BRS
136 [OPQ]: SCLT_Xfer :: SCI × CI × LTI
137 [X]: VS_Dept :: VI
138 Skip :: nil

```

#### value

```

139 calc_nxt_transaction: MCCI × mereoMCC × statMCC → MCCΣ → MCCTrans × MCCΣ
139 calc_nxt_transaction(mcci, mcmereo, mmstat)(mccσ) as (mcctrans, mccσ')
140 pre: $\mathcal{P}_{calc_nxt_trans}((mcci, mcmereo, mccstat)(mccσ))$
140 post: $\mathcal{Q}_{calc_nxt_trans}((mcci, mcmereo, mccstat)(mccσ))(mcctrans, mccσ')$

```

The above mentioned actions are invoked by the command center in its endeavour to see containers moved from vessels to customers. A similar set of actions affording movement of containers customers to vessels, i.e., in the reverse direction: from land trucks to stack cranes, from stacks to quay trucks, from quay trucks to quay cranes, and from quay cranes to vessels, round off the full picture of all command center actions.



**K.6.6.1.3 Command Center Action [A]: update\_mcc\_from\_vessel:**

- 141. Command centers
- 142. upon receiving arrival information,  $v\_info$ , from arriving vessels,  $v\_i$ , can update their state “accordingly”.
- 143. We leave undefined the pre- and post-conditions.

**value**

- 141  $update\_mcc\_from\_vessel: VSMCC\_MSG \times MCC\_Σ \rightarrow MCC\_Σ$
- 142  $update\_mcc\_from\_vessel((vs\_i, vir\_csa, vs\_info), mcc\_σ) \text{ as } mcc\_σ'$
- 143 **pre:**  $P_{upd\_mcc\_f\_v}((vs\_i, vir\_csa, vs\_info), mcc\_σ)$
- 143 **post:**  $Q_{upd\_mcc\_f\_v}((vs\_i, vir\_csa, vs\_info), mcc\_σ)(mcc\_σ')$

**K.6.6.1.4 Command Center Action [B]: calc\_ves\_pos:**

- 144. Command centers
- 145. can calculate,  $q\_pos$ , the quay segment and quay positions for an arriving vessel,  $v\_i$ .
- 146. We leave undefined the pre- and post-conditions.

**value**

- 144  $calc\_ves\_pos: MCCI \times MCC\_mereo \times TopLog \times MCCΣ \times VI \rightarrow (QSIId \times QP^*) \times MCCΣ$
- 145  $calc\_ves\_pos(mcc\_i, mcc\_mereo, toplog, mcc\_σ, v\_i) \text{ as } (q\_pos, mcc\_σ')$
- 146 **pre:**  $P_{calc\_ves\_pos}(mcc\_i, mcc\_mereo, toplog, mcc\_σ, v\_i)$
- 146 **post:**  $Q_{calc\_ves\_pos}(mcc\_i, mcc\_mereo, toplog, mcc\_σ, v\_i)(q\_pos, mcc\_σ')$

**K.6.6.1.5 Command Center Action [C-D-E]: calc\_ves\_qc**

- 147. The command center non-deterministically internally calculates
- 148. a pair of a triplet: the bay-row-stack coordinates,  $brs$ , from which a top container, supposedly  $ci$ , is to be removed by quay crane  $qci$ , and a next command center state reflecting that calculation (and that the identified quay crane is being so alerted).
- 149. We leave undefined the relevant pre- and post-conditions

**value**

- 147  $calc\_ves\_qc: MCCΣ \rightarrow (BRS \times CI \times QCI) \times MCCΣ$
- 148  $calc\_ves\_qc(mccσ) \text{ as } ((brs, ci, qci), mccσ')$
- 149 **pre:**  $P_{calc\_ves\_qc}(mccσ)$
- 149 **post:**  $Q_{calc\_ves\_qc}(mccσ)((brs, ci, qci), mccσ')$

**K.6.6.1.6 Command Center Action [F-G-H]: calc\_qc qt**

- 150. The command center non-deterministically internally
- 151. calculates a pair of a triplet: the identities of the quay crane from which and the quay truck to which the quay crane is to transfer a container, and an update command center state reflecting that calculation (and that the identified quay crane, container and truck are being so alerted).
- 152. We leave undefined the relevant pre- and post-conditions

**value**

```

150 calc_qc_qt: $MCC\Sigma \rightarrow (QCI \times CI \times QTI) \times MCC\Sigma$
151 calc_qc_qt(mcc σ) as ((qci,ci,qti),mcc σ')
152 pre: $\mathcal{P}_{calc_qc_qt}(mcc\sigma)$
152 post: $\mathcal{Q}_{calc_qc_qt}(mcc\sigma)((qci,ci,qti),mcc\sigma')$

```

**K.6.6.1.7 Command Center Action [I-J-K]: calc\_qt\_sc**

- 153. The command center non-deterministically internally
- 154. calculates a pair of a triplet: the identities of a quay truck, a container, and a stack crane, and an update command center state reflecting that calculation (and that the identified quay truck, container and stack crane are being so alerted).
- 155. We leave undefined the relevant pre- and post-conditions

**value**

```

153 calc_qt_sc: $MCC\Sigma \rightarrow (QTI \times CI \times SCI) \times MCC\Sigma$
154 calc_qt_sc(mcc σ) as ((qti,ci,sci),mcc σ')
155 pre: $\mathcal{P}_{calc_qt_sc}(mcc\sigma)$
155 post: $\mathcal{Q}_{calc_qt_sc}(mcc\sigma)((qti,ci,sci),mcc\sigma')$

```

**K.6.6.1.8 Command Center Action [L-M-N]: calc\_sc\_stack**

- 156. The command center non-deterministically internally calculates a pair:
- 157. a triplet of the identities of a stack crane, a container and a terminal bay/row/stack triplet and a new state that reflects this action.
- 158. We leave undefined the relevant pre- and post-conditions

**value**

```

156 calc_sc_stack: $MCC\Sigma \rightarrow (SCI \times CI \times BRS) \times MCC\Sigma$
157 calc_sc_stack(mcc σ) as ((sci,ci,brs),mcc σ')
158 pre: $\mathcal{P}_{calc_sc_stack}(mcc\sigma)$
158 post: $\mathcal{Q}_{calc_sc_stack}(mcc\sigma)((sci,ci,brs),mcc\sigma')$

```

**K.6.6.1.9 Command Center Action [N-M-L]: calc\_stack\_sc**

- 159. The command center non-deterministically internally calculates a pair:
- 160. a triplet of a terminal bay/row/stack triplet and the identities of a container and a stack crane, and a new state that reflects this action.
- 161. We leave undefined the relevant pre- and post-conditions

**value**

```

159 calc_stack_sc: $MCC\Sigma \rightarrow (BRS \times CI \times SCI) \times MCC\Sigma$
160 calc_stack_sc(mcc σ) as ((brs,ci,sci),mcc σ')
161 pre: $\mathcal{P}_{calc_stack_sc}(mcc\sigma)$
161 post: $\mathcal{Q}_{calc_stack_sc}(mcc\sigma)((brs,ci,sci),mcc\sigma')$

```

**K.6.6.1.10 Command Center Action [O-P-Q]: `calc_sc_lt`**

162. The command center non-deterministically internally calculates a pair:
163. a triplet of the identities of a stack crane, a container and a land truck, and a new state that reflects this action.
164. We leave undefined the relevant pre- and post-conditions.

**value**

```

162 calc_sc_lt: MCCΣ → (BRS×CI×SCI)×MCCΣ
163 calc_sc_lt(mccσ) as ((sci,ci,lti),mccσ')
164 pre: Pcalc_sc_lt(mccσ)
164 post: Qcalc_sc_lt(mccσ)((sci,ci,lti),mccσ')
```

**K.6.6.1.11 Command Center Action [Q-P-O]: `calc_lt_sc`**

165. The command center non-deterministically internally calculates a pair:
166. a triplet of the identities of a land truck, a container and a stack crane, and a new state that reflects this action.
167. We leave undefined the relevant pre- and post-conditions.

**value**

```

165 calc_lt_sc: MCCΣ → (BRS×CI×SCI)×MCCΣ
166 calc_lt_sc(mccσ) as ((lti,ci,sci),mccσ')
167 pre: Pcalc_lt_sc(mccσ)
167 post: Qcalc_lt_sc(mccσ)((lti,ci,sci),mccσ')
```

**K.6.6.1.12 Command Center: Further Observations** Please observe the following: any terminal command center repeatedly and non-deterministically alternates between any and all of these actions. Observe further that: The intention of the pre- and post-conditions [Items 143, 146, 149, 152, 155, 158, 161, 167, and 164], express requirements to the command center states,  $mccσ:mccΣ$ , w.r.t. the information it must handle. Quite a complex state.

**K.6.6.2 Container Storage Area Actions**

We define two operations on virtual CSAs:

168. one of stacking (loading) a container, referred to by its unique identifier in a virtual CSA,
169. and one of unstacking (unloading) a container;
170. both operations involving bay/row/stack references.

**type**

```
170 BRS = BI × RI × SI
```

**value**

```

168 load_CI: vir_CSA × BRS × CI → vir_CSA
168 load_CI(vir_csa,(bi,ri,si),ci) as vir_csa'
168 pre: Pload(vir_csa,(bi,ri,si),ci)
168 post: Qload(vir_csa,(bi,ri,si),ci)(vir_csa')
169 unload_CI: vir_CSA × BRS $\tilde{\rightarrow}$ CI × vir_CSA
169 unload_CI(vir_csa,(bi,ri,si)) as (ci,vir_csa')
169 pre: Punload(vir_csa,(bi,ri,si))
169 post: Qunload(vir_csa,(bi,ri,si))(ci,vir_csa')
```

### K.6.6.2.1 The Load Pre-/Post-Conditions

171. The virtual  $\text{vir\_CSA}$ , i.e.,  $\text{vir\_csa}$ , must be well-formed;
172. the  $\text{ci}$  must not be embodied in that  $\text{vir\_csa}$ ; and
173. the bay/row/stack reference,  $(\text{bi}, \text{ri}, \text{si})$  must be one of the [virtual] container stowage area.

#### value

- 168  $\mathcal{P}_{load}(\text{vir\_csa}, (\text{bi}, \text{ri}, \text{si}), \text{ci}) \equiv$
- 171  $\text{well\_formed}(\text{vir\_csa})$  cf. 25– 27 on page 320
- 172  $\wedge \text{ci} \notin \text{xtr\_Cls}(\text{vir\_csa})$  cf. 49 on page 323
- 174  $\wedge \text{valid\_BRS}(\text{bi}, \text{ri}, \text{si})(\text{vir\_csa})$

- 174  $\text{valid\_BRS}: \text{BRS} \rightarrow \text{iCSA} \rightarrow \mathbf{Bool}$
- 174  $\text{valid\_BRS}(\text{bi}, \text{ri}, \text{si})(\text{vir\_csa}) \equiv$
- 174  $\text{bi} \in \mathbf{dom} \text{vir\_csa} \wedge \text{ri} \in \mathbf{dom} \text{vir\_csa}(\text{bi}) \wedge \text{si} \in \mathbf{dom}(\text{vir\_csa}(\text{bi}))(\text{ri})$

174. The resulting  $\text{vir\_CSA}$ , i.e.,  $\text{vir\_csa}'$ , must have the same bay, row and stack identifications, and
175. except for the designated bay, row and stack, must be unchanged.
176. The designated “before”, i.e., the stack before loading, must equal the tail of the “after”, i.e., the loaded stack, and
177. the top of the “after” stack must equal the “input” argument container identifier.,

#### value

- 169  $\mathcal{Q}_{load}(\text{vir\_csa}, (\text{bi}, \text{ri}, \text{si}), \text{ci})(\text{vir\_csa}') \equiv$
- 174  $\mathbf{dom} \text{vir\_csa} = \mathbf{dom} \text{vir\_csa}'$
- 174  $\wedge \forall \text{bi}': \text{BI} \bullet \text{bi}' \in \mathbf{dom} \text{vir\_csa}(\text{bi}')$
- 174  $\Rightarrow \mathbf{dom} \text{vir\_csa}(\text{bi}') = \mathbf{dom} \text{vir\_csa}'(\text{bi}')$
- 174  $\wedge \forall \text{ri}': \text{RI} \bullet \text{ri}' \in \mathbf{dom}(\text{vir\_csa}(\text{bi}'))()$
- 174  $\Rightarrow \mathbf{dom}(\text{vir\_csa}(\text{bi}'))(\text{ri}') = (\mathbf{dom} \text{vir\_csa}'(\text{bi}'))(\text{ri}')$
- 174  $\wedge \forall \text{si}': \text{SI} \bullet \text{si}' \in \mathbf{dom} \text{vir\_csa}(\text{bi}')$
- 174  $\Rightarrow \mathbf{dom}((\text{vir\_csa}(\text{bi}'))(\text{ri}'))(\text{si}') = \mathbf{dom}((\text{vir\_csa}'(\text{bi}'))(\text{ri}'))(\text{si}')$
- 175  $\wedge \forall \text{bi}': \text{BI} \bullet \text{bi}' \in \mathbf{dom} \text{vir\_csa} \setminus \{\text{bi}\}$
- 175  $\Rightarrow \text{vir\_csa} \setminus \{\text{bi}\} = \text{vir\_csa}' \setminus \{\text{bi}\}$
- 175  $\wedge \forall \text{ri}': \text{RI} \bullet \text{ri}' \in \mathbf{dom} \text{vir\_csa}(\text{bi}) \setminus \{\text{ri}\}$
- 175  $\Rightarrow (\text{vir\_csa}(\text{bi}))(\text{ri}') = (\text{vir\_csa}'(\text{bi}))(\text{ri}')$
- 175  $\wedge \forall \text{si}': \text{SI} \bullet \text{si}' \in \mathbf{dom}(\text{vir\_csa}(\text{ri}')) \setminus \{\text{si}\}$
- 175  $\Rightarrow ((\text{vir\_csa}(\text{bi}'))(\text{ri}'))(\text{si}') = ((\text{vir\_csa}'(\text{bi}'))(\text{ri}'))(\text{si}')$
- 176  $\wedge \mathbf{tl}((\text{vir\_csa}'(\text{bi}'))(\text{si}')) = ((\text{vir\_csa}'(\text{bi}'))(\text{si}'))$
- 177  $\wedge \mathbf{hd}((\text{vir\_csa}'(\text{bi}'))(\text{si}')) = \text{ci}$

### K.6.6.2.2 The Unload Pre-/Post-Conditions

178. The virtual  $\text{vir\_csa}$ , i.e.,  $\text{vir\_csa}$ ,
179. must be wellformed; and
180. the bay/row/stack reference,  $(\text{bi}, \text{ri}, \text{si})$  must be one of the [virtual] container stowage area.

**value**

```

178 $\mathcal{P}_{\text{unload}}(\text{vir_csa}, (\text{bi}, \text{ri}, \text{si})) \equiv$
179 $\text{well_formed}(\text{vir_csa})$
180 $\wedge \text{valid_BRS}(\text{bi}, \text{ri}, \text{si})(\text{vir_csa})$

```

181.

182.

183.

184.

185.

**value**

```

169 $\mathcal{Q}_{\text{unload}}(\text{vir_csa}, (\text{bi}, \text{ri}, \text{si}))(\text{ci}, \text{vir_csa}') \equiv$
181 $\text{dom } \text{vir_csa} = \text{dom } \text{vir_csa}'$
182 $\wedge \forall \text{bi}': \text{BI} \bullet \text{bi}' \in \text{dom } \text{vir_csa} \setminus \{\text{bi}\}$
182 $\Rightarrow \text{vir_csa} \setminus \{\text{bi}\} = \text{vir_csa}' \setminus \{\text{bi}\}$
183 $\wedge \forall \text{ri}': \text{RI} \bullet \text{ri}' \in \text{dom } \text{vir_csa}(\text{bi}) \setminus \{\text{ri}\}$
183 $\Rightarrow (\text{vir_csa}(\text{bi}))(\text{ri}') = (\text{vir_csa}'(\text{bi}))(\text{ri}')$
184 $\wedge \forall \text{si}': \text{SI} \bullet \text{si}' \in \text{dom } (\text{vir_csa})(\text{ri}') \setminus \{\text{si}\}$
184 $\Rightarrow ((\text{vir_csa})(\text{bi}'))(\text{si}') = ((\text{vir_csa}')(\text{bi}'))(\text{si}')$
185 $\wedge ((\text{vir_csa}')(\text{bi}'))(\text{si}') = \text{tl}((\text{vir_csa}')(\text{bi}'))(\text{si}')$
185 $\wedge \text{hd}((\text{vir_csa})(\text{bi}'))(\text{si}') = \text{ci}$

```

### K.6.6.3 Vessel Actions

Vessels (and land trucks) are in a sense, the primary movers in understanding the terminal container domain. Containers are, of course, at the very heart of this domain. But without container vessels (and land trucks) arriving at ports *nothing would happen!* So the actions of vessels are those of actively announcing their arrivals at and departures from ports, and participating, more passively, in the unloading and loading of containers.

#### K.6.6.3.1 Action [A]: `calc_next_port`:

186. Vessels can calculate, `calc_next_port`, the unique identifier, `mcc_i`, of that ports' monitoring & control center.

187. We do not further define the pre- and post-conditions of the `calc_next_port` action.

**value**

```

186 $\text{calc_next_port}: \text{VI} \times \text{VS_Mereo} \times \text{VS_Stat} \rightarrow \text{vir_CSA} \times \text{VS}\Sigma \rightarrow \text{MCCI} \times \text{VS}\Sigma$
186 $\text{calc_next_port}(\text{vs}_i, \text{vs_mereo}, \text{vs_stat})(\text{vir_csa}, \text{vs}\sigma) \text{ ia } (\text{mcc}_i, \text{vs}\sigma')$
187 pre: $\mathcal{P}_{\text{calc-next-port}}(\text{vs}\sigma, \text{vs_mereo}, \text{vs_stat})$
187 post: $\mathcal{Q}_{\text{calc-next-port}}(\text{vs}\sigma, \text{vs_mereo}, \text{vs_stat})(\text{mcc}_i, \text{vs}\sigma')$

```

**K.6.6.3.2 Vessel Action [B]: `calc-ves-msg`:**

188. Vessels can calculate, `calc-ves-info`, the vessel information, `vs-info:VS_Info`, to be handed to the next ports' command center.
189. This information is combined with the vessel identifier and its virtual CSA,
190. We leave undefined the pre- and post-conditions over vessel states and vessel information.

**type**

188 VS\_Info

189 VS\_MCC\_MSG :: VI×vir\_CSA×VS\_Info

**value**188 `calc-ves-msg`: VI×VMereo×VStat → VS\_Pos×vir\_CSA×VSΣ → VS\_MCC\_MSG×VSΣ188 `calc-ves-msg(vs_i,vs_mereo,vs_stat)(vpos,vir_csa,vσ) as (vs_mcc_msg,vσ')`190 **pre**:  $\mathcal{P}_{calc-ves-mcc-msg}(vs_i,vs_mereo,vs_stat)(vpos,vir_csa,v\sigma)$ 190 **post**:  $\mathcal{Q}_{calc-ves-mcc-msg}(vs_i,vs_mereo,vs_stat)(vpos,vir_csa,v\sigma)(vs_mcc_msg,v\sigma')$ **K.6.6.4 Land Truck Actions**

Land trucks can initiate the following actions vis-a-vis a targeted terminal port command center: announce, to a terminal command center, its arrival with a container; announce, to a terminal command center, its readiness to haul a container. Land trucks furthermore interacts with stack cranes – as so directed by terminal command centers.

**K.6.6.4.1 Land Truck Action [R]: `calc-truck-delivery`:**

191. Land trucks, upon approaching, from an outside, terminal ports, calculate
192. the identifier of the next port's command center and a next land truck state.
- We do not define the
193. pre- and
194. post conditions of this calculation.

**value**191 `calc-truck-delivery`: CI × TRUCKΣ → MCCI × LTΣ192 `calc-truck-delivery(ci,ltσ) as (mcci,ltσ')`193 **pre**:  $\mathcal{P}_{calc-truck-deliv}(ci,lt\sigma)$ 194 **post**:  $\mathcal{Q}_{calc-truck-deliv}(ci,lt\sigma)(mcci,lt\sigma')$ **K.6.6.4.2 Land Truck Action [S]: `calc-truck-avail`:**

195. Land trucks, when free, i.e., available for a next haul, calculate
196. the identifier of a suitable port's command center and a next land truck state.
- We do not define the
197. pre- and
198. post conditions of this calculation.

**value**195 `calc-truck-avail`: LTI × LTΣ → MCCI × LTΣ196 `calc-truck-avail(lti,ltσ) as (mcci,ltσ')`197 **pre**:  $\mathcal{P}_{calc-truck-avail}(lti,lt\sigma)$ 198 **post**:  $\mathcal{Q}_{calc-truck-avail}(lti,lt\sigma)(mcci,lt\sigma')$

### K.6.7 Events

We refer to [56, Sect. 7.1.6 and 7.3.2]. Events occur to all entities. For reasons purely of presentation we separate events into active part initiation events and active part completion events. Active part initiation events are those events that signal the initiation of actions. (Let  $[\Theta]$  designate an action, then  $[\Theta']$  designates the completion of that action.) Active part completion events are those events that signal the completion of actions. We do not show the lower case [**d, f, g, h, i, j, k, l, m, n, o**] in Fig. 3.

#### K.6.7.1 Active Part Initiation Events

##### Vessels:

- |                                                                                                |                                                                                                       |
|------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------|
| 199. $[\alpha_{vessel}]$ approaching terminal port;                                            | as to these unloads and loadings – and these actual unloads/loadings;                                 |
| 200. <b>[A]</b> informing the command center, mcc, of a terminal port, of arrival;             | 203. <b>[X]</b> receiving from an mcc directions of completion of stowage (no more unloads/loadings); |
| 201. <b>[B]</b> receiving from an mcc directions as to quay berth positions;                   | 204. <b>[Y]</b> informing the mcc of its departure from terminal port; or                             |
| 202. <b>[C]</b> receiving from an mcc, for each container to be unloaded or loaded, directions | 205. $[\omega_{vessel}]$ leaving a terminal port.                                                     |

##### Land Trucks:

- |                                                                                    |                                                                |
|------------------------------------------------------------------------------------|----------------------------------------------------------------|
| 206. $[\alpha_{land\_truck}]$ approaching a terminal port;                         | container;                                                     |
| 207. <b>[W]</b> informing its mcc of its arrival;                                  | 210. <b>[T]</b> the loading of a container from a stack crane; |
| 208. <b>[V]</b> being directed, by an mcc, as to the stack (crane) of destination; | 211. <b>[R]</b> informing its mcc of its departure; or         |
| 209. <b>[S]</b> the unloading, to a stack crane, of a                              | 212. $[\omega_{land\_truck}]$ leaving a terminal port.         |

##### Containers: the transfers from

- |                                            |                                                    |
|--------------------------------------------|----------------------------------------------------|
| 213. <b>[D]</b> vessel to quay crane;      | 218. <b>[j]</b> stack crane to quay truck;         |
| 214. <b>[d]</b> quay crane to vessel;      | 219. <b>[M]</b> stack crane to stack;              |
| 215. <b>[G]</b> quay crane to quay truck;  | 220. <b>[m]</b> stack to stack crane;              |
| 216. <b>[g]</b> quay truck to quay crane;  | 221. <b>[P]</b> stack crane to land truck; or from |
| 217. <b>[J]</b> quay truck to stack crane; | 222. <b>[p]</b> land truck to stack crane.         |

##### Quay Cranes: being informed, by the command center, mcc, of a container to be

- |                                          |                                              |
|------------------------------------------|----------------------------------------------|
| 223. <b>[E]</b> picked-up from a vessel; | 225. <b>[F]</b> set-down on a quay truck; or |
| 224. <b>[e]</b> set-down on a vessel;    | 226. <b>[f]</b> picked-up from a quay truck. |

##### Quay Trucks: being informed, by the command center, mcc, of a container to be

227. **[H]** loaded from a quay crane;                      229. **[I]** picked-up by a stack crane; or  
 228. **[h]** picked-up by a quay crane;                      230. **[i]** loaded from a stack crane.

**[Terminal] Stack Cranes:** being informed, by the command center, mcc, of a container to be

231. **[K]** picked-up from a quay truck;                      234. **[I]** loaded on to a stack;  
 232. **[k]** loaded on to a quay truck;                      235. **[O]** picked-up from a land truck; or  
 233. **[L]** picked-up from a stack;                      236. **[o]** loaded on to a land truck.

**[Terminal Bay] Stacks:** being informed, by the command center, mcc, of a container to be

237. **[N]** set-down, of a container, from a stack crane; or                      238. **[n]** picked-up, of a container, by a stack crane.

These events, in most cases, prompt interaction with the terminal command center.

#### K.6.7.2 Active Part Completion Events:

We do not show, in Fig. 3, the **c', e', h', o', q', t'** events.

239. **[C']**  
 240. **[E']**  
 241. **[H']**  
 242. **[O']**  
 243. **[Q']**  
 244. **[T']**

### K.6.8 Channels

We refer to [56, Sect. 7.2], and we refer to Sect. K.5.2 and to Fig. 2 on Page 336.

#### K.6.8.1 Channel Declarations

There are channels between terminal port monitoring & control command center (mcci) and that command centers and that terminal port's

245. all the **containers** (ci), that might visit the terminal port; `ch_mcc_con[mcci,ci]`<sup>12</sup>;  
 246. **vessels** (vi) that might visit that port, `ch_mcc[mcci,vi]`<sup>13</sup>;  
 247. **quay cranes** (qci) of that port, `ch_mcc[mcci,qci]`<sup>14</sup>;  
 248. **quay trucks** (qti) of that port , `ch_mcc[mcci,qti]`<sup>15</sup>;  
 249. **stack cranes** (sci) of that port, `ch_mcc[mcci,sci]`<sup>16</sup>;

<sup>12</sup>cf. Item 98 on page 330

<sup>13</sup>cf. Item 76 on page 326

<sup>14</sup>cf. Item 81 on page 326

<sup>15</sup>cf. Item 83 on page 327

<sup>16</sup>cf. Item 87 on page 327



250. **stacks [bays]** (stki) of that port, `ch_mcc[mcci,stki]`<sup>17</sup>; and
251. **land trucks** (lti) of, in principle, any port, `ch_mcc[mcci,lti]`<sup>18</sup>.
252. We shall define the concrete types of messages communicated by these channels subsequently (Sect. K.6.8.2).

**channel**

```

245 {ch_mcc_con[mcci,ci]|mcci:MCCI,ci:CI•mcci∈mcc_uis∧ci∈c_uis}:MCC_Con_Cmd
246-251 {ch_mcc[mcci,ui]|mcci:MCCI,ui:(VI|QCI|QTI|SCI|STKI|LTI)•mcci∈mcc_uis∧ui∈uis}:MCC_Msg
type
252 MCC_Con_Msg, MCC_Msg

```

**K.6.8.2 Channel Messages**

We present a careful analysis description, for the channels declared above, of the rather rich variety of messages communicated over channels. All messages “goes to” (a few) or “comes from” (the rest) the command center. Messages from quay cranes, quay trucks, stack cranes, and land trucks – directed at the command center – are all in response to the *events* of their being loaded or unloaded.

**K.6.8.2.1 A,B,X,Y,C': Vessel Messages**

253. There are a number **command center – vessel** and vice-versa messages:

- (a) **A**: Vessels announce their (forthcoming) arrival to the next destination terminal by sending such information, **VSArrv**, to its monitoring & control (also referred to as command) center, that enables it to handle those vessels’ berthing, unloading and loading (of container stowage).<sup>19</sup>
- (b) **B**: The terminal command center informs such arriving vessels of their quay segment positions, **VSQPos**.
- (c) **X**: The terminal command center informs vessels of completion of stowage handling, **VSComp**.
- (d) **Y**: Vessels inform the terminal of their departure, **VesDept**.

**type**

```

253 MCC_Cmd == VSArrv|VSQPos|VSComp|VSDept|...
253a A: VSArrv :: VI × vir_CSA
253b B: VSQPos :: VI × (QSIId × QP+)
253c X: VSComp :: MCCI × VI
253d Y: VSDept :: MCCI × VI

```

**K.6.8.2.2 C,D,E,E': Vessel/Container/Quay Crane Messages**

254. The terminal command center, at a time it so decides, “triggers” the simultaneous transitions, **C,D,E**, of
- (a) **C**: unloading (loading) from (to) a vessel stack position of a container (surrogate), **VSQC\_Xfer**, **QCVS\_Xfer**),

<sup>17</sup>cf. Item 94 on page 329

<sup>18</sup>cf. Item 96 on page 329

<sup>19</sup>What exactly that information is, i.e., any more concrete type model of **Ves\_Info** cannot be given at this early stage in our development of *what a terminal is*.

- (b) **D**: notifying the physical, i.e., the actual container that it is being unloaded (loaded), C\_VStoQC (C\_QCtoVS), and
  - (c) **E**: loading (unloading) the container (surrogate) onto (from) a quay crane, VStoQC (QCtoVS).
255. **C',E'**: The vessel and the quay crane, in response to their being unloaded, respectively loaded with a container “moves” that load, from its top vessel bay/row/stack position to the quay crane and notifies the terminal command center of the completion of that move, VSQC\_Compl.

**type**

253 MCC\_Cmd == ... | VSQC\_Xfer | QCVS\_Xfer | C\_VtoQC | C\_QCtoV | VQC\_Compl  
 254a VSQC\_Xfer, QCVS\_Xfer :: VI × (BRS × CI) × QCI  
 254b C\_VStoQC, C\_QCtoVS :: VI × CI × QCI  
 254c VStoQC, QCtoVS :: VI × CI × QCI  
 255 VSQC\_Compl == VS\_UnLoad | VS\_Load  
 255 VS\_UnLoad, VS\_Load :: VI × CI × QCI

**K.6.8.2.3 F,G,H,H': Quay Crane/Container/Quay Truck Messages**

256. The terminal command center, at a time it so decides “triggers” the simultaneous transitions, **F,G,H**: QCtoQT, of
- (a) **F**: the removal of the container from the quay crane,
  - (b) **G**: the notification of the physical container that it is now being transferred to a quay truck, and
  - (c) **H**: the loading of that container to a quay truck.
  - (d) **H'**: The quay truck, in response to it being loaded notifies the terminal command center of the completion of that move.

**type**

253 MCC\_Cmd == ... | QCtoQT | ...  
 256 QCtoQT == UnloadCQC | NowConQT | LoadCQT | QCtoQTCompl  
 256a UnloadCQC :: CI × QCI  
 256b NowConQT :: CI × QTI  
 256c LoadCQT :: CI × QTI  
 256d QCtoQTCompl :: ...

**K.6.8.2.4 I,J,K,K': Quay Truck/Container/Stack Crane Messages**

257. The terminal command center, at a time it so decides “triggers” the simultaneous transitions, **I,J,K**: QTtoSC, of
- (a) **I**: the removal of a container from a quay truck,
  - (b) **J**: the notification of the physical container that it is now being transferred to a stack crane, and
  - (c) **K**: the loading of that container to a stack crane.
258. **K'**: The stack crane, in response to it being loaded notifies the terminal command center of the completion of that move.

**type**

```

257 MCC_Cmd = ... | QTtoSC | ...
257 QTtoSC == UnLoadCQT | NowConSC | | QCQTCompl
257a UnLoadCQT :: CI × QRI
257b NowConSC :: CI × SCI
257c LoadCSC :: CI × SCI
258 QCSCCompl :: ...

```

**K.6.8.2.5 L,M,N,N': Stack Crane/Container/Stack Messages**

259. The terminal command center, at a time it so decides “triggers” the simultaneous transitions, **L,M,N**: SCtoStack, of
- (a) **L**: the unloading of the container from a stack crane;
  - (b) **M**: the notification of the physical container that it is now being transferred to a stack, and
  - (c) **N**: the loading of that container to a stack.
260. **N'**: The stack, in response to it being loaded, notifies the terminal command center of the completion of that move.

**type**

```

259 MCC_Cmd = ... | SCtoStack | ...
259 SCtoStack == UnLoadCSC | NowConSTK | LoadConSTK | SCStkCompl
259a UnLoadCSC :: CI × SCI
259b NowConSTK :: CI × BRS
259c LoadConSTK :: CI × BRS
260 SCStkCompl :: ...

```

**K.6.8.2.6 O,P,Q,Q': Land Truck/Container/Stack Crane Messages**

261. The terminal command center, at a time it so decides “triggers” the simultaneous transitions, **O,P,Q**: LTtoSC, of
- (a) **Q**: the unloading of the container from a land truck to a stack crane;
  - (b) **P**: the notification of the physical container that it is now being transferred to a stack crane, and
  - (c) **O**: the loading of that container to a stack crane.
  - (d) **O'**: The stack crane, in response to it being loaded, notifies the terminal command center of the completion of that move.<sup>20</sup>

**type**

```

261 MCC_Cmd = ... | LTtoSC | ...
261 LTtoSC == UnLoadCLT | NowConSC | LoadConSC | LTtoSCCompl
261a UnLoadCLT :: CI × LTI
261b NowConSC :: CI × SCI
261c LoadConSC :: CI × SCI
261d LTtoSCCompl :: ...

```

<sup>20</sup>The **O'** event is “the same” as the **K'** event.

### K.6.8.2.7 R,S,T,U,Q,V: Land Truck Messages

262. These are the messages that are communicated either from land trucks to command centers or vice versa:

- (a) **R**: Land trucks, when approaching a terminal port, informs that port of its offer to deliver an identified container to stowage.
- (b) **S**: Land trucks, when approaching a terminal port, informs that port of its offer to accept (load) an identified container from stowage.
- (c) **T**: Land trucks, at a terminal, are informed by the terminal of the stack crane at which to deliver (unload) an identified container.
- (d) **U**: Land trucks, at a terminal, are informed by the terminal of the stack crane from which to accept an identified container.
- (e) **Q**: Land trucks, at a terminal, are informed by the terminal of the stack crane at which to unload (deliver) an identified container.
- (f) **q**: Land trucks, at a terminal, are informed by the terminal of the stack crane at which to load (accept) an identified container.
- (g) **V**: Land trucks, at a terminal, inform the terminal of their departure.

#### type

```

262 MCC_Cmd = ... | LTCmd | ...
262 LTCmd == LTDlvr | LTFtch | LTtoSC | LTfrSC | LTDept
262a LTDlvr :: LTI × CI
262b LTFtch :: LTI × CI
262c LTtoSC :: LTI × CI
262d LTfrSC :: LTI × CI
262g LTDept :: LTI

```

## K.6.9 Behaviours

We refer to [56, Sects. 7.1.7, 7.3.3-4-5, and 7.4].

To every part of the domain we associate a behaviour. Parts are in space: there are the manifest parts, and there are the notion of their corresponding behaviours. Behaviours are in space and time. We model behaviours as processes defined in [AMoL](#). We cannot see these processes. We can, however, define their effects.

Parts may move in space: vessels, cranes, trucks and containers certainly do move in space; processes have no notion of spatial location. So we must “fake” the movements of movable parts. We do so as follows: We associate with containers the programmable attribute of location, as outlined in Items 126–126g on page 334. We omit, for this model, the more explicit modelling of vessels, cranes and trucks but refer to their physical mereologies.

In the model of endurants, cf. Page 319, we modelled vessel and terminal container stowage areas as physically embodying containers, and we could move containers: push and pop them onto, respectively from bay stacks. This model must now, with containers being processes, be changed. The stacks, **STACK**, of container stowage areas, **CAS**, now embody unique container identifiers! We rename these stacks into **cistack:CiSTACK**

### K.6.9.1 Terminal Command Center

The terminal command center is at the core of activities of a terminal port. We refer to the figure on Page 336. “Reading” that figure left-to-right illustrates the movements of containers from **[C-D-E]** vessels to quay cranes, **[F-G-H]** quay cranes to quay trucks, **[I-J-K]** quay trucks to stack cranes, **[L-M-N]** stack cranes to stacks, and from **[O-P-Q]** land truck to stack cranes.

A similar “reading” of that figure from right-to-left would illustrate the movements of containers from [q-p-o] stack cranes to land trucks; [n-m-l] stacks to stack cranes; [k-j-i] stack cranes to quay trucks; [h-g-f] quay trucks to quay cranes; and from [e-d-c] quay cranes to vessels. We have not show the [c-d-e-f-g-h-i-j-k-l-m-n-o-p-q] labels, but their points should be obvious (!).

**K.6.9.1.1 The Command Center Behaviour:** We distinguish between the command center behaviour offering to *monitor* primarily vessels and land trucks, secondarily cranes, quay cranes and stacks, and offering to *control* vessels, cranes, trucks and containers.

263. The signature of the command center behaviour is a triple of the command center identifier, the conceptual command center mereology and the static command center attributes (i.e., the topological description of the terminal); the programmable command center attributes (i.e., the command center state); and the input/output channels for the command center.

The command center behaviour non-deterministically (externally) chooses between

264. either monitoring inputs from

265. or controlling (i.e., outputs to)

vessels, cranes, trucks, stacks and containers.

**value**

```

263 command_center:
263 mcci:MCCI×(vis,qcis,qtis,scis,bis,ltis,cis):MCC_Mer×MCC_Stat
263 → MCCΣ →
263 in,out { ch_mcc[mcci,ui]n
263 | mcci:MCCI,ui:(V|QCI|QTI|SCI|BI|LTI)
263 • ui∈vis∪qcis∪qtis∪scis∪bis∪ltis }
263 out { ch_mcc_con[mcci,ci] | ci:CI•ci ∈ cis } Unit
263 command_center(mcci,(vis,qcis,scis,bis,ltis,cis),mcc_stat)(mccσ) ≡
264 monitoring(mcci,(vis,qcis,scis,bis,ltis,cis),mcc_stat)(mccσ)
263 []
265 control(mcci,(vis,qcis,scis,bis,ltis,cis),mcc_stat)(mccσ)

```

**K.6.9.1.2 The Command Center Monitor Behaviours:** The command center monitors the behaviours of vessels, cranes and trucks: [A,Y',C',E',F',H',I',K',L',N',O',Q']. The input message thus received is typed:

**type**

VCT\_Info = ...

That information is used by the command center to update its state:

**value**

update\_MCCΣ: VCT\_Infor → MCCΣ → MCCΣ

The definition of monitoring is simple.

266. The signature of the monitoring behaviour is the same as the command center behaviour.
267. The monitor non-deterministically externally ([]) offers to accept any input, vct\_info, message from any vessel, any land truck and from local terminal port quay trucks and cranes.
268. That input, vct\_info, enters the update of the command center state, from mccσ to mccσ'.
269. Whereupon the monitoring behaviour resumes being the command center behaviour with an updated state.

**value**

```

266 monitoring: mcci:MCCI × mis:MCC_Mereo × MCC_Stat
266 → MCCΣ
266 → in,out {chan_mcc[mcci,i] | i ∈ mis} Unit
266 monitoring(mcci,mis,mcc_stat)(mccσ) ≡
267 let vct_info = [] { chan_mcc[mcci,i] ? | i ∈ mis } in
268 let mcccσ' = update_MCCΣ((vct_info,ui))(mccσ) in
269 command_center(mcci,mis,mcc_stat)(mcccσ') end end

```

**K.6.9.1.3 The Command Center Control Behaviours:**

270. The command center control behaviour has the same signature as the command center behaviour (formula Items 263).

271. In each iteration of the **command center** behaviour in which it chooses the **control** alternative it calculates<sup>21</sup> a next [output] transaction. This calculation is at the very core of the overall terminal port. We shall have more to say about this in Sect. K.7.1 on page 360.

Items, 272a–272j represent 10 alternative transactions.

272. They are “selected” by the **case** clause (Item 272).

So for each of these 10 alternatives there the command center offers a communication. For the **[CDE, FGH, IJK, LMN, OPQ, opq]** cases there is the same triple of concurrently synchronised events. For the **[B,T,X]** clauses there are only a single synchronisation effort. The command center events communicates:

- (a) **[B]** the quay positions to arriving vessels,  
the transfer of containers
- (b) **[CDE]** from vessel stacks to quay cranes,
- (c) **[FGH]** quay cranes to quay trucks,
- (d) **[IJK]** quay trucks to stack cranes,
- (e) **[LMN]** stack cranes to stacks,
- (f) **[OPQ]** stack cranes to land trucks, and
- (g) **[opq]** land trucks to stack cranes.

We also illustrate

- (h) **[T]** the bays to which a land truck is to deliver, or fetch a container, and
- (i) **[X]** the “signing off” of a vessel by the command center.
- (j) For the case that the next transaction cannot be determined [at any given point in time] there is nothing to act upon.

273. After any of these alternatives the command center control behaviour resumes being the command center behaviour with the state updated from the next transaction calculation.

**value**

```

270 control: mcci:MCCI × (vis,qcis,qtis,scis,bis,ltis,cis):MCC_Mer × MCC_Stat → MCCΣ →
263 in,out {ch_mcc[mcci,ui] | mcci:MCCI,ui:(VI|QCI|QTI|SCI|BI|LTI)•ui ∈ vis ∪ qcis ∪ qtis ∪ scis ∪ bis ∪ ltis}
263 out { ch_mcc_con[mcci,ci] | ci:CI•ci ∈ cis } Unit
270 control(mcci,(vis,qcis,scis,bis,ltis,cis),mcc_stat)(mccσ) ≡
271 let (mcc_trans,mcccσ') = calc_nxt_transaction(mcci,mcc_mereo,mcc_stat)(mccσ) in
272 case mcc_trans of
272a [B] mkVSQPos(vi,qp) → ch_mcc[mcci,vi] ! mkVSQPos(vi,qp),

```

<sup>21</sup>For `calc_nxt_transaction` see Items 130 – 140 on page 338

```

272b [CDE] mkVSQC_Xfer(vi,(brs,ci),qci) →
272b [C] ch_mcc[mcci,vi] ! mkVes_UnLoad(ci,brs)
272b [D] || ch_mcc_con[mcci,ci] ! mkNewPos(mcci,qci)
272b [E] || ch_mcc[mcci,qci] ! mkQC_Load(ci),
272c [FGH] mkQCQT_Xfer(qci,ci,qti) →
272c [F] ch_mcc[mcci,qci] ! mkQC_UnLoad(ci)
272c [G] || ch_mcc_con[mcci,ci] ! mkNewPos(mcci,qti)
272c [H] || ch_mcc[mcci,qti] ! mkQT_Load(ci),
272d [IJK] mkQTSC_Xfer(qti,ci,sci) →
272d [I] ch_mcc[mcci,qci] ! mkQT_UnLoad(ci)
272d [J] || ch_mcc_con[mcci,ci] ! mkNewPos(mcci,sci)
272d [K] || ch_mcc[mcci,qti] ! mkSC_Load(ci),
272e [LMN] mkSCSTK_Xfer(brs,ci,sci,sti) →
272e [L] ch_mcc[mcci,sci] ! mkSC_UnLoad(ci)
272e [M] || ch_mcc_con[mcci,ci] ! mkNewPos(mcci,brs)
272e [N] || ch_mcc[mcci,sti] ! mkSTK_Load(ci,brs),
272f [OPQ] mkSCLT_Xfer(sci,ci,lti) →
272f [O] ch_mcc[mcci,sci] ! mkSC_UnLoad(ci)
272f [P] || ch_mcc_con[mcci,ci] ! mkNewPos(mcci,lti)
272f [Q] || ch_mcc[mcci,lti] ! mkLT_Load(ci),
272g [opq] mkLTSC_Xfer(sci,ci,lti) →
272g [o] ch_mcc[mcci,sci] ! mkSC_Load(ci)
272g [p] || ch_mcc_con[mcci,ci] ! mkNewPos(mcci,cti)
272g [q] || ch_mcc[mcci,lti] ! mkLT_UnLoad(ci),
272h [T] mkLT_Dept(lti) → ch_mcc[mcci,lti] ! LTDept(mcci,lti),
272i [Y] mkVSComp(mcci,vi) → ch_mcc[mcci,vi] ! VSComp(mcci,vi),
272i [X] mkVSDept(mcci,vi) → ch_mcc[mcci,vi] ! VSDept(mcci,vi),
272j _ → skip
272 end ; command_center(mcci,(vis,qcis,scis,bis,ltis,cis),mcc_stat)(mccσ') end

```

### K.6.9.2 Vessels

274. The signature of the vessel behaviour is a triple of the vessel identifier, the conceptual vessel mereology, the static vessel attributes, and the programmable vessel attributes. [We presently leave static attributes unspecified: ...]

Nondeterministically externally,  $\square$ , the vessel decides between

- 275. [A] either approaching a port,
- 276.  $\square$  or [subsequently] arriving at that port,  
or [subsequently] participating in the
- 277.  $\square$  unloading and
- 278.  $\square$  loading of containers of containers,
- 279.  $\square$  or [finally] departing from that port.

#### value

```

274 vessel: vi:V1×mccis:V_Mereo×V_Sta_Attrs → (V_Pos×vir_CSA×VΣ)
274 → in,out {ch_mcc[mcci,vi]|mcci:MCCI•mcci∈mccis} Unit
274 vessel(vi,mccis,...)(vpos,vir_csa,vσ) ≡
275 port_approach(vi,mccis,...)(vpos,vir_csa,vσ)
276 □ port_arrival(vi,mccis,...)(vpos,vir_csa,vσ)
277 □ unload_container(vi,mccis,...)(vpos,vir_csa,vσ)
278 □ load_container(vi,mccis,...)(vpos,vir_csa,vσ)
279 □ port_departure(vi,mccis,...)(vpos,vir_csa,vσ)

```

#### K.6.9.2.1 Port Approach

280. The signature of `port_approach` behaviour is identical to that of `vessel` behaviour.

281. On approaching any port the vessel calculates the identity of that port's command center.

282. Then, with an updated state, it calculates the information to be handed over to the designated terminal –

283. [A] which is then communicated from the vessel to the command center;

284. whereupon the vessel resumes being a vessel albeit with a doubly updated state.

**value**

```

280 port_approach: vi:VI×vs_mer:VS_Mereo×VS_Stat→(VS_Pos×vir_CSA×VΣ)
280 → in,out {ch_mcc[mcci,vi]|mcci:MCCI•mcci∈mccis} Unit
280 port_approach(vi,vs_mer,vs_stat)(vpos,vir_csa,vσ) ≡
281 let (mcci,vσ') = calc_next_port(vi,vs_mer,vs_stat)(vpos,vir_csa,vσ) in
282 let (mkVInfo(vi,vir_csa,vs_info),vσ'') = calc_ves_msg(vpos,vir_csa,vσ') in
283 ch_mcc[mcci,vi] ! mkVS_Info(vi,vir_csa,vs_info) ;
284 vessel(vi,vs_mer,vs_stat)(vpos,vir_csa,vσ'') end end

```

**K.6.9.2.2 Port Arrival**

285. The signature of `port_arrival` behaviour is identical to that of `vessel` behaviour.
286. **[B]** Non-deterministically externally the vessel offers to accept a terminal port quay position from any terminal port's command center.
287. The vessel state is updated accordingly.
288. Whereupon the vessel resumes being a vessel albeit with a state updated with awareness of its quay position.
289. The vessel is ready to receive such quay position from any terminal port.

**value**

```

285 port_arrival: vi:VI×mccis:V_Mereo×V_Sta_Attrs → (V_Pos×vir_CSA×VΣ)
285 → in,out {ch_mcc[mcci,vi]|mcci:MCCI•mcci∈mccis} Unit
285 port_arrival(vi,mccis,...)(vpos,vir_csa,vσ) ≡
286 { let mkVSQPos(vi,(qs,cpl)) = ch_mcc[mcci,vi] ? in
287 let vσ' = upd_ves_state(mcci,(qs,cpl))(vσ) in
288 vessel(vi,mccis,...)(mkInPort(mcci,mkVSQPos(qs,cpl)),vir_csa,vσ') end end
289 | mcci:MCCI•mcci∈mccis }

```

**K.6.9.2.3 Unloading of Containers**

290. The signature of `port_arrival` behaviour is identical to that of `vessel` behaviour.
291. **[C]** The vessel offers to accept, `ch_mcc_v[mcci,vi] ?`, a directive from the command center of the terminal port at which it is berthed, to unload, `mkUnload((bi,ri,si),ci)`, a container, identified by `ci`, at some container stowage area location `((bi,ri,si))`.
292. The vessel unloads the container – identified by `ci'`.
293. If the unloaded container identifier is different from the expected **chaos** erupts!
294. The vessel state, `vσ'`, is updated accordingly.
295. **[C']** “Some time has elapsed since the unload directive, modelling” the completion, from the point of view of the vessel, of the unload operation –
296. whereupon the command center is informed of this completion (**[!]**).
297. The vessel resumes being the vessel in a state reflecting the unload.

**value**

```

290 unload_container: vi:VI × mccis:V_Mereo × V_Sta_Attrs → (V_Pos × iCSA × VΣ) →
290 in,out {ch_mcc[mcci,vi]|mcci:MCCI•mcci∈mccis} Unit
290 unload_container(vi,mccis,...)(vpos,vir_csa,vσ) ≡
291 let mkVes_UnLoad(ci,(bi,ri,si)) = ch_mcc[mcci,vi] ? in
292 let (ci',vir_csa') = unload_CI((bi,ri,si),vir_csa) in
293 if ci' ≠ ci then chaos end;
294 let vσ'' = unload_update_VΣ((bi,ri,si),ci)(vir_csa') in
295 wait sometime ;
296 ch_mcc[mcci,vi] ! mkCompl(mkV_UnLoad((bi,ri,si),ci)) ;
297 vessel(vi,mccis,...)(vpos,vir_csa',vσ'') end end end

```



### K.6.9.2.4 Loading of Containers

298. The signature of `load_container` behaviour is identical to that of `vessel` behaviour.
299. [c] The vessel offers to accept, `ch_mcc_v[mcci,vi] ?`, a directive from the command center of the terminal port at which it is berthed, to load, `mkLoad((bi,ri,si),ci)`. a container, identified by `ci`, at some container stowage area location `((bi,ri,si))`.
300. The vessel (in co-operation with a quay crane, see later) then **unloads** the container – identified by `ci`.
301. The vessel state,  $v\sigma'$ , is updated accordingly.
302. [c'] “Some time has elapsed since the unload directive, modelling” the completion, from the point of view of the vessel, of the unload operation – whereupon the command center is informed of this completion ([!]).
303. and the vessels resumes being the vessel in a state reflecting the load.

**value**

```

298 load_container: vi:VI × mccis:V_Mereo × V_Sta_Attrs → (V_Pos × vir_CSA × VΣ)
298 → in,out {ch_mcc[mcci,vi] | mcci:MCCL • mcci ∈ mccis} Unit
298 load_container(vi,mccis,...)(vpos,vir_csa,vσ) ≡
299 let mkV_Load((bi,ri,si),ci) = ch_mcc[mcci,vi] ? in
300 let vir_csa' = load_CI(vir_csa,(bi,ri,si),ci) in
301 let vσ' = load_update_VΣ((bi,ri,si),ci) in
302 ch_mcc[mcci,vi] ! mkCompl(mkV_Load((bi,ri,si),ci)) ;
303 vessel(vi,mccis,...)(vpos,vir_csa',vσ') end end end

```

### K.6.9.2.5 Port Departure

304. The signature of `port_departure` behaviour is identical to that of `vessel` behaviour.
305. [Y] At some time some command center informs a vessel that *stowage*, i.e., the unloading and loading of containers has ended.
306. Vessels update their states accordingly.
307. [Y'] Vessels respond by informing the command center of their departure.
308. Whereupon vessels resume being vessels.

**value**

```

304 port_departure: vi:VI × mccis:V_Mereo × V_Sta_Attrs → (V_Pos × vir_CSA × VΣ)
304 → in,out {ch_mcc[mcci,vi] | mcci:MCCL • mcci ∈ mccis} Unit
304 port_departure(vi,mccis,v_sta)(vpos,vir_csa,vσ) ≡
305 let mkStow_Comp(mcci,vi) [] { ch_mcc[mcci,vi] ? | mcci:MCCL • mcci ∈ mccis } in
306 let vσ' = update_vessel_state(mkVes_Dept(mcci,vi))(vσ) in
307 ch_mcc[mcci,vi] ! mkVes_Dept(mcci,vi) ;
308 vessel(vi,mccis,v_sta)(vpos,vir_csa,vσ') end end

```

• • •

The next three behaviours: `quay_crane`, `quay_truck` and `stack_crane`, are very similar. One substitutes, line-by-line, command center/quay crane, quay crane/quay truck, quay truck/stack crane et cetera!

### K.6.9.3 Quay Cranes

309. The signature of the `quay_crane` behaviour is a triple of the quay crane identifier, the conceptual quay crane mereology, the static quay crane attributes, the programmable quay crane attributes – and the ‘command center’/‘quay crane’ channel.
310. The quay crane offers, non-deterministically externally, to
311. either, [E], accept a directive of a ‘*container transfer from vessel to quay crane*’.
- (a) The quay crane then resumes being a quay crane now holding (a surrogate of) the transferred container.
312. or, [F] accept a directive of a transfer ‘*container from quay crane to quay truck*’.
- (a) The quay crane then resumes being a quay crane now holding (a surrogate of) the transferred container.

**value**

```

309 quay_crane: qci:QCI × mcci:QC_Mer × QC_Sta → (QCHold × QCPos)
309 → ch_mcc[mcci,qci] Unit
309 quay_crane(qci,mcci,qc_sta)(qchold,qcpos) ≡
311 let mkVSQC(ci) = ch_mcc[mcci,qci] ? in
311a quay_crane(qci,mcci,qc_sta)(mkCon(ci),qcpos) end
310 []
312 let mkQCVS(ci) = ch_mcc[mcci,qci] ? in
312a quay_crane(qci,mcci,qc_sta)(mkCon(ci),qcpos) end

```

#### K.6.9.4 Quay Trucks

313. The signature of the `quay_truck` behaviour is a triple of the quay truck identifier, the conceptual quay truck mereology, the static quay truck attributes, the programmable quay truck attributes – and the 'command center'/'quay truck' channel.
314. The quay truck offers, non-deterministically externally, to
315. either, **[H]**, accept a directive of a '*container transfer from quay crane to quay truck*'.
- (a) The quay truck then resumes being a quay truck now holding (a surrogate of) the transferred container.
316. or, **[I]**, accept a directive of a '*container transfer from quay truck to quay crane*'.
- (a) The quay truck then resumes being a quay truck now holding (a surrogate of) the transferred container.

##### value

```

313 quay_truck: qti:QTl × mcci:QC_Mer × QT_Sta → (QTHold×QTPos)
313 → ch_mcc[mcci,qci] Unit
313 quay_truck(qti,mcci,qt_sta)(qthold,qtpos) ≡
315 let mkQCQT(ci) = ch_mcc[mcci,qti] ? in
315a quay_crane(qti,mcci,qc_sta)(mkCon(ci),qcpos) end
314 []
316 let mkQTQC(ci) = ch_mcc[mcci,qti] ? in
316a quay_crane(qti,mcci,qc_sta)(mkCon(ci),qcpos) end

```

#### K.6.9.5 Stack Crane

317. The signature of the `stack_crane` behaviour is a triple of the stack crane stack crane identifier, the conceptual mereology, the static stack crane attributes, the programmable stack crane attributes – and the 'command center'/'stack crane' channel.
318. The stack crane offers, non-deterministically externally, to
319. either, **[K]**, accept a directive of a '*container transfer from quay truck to stack crane*'.
- (a) The stack crane then resumes being a stack crane now holding (a surrogate of) the transferred container.
320. or, **[L]**, accept a directive of a '*container transfer from stack crane to quay truck*'.
- (a) The stack crane then resumes being a stack crane now holding (a surrogate of) the transferred container.

##### value

```

317 stack_crane: sci:SCI × mcci:SC_Mer × SC_Sta → (SCHold×SCPos)
317 → ch_mcc[mcci,sci] Unit
317 stack_crane(sci,mcci,sc_sta)(schold,scpos) ≡
319 let mkQTSC(ci) = ch_mcc[mcci,sci] ? in
319a stack_crane(sci,mcci,sc_sta)(mkCon(ci),scpos) end
318 []
320 let mkSCQT(ci) = ch_mcc[mcci,sci] ? in
320a stack_crane(sci,mcci,sc_sta)(mkCon(ci),scpos) end

```

#### K.6.9.6 Stacks

The stack behaviour is very much like the `unload_container` container behaviour of the `vessel`, cf. Items 290 – 294 on page 354.

321. The signature of the `stack` behaviour is a triple of the stack, i.e. terminal port bay identifier, the conceptual bay mereology, the static bay attributes, the programmable bay attributes and the 'command center'/'stack' channel.
322. The stack offers, **[N]**, to accept directive of a '*container transfer from stack crane to stack*'.
- (a) The stack behaviour loads the container, identified by  $ci'$ , to the bay/row/stack top, identified by  $(bi,ri,si)$ .
- (b) If the unloaded container identifier is different from the expected **chaos** erupts!
- (c) The stack state, `bay'`, is updated accordingly.
- (d) **[N']** "Some time has elapsed since the load directive, modelling" the completion, from the point of view of the vessel, of the unload operation –
- (e) whereupon the command center is informed of this completion (**[I]**).

- (f) The stack then resumes being a stack now holding (a surrogate of) the transferred container.

**value**

```

321 stack: tbi:TBI×mcci:STK_Mer×Stk_Sta_Attrs → (iCSA × Stk_Dir) →
321 in,out {ch_mcc[mcci,vi]|mcci:MCCI•mcci∈mccis} Unit
321 stack(tbi,mcci,stk_sta)(bay,dir) ≡
322 let mkUnload((bi,ri,si),ci) = ch_mcc[mcci,tbi] ? in
322a let (ci',bay') = unload_CI((bi,ri,si),bay) in
322b if ci' ≠ ci then chaos end ;
322c let bay'' = unload_update_BAY((bi,ri,si),ci)(bay') in
322d wait sometime ;
322e ch_mcc[mcci,tbi] ! mkCompl(mkUnload((bi,ri,si),ci)) ;
322f stack(tbi,mcci,stk_sta)(bay'',dir) end end end

```

### K.6.9.7 Land Trucks

323. The signature of the `land_truck` behaviour is a triple of the land truck identifier, the conceptual land truck mereology and the static land truck attributes, and the programmable land truck attributes.

324. **R**

- (a) The land truck calculates the identifier of the next port's command center
- (b) and communicates with this center as to its intent to deliver a container identified by `ci`,
- (c) whereupon the land truck resumes being that.

325. **T**

- (a) The command center informs the land truck of the bay ('stack'), `brs`, at which to deliver the container,
- (b) whereupon the land truck resumes being that.

326. **Q**

- (a) The command center informs the land truck of the delivery of a container from a stack crane,
- (b) ...,
- (c) whereupon the land truck resumes being that.

327. **V**

- (a) The land truck informs the command center of its intent to depart from the terminal port,
- (b) whereupon the land truck resumes by leaving the terminal port.

**value**

```

323 land_truck:
323
323 land_truck(lti,lt_mer,lt_sta)(lt_pos,lt_hold) ≡
324 next_port(lti,lt_mer,lt_sta)(lt_pos,lt_hold)
325 [] stack_location(lti,lt_mer,lt_sta)(lt_pos,lt_hold)
326 [] stack_crane_to_land_truck(lti,lt_mer,lt_sta)(lt_pos,lt_hold)
327 [] land_truck_departure(lti,lt_mer,lt_sta)(lt_pos,lt_hold)

```

**value**

```

324 next_port(lti,lt_mer,lt_sta)(...,mkHold(ci,cσ)) ≡
324a let mcci = calc_truck_delivery(ci,cσ) in
324b ch_mcc[mcci,lti] ! mkDlvr(ci,cσ) ;
324c land_truck(lti,lt_mer,lt_sta)(...,...) end ???

```

**value**

```

325 stack_location(lti,lt_mer,lt_sta)(...,mkHold(ci,cσ)) ≡
325a let mkLT_Pos(mcci,brs) = { ch_mcc[mcci,lti] ? | mcci:MCCI • mcci ∈ mcc_uis }
325b land_truck(lti,lt_mer,lt_sta)(...,lt_hold) end ???

```

**value**

```

326 stack_crane_to_land_truck(lti,lt_mer,lt_sta)(lt_pos,lt_hold) ≡
326a
326b

```

**value**

```

327 land_truck_departure(lti,lt_mer,lt_sta)(.....) ???
327a ch_mcc[mcci,lti] ! mkDept(lti) ;
327b land_truck(lti,lt_mer,lt_sta)(.....) ???

```

**K.6.9.8 Containers**

In RSL, as with all formal specification languages one cannot “move” values. So we model containers of vessels and of terminal port stacks as separate behaviours and replace their “values”,  $C$  in vessel and terminal port stacks by their unique identifications,  $CI$ .

- 328. The signature of the container behaviour is simple: the container identifier, its mereology, its static values, its position and state<sup>22</sup>, and its input channels.
- 329. **[D,G,J,M,P]** The container is here simplified to just, at any moment, accepting a new position from any terminal ports command center;
- 330. whereupon the container resumes being that with that new position.

**value**

```

328 container: ci:CI × mcci_uis:C_Mer × C_Stat → (CPos × CΣ)
328 → in { ch_mcc_con[mcci,ci]
328 | mcci:MCCI • mcci ∈ mcci_uis } Unit
328 container(ci,mcci_uis,...)(pos,σ) ≡
329 let mkNewPos(p) = { ch_mcc_con[mcci,ci] ?
329 | mcci:MCCI • mcci ∈ mcci_uis } in
330 container(ci,mcci_uis,...)(mkNewPos(p),σ) end

```

**K.6.10 Initial System****K.6.10.1 The Distributed System**

We remind ourselves that the container line industry includes a set of vessels, a set of land trucks, a set of containers and a set of terminal ports. We rely on the states expounded in Sect. K.5.4.1’s Items 50 on page 323 – 54 on page 323.

- 331. The signature of  $\tau_{\text{initial\_system}}$  is that of a function from an enduring container line industry to its perdurant behaviour, i.e., **Unit**.

This behaviour is expressed as

- 332. the distributed composition of all vessel behaviours in parallel with
- 333. the distributed composition of all land truck behaviours in parallel with
- 334. the distributed composition of all container behaviours in parallel with
- 335. the distributed composition of all terminal port behaviours.

**value**

```

332 τ_initial_system: CLI → Unit
332 τ_initial_system(cli) ≡
332 || { τ_vessel(v) | v:V • v ∈ vs }
333 || { τ_land_truck(lt) | lt:LT • lt ∈ lts }
334 || { τ_container(c) | c:CON • c ∈ cs }
335 || { τ_terminal_port(tp) | tp:TP • tp ∈ tps }

```

**K.6.10.2 Initial Vessels**

- 336. The signature of the `i_vessel_translation` function is simple: a  $\tau_{\text{translator}}$  from enduring vessel parts  $v$  to perdurant vessel behaviours, i.e., **Unit**.
- 337. The transcendental deduction then consists of obtaining the proper arguments for the vessel behaviour –
- 338. and invoking that behaviour.

---

<sup>22</sup>As for state: I need to update the container attribute section, Sect. K.5.6.11 on page 334 to reflect a state (for example: the component contents of a container)

**value**

```

336 $\tau_{\text{vessel}}: V \rightarrow \mathbf{Unit}$
336 $\tau_{\text{vessel}}(v) \equiv$
337 let $v_{\text{ui}} = \text{uid}_V(v)$, $v_{\text{mer}} = \text{mereo}_V(v)$,
337 $v_{\text{sta}} = \text{attr}_V\text{Sta}(v)$, $v_{\text{pos}} = \text{attr}_V\text{Pos}(v)$,
337 $v_{\text{csa}} = \text{attr}_V\text{CSA}(v)$, $v\sigma = \text{attr}_V\Sigma(v)$ in
338 $\text{vessel}(v_{\text{ui}}, v_{\text{mer}}, v_{\text{sta}})(v_{\text{pos}}, v_{\text{csa}}, v\sigma)$ end

```

### K.6.10.3 Initial Land Trucks

Similarly:

```

 $\tau_{\text{land_truck}}: \text{LT} \rightarrow \mathbf{Unit}$
 $\tau_{\text{land_truck}}(\text{lt}) \equiv$
 let $\text{lt}_{\text{ui}} = \text{uid}_{\text{LT}}(\text{lt})$, $\text{lt}_{\text{mer}} = \text{mereo}_{\text{LT}}(\text{lt})$,
 $\text{lt}_{\text{sta}} = \text{attr}_{\text{LT}}\text{Sta}(\text{lt})$, $\text{lt}_{\text{pos}} = \text{attr}_{\text{LT}}\text{Pos}(\text{lt})$,
 $\text{lt}_{\text{hold}} = \text{attr}_{\text{LT}}\text{Hold}(v)$, $\text{lt}\sigma = \text{attr}_{\text{LT}}\Sigma(\text{lt})$ in
 $\text{vessel}(\text{lt}_{\text{ui}}, \text{lt}_{\text{mer}}, \text{lt}_{\text{sta}})(\text{lt}_{\text{pos}}, \text{lt}_{\text{hold}}, \text{lt}\sigma)$ end

```

### K.6.10.4 Initial Containers

Similarly:

```

 $\tau_{\text{container}}: \text{CON} \rightarrow \mathbf{Unit}$
 $\tau_{\text{container}}(\text{con}) \equiv$
 let $c_{\text{ui}} = \text{uid}_{\text{CON}}(\text{con})$, $c_{\text{mer}} = \text{mereo}_{\text{CON}}(\text{con})$,
 $c_{\text{sta}} = \text{attr}_C\text{Sta}(\text{con})$, $c_{\text{pos}} = \text{attr}_C\text{Pos}(\text{con})$,
 $c\sigma = \text{attr}_{\text{CON}}\Sigma(\text{con})$ in
 $\text{container}(c_{\text{ui}}, c_{\text{mer}}, c_{\text{sta}})(c_{\text{pos}}, c\sigma)$ end

```

### K.6.10.5 Initial Terminal Ports

Terminal ports consists of a set of quay cranes, a set of quay trucks a set of stack cranes, and a set of stacks. They translate accordingly:

```

 $\tau_{\text{terminal_port}}: \text{TP} \rightarrow \mathbf{Unit}$
 $\tau_{\text{terminal_port}}(\text{tp}) \equiv$
 let $\text{qcs} = \text{obs_QCs}(\text{obs_QCS}(\text{tp}))$,
 $\text{qts} = \text{obs_QTS}(\text{obs_QTS}(\text{tp}))$,
 $\text{scs} = \text{obs_SCs}(\text{obs_SCS}(\text{tp}))$,
 $\text{stks} = \text{obs_STKs}(\text{obs_STKS}(\text{tp}))$ in
	{ $\tau_{\text{quay_crane}}(\text{qc}) \mid \text{qc}:\text{QC} \bullet \text{qc} \in \text{qcs}$ }	
	{ $\tau_{\text{quay_truck}}(\text{qt}) \mid \text{qt}:\text{QT} \bullet \text{qt} \in \text{qts}$ }	
	{ $\tau_{\text{stack_crane}}(\text{sc}) \mid \text{sc}:\text{SC} \bullet \text{sc} \in \text{scs}$ }	
	{ $\tau_{\text{stack}}(\text{stk}) \mid \text{stk}:\text{STK} \bullet \text{stk} \in \text{stks}$ } end	

```

### K.6.10.6 Initial Quay Cranes

```

 $\tau_{\text{quay_crane}}: \text{QC} \rightarrow \mathbf{Unit}$
 $\tau_{\text{quay_crane}}(\text{qc}) \equiv$
 let $\text{qc}_{\text{ui}} = \text{uid}_{\text{QC}}(\text{qc})$, $\text{qc}_{\text{mer}} = \text{mereo}_{\text{QC}}(\text{qc})$,
 $\text{qc}_{\text{sta}} = \text{attr}_{\text{QC}}\text{Sta}(\text{qc})$, $\text{qc}_{\text{pos}} = \text{attr}_{\text{QC}}\text{Pos}(\text{qc})$,
 $\text{qc}\sigma = \text{attr}_{\text{QC}}\Sigma(\text{qc})$ in
 $\text{quay_crane}(\text{qc}_{\text{ui}}, \text{qc}_{\text{mer}}, \text{qc}_{\text{sta}})(\text{qc}_{\text{pos}}, \text{qc}\sigma)$ end

```

### K.6.10.7 Initial Quay Trucks

```

 $\tau_{\text{quay_truck}}: \text{QT} \rightarrow \mathbf{Unit}$
 $\tau_{\text{quay_truck}}(\text{qt}) \equiv$
 let $\text{qt}_{\text{ui}} = \text{uid}_{\text{QT}}(\text{qt})$, $\text{qt}_{\text{mer}} = \text{mereo}_{\text{QT}}(\text{qt})$,
 $\text{qt}_{\text{sta}} = \text{attr}_{\text{QT}}\text{Sta}(\text{qt})$, $\text{qt}_{\text{pos}} = \text{attr}_{\text{QT}}\text{Pos}(\text{qt})$,
 $\text{qt}\sigma = \text{attr}_{\text{QT}}\Sigma(\text{qt})$ in
 $\text{quay_truck}(\text{qt}_{\text{ui}}, \text{qt}_{\text{mer}}, \text{qt}_{\text{sta}})(\text{qt}_{\text{pos}}, \text{qt}\sigma)$ end

```

**K.6.10.8 Initial Stack Cranes**

```

 $\tau_{\text{stack_crane}}$: SC \rightarrow Unit
 $\tau_{\text{stack_crane}}(\text{sc}) \equiv$
 let sc_ui = uid_SC(sc), sc_mer = mereo_SC(sc),
 sc_sta = attr_SC_Sta(sc), sc_pos = attr_SC_Pos(sc),
 sc σ = attr_SC Σ (sc) in
 container(sc_ui,sc_mer,sc_sta)(sc_pos,sc σ) end

```

**K.6.10.9 Initial Stacks**

```

 τ_{stack} : STK \rightarrow Unit
 $\tau_{\text{stack}}(\text{stk}) \equiv$
 let stk_ui = uid_STK(stk), stk_mer = mereo_STK(stk),
 stk_sta = attr_STK_Sta(stk),
 stk σ = attr_STK Σ (stk) in
 stack(stk_ui,stk_mer,stk_sta)(stk σ) end

```

**K.7 Conclusion**

|               |
|---------------|
| TO BE WRITTEN |
|---------------|

**K.7.1 An Interpretation of the Behavioural Description**

|               |
|---------------|
| TO BE WRITTEN |
|---------------|

**K.7.2 What Has Been Done**

|               |
|---------------|
| TO BE WRITTEN |
|---------------|

**K.7.3 What To Do Next**

|               |
|---------------|
| TO BE WRITTEN |
|---------------|

**K.7.4 Acknowledgements**

This report was begun when I was first invited to lecture, for three weeks in November 2018, at ECNU<sup>23</sup>, Shanghai, China. For this and for my actual stay at ECNU, I gratefully acknowledge Profs. He JiFeng, Zhu HuiBiao, Wang XiaoLing and Min Zhang. I chose at the time of the invitation to lead the course students through a major, non-trivial example. Since Shanghai is also one of the major container shipping ports of the world, and since the Danish company Maersk, through its subsidiary, APMTerminals, operates a major container terminal port, I decided on the subject for this experimental report. I gratefully acknowledge the support the ECNU course received from APMTerminals, through its staff, Messrs Henry Bai and Niels Roed.

---

<sup>23</sup>ECNU: East China Normal University

# Appendix L

## The Blue Skies

### Contents

---

|            |                               |            |
|------------|-------------------------------|------------|
| <b>L.1</b> | <b>Introduction</b>           | <b>361</b> |
| <b>L.2</b> | <b>Endurants</b>              | <b>362</b> |
| L.2.1      | <b>External Qualities</b>     | 362        |
| L.2.1.1    | <b>Parts and Fluids</b>       | 362        |
| L.2.1.2    | <b>The Air State</b>          | 362        |
| L.2.2      | <b>Internal Qualities</b>     | 362        |
| L.2.2.1    | <b>Unique Identifiers</b>     | 362        |
| L.2.2.1.1  | <b>Observers</b>              | 362        |
| L.2.2.1.2  | <b>All Unique Identifiers</b> | 362        |
| L.2.2.1.3  | <b>Axioms</b>                 | 362        |
| L.2.2.2    | <b>Mereology</b>              | 362        |
| L.2.2.2.1  | <b>Observers</b>              | 362        |
| L.2.2.2.2  | <b>Axioms</b>                 | 362        |
| L.2.2.3    | <b>Attributes</b>             | 362        |
| <b>L.3</b> | <b>Perdurants</b>             | <b>362</b> |
| L.3.1      | <b>Channels</b>               | 362        |
| L.3.2      | <b>Behaviours</b>             | 362        |
| L.3.3      | <b>Signatures</b>             | 362        |
| L.3.4      | <b>Definitions</b>            | 362        |
| L.3.5      | <b>System</b>                 | 362        |
| <b>L.4</b> | <b>Conclusion</b>             | <b>362</b> |

---

### L.1 Introduction

Some early work on this domain was reported in 1995 [19]. From Appendix B of [58] we “lift” Fig. B.1 Page 349, cf. Fig. L.1 on the following page.

The aim of this chapter is to [eventually] present a model of the air traffic domain hinted at in Fig. L.1 on the next page.

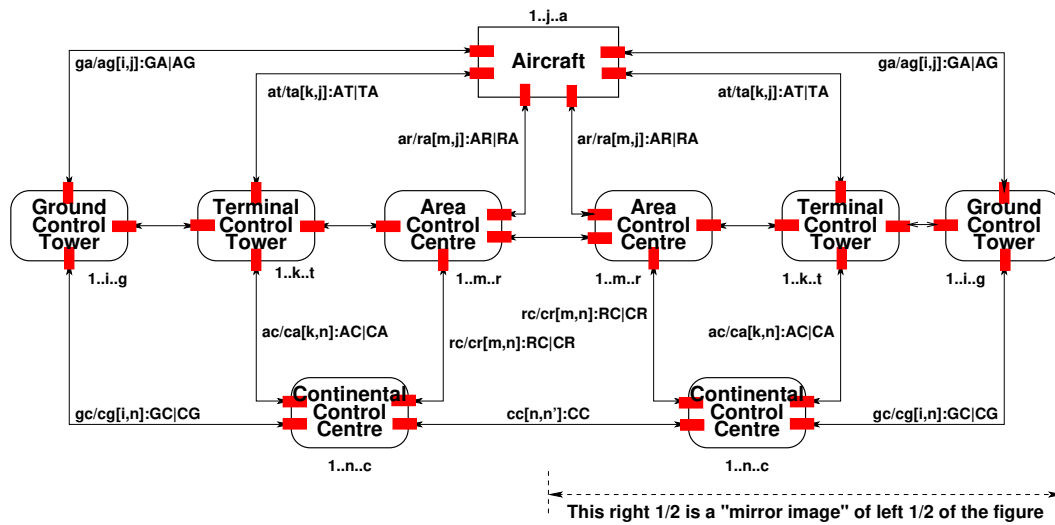


Figure L.1: A schematic air traffic system

## L.2 Endurants

### L.2.1 External Qualities

#### L.2.1.1 Parts and Fluids

#### L.2.1.2 The Air State

### L.2.2 Internal Qualities

#### L.2.2.1 Unique Identifiers

##### L.2.2.1.1 Observers

##### L.2.2.1.2 All Unique Identifiers

##### L.2.2.1.3 Axioms

#### L.2.2.2 Mereology

##### L.2.2.2.1 Observers

##### L.2.2.2.2 Axioms

#### L.2.2.3 Attributes

## L.3 Perdurants

### L.3.1 Channels

### L.3.2 Behaviours

### L.3.3 Signatures

### L.3.4 Definitions

### L.3.5 System

## L.4 Conclusion



# Appendix M

## Document Systems

I had, over the years, since mid 1990s, reflected upon the idea of “*what is a document ?*”. A most recent version, as I saw it in 2017, was “documented” in Chapter 7 [66]. But, preparing for my work, at TongJi University, Shanghai, September 2017, see Chapter Q, I reworked my earlier notes [66] into what is now this chapter.

### Contents

---

|        |                                                                  |     |
|--------|------------------------------------------------------------------|-----|
| M.1    | <b>Introduction</b>                                              | 364 |
| M.2    | <b>Managing, Archiving and Handling Documents</b>                | 365 |
| M.3    | <b>Principal Endurants</b>                                       | 365 |
| M.4    | <b>Unique Identifiers</b>                                        | 365 |
| M.5    | <b>Mereology</b>                                                 | 366 |
| M.6    | <b>Documents: A First View</b>                                   | 366 |
| M.6.1  | <b>Document Identifiers</b>                                      | 366 |
| M.6.2  | <b>Document Descriptors</b>                                      | 366 |
| M.6.3  | <b>Document Annotations</b>                                      | 366 |
| M.6.4  | <b>Document Contents: Text/Graphics</b>                          | 366 |
| M.6.5  | <b>Document Histories</b>                                        | 366 |
| M.6.6  | <b>A Summary of Document Attributes</b>                          | 366 |
| M.7    | <b>Behaviours: An Informal, First View</b>                       | 367 |
| M.8    | <b>Channels, A First View</b>                                    | 368 |
| M.9    | <b>An Informal Graphical System Rendition</b>                    | 369 |
| M.10   | <b>Behaviour Signatures</b>                                      | 369 |
| M.11   | <b>Time</b>                                                      | 369 |
| M.11.1 | <b>Time and Time Intervals: Types and Functions</b>              | 369 |
| M.11.2 | <b>A Time Behaviour and a Time Channel</b>                       | 370 |
| M.11.3 | <b>An Informal RSL Construct</b>                                 | 370 |
| M.12   | <b>Behaviour “States”</b>                                        | 370 |
| M.13   | <b>Inter-Behaviour Messages</b>                                  | 371 |
| M.13.1 | <b>Management Messages with Respect to the Archive</b>           | 371 |
| M.13.2 | <b>Management Messages with Respect to Handlers</b>              | 371 |
| M.13.3 | <b>Document Access Rights</b>                                    | 372 |
| M.13.4 | <b>Archive Messages with Respect to Management</b>               | 372 |
| M.13.5 | <b>Archive Message with Respect to Documents</b>                 | 372 |
| M.13.6 | <b>Handler Messages with Respect to Documents</b>                | 372 |
| M.13.7 | <b>Handler Messages with Respect to Management</b>               | 372 |
| M.13.8 | <b>A Summary of Behaviour Interactions</b>                       | 373 |
| M.14   | <b>A General Discussion of Handler and Document Interactions</b> | 373 |
| M.15   | <b>Channels: A Final View</b>                                    | 373 |

|                                                                                    |            |
|------------------------------------------------------------------------------------|------------|
| <b>M.16 An Informal Summary of Behaviours</b> . . . . .                            | <b>373</b> |
| M.16.1 <b>The Create Behaviour: Left Fig. M.3 on page 374</b> . . . . .            | 373        |
| M.16.2 <b>The Edit Behaviour: Right Fig. M.3 on page 374</b> . . . . .             | 374        |
| M.16.3 <b>The Read Behaviour: Left Fig. M.4 on page 375</b> . . . . .              | 374        |
| M.16.4 <b>The Copy Behaviour: Right Fig. M.4 on page 375</b> . . . . .             | 374        |
| M.16.5 <b>The Grant Behaviour: Left Fig. M.5 on page 375</b> . . . . .             | 375        |
| M.16.6 <b>The Shred Behaviour: Right Fig. M.5 on page 375</b> . . . . .            | 375        |
| <b>M.17 The Behaviour Actions</b> . . . . .                                        | <b>376</b> |
| M.17.1 <b>Management Behaviour</b> . . . . .                                       | 376        |
| M.17.1.1 <b>Management Create Behaviour: Left Fig. M.3 on page 374</b> . . . . .   | 376        |
| M.17.1.2 <b>Management Copy Behaviour: Right Fig. M.4 on page 375</b> . . . . .    | 377        |
| M.17.1.3 <b>Management Grant Behaviour: Left Fig. M.5 on page 375</b> . . . . .    | 378        |
| M.17.1.4 <b>Management Shared Behaviour: Right Fig. M.5 on page 375</b> . . . . .  | 378        |
| M.17.2 <b>Archive Behaviour</b> . . . . .                                          | 378        |
| M.17.2.1 <b>The Archive Create Behaviour: Left Fig. M.3 on page 374</b> . . . . .  | 379        |
| M.17.2.2 <b>The Archive Copy Behaviour: Right Fig. M.4 on page 375</b> . . . . .   | 379        |
| M.17.2.3 <b>The Archive Shred Behaviour: Right Fig. M.5 on page 375</b> . . . . .  | 379        |
| M.17.3 <b>Handler Behaviours</b> . . . . .                                         | 380        |
| M.17.3.1 <b>The Handler Create Behaviour: Left Fig. M.3 on page 374</b> . . . . .  | 380        |
| M.17.3.2 <b>The Handler Edit Behaviour: Right Fig. M.3 on page 374</b> . . . . .   | 380        |
| M.17.3.3 <b>The Handler Read Behaviour: Left Fig. M.4 on page 375</b> . . . . .    | 381        |
| M.17.3.4 <b>The Handler Copy Behaviour: Right Fig. M.4 on page 375</b> . . . . .   | 381        |
| M.17.3.5 <b>The Handler Grant Behaviour: Left Fig. M.5 on page 375</b> . . . . .   | 381        |
| M.17.4 <b>Document Behaviours</b> . . . . .                                        | 381        |
| M.17.4.1 <b>The Document Edit Behaviour: Right Fig. M.3 on page 374</b> . . . . .  | 382        |
| M.17.4.2 <b>The Document Read Behaviour: Left Fig. M.4 on page 375</b> . . . . .   | 382        |
| M.17.4.3 <b>The Document Shred Behaviour: Right Fig. M.5 on page 375</b> . . . . . | 382        |
| M.17.5 <b>Conclusion</b> . . . . .                                                 | 382        |
| <b>M.18 Documents in Public Government</b> . . . . .                               | <b>383</b> |
| <b>M.19 Documents in Urban Planning</b> . . . . .                                  | <b>383</b> |

We domain analyse and suggest a description of a domain of documents. We emphasize that the model is one of several possible. Common to these models is that we model “all” we can say about documents – irrespective of whether it can also be “implemented”! The model(s) are not requirements prescriptions – but we can develop such from our domain description.

You may find that the model is overly detailed with respect to a number of “operations” and properties of documents. We find that these operations must be part of the very basis of a document domain in order to cope with documents such as they occur in, for example, public government, see Appendix sect. M.18, or in urban planning, see Appendix Sect. M.19.

## M.1 Introduction

We analyse a notion of documents. Documents such as they occur in daily life. What can we say about documents – regardless of whether we can actually provide compelling evidence for what we say! That is: we model documents, not as electronic entities — which they are becoming, more-and-more, but as if they were manifest entities. When we, for example, say that “*this document was recently edited by such-and-such and the changes of that editing with respect to the text before is such-and-such*”, then we can, of course, always claim so, even if it may be difficult or even impossible to verify the claim. It is a fact, although maybe not demonstrably so, that there was a version of any document before an edit of that document. It is a fact that some handler did the editing. It is a fact that the editing took place at (or in) exactly such-and-such a time (interval), etc. We model such facts.

This research note unravels its analysis &<sup>1</sup> description in stages.

<sup>1</sup>We use the logo gram & between two terms, A & B, when we mean to express one meaning.

## M.2 Managing, Archiving and Handling Documents

The title of this section: *A System for Managing, Archiving and Handling Documents* immediately reveals the major concepts: That we are dealing with a *system* that **manages**, **archives** and **handles documents**. So what do we mean by **managing**, **archiving** and **handling** documents, and by **documents**? We give an ultra short survey. The survey relies on your prior knowledge of what you think documents are! **Management** decides<sup>2</sup> to direct **handlers** to work on **documents**. **Management** first directs the document archive to **create documents**. The document **archive creates documents**, as requested by **management**, and informs management of the **unique document identifiers** (by means of which handlers can handle these documents). **Management** then **grants** its designated **handler(s) access rights to documents**, these access rights enable handlers to **edit**, **read** and **copy** documents. The **handlers' editing and reading of documents** is accomplished by the **handlers** "working directly" with the **documents** (i.e., synchronising and communicating with **document behaviours**). The **handlers' copying of documents** is accomplished by the **handlers** requesting **management**, in collaboration with the **archive** behaviour, to do so.

## M.3 Principal Endurants

By an *endurant* we shall understand "an entity that can be observed or conceived and described as a "complete thing" at no matter which given snapshot of time." Were we to "freeze" time we would still be able to observe the entire endurant. This characterisation of what we mean by an 'endurant' is from [51, Manifest Domains: Analysis & Description]. We begin by identifying the principal endurants.

339. From document handling systems one can observe aggregates of handlers and documents.

We shall refer to 'aggregates of handlers' by M, for management, and to 'aggregates of documents' by A, for archive.

340. From aggregates of handlers (i.e., M) we can observe sets of handlers (i.e., H).

341. From aggregates of documents (i.e., A) we can observe sets of documents (i.e., D).

**type**

339 S, M, A

**value**

339 obs\_M: S → M

339 obs\_A: S → A

**type**

340 H, Hs = H-set

341 D, Ds = D-set

**value**

340 obs\_Hs: M → Hs

341 obs\_Ds: A → Ds

## M.4 Unique Identifiers

The notion of unique identifiers is treated, at length, in [51, Manifest Domains: Analysis & Description].

342. We associate unique identifiers with aggregate, handler and document endurants.

343. These can be observed from respective parts<sup>3</sup>.

**type**

342 MI<sup>4</sup>, AI<sup>5</sup>, HI, DI

**value**

343 uid\_MI<sup>6</sup>: M → MI

343 uid\_AI<sup>7</sup>: A → AI

343 uid\_HI: H → HI

343 uid\_DI: D → DI

As reasoned in [51, Manifest Domains: Analysis & Description], the unique identifiers of endurant parts are indeed unique: No two parts, whether composite, as are the aggregates, or atomic, as are handlers and documents, can have the same unique identifiers.

<sup>2</sup>How these decisions come about is not shown in this research note – as it has nothing to do with the essence of document handling, but, perhaps, with 'management'.

<sup>3</sup>[51, Manifest Domains: Analysis & Description] explains how 'parts' are the discrete endurants with which we associate the full complement of properties: unique identifiers, mereology and attributes.

<sup>4</sup>We shall not, in this research note, make use of the (one and only) management identifier.

<sup>5</sup>We shall not, in this research note, make use of the (one and only) archive identifier.

<sup>6</sup>Cf. Footnote 4: hence we shall not be using the uid\_MI observer.

<sup>7</sup>Cf. Footnote 5: hence we shall not be using the uid\_AI observer.

## M.5 Mereology

TO BE WRITTEN

## M.6 Documents: A First View

A document is a written, drawn, presented, or memorialized representation of thought. The word originates from the Latin *documentum*, which denotes a “teaching” or “lesson”.<sup>8</sup> We shall, for this research note, take a document in its written and/or drawn form. In this section we shall survey the concept a documents.

### M.6.1 Document Identifiers

Documents have *unique identifiers*. If two or more documents have the same document identifier then they are the same, one (and not two or more) document(s).

### M.6.2 Document Descriptors

With documents we associate *document descriptors*. We do not here stipulate what document descriptors are other than saying that when a document is **created** it is provided with a descriptor and this descriptor “remains” with the document and never changes value. In other words, it is a static attribute.<sup>9</sup> We do, however, include, in document descriptors, that the document they describe was initially based on a set of zero, one or more documents – identified by their unique identifiers.

### M.6.3 Document Annotations

With documents we also associate *document annotations*. By a document annotation we mean a programmable attribute, that is, an attribute which can be ‘augmented’ by document handlers. We think of document annotations as “incremental”, that is, as “adding” notes “on top of” previous notes. Thus we shall model document annotations as a repository: notes are added, i.e., annotations are augmented, previous notes are not edited, and no notes are deleted. We suggest that notes be time-stamped. The notes (of annotations) may be such which record handlers work on documents. Examples could be: “March 12, 2024: 10:48 am: This is version V.”, “This document was released on March 12, 2024: 10:48 am.”, “March 12, 2024: 10:48 am: Section X.Y.Z of version III was deleted.”, “March 12, 2024: 10:48 am: References to documents  $doc_i$  and  $doc_j$  are inserted on Pages  $p$  and  $q$ , respectively.” and “March 12, 2024: 10:48 am: Final release.”

### M.6.4 Document Contents: Text/Graphics

The main idea of a document, to us, is the *written* (i.e., text) and/or *drawn* (i.e., graphics) *contents*. We do not characterise any format for this *contents*. We may wish to insert, in the *contents*, references to locations in the *contents* of other documents. But, for now, we shall not go into such details. The main operations on documents, to us, are concerned with: their **creation, editing, reading, copying** and **shredding**. The **editing** and **reading** operations are mainly concerned with document *annotations* and *text/graphics*.

### M.6.5 Document Histories

So documents are **created, edited, read, copied** and **shredded**. These operations are initiated by the management (**create**), by the archive (**create**), and by handlers (**edit, read, copy**), and at specific times.

### M.6.6 A Summary of Document Attributes

- 344. As separate attributes of documents we have document descriptors, document annotations, document contents and document histories.
- 345. Document annotations are lists of document notes.
- 346. Document histories are lists of time-stamped document operation designators.
- 347. A document operation designator is either a **create**, or an **edit**, or a **read**, or a **copy**, or a **shred** designator.
- 348. A **create** designator identifies

<sup>8</sup>From: <https://en.wikipedia.org/wiki/Document>

<sup>9</sup>You may think of a document descriptor as giving the document a title; perhaps one or more authors; perhaps a physical address (of, for example, these authors); an initial date; as expressing whether the document is a research, or a technical report, or other; who is issuing the document (a public institution, a private firm, an individual citizen, or other); etc.

- (a) a handler and a time (at which the create request first arose), and presents
  - (b) elements for constructing a document descriptor, one which
    - i. besides some further undefined information
    - ii. refers to a set of documents (i.e., embeds reference to their unique identifiers),
  - (c) a (first) document note, and
  - (d) an empty document contents.
349. An edit designator identifies a handler, a time, and specifies a pair of edit/undo functions.
350. A read designator identifies a handler.
351. A copy designator identifies a handler, a time, the document to be copied (by its unique identifier, and a document note to be inserted in both the master and the copy document).
352. A shred designator identifies a handler.
353. An edit function takes a triple of a document annotation, a document note and document contents and yields a pair of a document annotation and a document contents.
354. An undo function takes a pair of a document note and document contents and yields a triple of a document annotation, a document note and a document contents.
355. Proper pairs of (edit,undo) functions satisfy some inverse relation.

There is, of course, no need, in any document history, to identify the identifier of that document.

**type**  
 344 DD, DA, DC, DH  
**value**  
 344 attr\_DD: D → DD  
 344 attr\_DA: D → DA  
 344 attr\_DC: D → DC  
 344 attr\_DH: D → DH  
**type**  
 345 DA = DN\*  
 346 DH = (TIME × DO)\*  
 347 DO == Crea | Edit | Read | Copy | Shre  
 348 Crea :: (HI × TIME) × (DI-set × Info) × DN × {"empty\_DC"}  
 348(b)i Info = ...  
**value**  
 348(b)ii embed\_DIs\_in\_DD: DI-set × Info → DD  
**axiom**  
 348d "empty\_DC" ∈ DC  
**type**  
 349 Edit :: (HI × TIME) × (EDIT × UNDO)  
 350 Read :: (HI × TIME) × DI  
 351 Copy :: (HI × TIME) × DI × DN  
 352 Shre :: (HI × TIME) × DI  
 353 EDIT = (DA × DN × DC) → (DA × DC)  
 354 UNDO = (DA × DC) → (DA × DN × DC)  
**axiom**  
 355  $\forall$  mkEdit(⟦, (e,u)):Edit •  
 355  $\forall$  (da,dn,dc):(DA×DN×DC) •  
 355 u(e(da,dn,dc))=(da,dn,dc)

## M.7 Behaviours: An Informal, First View

In [51, Manifest Domains: Analysis & Description] we show that we can associate behaviours with parts, where parts are such discrete endurants for which we choose to model all its observable properties: unique identifiers, mereology and attributes, and where behaviours are sequences of actions, events and behaviours.

- The overall document handler system behaviour can be expressed in terms of the parallel composition of the behaviours
  - 356. of the system core behaviour,
  - 357. of the handler aggregate (the management) behaviour
  - 358. and the document aggregate (the archive) behaviour,
 with the (distributed) parallel composition of

- 359. all the behaviours of handlers and,
  - the (distributed) parallel composition of
  - 360. at any one time, zero, one or more behaviours of documents.
- To express the latter
  - 361. we need introduce two “global” values: an indefinite set of handler identifiers and an indefinite set of document identifiers.

**value**

361 his:HI-set, dis:DI-set

```

356 sys(...)
357 || mgmt(...)
358 || arch(...)
359 || ||{hdlri(...)|i:HI•i∈his}
360 || ||{docui(dd)(da,dc,dh)|i:DI•i∈dis}
```

For now we leave undefined the arguments, (...) etc., of these behaviours. The arguments of the document behaviour, (dd)(da,dc,dh), are the static, respectively the three programmable (i.e., dynamic) attributes: *document descriptor*, *document annotation*, *document contents* and *document history*. The above expressions, Items 357–360, do not define anything, they can be said to be “snapshots” of a “behaviour state”. Initially there are no document behaviours, docu<sub>i</sub>(dd)(da,dc,dh), Item 360. Document behaviours are “started” by the archive behaviour (on behalf of the management and the handler behaviours). Other than mentioning the system (core) behaviour we shall not model that behaviour further.

## M.8 Channels, A First View

Channels are means for behaviours to synchronise and communicate values (such as unique identifiers, mereologies and attributes).

362. The management behaviour, **mgmt**, need to (synchronise and) communicate with the archive behaviour, **arch**, in order, for the management behaviour, to request the archive behaviour
- to **create** (ab initio or due to **copying**)
  - or **shred** document behaviours, docu<sub>j</sub>,
- and for the archive behaviour
- to inform the management behaviour of the identity of the document( behaviour)s that it has created.

**channel**

362 mgmt\_arch\_ch:MA

363. The management behaviour, **mgmt**, need to (synchronise and) communicate with all handler behaviours, hdlr<sub>i</sub> and they, in turn, to (synchronised) communicate with the handler management behaviour, **mgmt**. The management behaviour need to do so in order
- to inform a handler behaviour that it is granted access rights to a specific document, subsequently these access rights may be modified, including revoked.

**channel**

363 {mgmt\_hdlr\_ch[i]:MH|i:HI•i ∈ his}

364. The document archive behaviour, **arch**, need (synchronise and) communicate with all document behaviours, docu<sub>j</sub> and they, in turn, to (synchronise and) communicate with the archive behaviour, **arch**.

**channel**

364 {arch\_docu\_ch[j]:AD|h:DI•j ∈ dis}

365. Handler behaviours, hdlr<sub>i</sub>, need (synchronise and) communicate with all the document behaviours, docu<sub>j</sub>, with which it has operational allowance to so do so<sup>10</sup>, and document behaviours, docu<sub>j</sub>, need (synchronise and) communicate with potentially all handler behaviours, hdlr<sub>i</sub>, namely those handler behaviours, hdlr<sub>i</sub> with which they have (“earlier” synchronised and) communicated.

**channel**

365 {hdlr\_docu\_ch[i,j]:HD|i:HI,j:DI•i ∈ his∧j ∈ dis}

366. At present we leave undefined the type of messages that are communicated.

**type**

366 MA, MH, AD, HD

<sup>10</sup>The notion of operational allowance will be explained below.

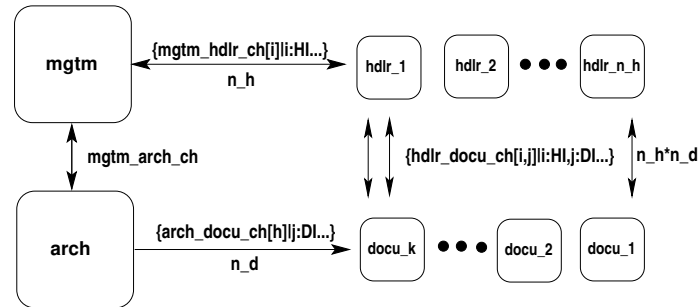


Figure M.1: An Informal Snapshot of System Behaviours

## M.9 An Informal Graphical System Rendition

Figure M.1 is an informal rendition of the “state” of a number of behaviours: a single management behaviour, a single archive behaviour, a fixed number,  $n_h$ , of one or more handler behaviours, and a variable, initially zero number of document behaviours, with a maximum of these being  $n_d$ . The figure also indicates, again rather informally, the channels between these behaviours: one channel between the management and the archive behaviours;  $n_h$  channels ( $n_h$  is, again, informally indicated) between the management behaviour and the  $n_h$  handler behaviours;  $n_d$  channels ( $n_d$  is, again, informally indicated) between the archive behaviour and the  $n_d$  document behaviours; and  $n_h \times n_d$  channels ( $n_h \times n_d$  is, again, informally indicated) between the  $n_h$  handler behaviours and the  $n_d$  document behaviours

## M.10 Behaviour Signatures

367. The `mgmt` behaviour (synchronises and) communicates with the `arch` behaviour and with all of the handler behaviours, `hdri`.
368. The `arch` behaviour (synchronises and) communicates with the `mgmt` behaviour and with all of the document behaviours, `docuj`.
369. The signature of the generic handler behaviours, `hdri` expresses that they [occasionally] receive “orders” from management, and otherwise [regularly] interacts with document behaviours.
370. The signature of the generic document behaviours, `docuj` expresses that they [occasionally] receive “orders” from the `arch` behaviour and that they [regularly] interacts with handler behaviours.

### value

- 367 `mgmt`: ... → **in,out** `mgmt_arch_ch`, {`mgmt_hdr_ch[i]:HI`• $i \in \text{his}$ } **Unit**  
 368 `arch`: ... → **in,out** `mgmt_arch_ch`, {`arch_docu_ch[j]:DI`• $j \in \text{dis}$ } **Unit**  
 369 `hdri`: ... → **in** `mgmt_hdr_ch[i]`, **in,out** {`hdr_docu_ch[i,j]:DI`• $j \in \text{dis}$ } **Unit**  
 370 `docuj`: ... → **in** `mgmt_arch_ch`, **in,out** {`hdr_docu_ch[i,j]:HI`• $i \in \text{his}$ } **Unit**

## M.11 Time

### M.11.1 Time and Time Intervals: Types and Functions

371. We postulate a notion of time, one that covers both a calendar date (from before Christ up till now and beyond). But we do not specify any concrete type (i.e., format such as: YY:MM:DD, HH:MM:SS).
372. And we postulate a notion of (signed) time interval — between two times (say:  $\pm$ YY:MM:DD:HH:MM:SS).
373. Then we postulate some operations on time: Adding a time interval to a time obtaining a time; subtracting one time from another time obtaining a time interval, multiplying a time interval with a natural number; etc.
374. And we postulate some relations between times and between time intervals.

**type**

371 TIME

372 TIME\_INTERVAL

**value**373 add: TIME\_INTERVAL  $\times$  TIME  $\rightarrow$  TIME373 sub: TIME  $\times$  TIME  $\rightarrow$  TIME\_INTERVAL373 mpy: TIME\_INTERVAL  $\times$  Nat  $\rightarrow$  TIME\_INTERVAL374 <,≤,=,≠,≥,>: ((TIME $\times$ TIME)|(TIME\_INTERVAL $\times$ TIME\_INTERVAL))  $\rightarrow$  Bool

## M.11.2 A Time Behaviour and a Time Channel

375. We postulate a [n “ongoing”] time behaviour: it either keeps being a time behaviour with unchanged time,  $t$ , or – internally non-deterministically – chooses being a time behaviour with a time interval incremented time,  $t+ti$ , or – internally non-deterministically – chooses to [first] offer its time on a [global] channel, `time_ch`, then resumes being a time behaviour with unchanged time,  $t$

376. The time interval increment,  $ti$ , is likewise internally non-deterministically chosen. We would assume that the increment is “infinitesimally small”, but there is no need to specify so.

377. We also postulate a channel, `time_ch`, on which the time behaviour offers time values to whoever so requests.

**value**375 `time`: TIME  $\rightarrow$  `time_ch` TIME Unit375 `time`( $t$ )  $\equiv$  (`time`( $t$ )  $\square$  `time`( $t+ti$ )  $\square$  `time_ch`! $t$  ; `time`( $t$ ))376 `ti`:TIME\_INTERVAL ...**channel**377 `time_ch`:TIME

## M.11.3 An Informal RSL Construct

The formal-looking specifications of this report appear in the style of the RAISE [101] Specification Language, RSL [100]. We shall be making use of an informal language construct:

- **wait**  $ti$ .

**wait** is a keyword;  $ti$  designates a time interval. A typical use of the wait construct is:

- ...  $ptA$  ; **wait**  $ti$ ;  $ptB$  ; ...

If at specification text point  $ptA$  we may assert that time is  $t$ , then at specification text point  $ptB$  we can assert that time is  $t+ti$ .

## M.12 Behaviour “States”

We recall that the enduring parts, Management, Archive, Handlers, and Documents, have properties in the form of *unique identifiers*, *mereologies* and *attributes*. We shall not, in this research note, deal with possible mereologies of these enduring parts. In this section we shall discuss the enduring attributes of `mgmt` (management), `arch` (archive), `hdlrs` (handlers), and `docu` (documents). Together the values of these properties, notably the attributes, constitute states – and, since we associate behaviours with these enduring parts, we can refer to these states also a behaviour states. Some attributes are static, i.e., their value never changes. Other attributes are dynamic.<sup>11</sup> Document handling systems are rather conceptual, i.e., abstract in nature. The dynamic attributes, therefore, in this modeling “exercise”, are constrained to just the *programmable* attributes. Programmable attributes are those whose value is set by “their” behaviour. For a behaviour  $\beta$  we shall show the static attributes as one set of parameters and the programmable attributes as another set of parameters.

**value**  $\beta$ : Static  $\rightarrow$  Program  $\rightarrow$  ... Unit

378. For the management enduring/behaviour we focus on one programmable attribute. The management behaviour needs keep track of all the handlers it is charged with, and for each of these which zero, one or more documents they have been granted access to (cf. Sect. M.13.3 on page 372). Initially that management directory lists a number of handlers, by their identifiers, but with no granted documents.

379. For the archive behaviour we similarly focus on one programmable attribute. The archive behaviour needs keep track of all the documents it has used (i.e., created), those that are available (and not yet used), and of those it has shredded. Initially all these three archive directory sets are empty.

<sup>11</sup>We refer to Sect. 3.4 of [51], and in particular its subsection 3.4.4.



380. For the handler behaviour we similarly focus on one programmable attribute. The handler behaviour needs keep track of all the documents it has been charged with and its access rights to these.
381. Document attributes we mentioned above, cf. Items 344–347.

**type**

378 MDIR = HI  $\overrightarrow{m}$  (DI  $\overrightarrow{m}$  ANm-set)  
 379 ADIR = avail:DI-set  $\times$  used:DI-set  $\times$  gone:DI-set  
 380 HDIR = DI  $\overrightarrow{m}$  ANm-set  
 381 SDATR = DD, PDATR = DA  $\times$  DC  $\times$  DH

**axiom**

379  $\forall$  (avail,used,gone):ADIR • avail  $\cap$  used = {}  $\wedge$  gone  $\subseteq$  used

We can now “complete” the behaviour signatures. We omit, for now, static attributes.

**value**

367 mgmt: MDIR  $\rightarrow$  in,out mgmt\_arch\_ch, {mgmt\_hdlr\_ch[i]|i:HI•i  $\in$  his} **Unit**  
 368 arch: ADIR  $\rightarrow$  in,out mgmt\_arch\_ch, {arch\_docu\_ch[j]|j:DI•j  $\in$  dis} **Unit**  
 369 hdlr<sub>i</sub>: HDIR  $\rightarrow$  in mgmt\_hdlr\_ch[i], in,out {hdlr\_docu\_ch[i,j]|j:DI•j  $\in$  dis} **Unit**  
 370 docu<sub>j</sub>: SDATR  $\rightarrow$  PDATR  $\rightarrow$  in mgmt\_arch\_ch, in,out {hdlr\_docu\_ch[i,j]|i:HI•i  $\in$  his} **Unit**

## M.13 Inter-Behaviour Messages

Documents are not “fixed, innate” entities. They embody a “history”, they have a “past”. Somehow or other they “carry a trace of all the ”things” that have happened/occurred to them. And, to us, these things are the manipulations that management, via the archive and handlers perform on documents.

### M.13.1 Management Messages with Respect to the Archive

382. Management **create** documents. It does so by requesting the archive behaviour to allocate a document identifier and initialize the document “state” and start a document behaviour, with initial information, cf. Item 348 on page 366:
- the identity of the initial handler of the document to be created,
  - the time at which the request is being made,
  - a document descriptor which embodies a (finite) set of zero or more (used) document identifiers (dis),
  - a document annotation note dn, and
  - an initial, i.e., “empty” contents, “empty\_DC”.

**type**

348. Crea :: (HI  $\times$  TIME)  $\times$  (DI-set  $\times$  Info)  $\times$  DN  $\times$  {“empty\_DC”} [cf. formula Item 348, Page 367]

383. The management behaviour passes on to the archive behaviour, requests that it accepts from handlers behaviours, for the copying of document:

383 Copy :: DI  $\times$  HI  $\times$  TIME  $\times$  DN [cf. Item 393 on the following page]

384. Management **schreds** documents by informing the archive behaviour to do so.

**type**

384 Shred :: TIME  $\times$  DI

### M.13.2 Management Messages with Respect to Handlers

385. Upon receiving, from the archive behaviour, the “feedback” the identifier of the created document (behaviour):

**type**

385 Create\_Reply :: NewDocID(di:DI)

386. the management behaviour decides to **grant** access rights, acrs:ACRS<sup>12</sup>, to a document handler, hi:HI.

**type**

386 Gran :: HI  $\times$  TIME  $\times$  DI  $\times$  ACRS

<sup>12</sup>For the concept of access rights see Sect. M.13.3 on the next page.

### M.13.3 Document Access Rights

Implicit in the above is a notion of document access rights.

387. By document access rights we mean a set of action names.

388. By an action name we mean such tokens that indicate either of the document handler operations indicate above.

**type**

387 ACRS = ANm-set

388 ANm = {"edit", "read", "copy"}

### M.13.4 Archive Messages with Respect to Management

To create a document management provides the archive with some initial information. The archive behaviour selects a document identifier that has not been used before.

389. The archive behaviour informs the management behaviour of the identifier of the created document.

**type**

389 NewDocID :: DI

### M.13.5 Archive Message with Respect to Documents

390. To shred a document the archive behaviour must access the designated document in order to **stop** it. No "message", other than a symbolic "**stop**", need be communicated to the document behaviour.

**type**

390 Shred :: {"stop"}

### M.13.6 Handler Messages with Respect to Documents

Handlers, generically referred to by  $hdlr_i$ , may perform the following operations on documents: **edit**, **read** and **copy**. (Management, via the archive behaviour, **creates** and **shreds** documents.)

391. To perform an **edit** action handler  $hdlr_i$  must provide the following:

- the document identity – in the form of a (i:HI,j:DI) channel `hdlr_docu_ch` index value,
- the handler identity,  $i$ ,
- the time of the edit request,
- and a pair of functions: one which performs the editing and one which un-does it!

**type**

391 Edit :: DI × HI × TIME × (EDIT × UNDO)

392. To perform a **read** action handler  $hdlr_i$  must provide the following information:

- the document identity – in the form of a di:DI channel `hdlr_docu_ch` index value,
- the handler identity and
- the time of the read request.

**type**

392 Read :: DI × HI × TIME

### M.13.7 Handler Messages with Respect to Management

393. To perform a **copy** action, a handler,  $hdlr_i$ , must provide the following information to the management behaviour, `mgmtm`:

- the document identity,
- the handler identity – in the form of an hi:HI channel `mgmtm_hdlr_ch` index value,
- the time of the copy request, and
- a document note (to be affixed both the master and the copy documents).

393 Copy :: DI × HI × TIME × DN [cf. Item 383 on the preceding page]

How the handler, the management, the archive and the "named other" handlers then enact the copying, etc., will be outlined later.

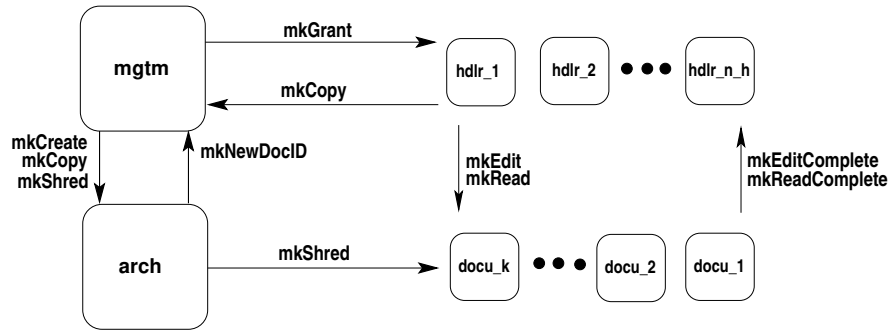


Figure M.2: A Summary of Behaviour Interactions

### M.13.8 A Summary of Behaviour Interactions

Figure M.2 summarises the sources, **out**, resp. !, and the targets, **in**, resp. ?, of the messages covered in the previous sections.

## M.14 A General Discussion of Handler and Document Interactions

We think of documents being manifest. Either a document is in paper form, or it is in electronic form. In paper form we think of a document as being in only one – and exactly one – physical location. In electronic form a document is also in only one – and exactly one – physical location. No two handlers can access the same document at the same time or in overlapping time intervals. If your conventional thinking makes you think that two or more handlers can, for example, read the same document “at the same time”, then, in fact, they are reading either a master and a copy of that master, or they are reading two copies of a common master.

## M.15 Channels: A Final View

We can now summarize the types of the various channel messages first referred to in Items 362, 363, 364 and 365.

- type
- 362 MA = Create (Item 382 on page 371)
- 362 | Shred (Item 382d on page 371)
- 362 | NewDocID (Item 389 on the facing page)
- 363 MH = Grant (Item 382c on page 371)
- 363 | Copy (Item 393 on the facing page)
- 364 AD = Shred (Item 390 on the preceding page)
- 365 HD = Edit (Item 391 on the facing page)
- 365 | Read (Item 392 on the preceding page)
- 365 | Copy (Item 393 on the facing page)

## M.16 An Informal Summary of Behaviours

### M.16.1 The Create Behaviour: Left Fig. M.3 on the next page

- 394. [1] The management behaviour, at its own volition, initiates a create document behaviour. It does so by offering a create document message to the archive behaviour.
  - (a) [1.1] That message contains a meaningful document descriptor,
  - (b) [1.2] an initial document annotation,
  - (c) [1.3] an “empty” document contents and

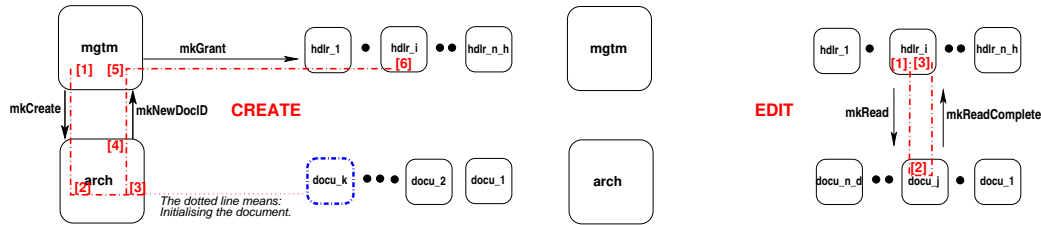


Figure M.3: Informal Snapshots of Create and Edit Document Behaviours

(d) [1.4] a single element document history.

(We refer to Sect. M.13.1 on page 371, Items 382–382e.)

395. [2] The archive behaviour offers to accept that management message. It then selects an available document identifier (here shown as  $k$ ), henceforth marking  $k$  as used.

396. [3] The archive behaviour then “spawns off” document behaviour  $\text{docu}_k$  – here shown by the “dash-dotted” rounded edge square.

397. [4] The archive behaviour then offers the document identifier  $k$  message to the management behaviour.

(We refer to Sect. M.13.4 on page 372, Item 389.)

398. [5] The management behaviour then

(a) [5.1] selects a handler, here shown as  $i$ , i.e.,  $\text{hdr}_i$ ,

(b) [5.2] records that that handler is granted certain access rights to document  $k$ ,

(c) [5.3] and offers that granting to handler behaviour  $i$ .

(We refer to Sect. M.13.2 on page 371, Item 386 on page 371.)

399. [6] Handler behaviour  $i$  records that it now has certain access rights to document  $i$ .

## M.16.2 The Edit Behaviour: Right Fig. M.3

1 Handler behaviour  $i$ , at its own volition, initiates an edit action on document  $j$  (where  $i$  has editing rights for document  $j$ ). Handler  $i$ , optionally, provides document  $j$  with a(annotation) note. While editing document  $j$  handler  $i$  also “selects” an appropriate pair of *edit/undo* functions for document  $j$ .

2 Document behaviour  $j$  accepts the editing request, enacts the editing, optionally appends the (annotation) note, and, with handler  $i$ , completes the editing, after some time interval  $t_i$ .

3 Handler behaviour  $i$  completes its edit action.

## M.16.3 The Read Behaviour: Left Fig. M.4 on the next page

1 Handler behaviour  $i$ , at its own volition, initiates a read action on document  $j$  (where  $i$  has reading rights for document  $j$ ). Handler  $i$ , optionally, provides document  $j$  with a(annotation) note.

2 Document behaviour  $j$  accepts the reading request, enacts the reading by providing the handler,  $i$ , with the document contents, and optionally appends the (annotation) note, and, with handler  $i$ , completes the reading, after some time interval  $t_i$ .

3 Handler behaviour  $i$  completes its read action.

## M.16.4 The Copy Behaviour: Right Fig. M.4 on the facing page

1 Handler behaviour  $i$ , at its own volition, initiates a copy action on document  $j$  (where  $i$  has copying rights for document  $j$ ). Handler  $i$ , optionally, provides master document  $j$  as well as the copied document (yet to be identified) with respective (annotation) notes.

2 The management behaviour offers to accept the handler message. As for the create action, the management behaviour offers a combined *copy and create* document message to the archive behaviour.

3 The archive behaviour selects an available document identifier (here shown as  $k$ ), henceforth marking  $k$  as used.

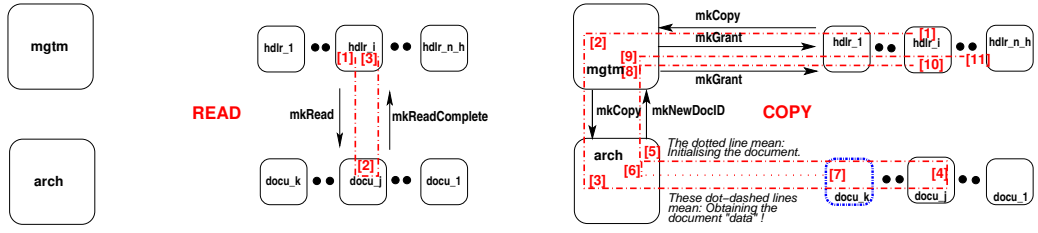


Figure M.4: Informal Snapshots of Read and Copy Document Behaviours

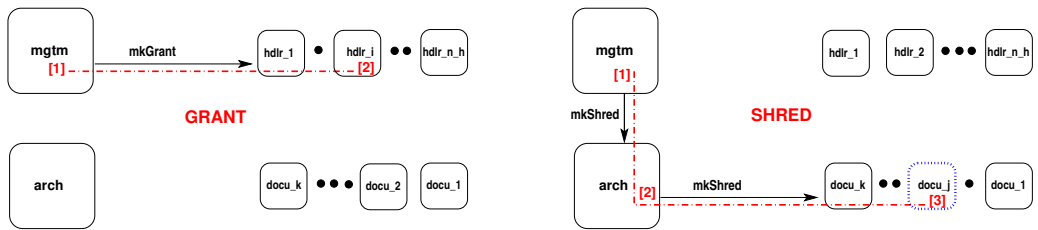


Figure M.5: Informal Snapshots of Grant and Shred Document Behaviours

- 4 The archive behaviour then obtains, from the master document  $j$  its *document descriptor*,  $dd_j$ , its *document annotations*,  $da_j$ , its *document contents*,  $dc_j$ , and its *document history*,  $dh_j$ .
- 5 The archive behaviour informs the management behaviour of the identifier,  $k$ , of the (new) document copy,
- 6 while assembling the attributes for that (new) document copy: its *document descriptor*,  $dd_k$ , its *document annotations*,  $da_k$ , its *document contents*,  $dc_k$ , and its *document history*,  $dh_k$ , from these “similar” attributes of the master document  $j$ ,
- 7 while then “spawning off” document behaviour  $docu_k$  – here shown by the “dash-dotted” rounded edge square.
- 8 The management behaviour accepts the identifier,  $k$ , of the (new) document copy, recording the identities of the handlers and their access rights to  $k$ ,
- 9 while informing these handlers (informally indicated by a “dangling” dash-dotted line) of their grants,
- 10 while also informing the master copy of the copy identity (et cetera).
- 11 The handlers granted access to the copy record this fact.

### M.16.5 The Grant Behaviour: Left Fig. M.5

This behaviour has its

- 1 Item [1] correspond, in essence, to Item [9] of the copy behaviour – see just above – and
- 2 Item [2] correspond, in essence, to Item [11] of the copy behaviour.

### M.16.6 The Shred Behaviour: Right Fig. M.5

- 1 The management, at its own volition, selects a document,  $j$ , to be shredded. It so informs the archive behaviour.
- 2 The archive behaviour records that document  $j$  is to be no longer in use, but shredded, and informs document  $j$ 's behaviour.
- 3 The document  $j$  behaviour accepts the shred message and **stops** (indicated by the dotted rounded edge box).

## M.17 The Behaviour Actions

To properly structure the definitions of the four kinds of (management, archive, handler and document) behaviours we single each of these out “across” the six behaviour traces informally described in Sects. M.16.1–M.16.6. The idea is that if behaviour  $\beta$  is involved in  $\tau$  traces,  $\tau_1, \tau_2, \dots, \tau_r$ , then behaviour  $\beta$  shall be defined in terms of  $r$  non-deterministic alternative behaviours named  $\beta_{\tau_1}, \beta_{\tau_2}, \dots, \beta_{\tau_r}$ .

### M.17.1 Management Behaviour

400. The management behaviour is involved in the following action traces:

- |                   |                                      |
|-------------------|--------------------------------------|
| (a) <b>create</b> | Fig. M.3 on page 374 Left            |
| (b) <b>copy</b>   | Fig. M.4 on the preceding page Right |
| (c) <b>grant</b>  | Fig. M.5 on the previous page Left   |
| (d) <b>shred</b>  | Fig. M.5 on the preceding page Right |

#### value

```

400 mgtm: MDIR → in,out mgtm_arch_ch, {mgtm_hdlr_ch[hi]|hi:HI•hi ∈ his} Unit
400 mgtm(mdir) ≡
400a mgtm_create(mdir)
400b [] mgtm_copy(mdir)
400c [] mgtm_grant(mdir)
400d [] mgtm_shred(mdir)

```

#### M.17.1.1 Management Create Behaviour: Left Fig. M.3 on page 374

401. The **management create** behaviour
402. initiates a create document behaviour (i.e., a request to the archive behaviour),
403. and then awaits its response.

#### value

```

401 mgtm_create: MDIR → in,out mgtm_arch_ch, {mgtm_hdlr_ch[hi]|hi:HI•hi ∈ his} Unit
401 mgtm_create(mdir) ≡
402 [1] let hi = mgtm_create_initiation(mdir) ; [Left Fig. M.3 on page 374]
403 [5] mgtm_create_awaits_response(mdir)(hi) end [Left Fig. M.3 on page 374]

```

The **management create initiation** behaviour

404. selects a handler on behalf of which it requests the document creation,
405. assembles the elements of the create message:
- by embedding a set of zero or more document references, **dis**, with some information, **info**, into a document descriptor, adding
  - a document note, **dn**, and
  - and initial, that is, empty document contents, "**empty\_DC**",
406. offers such a create document message to the archive behaviour, and
407. yields the identifier of the chosen handler.

#### value

```

402 mgtm_create_initiation: MDIR → in,out mgtm_arch_ch, {mgtm_hdlr_ch[hi]|hi:HI•hi ∈ his} HI
402 mgtm_create_initiation(mdir) ≡
404 let hi:HI • hi ∈ dom mdir,
405 [1.2–4] (dis,info):(DI-set×Info),dn:DN • is_meaningful(embed_DIs_in_DD(dis,info))(mdir) in
406 [1.1] mgtm_arch_ch ! mkCreate(embed_DIs_in_DD(ds,info),dn,"empty_DC")
407 hi end

```

405 is\_meaningful: DD → MDIR → Bool [left further undefined]

The **management create awaits response** behaviour

408. starts by awaiting a reply from the archive behaviour with the identity, **di**, of the document (that that behaviour has created).
409. It then selects suitable access rights,

- 410. with which it updates its handler/document directory
- 411. and offers to the chosen handler
- 412. whereupon it resumes, with the updated management directory, being the management behaviour.

**value**

```

403 mgmtm_create_awaits_response: MDIR → HI → in,out mgmtm_arch_ch, {mgmtm_hdlr_ch[hi]|hi:HI•hi ∈ his} Unit
403 mgmtm_create_awaits_response(mdir) ≡
408 [5] let mkNewDocID(di) = mgmtm_arch_ch ? in
409 [5.1] let acrs:ANm-set in
410 [5.2] let mdir' = mdir † [hi ↦ [di ↦ acrs]] in
411 [5.3] mgmtm_hdlr_ch[hi] ! mkGrant(di,acrs)
412 mgmtm(mdir') end end end

```

### M.17.1.2 Management Copy Behaviour: Right Fig. M.4 on page 375

- 413. The **management copy** behaviour
- 414. accepts a copy document request from a handler behaviour (i.e., a request to the archive behaviour),
- 415. and then awaits a response from the archive behaviour;
- 416. after which it grants access rights to handlers to the document copy.

**value**

```

413 mgmtm_copy: MDIR → in,out mgmtm_arch_ch, {mgmtm_hdlr_ch[hi]|hi:HI•hi ∈ his} Unit
413 mgmtm_copy(mdir) ≡
414 [2] let hi = mgmtm_accept_copy_request(mdir) in
415 [8] let di = mgmtm_awaits_copy_response(mdir)(hi) in
416 [9] mgmtm_grant_access_rights(mdir)(di) end end

```

- 417. The **management accept copy** behaviour non-deterministically externally ( $\square$ ) awaits a copy request from a[ny] handler ( $i$ ) behaviour –
- 418. with the request identifying the master document,  $j$ , to be copied.
- 419. The management accept copy behaviour forwards (!) this request to the archive behaviour –
- 420. while yielding the identity of the requesting handler.

```

417. mgmtm_accept_copy_request: MDIR →
417. in,out mgmtm_arch_ch, {mgmtm_hdlr_ch[hi]|hi:HI•hi ∈ his} HI
417. mgmtm_accept_copy_request(mdir) ≡
418. let mkCopy(di,hi,t,dn) = \square {mgmtm_hdlr_ch[i]?i:HI•i ∈ his} in
419. mgmtm_arch_ch ! mkCopy(di,hi,t,dn) ;
419. hi end

```

The **management awaits copy response** behaviour

- 421. awaits a reply from the archive behaviour as to the identity of the newly created copy ( $di$ ) of master document  $j$ .
- 422. The management awaits copy response behaviour then informs the ‘copying-requesting’ handler,  $hi$ , that the copying has been completed and the identity of the copy ( $di$ ) –
- 423. while yielding the identity,  $di$ , of the newly created copy.

```

400b. mgmtm_awaits_copy_response: MDIR → HI →
400b. in,out mgmtm_arch_ch, {mgmtm_hdlr_ch[hi]|hi:HI•hi ∈ his} DI
400b. mgmtm_awaits_copy_response(mdir)(hi) ≡
421. [8] let mkNewDocID(di) = mgmtm_arch_ch ? in
422. mgmtm_hdlr_ch[hi] ! mkCopy(di) ;
423. di end

```

The **management grants access rights** behaviour

- 424. selects suitable access rights for a suitable number of selected handlers.
- 425. It then offers these to the selected handlers.

```

416. mgmtm_grant_access_rights: MDIR → DI →
416. in,out {mgmtm_hdlr_ch[hi]|hi:HI•hi ∈ his} Unit
416. mgmtm_grant_access_rights(mdir)(di) ≡
424. let diarm = [hi→acrs|hi:HI,acrs:ANm-set• hi ∈ dom mdir∧acrs⊆(diarm(hi))(di)] in
425. || {mgmtm_hdlr_ch[hi]!mkGrant(hi,time_ch?,di,acrs) |
425. hi:HI,acrs:ANm-set•hi ∈ dom diarm∧acrs⊆(diarm(hi))(di)} end

```

### M.17.1.3 Management Grant Behaviour: Left Fig. M.5 on page 375

The **management grant** behaviour

- 426. is a variant of the `mgmt_grant_access_rights` function, Items 424–425.
- 427. The management behaviour selects a suitable subset of known handler identifiers, and
- 428. for these a suitable subset of document identifiers from which
- 429. it then constructs a map from handler identifiers to subsets of access rights.
- 430. With this the management behaviour then issues appropriate grants to the chosen handlers.

**type**

$\text{MDIR} = \text{HI} \multimap (\text{DI} \multimap \text{ANm-set})$

**value**

```

426 mgmt_grant: MDIR → in,out {mgmt_hdlr_ch[hi]|hi:HI•hi ∈ his} Unit
426 mgmt_grant(mdir) ≡
427 let his ⊆ dom dir in
428 let dis ⊆ ∪{dom mdir(hi)|hi:HI•hi ∈ his} in
429 let diarm = [hi→acrs|hi:HI,di:DI,acrs:ANm-set• hi ∈ his∧di ∈ dis∧acrs⊆(diarm(hi))(di)] in
430 [!{mgmt_hdlr_ch[hi]!mkGrant(di,acrs) |
430 hi:HI,di:DI,acrs:ANm-set•hi ∈ dom diarm∧di ∈ dis∧acrs⊆(diarm(hi))(di)}
426 end end end

```

### M.17.1.4 Management Shred Behaviour: Right Fig. M.5 on page 375

The **management shred** behaviour

- 431. initiates a request to the archive behaviour.
- 432. First the management shred behaviour selects a document identifier (from its directory).
- 433. Then it communicates a shred document message to the archive behaviour;
- 434. then it notes the (to be shredded) document in its directory
- 435. whereupon the management shred behaviour resumes being the management behaviour.

**value**

```

431 mgmt_shred: MDIR → out mgmt_arch_ch Unit
431 mgmt_shred(mdir) ≡
432 let di:DI • is_suitable(di)(mdir) in
433 [1] mgmt_arch_ch ! mkShred(time_ch?,di) ;
434 let mdir' = [hi→mdir(hi)\{di}|hi:HI•hi ∈ dom mdir] in
435 mgmt(mdir') end end

```

## M.17.2 Archive Behaviour

436. The archive behaviour is involved in the following action traces:

- (a) **create**
- (b) **copy**
- (c) **shred**

Fig. M.3 on page 374 Left

Fig. M.4 on page 375 Right

Fig. M.5 on page 375 Right

**type**

$\text{ADIR} = \text{avail:DI-set} \times \text{used:DI-set} \times \text{gone:DI-set}$

**axiom**

$\forall (\text{avail,used,gone}): \text{ADIR} \bullet \text{avail} \cap \text{used} = \{\} \wedge \text{gone} \subseteq \text{used}$

**value**

```

436 arch: ADIR → in,out mgmt_arch_ch, {arch_docu_ch[di]|di:DI•di ∈ dis} Unit
436a arch(adir) ≡
436a arch_create(adir)
436b [] arch_copy(adir)
436c [] arch_shred(adir)

```



**M.17.2.1 The Archive Create Behaviour: Left Fig. M.3 on page 374**

The **archive create** behaviour

```

437. accepts a request, from the management behaviour to create a document;
438. it then selects an available document identifier;
439. communicates this new document identifier to the management behaviour;
440. while initiating a new document behaviour, docu_{di} , with the document descriptor, dd , the initial document
 annotation being the singleton list of the note, an , and the initial document contents, dc – all received from
 the management behaviour – and an initial document history of just one entry: the date of creation, all
441. in parallel with resuming the archive behaviour with updated programmable attributes.

436a. arch_create: AATTR \rightarrow in,out mgmt_arch_ch, {arch_docu_ch[di]|di:DI•di \in dis} Unit
436a. arch_create(avail,used,gone) \equiv
437. [2] let mkCreate((hi,t),dd,an,dc) = mgmt_arch_ch ? in
438. let di:DI•di \in avail in
439. [4] mgmt_arch_ch ! mkNewDocID(di) ;
440. [3] $\text{docu}_{di}(dd)((an),dc,<(date_of_creation)>)$
441. || arch(avail\{di},used \cup \{di},gone)
436a. end end

```

**M.17.2.2 The Archive Copy Behaviour: Right Fig. M.4 on page 375**

The **archive copy** behaviour

```

442. accepts a copy document request from the management behaviour with the identity, j , of the master docu-
 ment;
443. it communicates (the request to obtain all the attribute values of the master document, j) to that document
 behaviour;
444. whereupon it awaits their communication (i.e., (dd,da,dc,dh));
445. (meanwhile) it obtains an available document identifier,
446. which it communicates to the management behaviour,
447. while initiating a new document behaviour, docu_{di} , with the master document descriptor, dd , the master
 document annotation, and the master document contents, dc , and the master document history, dh (all
 received from the master document),
448. in parallel with resuming the archive behaviour with updated programmable attributes.

436b. arch_copy: AATTR \rightarrow in,out mgmt_arch_ch, {arch_docu_ch[di]|di:DI•di \in dis} Unit
436b. arch_copy(avail,used,gone) \equiv
442. [3] let mkDocID(j,hi) = mgmt_arch_ch ? in
443. arch_docu_ch[j] ! mkReqAttrs() ;
444. let mkAttrs(dd,da,dc,dh) = arch_docu_ch[j] ? in
445. let di:DI • di \in avail in
446. mgmt_arch_ch ! mkCopyDocID(di) ;
447. [6,7] $\text{docu}_{di}(\text{augment}(dd, \text{"copy"}, j, hi),$
447. $\text{augment}(da, \text{"copy"}, hi), dc,$
447. $\text{augment}(dh, (\text{"copy"}, \text{date_and_time}, j, hi)))$
448. || arch(avail\{di},used \cup \{di},gone)
436b. end end end

```

where we presently leave the [overloaded] **augment** functions undefined.

**M.17.2.3 The Archive Shred Behaviour: Right Fig. M.5 on page 375**

The **archive shred** behaviour

```

449. accepts a shred request from the management behaviour.
450. It communicates this request to the identified document behaviour.
451. And then resumes being the archive behaviour, noting however, that the shredded document has been
 shredded.

436c. arch_shred: AATTR \rightarrow in,out mgmt_arch_ch, {arch_docu_ch[di]|di:DI•di \in dis} Unit
436c. arch_shred(avail,used,gone) \equiv
449. [2] let mkShred(j) = mgmt_arch_ch ? in
450. arch_docu_ch[j] ! mkShred() ;
451. arch(avail,used,gone \cup \{j})
436c. end

```

### M.17.3 Handler Behaviours

452. The handler behaviour is involved in the following action traces:

- |                   |                            |
|-------------------|----------------------------|
| (a) <b>create</b> | Fig. M.3 on page 374 Left  |
| (b) <b>edit</b>   | Fig. M.3 on page 374 Right |
| (c) <b>read</b>   | Fig. M.4 on page 375 Left  |
| (d) <b>copy</b>   | Fig. M.4 on page 375 Right |
| (e) <b>grant</b>  | Fig. M.5 on page 375 Left  |

**value**

```

452 hdlrhi: HATTRS → in,out mgmt_hdlr_ch[hi],{hdlr_docu_ch[hi,di]|di:DI•di∈dis} Unit
452 hdlrhi(hattrs) ≡
452a hdlr_createhi(hattrs)
452b [] hdlr_edithi(hattrs)
452c [] hdlr_readhi(hattrs)
452d [] hdlr_copyhi(hattrs)
452e [] hdlr_granthi(hattrs)

```

#### M.17.3.1 The Handler Create Behaviour: Left Fig. M.3 on page 374

453. The **handler create** behaviour offers to accept the granting of access rights, *acrs*, to document *di*.  
 454. It according updates its programmable *hattrs* attribute;  
 455. and resumes being a handler behaviour with that update.

```

452a hdlr_createhi: HATTRS × HHIST → in,out mgmt_hdlr_ch[hi] Unit
452a hdlr_createhi(hattrs,hhist) ≡
453 let mkGrant(di,acrs) = mgmt_hdlr_ch[hi] ? in
454 let hattrs' = hattrs † [hi ↦ acrs] in
455 hdlr_createhi(hattrs',augment(hhist,mkGrant(di,acrs))) end end

```

#### M.17.3.2 The Handler Edit Behaviour: Right Fig. M.3 on page 374

456. The handler behaviour, on its own volition, decides to edit a document, *di*, for which it has editing rights.  
 457. The handler behaviour selects a suitable (...) pair of *edit/undo* functions and a suitable (annotation) note.  
 458. It then communicates the desire to edit document *di* with (*e,u*) (at time  $t=time\_ch?$ ).  
 459. Editing take some time, *ti*.  
 460. We can therefore assert that the time at which editing has completed is  $t+ti$ .  
 461. The handler behaviour accepts the edit completion message from the document handler.  
 462. The handler behaviour can therefore resume with an updated document history.

```

452b hdlr_edithi: HATTRS × HHIST → in,out {hdlr_docu_ch[hi,di]|di:DI•di∈dis} Unit
452b hdlr_edithi(hattrs,hhist) ≡
456 [1] let di:DI • di ∈ dom hattrs ∧ "edit" ∈ hattrs(di) in
457 [1] let (e,u):(EDIT×UNDO) • ... , n:AN • ... in
458 [1] hdlr_docu_ch[hi,di] ! mkEdit(hi,t=time_ch?,e,u,n) ;
459 [2] let ti:TIMEINTERVAL • ... in
460 [2] wait ti ; assert: time_ch? = t+ti
461 [3] let mkEditComplete(ti',...) = hdlr_docu_ch[hi,di] ? in assert ti' ≅ ti
462 hdlrhi(hattrs,augment(hhist,(di,mkEdit(hi,t,ti,e,u))))
452b end end end end

```

**M.17.3.3 The Handler Read Behaviour: Left Fig. M.4 on page 375**

463. The **handler behaviour**, on its own volition, decides to read a document,  $di$ , for which it has reading rights.  
 464. It then communicates the desire to read document  $di$  with at time  $t=time\_ch?$  – with an annotation note ( $n$ ).  
 465. Reading take some time,  $ti$ .  
 466. We can therefore assert that the time at which reading has completed is  $t+ti$ .  
 467. The handler behaviour accepts the read completion message from the document handler.  
 468. The handler behaviour can therefore resume with an updated document history.

```

452c hdlr_edithi: HATTRS × HHIST → in,out {hdlr_docu_ch[hi,di]|di:DI•di∈dis} Unit
452c hdlr_edithi(hattrs,hhist) ≡
463 [1] let di:DI • di ∈ dom hattrs ∧ "read" ∈ hattrs(di), n:N • ... in
464 [1] hdlr_docu_ch[hi,di] ! mkRead(hi,t=time_ch?,n) ;
465 [2] let ti:TIME_INTERVAL • ... in
466 [2] wait ti ; assert: time_ch? = t+ti
467 [3] let mkReadComplete(ti,...) = hdlr_docu_ch[hi,di] ? in
468 [3] hdlrhi(hattrs,augment(hhist,(di,mkRead(di,t,ti))))
452c end end end

```

**M.17.3.4 The Handler Copy Behaviour: Right Fig. M.4 on page 375**

469. The **handler [copy] behaviour**, on its own volition, decides to copy a document,  $di$ , for which it has copying rights.  
 470. It communicates this copy request to the management behaviour.  
 471. After a while the handler [copy] behaviour receives acknowledgment of a completed copying from the management behaviour.  
 472. The handler [copy] behaviour records the request and acknowledgment in its, thus updated whereupon the handler [copy] behaviour resumes being the handler behaviour.

```

452d hdlr_copyhi: HATTRS × HHIST → in,out mgmt_hdlr_ch[hi] Unit
452d hdlr_copyhi(hattrs,hhist) ≡
469 [1] let di:DI • di ∈ dom hattrs ∧ "copy" ∈ hattrs(di) in
470 [1] mgmt_hdlr_ch[hi] ! mkCopy(di,hi,t=time_ch?) ;
471 [10] let mkCopyComplete(di',di) = mgmt_hdlr_ch[hi] ? in
472 [10] hdlrhi(hattrs,augment(hhist,time_ch?,(mkCopy(di,hi,t),mkCopyComplete(di'))))
452d end end

```

**M.17.3.5 The Handler Grant Behaviour: Left Fig. M.5 on page 375**

473. The **handler [grant] behaviour** offers to accept grant permissions from the management behaviour.  
 474. In response it updates its handler attribute while resuming being a handler behaviour.

```

452e hdlr_granthi: HATTRS × HHIST → in,out mgmt_hdlr_ch[hi] Unit
452e hdlr_granthi(hattrs,hhist) ≡
473 [2] let mkGrant(di,acrs) = mgmt_hdlr_ch[hi] ? in
474 [2] hdlrhi(hattrs†[di→acrs],augment(hhist,time_ch?,mkGrant(di,acrs)))
452e end

```

**M.17.4 Document Behaviours**

475. The document behaviour is involved in the following action traces:

- (a) **edit**
- (b) **read**
- (c) **shred**

Fig. M.3 on page 374 Right

Fig. M.4 on page 375 Left

Fig. M.5 on page 375 Right

**value**

```

475 docudi: DD × (DA × DC × DH) →
475 in,out arch_docu_ch[di], {hdlr_docu_ch[hi,di]|hi:HI•hi∈his} Unit
475 docudi(dattrs) ≡
475a docu_editdi(dd)(da,dc,dh)
475b [] docu_readdi(dd)(da,dc,dh)
475c [] docu_shreddi(dd)(da,dc,dh)

```

### M.17.4.1 The Document Edit Behaviour: Right Fig. M.3 on page 374

476. The **document [edit] behaviour** offers to accept edit requests from document handlers.
- (a) The document contents is edited, over a time interval of  $ti$ , with respect to the handlers edit function ( $e$ ),
  - (b) the document annotations are augmented with respect to the handlers note ( $n$ ), and
  - (c) the document history is augmented with the fact that an edit took place, at a certain time, with a pair of *edit/undo* functions.
477. The edit (etc.) function(s) take some time,  $ti$ , to do.
478. The handler behaviour is notified, `mkEditComplete(...)` of the completion of the edit, and
479. the document behaviour is then resumed with updated programmable attributes.

**value**

```

475a docu_editdi: DD × (DA × DC × DH) → in,out {hdlr_docu_ch[hi,di]|hi:HI•hi∈his} Unit
475a docu_editdi(dd)(da,dc,dh) ≡
476 [2] let mkEdit(hi,t,e,u,n) = [] {hdlr_docu_ch[hi,di]?|hi:HI•hi∈his} in
476a [2] let dc' = e(dc),
476b da' = augment(da,((hi,t),(''edit'',e,u),n)),
476c dh' = augment(dh,((hi,t),(''edit'',e,u))) in
477 let ti = time_ch? - t in
478 hdlr_docu_ch[hi,di] ! mkEditComplete(ti,...) ;
479 docudi(dd)(da',dc',dh')
475a end end end

```

### M.17.4.2 The Document Read Behaviour: Left Fig. M.4 on page 375

480. The **document [read] behaviour** offers to receive a read request from a handler behaviour.
481. The reading takes some time to do.
482. The handler behaviour is advised on completion.
483. And the document behaviour is resumed with appropriate programmable attributes being updated.

**value**

```

475b docu_readdi: DD × (DA × DC × DH) → in,out {hdlr_docu_ch[hi,di]|hi:HI•hi∈his} Unit
475b docu_readdi(dd)(da,dc,dh) ≡
480 [2] let mkRead(hi,t,n) = {hdlr_docu_ch[hi,di]?|hi:HI•hi∈his} in
481 [2] let ti:TIME_INTERVAL • ... in
481 [2] wait ti ;
482 [2] hdlr_docu_ch[hi,di] ! mkReadComplete(ti,...) ;
483 [2] docudi(dd)(augment(da,n),dc,augment(dh,(hi,t,ti,'read'')))
475b end end

```

### M.17.4.3 The Document Shred Behaviour: Right Fig. M.5 on page 375

484. The **document [shred] behaviour** offers to accept a document shred request from the archive behaviour –
485. whereupon it **stops!**

**value**

```

475c docu_shreddi: DD × (DA × DC × DH) → in,out arch_docu_ch[di] Unit
475c docu_shreddi(dd)(da,dc,dh) ≡
484 [3] let mkShred(...) = arch_docu_ch[di] ? in
485 stop
475c [3] end

```

## M.17.5 Conclusion

This completes a first draft version of this document. The date time is: March 12, 2024: 10:48 am. Many things need to be done. First a careful checking of all types and functions: that all used names have been defined. The internal non-deterministic choices in formula Items 400 on page 376, 436 on page 378, 452 on page 380 and 475 on the preceding page, need be checked. I suspect there should, instead, be some mix of both internal and external non-deterministic choices. Then a careful motivation for all the other non-deterministic choices.

## M.18 Documents in Public Government

Public government, in the spirit of *Charles-Louis de Secondat, Baron de La Brède et de Montesquieu* (or just *Montesquieu*), has three branches:

- the **legislative**,
- the **executive**, and
- the **judicial**.

Our interpretation of these, with respect to documents, are as follows.

- The **legislative** branch produces laws, i.e., documents. To do so many preparatory documents are created, edited, read, copied, etc. Committees, subcommittees, individual lawmakers and ministry law office staff handles these documents. Parliament staff and legislators are granted limited or unlimited access rights to these documents. Finally laws are put into effect, are amended, changed or abolished.  
The legislative branch documents refer to legislative, executive and judicial branch documents.
- The **executive** branch produces guide lines, i.e., documents. Instructions on interpretation and implementation of laws; directives to ministry services on how to handle the laws; et cetera.  
These executive branch documents refer to legislative, executive and judicial branch documents.
- The **judicial** branch produces documents. Police cite citizens and enterprises for breach of law. Citizens and enterprises sue other citizens and/or enterprises. Attorneys on behalf of the governments, or citizens or enterprises prepare statements. Court proceedings are recorded. Justices pass verdicts.  
The judicial branch documents refer to legislative, executive and judicial branch documents.

## M.19 Documents in Urban Planning

A separate research note [78, Urban Planning Processes] analyses & describes a domain of urban planning. There are the geographical documents:

- geodetic,
- geotechnic,
- meteorological,
- and other types of geographical documents.

In order to perform an informed urban planning further documents are needed:

- auxiliary documents which
- requirements documents which

Auxiliary documents presents such information that “fill in” details concerning current ownership of the land area, current laws affecting this ownership, the use of the land, et cetera. Requirements documents express expectations about the (base) urban plans that should result from the base urban planning. As a first result of base urban planning we see the emergence of the following kinds of documents:

- base urban plans
- and ancillary notes.

The base urban plans deal with

- cadastral,
- cartographic and
- zoning

issues. The ancillary notes deal with such things as insufficiencies in the base plans, things that ought be improved in a next iteration base urban planning, etc. The base plans and ancillary notes, besides possible re-iteration of base urban planning, lead on to “derived urban planning” for

- light, medium and heavy industry zones,
- mixed shopping and residential zones,
- apartment building zones,
- villa zones,
- recreational zones,
- et cetera.

After these “first generation” derived urban plans are well underway, a “second generation” derived urban planning can start:

- transport infrastructure,
- water and waste resource management,

- electricity, natural gas, etc., infrastructure,
- et cetera.

And so forth. Literally “zillions upon zillions” of strongly and crucially interrelated documents accrue.

Urban planning evolves and revolves around documents.

Documents are the only “tangible” results of urban planning.<sup>13</sup>

---

<sup>13</sup>Once urban plans have been agreed upon by all relevant authorities and individuals, then urban development (“build”) and, finally, “operation” of the developed, new urban “landscape”. For development, the urban plans form one of the “tangible” inputs. Others are of financial and human and other resource nature.

# Appendix N

## Swarms of Drones

### Contents

---

|             |                                                                        |            |
|-------------|------------------------------------------------------------------------|------------|
| <b>N.1</b>  | <b>An Informal Introduction</b>                                        | <b>387</b> |
| N.1.1       | <b>Describable Entities</b>                                            | 387        |
| N.1.1.1     | <b>The Endurants: Parts</b>                                            | 387        |
| N.1.1.2     | <b>The Perdurants</b>                                                  | 388        |
| N.1.2       | <b>The Contribution of [51]</b>                                        | 388        |
| N.1.3       | <b>The Contribution of This Report</b>                                 | 388        |
| <b>N.2</b>  | <b>Entities, Endurants</b>                                             | <b>388</b> |
| N.2.1       | <b>Parts, Atomic and Composite, Sorts, Abstract and Concrete Types</b> | 389        |
| N.2.1.1     | <b>Universe of Discourse</b>                                           | 389        |
| N.2.1.2     | <b>The Enterprise</b>                                                  | 389        |
| N.2.1.3     | <b>From Abstract Sorts to Concrete Types</b>                           | 389        |
| N.2.1.3.1   | <b>The Auxiliary Function <code>xtr_Ds</code>:</b>                     | 390        |
| N.2.1.3.2   | <b>Command Center</b>                                                  | 390        |
| N.2.1.3.2.1 | <b>A Simple Narrative</b>                                              | 390        |
| N.2.1.3.3   | <b>Command Center Decomposition</b>                                    | 390        |
| N.2.2       | <b>Unique Identifiers</b>                                              | 390        |
| N.2.2.1     | <b>The Enterprise, the Aggregates of Drones and the Geography</b>      | 391        |
| N.2.2.2     | <b>Unique Command Center Identifiers</b>                               | 391        |
| N.2.2.3     | <b>Unique Drone Identifiers</b>                                        | 391        |
| N.2.2.3.1   | <b>Auxiliary Function: <code>xtr_dis</code>:</b>                       | 391        |
| N.2.2.3.2   | <b>Auxiliary Function: <code>xtr_D</code>:</b>                         | 392        |
| N.2.3       | <b>Mereologies</b>                                                     | 392        |
| N.2.3.1     | <b>Definition</b>                                                      | 392        |
| N.2.3.2     | <b>Origin of the Concept of Mereology as Treated Here</b>              | 392        |
| N.2.3.3     | <b>Basic Mereology Principle</b>                                       | 392        |
| N.2.3.4     | <b>Engineering versus Methodical Mereology</b>                         | 392        |
| N.2.3.5     | <b>Planner Mereology</b>                                               | 393        |
| N.2.3.6     | <b>Monitor Mereology</b>                                               | 393        |
| N.2.3.7     | <b>Actuator Mereology</b>                                              | 393        |
| N.2.3.8     | <b>Enterprise Drone Mereology</b>                                      | 394        |
| N.2.3.9     | <b>'Other' Drone Mereology</b>                                         | 394        |
| N.2.3.10    | <b>Geography Mereology</b>                                             | 395        |
| N.2.4       | <b>Attributes</b>                                                      | 395        |
| N.2.4.1     | <b>The Time Sort</b>                                                   | 395        |
| N.2.4.2     | <b>Positions</b>                                                       | 395        |

|            |                                                   |            |
|------------|---------------------------------------------------|------------|
| N.2.4.2.1  | <b>A Neighbourhood Concept</b>                    | 396        |
| N.2.4.3    | <b>Flight Plans</b>                               | 396        |
| N.2.4.4    | <b>Enterprise Drone Attributes</b>                | 396        |
| N.2.4.4.1  | <b>Constituent Types</b>                          | 396        |
| N.2.4.4.2  | <b>Attributes</b>                                 | 397        |
| N.2.4.4.3  | <b>Enterprise Drone Attribute Categories:</b>     | 397        |
| N.2.4.5    | <b>'Other' Drones Attributes</b>                  | 397        |
| N.2.4.5.1  | <b>Constituent Types</b>                          | 397        |
| N.2.4.5.2  | <b>Attributes</b>                                 | 397        |
| N.2.4.6    | <b>Drone Dynamics</b>                             | 398        |
| N.2.4.7    | <b>Drone Positions</b>                            | 398        |
| N.2.4.8    | <b>Monitor Attributes</b>                         | 398        |
| N.2.4.9    | <b>Planner Attributes</b>                         | 398        |
| N.2.4.9.1  | <b>Swarms and Businesses:</b>                     | 398        |
| N.2.4.9.2  | <b>Planner Directories:</b>                       | 398        |
| N.2.4.10   | <b>Actuator Attributes</b>                        | 400        |
| N.2.4.11   | <b>Geography Attributes</b>                       | 400        |
| N.2.4.11.1 | <b>Constituent Types:</b>                         | 400        |
| N.2.4.11.2 | <b>Attributes</b>                                 | 400        |
| <b>N.3</b> | <b>Operations on Universe of Discourse States</b> | <b>401</b> |
| N.3.1      | <b>The Notion of a State</b>                      | 401        |
| N.3.2      | <b>Constants</b>                                  | 401        |
| N.3.3      | <b>Operations</b>                                 | 401        |
| N.3.3.1    | <b>A Drone Transfer</b>                           | 401        |
| N.3.3.2    | <b>An Enterprise Drone Changing Course</b>        | 402        |
| N.3.3.3    | <b>A Swarm Splitting into Two Swarms</b>          | 402        |
| N.3.3.4    | <b>Two Swarms Joining to form One Swarm</b>       | 402        |
| N.3.3.5    | <b>Etcetera</b>                                   | 402        |
| <b>N.4</b> | <b>Perdurants</b>                                 | <b>402</b> |
| N.4.1      | <b>System Compilation</b>                         | 402        |
| N.4.1.1    | <b>The Compile Functions</b>                      | 402        |
| N.4.1.2    | <b>Some CSP Expression Simplifications</b>        | 404        |
| N.4.1.3    | <b>The Simplified Compilation</b>                 | 404        |
| N.4.2      | <b>An Early Narrative on Behaviours</b>           | 405        |
| N.4.2.1    | <b>Either Endurants or Perdurants, Not Both!</b>  | 405        |
| N.4.2.2    | <b>Focus on Some Behaviours, Not All!</b>         | 405        |
| N.4.2.3    | <b>The Behaviours – a First Narrative</b>         | 405        |
| N.4.3      | <b>Channels</b>                                   | 405        |
| N.4.3.1    | <b>The Part Channels</b>                          | 405        |
| N.4.3.1.1  | <b>General Remarks:</b>                           | 405        |
| N.4.3.1.2  | <b>Part Channel Specifics</b>                     | 406        |
| N.4.3.2    | <b>Attribute Channels, General Principles</b>     | 407        |
| N.4.3.3    | <b>The Case Study Attribute Channels</b>          | 408        |
| N.4.3.3.1  | <b>'Other' Drones:</b>                            | 408        |
| N.4.3.3.2  | <b>Enterprise Drones:</b>                         | 408        |
| N.4.3.3.3  | <b>Geography:</b>                                 | 408        |
| N.4.4      | <b>The Atomic Behaviours</b>                      | 408        |
| N.4.4.1    | <b>Monitor Behaviour</b>                          | 408        |
| N.4.4.2    | <b>Planner Behaviour</b>                          | 409        |
| N.4.4.2.1  | <b>The Auxiliary transfer Function</b>            | 409        |



N.4.4.2 **The Auxiliary flight\_planning Function** . . . . . 410  
 N.4.4.3 **Actuator Behaviour** . . . . . 411  
 N.4.4.4 **'Other' Drone Behaviour** . . . . . 411  
 N.4.4.5 **Enterprise Drone Behaviour** . . . . . 412  
 N.4.4.6 **Geography Behaviour** . . . . . 414  
**N.5 Conclusion** . . . . . 415

We speculate<sup>1</sup> on a domain of *swarms* and *drones monitored and controlled* by a *command center* in some *geography*. Awareness of swarms is registered only in an enterprise command center. We think of these swarms of drones as an enterprise of either package deliverers, crop-dusters, insect sprayers, search & rescuers, traffic monitors, or wildfire fighters – or several of these, united in a notion of *an enterprise* possibly consisting of of “disjoint” *businesses*. We analyse & describe the properties of these phenomena as endurants and as perdurants: parts one can observe and behaviours that one can study. We do not yet examine the problem of drone air traffic management<sup>2</sup>. The analysis & description of this postulated domain follows the principles, techniques and tools laid down in [51].

## N.1 An Informal Introduction

### N.1.1 Describable Entities

#### N.1.1.1 The Endurants: Parts

In the universe of discourse we observe *endurants*, here in the form of parts, and *perdurants*, here in the form of behaviours.

The parts are *discrete endurants*, that is, can be seen or touched by humans, or that can be conceived as an abstraction of a discrete part.

We refer to Fig. N.1.

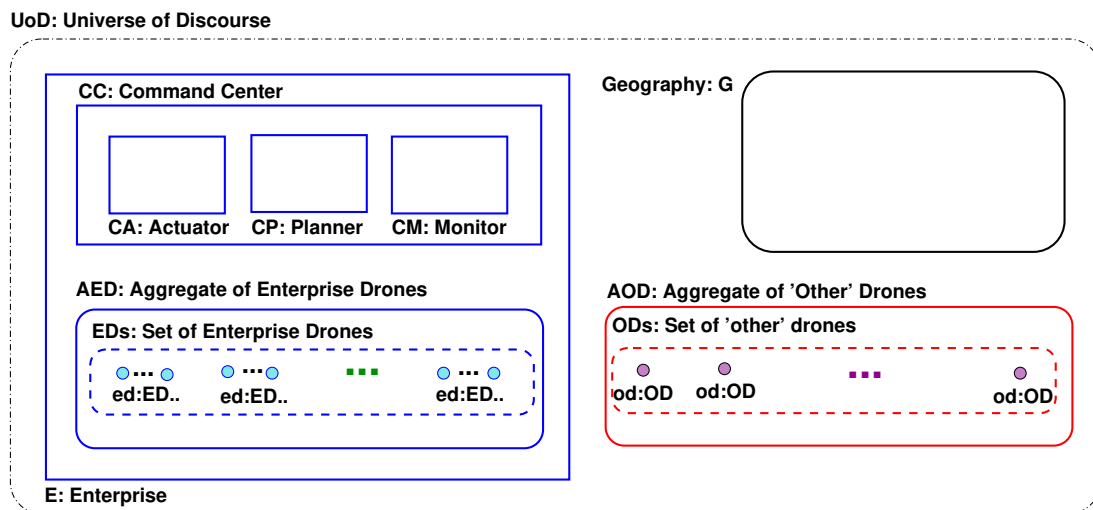


Figure N.1: Universe of Discourse

There is a *universe of discourse*, uod:UoD. The universe of discourse embodies: an *enterprise*, e:E. The enterprise consists of an *aggregate of enterprise drones*, aed:AED (which consists of a set, eds:EDs, of enterprise drones). and a *command center*, cc:CC; The universe of discourse also embodies a *geography*, g:G. The universe of discourse finally embodies an *aggregate of 'other' drones*, aod:AOD (which consists of a set, ods:ODs, of these 'other' drones). A

<sup>1</sup>A young researcher colleague, Dr. Yang ShaoFa, of the Software Institute of the Chinese Academy of Sciences in Beijing, at our meeting in Beijing, early November 2017, told me that he was then about to get involved in algorithms for drone maneuvering. So, true to me thinking, that, in order to reflect on such algorithms, one ought try understand the domain. So I sketched the model of this chapter in the week, attending the ICFEM'2017 conference in Xi'An, and presented the model to Dr. Yang upon my return to Beijing.

<sup>2</sup>[www.nasa.gov/feature/ames/first-steps-toward-drone-traffic-management](http://www.nasa.gov/feature/ames/first-steps-toward-drone-traffic-management), [www.sciencedirect.com/science/article/pii/S2046043016300260](http://www.sciencedirect.com/science/article/pii/S2046043016300260)

*drone* is an *unmanned aerial vehicle*.<sup>3</sup> We distinguish between *enterprise drones*, ed:ED, and ‘*other*’ *drones*, od:OD. The *pragmatics* of the enterprise swarms is that of providing enterprise drones for one or more of the following kinds of *businesses*:<sup>4</sup> delivering parcels (mail, packages, etc.)<sup>5</sup>, crop dusting<sup>6</sup>, aerial spraying<sup>7</sup>, wildfire fighting<sup>8</sup>, traffic control<sup>9</sup>, search and rescue<sup>10</sup>, etcetera. A notion of *swarm* is introduced. A swarm is a concept. As a concept a swarm is a set of drones. We associate swarms with businesses. A business has access to one or more swarms. The enterprise *command center*, cc:CC, can be seen as embodying three kinds of functions: a *monitoring* service, cm:CM, whose function it is to know the locations and dynamics of all drones, whether enterprise drones or ‘other’ drones; a *planning* service, cp:CP, whose function it is to plan the next moves of all that enterprise’s drones; and an *actuator* service, ca:CA, whose functions it is to guide that enterprise’s drones as to their next moves. The swarm concept “resides” in the command planner.

### N.1.1.2 The Perdurants

The perdurants are entities for which only a fragment exists if we look at or touch them at any given snapshot in time, that is, were we to freeze time we would only see or touch a fragment of the perdurant.

The major \*\*\*

MORE TO COME

### N.1.2 The Contribution of [51]

The major contributions of [51] are these: a methodology<sup>11</sup> for analysing & describing manifest domains<sup>12</sup>, where the methodology builds on an *ontological principle* of viewing the domains as consisting of *endurants* and *perdurants*. Endurants possess properties such as *unique identifiers*, *mereologies*, and *attributes*. Perdurants are then analysed & described as either *actions*, *events*, or *behaviours*. The *techniques* to go with the \*\*\*

MORE TO COME

The *tools* are \*\*\*

MORE TO COME

### N.1.3 The Contribution of This Report

TO BE WRITTEN

We relate our work to that of [139].

• • •

The main part of this report is contained in the next three sections: endurants; states, constants, and operations on states; and perdurants.

## N.2 Entities, Endurants

By an *entite* shall understand a *phenomenon*, *i.e.*, *something that can be observe d, i.e., be seen or touched by humans, or that can be conceived as an abstraction of an entity. We further demand that an entity can be objectively described.*

By an *endurante* shall understand *an entity that can be observed or conceived and described as a “complete thing” at no matter which given snapshot of time. Were we to “freeze” time we would still be able to observe the entire endurant.*

<sup>3</sup>Drones are also referred to as UAVs.

<sup>4</sup><http://www.latimes.com/business/la-fi-drone-traffic-20170501-htlmlstory.html>

<sup>5</sup><https://www.amazon.com/Amazon-Prime-Air/b?node=8037720011> and <https://www.digitaltrends.com/cool-tech/amazon-prime-air-delivery-drones-history-progress/>

<sup>6</sup><http://www.uavcropdustersprayers.com/>, <http://sprayingdrone.com/>

<sup>7</sup><https://abjdrones.com/commercial-drone-services/industry-specific-solutions/agriculture/>

<sup>8</sup><https://www.smithsonianmag.com/videos/category/innovation/drones-are-now-being-used-to-battle-wildfires/>

<sup>9</sup>[https://business.esa.int/sites/default/files/Presentation%20on%20UAV%20Road%20Surface%20Monitoring%20and%20Traffic%20Information\\_0.pdf](https://business.esa.int/sites/default/files/Presentation%20on%20UAV%20Road%20Surface%20Monitoring%20and%20Traffic%20Information_0.pdf)

<sup>10</sup><http://sardrones.org/>

<sup>11</sup>By a *methodology* we shall understand a set of *principles* for selecting and applying a number of *techniques*, using *tools*, to – in this case – analyse & describe a domain.

<sup>12</sup>A manifest domain is a human- and artifact-assisted arrangement of endurant, that is spatially “stable”, and perdurant, that is temporally “fleeting” entities. Endurant entities are either parts or components or materials. Perdurant entities are either actions or events or behaviours.

## N.2.1 Parts, Atomic and Composite, Sorts, Abstract and Concrete Types

By a *discrete endurant* we shall understand *an endurant which is separate, individual or distinct in form or concept*.

By a *parte* shall understand a *discrete endurant which the domain engineer chooses to endow with internal qualities such as unique identification, mereology, and one or more attributes*. We shall define the concepts of unique identifier, mereology and attribute later in this report.

*Atomic parts* are those which, in a given context, are deemed to not consist of meaningful, separately observable proper sub-parts.

*Sub-parts* are parts.

*Composite parts* are those which, in a given context, are deemed to indeed consist of meaningful, separately observable proper sub-parts.

By a *sorte* shall understand *an abstract type*.

By a *typee* shall here understand *a set of values "of the same kind"* – where we do not further define what we mean by *the same kind*".

By an *abstract type* we shall understand *a type about whose values we make no assumption* [as to their atomicity or composition].

By a *concrete type* we shall understand *a type about whose values we are making certain assumptions as to their atomicity or composition, and, if composed then how and from which other types they are composed*.

### N.2.1.1 Universe of Discourse

By a *universe of discourse* shall understand *that which we can talk about, refer to and whose entities we can name*. Included in that universe is the *geography*. By *geography* we shall understand a section of the globe, an area of land, its geodesy, its meteorology, etc.

486. In the **Universe of Discourse** we can observe the following parts:

- (a) an atomic **Geography**,
- (b) a composite **Enterprise**,
- (c) and an aggregate of '**Other**'<sup>13</sup> **Drones**.

#### type

486 UoD, G, E, AOD

#### value

486a obs\_G: UoD → G

486b obs\_E: UoD → E

486c obs\_AOD: UoD → AOD

### N.2.1.2 The Enterprise

487. From an enterprise one can observe:

- (a) a(n enterprise) command center. and
- (b) an aggregate of enterprise drones.

#### type

487a CC

487a AED

#### value

487a obs\_CC: E → CC

487b obs\_AED: E → AED

### N.2.1.3 From Abstract Sorts to Concrete Types

488. From an *aggregate of enterprise drones*, AED, we can observe a possibly empty set of drones, EDs

489. From an *aggregate of 'other' drones*, AOD, we can observe a possibly empty set, ODs, of '*other*' drones.

#### type

488 ED

488 EDs = ED-set

489 OD

489 ODs = OD-set

<sup>13</sup>We apologize for our using the term 'other' drones. These 'other' drones are not necessarily adversary or enemy drones. They are just there – coexisting with the enterprise drones.

**value**

488 obs\_EDs: AED  $\rightarrow$  EDs  
 489 obs\_ODs: AOD  $\rightarrow$  ODs

Drones, whether ‘other’ or ‘enterprise’, are considered atomic.

**N.2.1.3.1 The Auxiliary Function xtr\_Ds:** We define an auxiliary function, xtr\_Ds.

- 490. From the universe of discourse we can extract all its drones;
- 491. similarly from its enterprise;
- 492. similarly from the aggregate of enterprise drones; and
- 493. from an aggregate of ‘other’ drones.

490 xtr\_Ds: UoD  $\rightarrow$  (ED|OD)-set  
 490 xtr\_Ds(uod)  $\equiv$   
 490  $\cup\{xtr\_Ds(obs\_AED(obs\_E(uod)))\} \cup xtr\_Ds(obs\_AOD(uod))$   
 491 xtr\_Ds: E  $\rightarrow$  ED-set  
 491 xtr\_Ds(e)  $\equiv$  xtr\_Ds(obs\_AED(e))  
 492 xtr\_Ds: AED  $\rightarrow$  ED-set  
 492 xtr\_Ds(aed)  $\equiv$  obs\_EDs(obs\_EDs(aed))  
 493 xtr\_Ds: AOD  $\rightarrow$  OD-set  
 493 xtr\_Ds(aod)  $\equiv$  obs\_ODs(aod)

- 494. In the universe of discourse a drone cannot be both among the enterprise drones and among the ‘other’ drones.

**axiom**

494  $\forall uod:UoD, e:E, aed:ES, aod:AOD \bullet$   
 494  $e=obs\_E(uod) \wedge aed=obs\_AED(e) \wedge aod=obs\_AOD(uod)$   
 494  $\Rightarrow xtr\_Ds(aed) \cap xtr\_Ds(aod) = \{\}$

The functions are partial as the supplied swarm identifier may not be one of the universe of discourse, etc.

**N.2.1.3.2 Command Center**

**N.2.1.3.2.1 A Simple Narrative** Figure N.1 on page 387 shows a graphic rendition of a space of interest. The command center, CC, a composite part, is shown to include three atomic parts: An atomic part, the monitor, CM. It monitors the location and dynamics of all drones. An atomic part, the planner, CP. It plans the next, “friendly”, drone movements. The command center also has yet an atomic part, the actuator, CA. It informs “friendly” drones of their next movements. The planner is where “resides” the notion of an enterprise consisting of one or more businesses, where each business has access to zero, one or more swarms, where a swarm is a set of enterprise drone identifiers.

The purpose of the control center is to monitor the whereabouts and dynamics of all drones (done by CM); to plan possible next actions by enterprise drones (done by CP); and to instruct enterprise drones of possible next actions (done by CA).

**N.2.1.3.3 Command Center Decomposition** From the composite command center we can observe

- 495. the center monitor, CM;
- 496. the center planner, CP; and
- 497. the center actuator, CA .

**type**

495 CM  
 496 CP  
 497 CA

**value**

495 obs\_CM: CC  $\rightarrow$  CM  
 496 obs\_CP: CC  $\rightarrow$  CP  
 497 obs\_CA: CC  $\rightarrow$  CA

**N.2.2 Unique Identifiers**

Parts are distinguishable through their unique identifiers. A *unique identifiers* a further undefined quantity which we associate with parts such that no two parts of a universe of discourse are identical.

**N.2.2.1 The Enterprise, the Aggregates of Drones and the Geography**

498. Although we may not need it for subsequent descriptions we do, for completeness of description, introduce unique identifiers for parts and sub-parts of the universe of discourse:

- (a) Geographies,  $g:G$ , have unique identification.
- (b) Enterprises,  $e:E$ , have unique identification.
- (c) Aggregates of enterprise drones,  $aed:AED$ , have unique identification.
- (d) Aggregates of ‘other’ drones,  $aod:AOD$ , have unique identification.
- (e) Command centers,  $cc:CC$ , have unique identification.

**type**

498  $GI, EI, AEDI, AODI, CCI$

**value**

498a  $uid_G: G \rightarrow GI$

498b  $uid_E: E \rightarrow EI$

498c  $uid_{AED}: AED \rightarrow AEDI$

498d  $uid_{OD}: AOD \rightarrow AODI$

498e  $uid_{CC}: CC \rightarrow CCI$

**N.2.2.2 Unique Command Center Identifiers**

499. The monitor has a unique identifier.

500. The planner has a unique identifier.

501. The actuator has a unique identifier.

**type**

499  $CMI$

500  $CPI$

501  $CAI$

**value**

499  $uid_{CM}: CM \rightarrow CMI$

500  $uid_{CP}: CP \rightarrow CPI$

501  $uid_{CA}: CA \rightarrow CAI$

**N.2.2.3 Unique Drone Identifiers**

502. Drones have unique identifiers.

- (a) whether enterprise or
- (b) ‘other’ drones

**type**

502  $DI = EDI \mid ODI$

**value**

502a  $uid_{ED}: ED \rightarrow EDI$

502b  $uid_{OD}: OD \rightarrow ODI$

**N.2.2.3.1 Auxiliary Function:  $xtr\_dis$ :**

503. From the aggregate of enterprise drones;

504. From the aggregate of ‘other’ drones;

505. and from the two parts of a universe of discourse: the enterprise and the ‘other’ drones.

**value**

503  $xtr\_dis: AED \rightarrow DI\text{-set}$

503  $xtr\_dis(aed) \equiv \{uid_{ED}(ed) \mid ed:ED \bullet ed \in obs\_EDs(aed)\}$

504  $xtr\_dis: AOD \rightarrow DI\text{-set}$

504  $xtr\_dis(aod) \equiv \{uid_D(od) \mid od:OD \bullet od \in obs\_ODs(aod)\}$

505  $xtr\_dis: UoD \rightarrow DI\text{-set}$

505  $xtr\_dis(uod) \equiv xtr\_dis(obs\_AED(uod)) \cup xtr\_dis(obs\_AOD(uod))$

### N.2.2.3.2 Auxiliary Function: xtr\_D:

506. From the universe of discourse, given a drone identifier of that space, we can extract the identified drone;  
 507. similarly from the enterprise;  
 508. its aggregate of enterprise drones; and  
 509. and from its aggregate of ‘other’ drones;

```

506 xtr_D: UoD → DI $\overset{\sim}{\rightarrow}$ D
506 xtr_D(uod)(di) \equiv let d:D • d ∈ xtr_Ds(uod) \wedge uid_D(d)=di in d end
506 pre: di ∈ xtr_dis(soi)
507 xtr_D: E → DI $\overset{\sim}{\rightarrow}$ D
507 xtr_D(e)(di) \equiv let d:D • d ∈ xtr_Ds(obs_ES(e)) \wedge uid_D(d)=di in d end
507 pre: di ∈ xtr_dis(e)
508 xtr_D: AED → DI $\overset{\sim}{\rightarrow}$ D
508 xtr_D(aed)(di) \equiv let d:D • d ∈ xtr_Ds(aed) \wedge uid_D(d)=di in d end
508 pre: di ∈ xtr_dis(es)
509 xtr_D: AOD → DI $\overset{\sim}{\rightarrow}$ D
509 xtr_D(aod)(di) \equiv let d:D • d ∈ xtr_Ds(aod) \wedge uid_D(d)=di in d end
509 pre: di ∈ xtr_dis(ds)

```

## N.2.3 Mereologies

### N.2.3.1 Definition

*Mereology is the study and knowledge of parts and their relations (to other parts and to the “whole”) [81].*

### N.2.3.2 Origin of the Concept of Mereology as Treated Here

We shall [thus] deploy the concept of mereology as advanced by the Polish mathematician, logician and philosopher Stanisław Léchniewski. Douglas T. (“Doug”) Ross<sup>14</sup> also contributed along the lines of our approach [162] – hence [54] is dedicated to Doug.

### N.2.3.3 Basic Mereology Principle

The basic principle in modelling the mereology of a any universe of discourse is as follows: Let  $p'$  be a part with unique identifier  $p'_{id}$ . Let  $p$  be a sub-part of  $p'$  with unique identifier  $p_{id}$ . Let the immediate sub-parts of  $p$  be  $p_1, p_2, \dots, p_n$  with unique identifiers  $p_{1id}, p_{2id}, \dots, p_{nid}$ . That  $p$  has mereology  $(p'_{id}, \{p_{1id}, p_{2id}, \dots, p_{nid}\})$ . The parts  $p_j$ , for  $1 \leq j \leq n$  for  $n \geq 2$ , if atomic, have mereologies  $(p_{id}, \{p_{1id}, p_{2id}, \dots, p_{j-1id}, p_{j+1id}, \dots, p_{nid}\})$  – where we refer to the second term in that pair by  $m$ ; and if composite, have mereologies  $(p_{id}, (m, m'))$ , where the  $m'$  term is the set of unique identifiers of the sub-parts of  $p_j$ .

### N.2.3.4 Engineering versus Methodical Mereology

We shall restrict ourselves to an engineering treatment of the mereology of our universe of discourse. That is in contrast to a strict, methodical treatment. In a methodical description of the mereologies of the various parts of the universe of discourse one assigns a mereology to every part: to the enterprise, the aggregate of ‘other’ drones and the geography; to the command center of the enterprise and its aggregate of drones; to the monitor, the planner and the actuator of the command center; to the drones of the aggregate of enterprise drones, and to the drones of the aggregate of ‘other’ drones. We shall “shortcut” most of these mereologies. The reason is this: The *pragmatics* of our attempt to model *drones*, is rooted in our interest in the interactions between the command center’s monitor and actuator and the enterprise and ‘other’ drones. For “completeness” we also include interactions between the geography’s mereology and the above command center and drones. The mereologies of the enterprise, E, the enterprise aggregate of drones AED, and the set of (enterprise) drones, EDs, do not involve drone identifiers. The only “thing” that the monitor and actuator are interested in are the drone identifiers. So we shall thus model the mereologies of our universe of discourse by omitting mereologies for the enterprise, the aggregates of drones, the sets of these aggregates, and the geography, and only describe the mereologies of the monitor, planner and actuator, the enterprise drones and the ‘other’ drones.

<sup>14</sup>Doug Ross is the originator of the term CAD for *computer aided design*, of APT for *Automatically Programmed Tools*, a language to drive numerically controlled manufacturing, and also SADT for *Structure Analysis and Design Techniques*

### N.2.3.5 Planner Mereology

510. The planner mereology reflects the center planners awareness<sup>15</sup> of the monitor, the actuator,, and the geography of the universe of discourse.
511. The planner mereology further reflects that a *eureka*<sup>16</sup> is provided by, or from, an outside source reflected in the autonomous attribute *Cmdl*. The value of this attribute changes at its own volition and ranges over commands that directs the planner to perform either of a number of operations.

Eureka examples are: calculate and effect a new flight *plan* for one or more designated swarms of a designated business; effect the transfer of an enterprise drone from a designated swarm of a business to another, distinctly designated swarm of the same business; etcetera.

```

type
510 CPM = (CAI × CMI × GI) × Eureka
511 Eureka == mkNewFP(BI×SI-set×Plan)
511 | mkChgDB(fsi:SI×tsi:SI×di×DI)
511 | ...
value
510 mereo_CP: CP → CPM
511 Plan = ...

```

We omit expressing a suitable axiom concerning center planner mereologies. Our behavioural analysis & description of monitoring & control of operations on the space of drones will show that command center mereologies may change.

### N.2.3.6 Monitor Mereology

The monitor's mereology reflects its awareness of the drones whose position and dynamics it is expected to monitor.

512. The mereology of the center monitor is a pair: the set of unique identifiers of the drones of the universe of discourse, and the unique identifier of the center planner.

```

type
512 CMM = DI-set × CPI
value
512 mereo_CM: CM → CMM

```

513. For the universe of discourse it is the case that
- (a) the drone identifiers of the mereology of a monitor must be exactly those of the drones of the universe of discourse, and
  - (b) the planner identifier of the mereology of a monitor must be exactly that of the planner of the universe of discourse.

```

axiom
513 $\forall uod:UoD,e:E,cc:CC,cp:CP,cm:CM,g:G \bullet$
513 $e=obs_E(uod)\wedge cc=obs_CC(e)\wedge cp=obs_CP(cc)\wedge cm=obs_CM(cc) \Rightarrow$
513 let (dis,cpi) = mereo_CM(cm) in
513a dis = xtr_dis(uod)
513b $\wedge cpi = uid_CP(cp)$ end

```

### N.2.3.7 Actuator Mereology

The center actuator's mereology reflects its awareness of the enterprise drones whose position and dynamics it is expected to control.

514. The mereology of the center actuator is a pair: the set of unique identifiers of the business drones of the universe of discourse, and the unique identifier of the center planner.

<sup>15</sup>That "awareness" includes, amongst others, the planner obtaining information from the monitor of the whereabouts of all drones and providing the actuator with directives for the enterprise drones — all in the context of the *land* and "its" *meteorology*.

<sup>16</sup>"Eureka" comes from the Ancient Greek word *εὐρηκα* *heúrēka*, meaning "I have found (it)", which is the first person singular perfect indicative active of the verb *εὐρηκω* *heuriskō* "I find".[1] It is closely related to heuristic, which refers to experience-based techniques for problem solving, learning, and discovery.

**type**514 CAM = EDI-set  $\times$  CPI**value**514 mereo\_CA: CA  $\rightarrow$  CAM

515. For all universes of discourse

- (a) the drone identifiers of the mereology of a center actuator must be exactly those of the enterprise drones of the space of interest (of the monitor), and
- (b) the center planner identifier of the mereology of a center actuator must be exactly that of the center planner of the command center of the space of interest (of the monitor)

**axiom**515  $\forall$  uod:UoD,e:E,cc:CC,cp:CP,ca:CA •515  $e = \text{obs\_E}(uod) \wedge cc = \text{obs\_CC}(e) \wedge cp = \text{obs\_CP}(cc) \wedge ca = \text{obs\_CA}(cc) \Rightarrow$ 515  $\text{let } (dis, cpi) = \text{mereo\_CA}(ca) \text{ in}$ 515a  $dis = \text{tr\_dis}(e)$ 515b  $\wedge cpi = \text{uid\_CP}(cp) \text{ end}$ **N.2.3.8 Enterprise Drone Mereology**516. The mereology of an enterprise drone is the triple of the command center monitor, the command center actuator<sup>17</sup>, and the geography.**type**516 EDM = CMI  $\times$  CAI  $\times$  GI**value**516 mereo\_ED: ED  $\rightarrow$  EDM

517. For all universes of discourse the enterprise drone mereology satisfies:

- (a) the unique identifier of the first element of the drone mereology is that of the enterprise's command monitor,
- (b) the unique identifier of the second element of the drone mereology is that of the enterprise's command actuator, and
- (c) the unique identifier of the third element of the drone mereology is that of the universe of discourse's geography.

**axiom**517  $\forall$  uod:UoD,e:E,cm:CM,ca:CA,ed:ED,g:G •517  $e = \text{obs\_E}(uod) \wedge cm = \text{obs\_CM}(\text{obs\_CC}(e)) \wedge ca = \text{obs\_CA}(\text{obs\_CC}(e))$ 517  $\wedge ed \in \text{xtr\_Ds}(e) \wedge g = \text{obs\_G}(uod) \Rightarrow$ 517  $\text{let } (cmi, cai, gi) = \text{mereo\_D}(ed) \text{ in}$ 517a  $cmi = \text{uid\_CMM}(cmm)$ 517b  $\wedge cai = \text{uid\_CAI}(cai)$ 517c  $\wedge gi = \text{uid\_G}(g) \text{ end}$ **N.2.3.9 'Other' Drone Mereology**

518. The mereology of an 'other' drone is a pair: the unique identifier of the monitor and the unique identifier of the geography.

**type**518 ODM = CMI  $\times$  GI**value**518 mereo\_OD: OD  $\rightarrow$  ODM

We leave it to the reader to formulate a suitable axiom, cf. axiom 517.

<sup>17</sup>The command center monitor and the command center actuator and their unique identifiers will be defined in Items 495, 497 on page 390, 499 and 501 on page 391.



### N.2.3.10 Geography Mereology

519. The geography mereology is a pair<sup>18</sup> of the unique of the unique identifiers of the planner and the set of all drones.

**type**

519  $GM = CPI \times CMI \times DI\text{-set}$

**value**

519  $\text{mereo\_G}: G \rightarrow GM$

We leave it to the reader to formulate a suitable axiom, cf. axiom 517 on the facing page.

## N.2.4 Attributes

We analyse & describe attributes for the following parts: *enterprise drones* and *'other' drones*, *monitor*, *planner* and *actuator*, and the *geography*. The attributes, that we shall arrive at, are usually concrete in the sense that they comprise values of, as we shall call them, *constituent* types. We shall therefore first analyse & describe these constituent types. Then we introduce the part attributes as expressed in terms of the constituent types. But first we introduce three notions core notions: time, Sect. N.2.4.1, positions, Sect. N.2.4.2, and flight plans, Sect. N.2.4.3.

### N.2.4.1 The Time Sort

520. Let the special sort identifier  $\mathbb{T}$  denote times

521. and the special sort identifier  $\mathbb{TI}$  denote time intervals.

522. Let identifier *time* designate a “magic” function whose invocations yield times.

**type**

520  $\mathbb{T}$

520  $\mathbb{TI}$

**value**

520  $\text{time}: \mathbf{Unit} \rightarrow \mathbb{T}$

523. Two times can not be added, multiplied or divided, but subtracting one time from another yields a time interval.

524. Two times can be compared: smaller than, smaller than or equal, equal, not equal, etc.

525. Two time intervals can be compared: smaller than, smaller than or equal, equal, not equal, etc.

526. A time interval can be multiplied by a real number.

Etcetera.

**value**

523  $\ominus: \mathbb{T} \times \mathbb{T} \rightarrow \mathbb{TI}$

524  $\langle, \leq, =, \neq, \geq, \rangle: \mathbb{T} \times \mathbb{T} \rightarrow \mathbf{Bool}$

525  $\langle, \leq, =, \neq, \geq, \rangle: \mathbb{TI} \times \mathbb{TI} \rightarrow \mathbf{Bool}$

526  $\otimes: \mathbb{TI} \times \mathbf{Real} \rightarrow \mathbb{TI}$

### N.2.4.2 Positions

Positions (of drones) play a pivotal rôle.

527. Each *position* being designated by

528. *longitude*, *latitude* and *altitude*.

**type**

528  $LO, LA, AL$

527  $P = LO \times LA \times AL$

---

<sup>18</sup>30.11.2017: I think!

### N.2.4.2.1 A Neighbourhood Concept

529. Two positions are said to be *neighbours* if the *distance* between them is small enough for a drone to fly from one to the other in one to three minutes' time – for drones flying at a speed below Mach 1.

#### value

529 neighbours:  $P \times P \rightarrow \mathbf{Bool}$

We leave the neighbourhood proposition further undefined.

### N.2.4.3 Flight Plans

A crucial notion of our universe of discourse is that of flight plans.

530. A *flight plan element* is a pair of a time and a position.

531. A *flight plan* is a sequence of flight plan elements.

#### type

530 FPE =  $\mathbb{T} \times P$

531 FP = FLE\*

532. such that adjacent entries in flight plans

- (a) record increasing times and
- (b) neighbouring positions.

#### axiom

532  $\forall fp:FP, i:Nat \bullet \{i, i+1\} \subseteq \mathit{indsfp} \Rightarrow$

532  $\quad \mathit{let} (t, p) = fp[i], (t', p') = fp[i+1] \mathit{ in}$

532a  $\quad t \leq t'$

532b  $\quad \wedge \mathit{neighbours}(p, p')$

532  $\quad \mathit{end}$

### N.2.4.4 Enterprise Drone Attributes

#### N.2.4.4.1 Constituent Types

533. Enterprise drones have *positions* expressed, for example, in terms of *longitude*, *latitude* and *altitude*.<sup>19</sup>

534. Enterprise drones have *velocity* which is a vector of *speed* and three-dimensional, i.e., spatial, *direction*.

535. Enterprise drones have *acceleration* which is a vector of *increase/decrease of speed per time unit* and *direction*.

536. Enterprise drones have orientation which is expressed in terms of three quantities: *yaw*, *pitch* and *roll*.<sup>20</sup>

We leave *speed*, *direction* and *increase/decrease per time unit* unspecified.

#### type

533 POS = P

534 VEL = SPEED  $\times$  DIRECTION

535 ACC = IncrDecrSPEEDperTimeUnit  $\times$  DIRECTION

536 ORI = YAW  $\times$  PITCH  $\times$  ROLL

534 SPEED = ...

534 DIRECTION = ...

535 IncrDecrSPEEDperTimeUnit = ...

<sup>19</sup>*Longitude* is a geographic coordinate that specifies the east-west position of a point on the Earth's surface. It is an angular measurement, usually expressed in degrees and denoted by the Greek letter lambda. Meridians (lines running from the North Pole to the South Pole) connect points with the same longitude. *Latitude* is a geographic coordinate that specifies the northsouth position of a point on the Earth's surface. Latitude is an angle (defined below) which ranges from 0° at the Equator to 90° (North or South) at the poles. Lines of constant latitude, or parallels, run eastwest as circles parallel to the equator. *Altitude* or height (sometimes known as depth) is defined based on the context in which it is used (aviation, geometry, geographical survey, sport, and many more). As a general definition, altitude is a distance measurement, usually in the vertical or "up" direction, between a reference datum and a point or object. The reference datum also often varies according to the context.

<sup>20</sup>*Yaw*, *pitch* and *roll* are seen as symmetry axes of a drone: normal axis, lateral (or transverse) axis and longitudinal (or roll) axis. See Fig. N.2 on the next page.

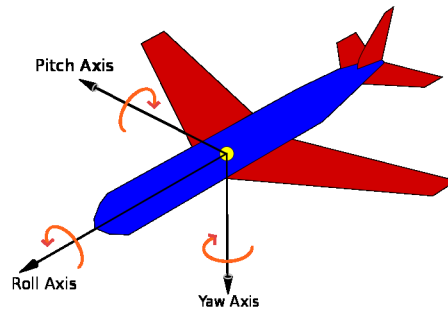


Figure N.2: Aircraft Orientation

#### N.2.4.4.2 Attributes

537. One of the enterprise properties is that of its *dynamics* which is seen as a quadruple of *velocity*, *acceleration*, *orientation* and *position*. It is recorded as a reactive attribute.
538. Enterprise drones follow a flight course, as prescribed in and recorded as a programmable attribute, referred to as the *future flight plan*, FFP.
539. Enterprise drones have followed a course recorded, also a programmable attribute, as a *past flight plan list*, PFPL.
540. Finally enterprise drones “remember”, in the form of a programmable attribute, the *geography* (i.e., the *area*, the *land* and the *weather*) it is flying over and in!

##### type

540  $\text{ImG} = A \times L \times W$

537  $\text{DYN} = s_{\text{vel}}:\text{VEL} \times s_{\text{acc}}:\text{ACC} \times s_{\text{ori}}:\text{ORI} \times s_{\text{pos}}:\text{POS}$

538  $\text{FPL} = \text{FP}$

539  $\text{PFPL} = \text{FP}^*$

##### value

537  $\text{attr\_DYN}: \text{ED} \rightarrow \text{DYN}$

538  $\text{attr\_FPL}: \text{ED} \rightarrow \text{FPL}$

539  $\text{attr\_PFPL}: \text{ED} \rightarrow \text{PFPL}$

540  $\text{attr\_ImG}: \text{ED} \rightarrow \text{ImG}$

Enterprise, as well as ‘other’ drone, positions must fall within the *Euclidian Point Space* of the geography of the universe of discourse. We leave that as an axiom to be defined – or we could decide that if a drone leaves that space then it is lost, and if drones suddenly “appear, out of the blue”, then they are either “brand new”, or “reappear”.

**N.2.4.4.3 Enterprise Drone Attribute Categories:** The position, velocity, acceleration, position and past position list attributes belong to the **reactive** category. The future position list attribute belong to the **programmable** category. Drones have a “zillion” more attributes – which may be introduced in due course.

#### N.2.4.5 ‘Other’ Drones Attributes

**N.2.4.5.1 Constituent Types** The constituent types of ‘other’ drones are similar to those of some of the enterprise drones.

##### N.2.4.5.2 Attributes

541. ‘Other’ drones have *dynamics*,  $\text{dyn}:\text{DYN}$ .

542. ‘Other’ drones “remember”, in the form of a programmable attribute, the *immediate geography*,  $\text{ImG}$  (i.e., the *area*, the *land* and the *weather*) it is flying over and in!

##### type

542  $A, L, W$

542  $\text{ImG} = A \times L \times W$

##### value

541  $\text{attr\_DYN}: \text{OD} \rightarrow \text{DYN}$

542  $\text{attr\_ImG}: \text{OD} \rightarrow \text{ImG}$

### N.2.4.6 Drone Dynamics

543. By a timed drone dynamics, TiDYN, we understand a quadruplet of *time*, *position*, *dynamics* and *immediate geography*.
544. By a *current drone dynamics* we shall understand a drone identifier-indexed set of timed drone dynamics.
545. By a *record of [traces of] timed drone dynamics* we shall understand a drone identifier-indexed set of sequences of timed drone dynamics.

#### type

- 543  $\text{TiDYN} = \mathbb{T} \times \text{POS} \times \text{DYN} \times \text{ImG}$   
 544  $\text{CuDD} = (\text{EDI} \xrightarrow{m} \text{TiDYN}) \cup (\text{ODI} \xrightarrow{m} \text{TiDYN})$   
 545  $\text{RoDD} = (\text{EDI} \xrightarrow{m} \text{TiDYN}^*) \cup (\text{ODI} \xrightarrow{m} \text{TiDYN}^*)$

We shall use the notion of *current drone dynamics* as the means whereby the *monitor* ascertains (obtains, by interacting with drones) the dynamics of drones, and the notion of a *record of [traces of] drone dynamics* in the *monitor*.

### N.2.4.7 Drone Positions

546. For all drones whether enterprise or ‘other’, their positions must lie within the geography of their universe of discourse.

#### axiom

- 546  $\forall \text{uod}:\text{UoD}, \text{e}:\text{E}, \text{g}:\text{G}, \text{d}:(\text{ED}|\text{OD}) \bullet$   
 546  $\text{e} = \text{obs\_E}(\text{uod}) \wedge \text{g} = \text{obs\_G}(\text{uod}) \wedge \text{d} \in \text{xtr\_Ds}(\text{uod}) \Rightarrow$   
 546  $\text{let } \text{eps} = \text{attr\_EPS}(\text{g}), (\_ , \_ , \rho) = \text{attr\_DYN}(\text{d}) \text{ in } \rho \in \text{eps} \text{ end}$

### N.2.4.8 Monitor Attributes

The *monitor* “sits between” the *drones* whose dynamics it monitors and the *planner* which it provides with records of drone dynamics. Therefore we introduce the following.

547. The monitor has just one, a programmable attribute: a trace of the most recent and all past time-stamped recordings of the dynamics of all drones, that is, an element  $\text{rodd}:\text{RoDD}$ , cf. Item 545.

#### type

- 547  $\text{MRoDD} = \text{RoDD}$

#### value

- 547  $\text{attr\_MRoDD}:\text{CM} \rightarrow \text{MRoDD}$

The monitor “obtains” current drone dynamics,  $\text{cudd}:\text{CuDD}$  (cf. Item 544) from the *drones* and offers records of [traces of] drone dynamics, (cf. Item 545)  $\text{rodd}:\text{RoDD}$ , to the *planner*.

### N.2.4.9 Planner Attributes

**N.2.4.9.1 Swarms and Businesses:** The *planner* is where all decisions are made with respect to where enterprise drones should be flying; which enterprise drones fly together, which no longer – (with this notion of “flying together” leading us to the concept of *swarms*); which swarms of enterprise drones do which kinds of work – (with this notion of work specialisation leading us to the concept of businesses.)

548. The is a notion of a *business identifier*, BI.

#### type

- 548 BI

**N.2.4.9.2 Planner Directories:** Planners have three directories. These are attributes, BDIR (businesses), SDIR (swarms) and DDIR (drones).

549. BDIR records which swarms are resources of which businesses;
550. SDIR records which drones “belong” to which swarms.
551. DDIR “keeps track” of past and present enterprise drone positions, as per enterprise drone identifier.
552. We shall refer to this triplet of directories by TDIR

**type**

549 BDIR = BI  $\mapsto$  SI-set  
 550 SDIR = SI  $\mapsto$  DI-set  
 551 DDIR = DI  $\mapsto$  RoDD  
 552 TDIR = BDIR  $\times$  SDIR  $\times$  DDIR

**value**

549 attr\_BDIR: CP  $\rightarrow$  BDIR  
 550 attr\_SDIR: CP  $\rightarrow$  SDIR  
 551 attr\_DDIR: CP  $\rightarrow$  DPL

All three directories are *programmable attributes*.

The business swarm concept can be visualized by grouping together drones of the same swarm in the visualization of the aggregate set of enterprise drones. Figure N.3 attempts this visualization.

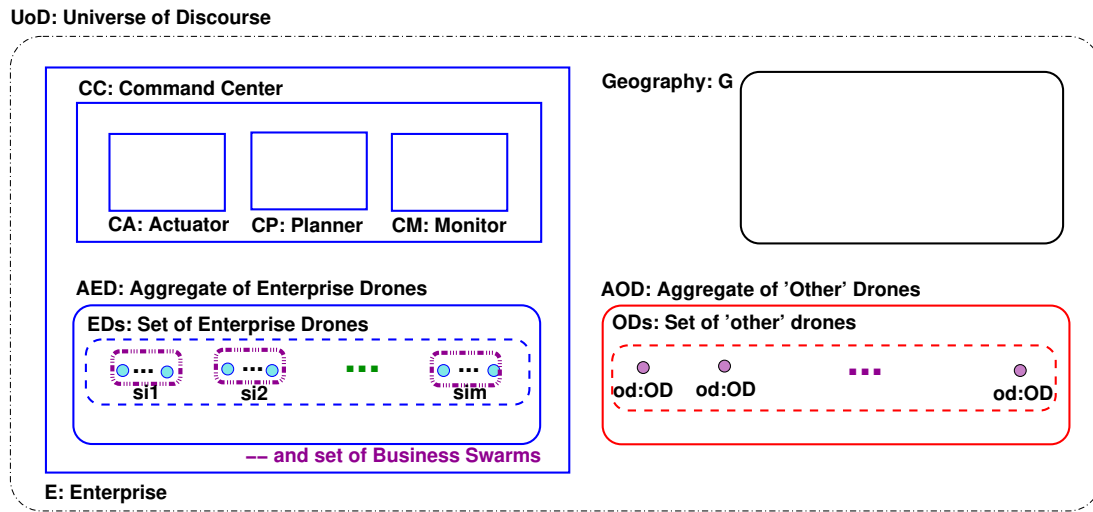


Figure N.3: Conceptual Swarms of the Universe of Discourse

553. For the planners of all universes of discourse the following must be the case.

- (a) The swarm directory must
  - i. have entries for exactly the swarms of the business directory,
  - ii. define disjoint sets of enterprise drone identifiers, and
  - iii. these sets must together cover all enterprise drones.
- (b) The drone directory must record the present position, the past positions, a list, dpl:DPL, and, besides satisfying axioms 546, satisfy some further constraints:
  - i. they must list exactly the drone identifiers of the aggregate of enterprise drones, and the sum total of its enterprise drone identifiers must be exactly those of the enterprise drones aggregate of enterprise swarms, and
  - ii. the head of a drone's present and past position list must similarly be within reasonable distance of that drone's current position.

**axiom**

553  $\forall uod:UpD, e:E, cp:CP, g:G \bullet$   
 553  $e=obs\_E(uod) \wedge cp=obs\_CP(obs\_CC(e)) \Rightarrow$   
 553a **let** (bdir,sdir,ddir) = (attr\_BDIR,attr\_SDIR,attr\_DDIR)(cp) **in**  
 553(a)i  $\cup rng\ bdir = dom\ sdir$   
 553(a)ii  $\wedge \forall si,si' \bullet \{si,si'\} \subseteq dom\ sdir \wedge si \neq si' \Rightarrow$   
 553(a)iii  $sdir(si) \cap sdir(si') = \{\}$   
 553(a)iii  $\wedge \cup rng\ sdir = xtr\_dis(e)$   
 553(b)i  $\wedge dom\ ddir = xtr\_dis(e)$   
 553(b)ii  $\wedge \forall di:Dl \bullet di \in dom\ ddir$   
 553(b)ii **let** (d,dpl) = (attr\_DDIR(cp))(di) **in**  
 553(b)ii  $dpl \neq \langle \rangle$

```

553(b)ii => neighbours(f,hd(dpl))
553(b)ii ^ neighbours(hd(dpl),
553(b)ii attr_EDPOS(xtr-D(obs_Ss(e))(di)))
553 end end

```

#### N.2.4.10 Actuator Attributes

The actuator receives, from the planner, flight directives as to which enterprise drones should be redirected. The actuator maintains a record of most recent and all past such flight directives. Finally, the actuator, effects the directives by informing designated enterprise drones as to their next flight plans.

554. Actuators have one programmable attribute: a flight directive directory. It lists, for each enterprise drone, by identifier, a pair: its current flight plan and a list of past flight plans.

##### type

```
554 FDDIR = EDI \mapsto (FP \times FP*)
```

##### value

```
554 attr_FDDIR: CA \rightarrow FDDIR
```

#### N.2.4.11 Geography Attributes

**N.2.4.11.1 Constituent Types:** The constituent types of *longitude*, *latitude* and *altitude* and *positions*, of a *geography*, were introduced in Items 488.

555. A further concept of geography is that of *area*.

556. An area,  $a:A$ , is a subset of positions within the geography.

##### type

```
555 A = P-infset
```

##### axiom

```
556 \forall uod:UoD,g:G,a:A • g=obs.G(uod) \Rightarrow a \subseteq attr_EPS(g)
```

#### N.2.4.11.2 Attributes

557. Geographies have, as one of their attributes, a *Euclidian Point Space*, in this case, a *compact*<sup>21</sup> infinite set of three-dimensional positions.

##### type

```
557 EPS = P-infset
```

##### value

```
557 attr_EPS: G \rightarrow EPS
```

Further geography attributes reflect the “lay of the land and the weather right now!”.

558. The “lay of the land”,  $L$  is a “conglomerate” further undefined geodetics and cadestra<sup>22</sup>

559. The “weather”  $W$  is another “conglomerate” of temperature, humidity, precipitation, air pressure, etc.

##### type

```
558 L
```

```
559 W
```

##### value

```
558 attr_L: G \rightarrow L
```

```
559 attr_W: G \rightarrow W
```

<sup>21</sup>In mathematics, and more specifically in general topology, compactness is a property that generalizes the notion of a subset of Euclidean space being closed (that is, containing all its limit points) and bounded (that is, having all its points lie within some fixed distance of each other). Examples include a closed interval, a rectangle, or a finite set of points.

<sup>22</sup>land surface altitude, streets, buildings (tall or not so tall), power lines, etc.

## N.3 Operations on Universe of Discourse States

Before we analyse & describe perdurants let us take a careful look at the actions that drone and swarm behaviours may take. We refer to this preparatory analysis & description as one of analysing & describing the state operations. From this analysis & description we move on to the analysis & description of behaviours, events and actions. The idea is to be able to prove some relations between the two analyses & descriptions: the state operation and the behaviour analyses & descriptions. We refer to [52, Sects. 2.3 and 2.5].

### N.3.1 The Notion of a State

A *state* is any subset of parts each of which contains one or more dynamic attributes. Following are examples of states of the present case study: a space of interest, an aggregate of ‘business’ swarms, an aggregate of ‘other’ swarms, a pair of the aggregates just mentioned, a swarm, or a drone.

### N.3.2 Constants

Some quantities of a given universe of discourse are constants. Examples are the unique identifiers of the:

- |                                          |                                             |
|------------------------------------------|---------------------------------------------|
| 560. enterprise, $e_i$ ;                 | 565. planner, $cp_i$ ;                      |
| 561. aggregate of ‘other’ drones, $oi$ ; | 566. actuator, $ca_i$ ;                     |
| 562. geography, $g_i$ ;                  | 567. set of ‘other’ drones, $od_{is}$ ;     |
| 563. command center, $cc_i$ ;            | 568. set of enterprise drones, $ed_{is}$ ;  |
| 564. monitor, $cm_i$ ;                   | 569. and the set of all drones, $ad_{is}$ . |

#### value

- ```

560 aedi:EI = uid_AED(obs_AED(uod))
561 aodi:OI = uid_AOD(obs_AOD(uod))
562 gi:GI = uid_G(obs_G(uod))
563 cci:CCI = uid_CC(obs_CC(obs_AED(uod)))
564 cmi:CMI = uid_CM(obs_CM(obs_CC(obs_AED(uod))))
565 cpi:CPI = uid_CP(obs_CP(obs_CC(obs_AED(uod))))
566 cai:CAI = uid_CA(obs_CA(obs_CC(obs_AED(uod))))
567 odis:ODIs = xtr_dis(obs_AOD(uod))
568 edis:EDIs = xtr_dis(obs_AED(uod))
569 adis:DI-set = odis ∪ edis

```

N.3.3 Operations

An *operation* is a function from states to states. Following are examples of operations of the present case study: a drone *transfer*: leaving a swarm to join another swarm, a drone *changing course*: an enterprise drone changing course, a swarm *split*: a swarm splitting into two swarms, and swarm *join*: two swarms joining to form one swarm.

N.3.3.1 A Drone Transfer

570. The *transfer* operator specifies two distinct and unique identifiers, si , si' , of two enterprise swarms, and the unique identifier, di , of an enterprise drone – all of the same universe of discourse. The *transfer* operation further takes a universe of discourse and yields a universe of discourse as follows:
571. The input argument ‘from’ and ‘to’ swarm identifiers are different.
572. The initial and the final state aggregates of enterprise drones, ‘other’ drones and geographies are unchanged.
573. The initial and final state monitors and actuators are unchanged.
574. The business and the drone directors of the initial and final planner are unchanged.
575. The ‘from’ and ‘to’ input argument swarm identifiers are in the swarm directory and the input argument drone identifiers is in the initial swarm directory entry for the ‘from’ swarm identifier.
576. The input argument drone identifier is in final the swarm directory entry for the ‘to’ swarm identifier.
577. And the final swarm directory is updated ...

value

- ```

570 transfer: DI × SI × SI → UoD $\xrightarrow{\sim}$ UoD
570 transfer(di,fsi,tsi)(uod) as uod'
571 fsi ≠ tsi ∧
570 let aed = obs_AED(uod), aed' = obs_AED(uod'), g = obs_G(uod), g' = obs_G(uod') in

```

```

570 let cc = obs_CC(aed), cc' = obs_CC(aed'), aod = obs_AOD(uod), aod' = obs_AOD(uod') in
570 let cm = obs_CM(cc), cm' = obs_CM(cc'), cp = obs_CP(cc), cp' = obs_CP(cc') in
570 let ca = obs_CA(cc), ca' = obs_CA(cc') in
570 let bdir = attr_BDIR(cc), bdir' = attr_BDIR(cc'),
570 sdir = attr_SDIR(cc), sdir' = attr_SDIR(cc'),
570 ddir = attr_DDIR(cc), ddir' = attr_DDIR(cc') in
572 post: aed = aed' ∧ aod = aod' ∧ g = g' ∧
573 cm = cm' ∧ ca = ca' ∧
574 bdir = bdir' ∧ ddir = ddir'
575 pre {fsi,tsi} ⊆ dom sdir ∧ di ∈ sdir(fsi)
576 post di ∉ sdir(fsi') ∧ di ∈ sdir(tsi') ∧
577 sdir' = sdir † [fsi→sdir(fsi) ∪ di] † [tsi→sdir(tsi)\di]
570 end end end end end

```

### N.3.3.2 An Enterprise Drone Changing Course

TO BE WRITTEN

### N.3.3.3 A Swarm Splitting into Two Swarms

TO BE WRITTEN

### N.3.3.4 Two Swarms Joining to form One Swarm

TO BE WRITTEN

### N.3.3.5 Etcetera

TO BE WRITTEN

## N.4 Perdurants

We observe that the term *train* can have the following “meanings”: the *train*, as an *endurant*, parked at the railway station platform, i.e., as a *composite part*; the *train*, as a *perdurant*, as it “speeds” down the railway track, i.e., as a *behaviour*; the *train*, as an *attribute*. This observation motivates that we “magically”, as it were, introduce a [Translate<sub>r</sub>](#) function, cf. [51, Sect. 4]

### N.4.1 System Compilation

The [Translate<sub>r</sub>](#) function “worms” its way, so-to-speak, “down” the “hierarchy” of parts, from the universe of discourse, via its immediate sup-parts, and from these to their sub-parts, and so on, until the [Translate<sub>r</sub>](#) reaches atomic parts. We shall henceforth do likewise.

#### N.4.1.1 The Compile Functions

578. Compilation of a *universe of discourse* results in

- (a) the [AMoL-text](#) of the *core* of the universe of discourse behaviour (which we set to **skip** – allowing us to ignore *core* arguments),
- (b) followed by the [AMoL-text](#) of the parallel composition of the compilation of the enterprise,
- (c) followed by the [AMoL-text](#) of the parallel composition of the compilation of the geography,
- (d) followed by the [AMoL-text](#) of the parallel composition of the compilation of the aggregate of ‘other’ drones.

```

578 TranslateUoD(uod) ≡
578a Muid_UoD(uod)(mereo_UoD(uod),sta(uod))(pro(uod))
578b || TranslateAED(obs_AED(uod))
578c || TranslateG(obs_G(uod))
578d || TranslateAOD(obs_AOD(uod))

```

579. Compilation of an *enterprise* results in

- (a) the [AMoL-text](#) of the *core* of the enterprise behaviour (which we set to **skip** – allowing us to ignore *core* arguments),



- (b) followed by the **AMoL-text** of the parallel composition of the compilation of the enterprise aggregate of enterprise drones,
- (c) followed by the **AMoL-text** of the parallel composition of the compilation of the enterprise command center.

579 **Translate**<sub>AED</sub>(e)  $\equiv$   
 579a  $\mathcal{M}_{\text{uid\_AED}}(e)(\text{mereo\_E}(e), \text{sta}(e))(\text{pro}(e))$   
 579b  $\parallel \text{Translate}_{EDs}(\text{obs\_EDs}(e))$   
 579c  $\parallel \text{Translate}_{CC}(\text{obs\_CC}(e))$

580. Compilation of an *enterprise aggregate of enterprise drones* results in

- (a) the **AMoL-text** of the *core* of the aggregate behaviour (which we set to **skip** – allowing us to ignore *core* arguments),
- (b) followed by the **AMoL-text** of the parallel composition of the distributed compilation of the enterprise aggregate's set of enterprise drones.

580 **Translate**<sub>EDs</sub>(es)  $\equiv$   
 580a  $\mathcal{M}_{\text{uid\_EDs}}(es)(\text{mereo\_EDS}(es), \text{sta}(es))(\text{pro}(es))$   
 580b  $\parallel \{ \text{Translate}_{ED}(\text{ed}) \mid \text{ed} : \text{ED} \bullet \text{ed} \in \text{obs\_EDs}(s) \}$

581. Compilation of an *enterprise drone* results in

- (a) the **AMoL-text** of the *core* of the enterprise drone behaviour – which is what we really wish to express – and since enterprise drones are here considered atomic, that is where the compilation of enterprise ends.

581 **Translate**<sub>ED</sub>(ed)  $\equiv$   
 581a  $\mathcal{M}_{\text{uid\_ED}}(\text{ed})(\text{mereo\_ED}(\text{ed}), \text{sta}(\text{ed}))(\text{pro}(\text{ed}))$

582. Compilation of an *aggregate of 'other' drones* results in

- (a) the **AMoL-text** of the *core* of the aggregate 'other' drones behaviour (which we set to **skip** – allowing us to ignore *core* arguments) –
- (b) followed by the **AMoL-text** of the parallel composition of the distributed compilation of the 'other' drones in the 'other' drones' aggregate set of 'other' drones.

582 **Translate**<sub>AOD</sub>(aod)  $\equiv$   
 582a  $\mathcal{M}_{\text{uid\_OD}}(\text{od})(\text{mereo\_S}(\text{ods}), \text{sta}(\text{ods}))(\text{pro}(\text{ods}))$   
 582b  $\parallel \{ \text{Translate}_{OD}(\text{od}) \mid \text{od} : \text{OD} \bullet \text{od} \in \text{obs\_ODs}(\text{ods}) \}$

583. Compilation of a(n) *'other' drone* results in

- (a) the **AMoL-text** of the *core* of the 'other' drone behaviour – which is what we really wish to express – and since 'other' drones are here considered atomic, that is where the compilation of the 'other' drones aggregate

583a **Translate**<sub>{OD}</sub>(ed)  $\equiv$   
 583a  $\mathcal{M}_{\text{uid\_OD}}(\text{od})(\text{mereo\_OD}(\text{od}), \text{sta}(\text{od}))(\text{pro}(\text{od}))$

584. Compilation of an atomic *geography* results in

- (a) the **AMoL-text** of the *core* of the geography behaviour.

584 **Translate**<sub>G</sub>(g)  $\equiv$   
 584a  $\mathcal{M}_{\text{uid\_G}}(g)(\text{mereo\_G}(g), \text{sta}(g))(\text{pro}(g))$

585. Compilation of a composite *command center* results in

- (a) the **AMoL-text** of the *core* of the command center behaviour (which we set to **skip** – allowing us to ignore *core* arguments)
- (b) followed by the **AMoL-text** of the parallel composition of the compilation of the command monitor,

- (c) followed by the **AMoL-text** of the parallel composition of the compilation of the command planner,
- (d) followed by the **AMoL-text** of the parallel composition of the compilation of the command actuator.

585 **Translate**<sub>M</sub>(cc)  $\equiv$   
585a  $\mathcal{M}_{\text{uid\_CC}}(\text{mereo\_CC}(\text{cc}), \text{sta}(\text{cc}))(\text{pro}(\text{cc}))$   
585b  $\parallel$  **Translate**<sub>CC</sub>(obs\_CM(cc))  
585c  $\parallel$  **Translate**<sub>CP</sub>(obs\_CP(cc))  
585d  $\parallel$  **Translate**<sub>CA</sub>(obs\_CA(cc))

586. Compilation of an atomic *command monitor* results in
- (a) the **AMoL-text** of the *core* of the monitor behaviour.

586 **Translate**<sub>CM</sub>(cm)  $\equiv$   
586a  $\mathcal{M}_{\text{uid\_CM}}(\text{mereo\_CM}(\text{cm}), \text{sta}(\text{cm}))(\text{pro}(\text{cm}))$

587. Compilation of an atomic *command planner* results in
- (a) the **AMoL-text** of the *core* of the planner behaviour.

587 **Translate**<sub>CP</sub>(cp)  $\equiv$   
587a  $\mathcal{M}_{\text{uid\_CP}}(\text{mereo\_CP}(\text{cp}), \text{sta}(\text{cp}))(\text{pro}(\text{cp}))$

588. Compilation of an atomic *command actuator* results in
- (a) the **AMoL-text** of the *core* of the actuator behaviour.

588 **Translate**<sub>CA</sub>(ca)  $\equiv$   
588a  $\mathcal{M}_{\text{uid\_CA}}(\text{mereo\_CA}(\text{ca}), \text{sta}(\text{ca}))(\text{pro}(\text{ca}))$

#### N.4.1.2 Some CSP Expression Simplifications

We can justify the following CSP simplifications [108, 112, 161, 164]:

589. **skip** in parallel with any CSP expression *csp* is *csp*.
590. The distributed parallel composition of the distributed parallel composition of CSP expressions,  $\text{csp}(i,j)$ , *i* indexed over *I*, i.e.,  $i:I$ , and *j*:*J* respectively, is the distributed parallel composition over CSP expressions,  $\text{csp}(i,j)$ , i.e., indexed over  $(i,j):I \times J$  – where the index sets *iset* and *jset* are assumed.

**axiom**

590 **skip**  $\parallel$  *csp*  $\equiv$  *csp*  
590  $\parallel \{ \{ \text{csp}(i,j) \mid i:I \in \text{iset} \} \mid j:J \in \text{jset} \} \equiv \parallel \{ \text{csp}(i,j) \mid i:I, j:J \in \text{iset} \wedge j \in \text{jset} \}$

#### N.4.1.3 The Simplified Compilation

591. The simplified compilation results in:

591 **Translate**(uod)  $\equiv$   
581a  $\{ \mathcal{M}_{\text{uid\_ED}}(\text{mereo\_ED}(\text{ed}), \text{sta}(\text{ed}))(\text{pro}(\text{ed}))$   
581a  $\mid \text{ed}:\text{ED} \bullet \text{ed} \in \text{xtr\_Ds}(\text{obs\_AED}(\text{uod})) \}$   
583a  $\parallel \{ \mathcal{M}_{\text{uid\_OD}}(\text{mereo\_OD}(\text{od}), \text{sta}(\text{od}))(\text{pro}(\text{od}))$   
583a  $\mid \text{od}:\text{OD} \bullet \text{od} \in \text{xtr\_ODs}(\text{obs\_AOD}(\text{uod})) \}$   
584a  $\parallel \mathcal{M}_{\text{uid\_G}}(\text{mereo\_G}(\text{g}), \text{sta}(\text{g}))(\text{pro}(\text{g}))$   
584a **where**  $\text{g} \equiv \text{obs\_G}(\text{uod})$   
586a  $\parallel \mathcal{M}_{\text{uid\_CM}}(\text{cm}) (\text{mereo\_CM}(\text{cm}), \text{sta}(\text{cm}))(\text{pro}(\text{cm}))$   
586a **where**  $\text{cm} \equiv \text{obs\_CM}(\text{obs\_CC}(\text{obs\_E}(\text{uod})))$   
587a  $\parallel \mathcal{M}_{\text{uid\_CP}}(\text{cp}) (\text{mereo\_CP}(\text{cp}), \text{sta}(\text{cp}))(\text{pro}(\text{cp}))$   
587a **where**  $\text{cp} \equiv \text{obs\_CP}(\text{obs\_CC}(\text{obs\_E}(\text{uod})))$   
588a  $\parallel \mathcal{M}_{\text{uid\_CA}}(\text{ca}) (\text{mereo\_CA}(\text{ca}), \text{sta}(\text{ca}))(\text{pro}(\text{ca}))$   
588a **where**  $\text{ca} \equiv \text{obs\_CA}(\text{obs\_CC}(\text{obs\_E}(\text{uod})))$

592. In Item 591's Items 581a, 583a, 584a, 586a, 587a, and 588a we replace the “anonymous” behaviour names  $\mathcal{M}$  by more meaningful names.

```

592 Translate(uod) ≡
581a { enterprise_drone_uid_ED(ed)(mereo_ED(ed),sta(ed))(pro(ed))
581a | ed:ED • ed ∈ xtr_Ds(obs_AED(uod)) }
583a || { other_drone_uid_OD(od)(mereo_OD(od),sta(od))(pro(od))
583a | od:OD • od ∈ xtr_ODs(obs_AOD(uod)) }
584a || geography_uid_G(g)(mereo_G(g),sta(g))(pro(g))
584a where g ≡ obs_G(uod)
586a || monitor_uid_CM(cm)(mereo_CM(cm),sta(cm))(pro(cm))
586a where cm ≡ obs_CM(obs_CC(obs_E(uod)))
587a || planner_uid_CP(cp)(mereo_CP(cp),sta(cp))(pro(cp))
587a where cp ≡ obs_CP(obs_CC(obs_E(uod)))
588a || actuator_uid_CA(ca)(mereo_CA(ca),sta(ca))(pro(ca))
588a where ca ≡ obs_CA(obs_CC(obs_E(uod)))

```

## N.4.2 An Early Narrative on Behaviours

### N.4.2.1 Either Endurants or Perdurants, Not Both !

First the reader should observe that the manifest parts, in some sense, do no longer “exist” ! They have all been replaced by their corresponding behaviours. These behaviours embody all the qualities of their “origin”: the unique identifiers, the mereology, and all the attributes – the latter in one form or another: the static attributes as constants (referred to in the bodies of the behaviour definitions); the programmable attributes as arguments (“carried over” from one invocation to the next); and the remaining dynamic attributes as “inputs” (whose varying values are “accessed” through [dynamic attribute] channels).

### N.4.2.2 Focus on Some Behaviours, Not All !

Secondly we focus, in this case study, only on the behaviour of the *planner*. The other behaviours, the ‘*other*’ *drones*, *enterprise drones*, *monitor*, *actuator*, and the *geography*, are, in this case study of less interest to us. That is, other case studies could focus on the behaviours of *drones*, or *geographies*, or *monitor*, or *actuator*.

### N.4.2.3 The Behaviours – a First Narrative

*Drones* “continuously” offer their identified dynamics (location, velocity, and possibly more) **to** the *monitor*. *Enterprise drones* “continuously”, and in addition, offers to accept flight guidance **from** the *actuator*. The *monitor* “continuously sweeps” the air space and collects the identities of all recognizable drones and their dynamics, and offers this **to** the *planner*. The *planner* does all the interesting work ! It effects the *allocation/reallocation* of drones to/from business swarms; it *calculates enterprise drone flights* and instructs the *actuator* to offer such flight plans to relevant drones; etcetera ! Finally the *actuator*, as instructed by the *planner*, offers flight guidance, as per instructions **from** the *planner*, **to** all or some *enterprise drones*.

## N.4.3 Channels

Channels is a concept of CSP [108, 111, 112].

CSP channels are a means for synchronising behaviours and for communicating values between synchronised behaviours, as well as, as a technicality, conveying values of most kinds of dynamic attributes of parts (i.e., endurants) to “their” behavioural counterparts.

There are thus two starting point for the analysis & description of channels: the mereologies and the dynamic attributes of parts. Here we shall single out the following parts and behaviours: the command *monitor*, *planner* and *actuator*, the *enterprise drones* and the ‘*other*’ *drones*, and the *geography*. We refer to Fig. N.4 on the following page, a slight “refinement” of Fig. N.1 on page 387.

### N.4.3.1 The Part Channels

**N.4.3.1.1 General Remarks:** Let there be given a *universe of discourse*. Let us analyse the *unique identifiers* and the *mereologies* of the *planner* cp: (cpi, cpm), *monitor* cm: (cmi, cmm) and *geography* g: (mi, mm), where cpm = (cai, cmi, gi), cmm = ({di<sub>1</sub>, di<sub>2</sub>, . . . , di<sub>n</sub>}, cpi) and gm = (cpi, {di<sub>1</sub>, di<sub>2</sub>, . . . , di<sub>n</sub>}).

We now interpret these facts. When the *planner mereology* specifies the unique identifiers of the *actuator*, the *monitor*, and the *geography*, then that shall mean there there is a way of communicating messages between the actuator, and the *geography*, and one side, and the planner on the other side.

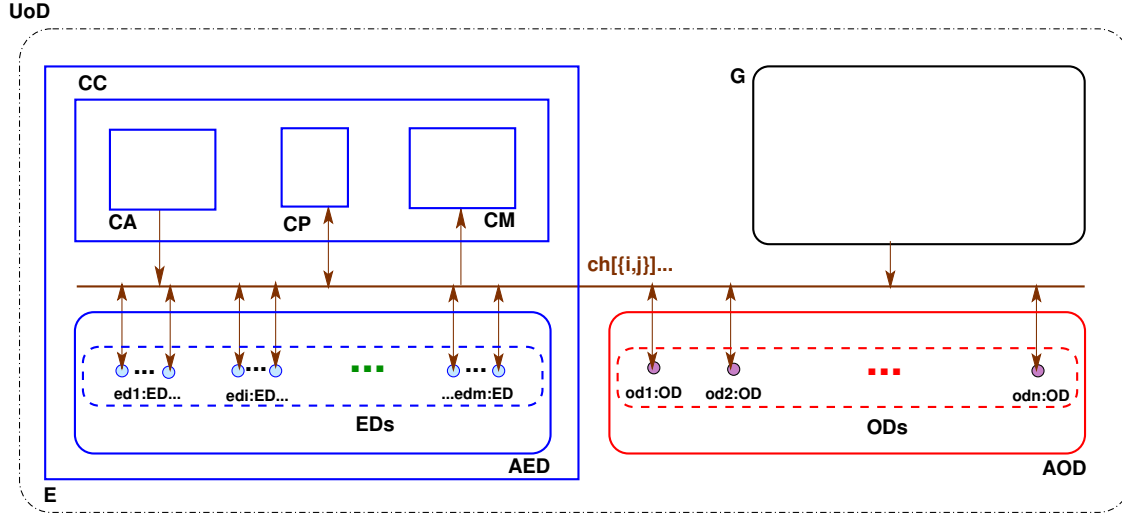


Figure N.4: Universe of Discourse with General Channel:  $\text{ch}[\{i,j\}] \dots$

593. We shall therefore, in a first step of specification development, think of a “grand” array channel over which all communication between behaviours take place. See Fig. N.4.
594. Example indexes into this array channel are shown in the formulas just below.

**type**

593 MSG

**channel**

593  $\{\text{ch}[\text{fui},\text{tui}]|\text{fui},\text{tui}:\text{PI} \bullet \dots\}:\text{MSG}$

**value**

594  $\text{ch}[\text{cpi},\text{cai}]!\text{msg}$  *output from planner to actuator.*

594  $\text{ch}[\text{cpi},\text{cai}]?\text{input}$  *from planner to actuator.*

594 etc.

We presently leave the type of messages, MSG, that can be communicated over this “grand” channel further unspecified. We also leave unspecified the pair of distinct unique identifiers that index the channel array. We emphasize that the uniqueness of all part identifiers allow us to use pairs of such as indices. Expression  $\text{ch}[\text{fui},\text{tui}]!\text{sg}$  thus expresses *output from* behaviour indexed by *fui* *to* behaviour indexed by *tui*, whereas expression  $\text{ch}[\text{tui},\text{fui}]?$  thus expresses *input from* behaviour indexed by *tui* *to* behaviour indexed by *fui*. Not all combinations of unique identifiers are needed. The channel array is “sparse”! That property allows us to refine the “grand” channel into the channels illustrated on Fig. N.5 on the next page. Some channels are array channels: The channels to the drones whether all drones, or just the enterprise drones. Other channels are “single” channels: these are the channels which are anchored in parts with a priori known, i.e., constant unique identifiers.

#### N.4.3.1.2 Part Channel Specifics

595. There is an array channel,  $\text{d\_cm\_ch}[\text{di},\text{cm}_i]:\text{D\_CM\_MSG}$ , from any *drone* ( $[\text{di}]$ ) behaviour to the *monitor* behaviour (whose unique identifier is  $\text{cm}_i$ ). The channel, as an array, forwards the current drone dynamics  $\text{D\_CM\_MSG} = \text{CuDD}$ .

**type**

595  $\text{D\_CM\_MSG} = \text{CuDD}$

**channel**

595  $\{\text{d\_cm\_ch}[\text{di},\text{cm}_i]|\text{di}:(\text{EDI}|\text{ODI})\bullet\text{di} \in \text{dis}\}:\text{D\_CM\_MSG}$

596. There is a channel,  $\text{cm\_cp\_ch}[\text{cm}_i,\text{cp}_i]$ , from the *monitor* behaviour ( $\text{cm}_i$ ) to the *planner* behaviour ( $\text{cp}_i$ ). It forwards the monitor’s records of drone dynamics  $\text{CM\_CP\_MSG} = \text{MRoDD}$ .

**type**

596  $\text{CM\_CP\_MSG} = \text{MRoDD}$

596 **channel**  $\text{m\_cp\_ch}[\text{cm}_i,\text{cp}_i]:\text{CM\_CP\_MSG}$

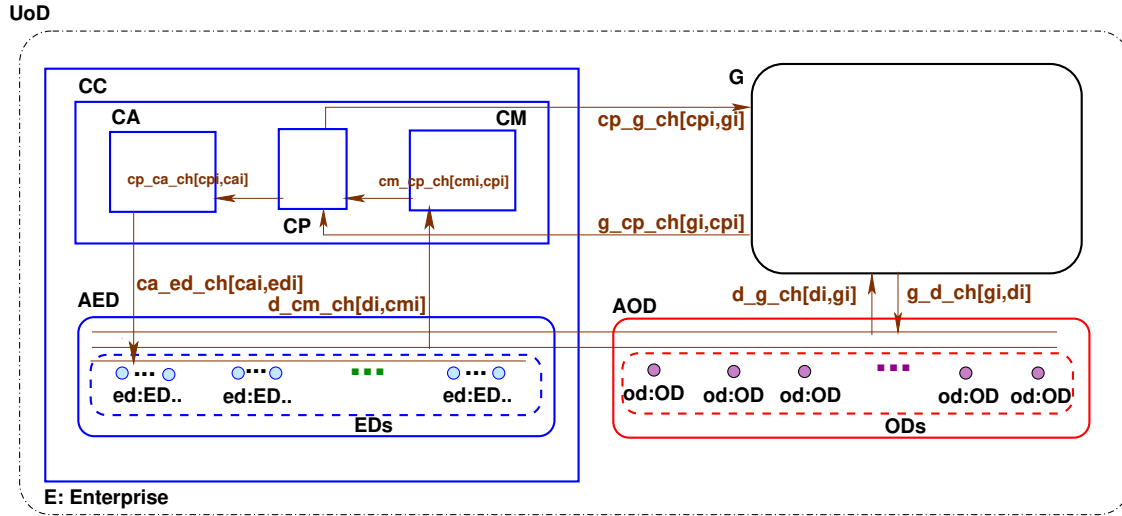


Figure N.5: Universe of Discourse with Specific Channels

597. There is a channel,  $cp\_ca\_ch[cp_i,ca_i]:CP\_CA\_MSG$ , from the *planner* behaviour ( $cp_i$ ) to the *actuator* behaviour ( $ca_i$ ). It forwards flight plans  $CP\_CA\_MSG = FP$ .

**type**  
 597  $CP\_CA\_MSG = EID \xrightarrow{m} FP$   
**channel**  
 597  $cp\_ca\_ch[cp_i,ca_i]:CM\_CP\_MSG$

598. There is an array channel,  $ca\_ed\_ch[cai,edi]$ , from the *actuator behaviour* ( $ca_i$ ) to the *enterprise drone* behaviours ( $edi$  for suitable edis). It forwards flight plans,  $CA\_ED\_MSG = FP$ , to enterprise drones in a designated set.

**type**  
 598  $CA\_ED\_MSG = EID \times FP$   
**channel**  
 598  $\{ca\_ed\_ch[cai,edi] | edi:EDI \bullet edi \in edis\}:CA\_ED\_MSG$

599. There is an array channel,  $d\_g\_ch[di,gi]:D\_G\_MSG$ , from all the *drone* behaviours ( $di$ ) to the *geography* behaviour. The channels convey, requests for an *immediate geography* for and around a *point*:  $D\_G\_MSG = P$ .

**type**  
 599  $D\_G\_MSG = P$   
**channel**  
 599  $\{d\_g\_ch[di,gi] | di:(EDI|ODI) \bullet di \in dis\}:D\_H\_MSG$

600. There is an array channel,  $g\_d\_ch[gi,di]:G\_D\_MSG$ , from the *geography* behaviour to all the *drone* behaviours. The channels convey, for a requested *point*, the immediate geography for that area:  $G\_D\_MSG = ImG$ .

**type**  
 600  $G\_D\_MSG = ImG$   
**channel**  
 600  $\{g\_d\_ch[gi,di] | di:(EDI|ODI) \bullet di \in dis\}:G\_D\_MSG$

### N.4.3.2 Attribute Channels, General Principles

Some of the drone attributes are *reactive*. Being reactive means that their values change surreptitiously. In the physical world of parts that means that these vales must be measured, or somehow ascertained, whenever needed, i.e., “on the fly”. Now “our world” is that of a domain description. When dealing with enduring, the value of an attribute,  $a:A$ , of part  $p:P$ , is expressed as  $attr\_A(p)$ . When dealing with perdurants, that same value is to be expressed as  $attr\_A\_ch[uid\_P(p)]?$ .

601. This means that we must declare a channel for each part with one or more *dynamic*, however not including *programmable*, attributes  $A_1, A_2, \dots, A_n$ .

**channel**

601  $\text{attr\_A1\_ch}[p_i]:A_1, \text{attr\_A2\_ch}[p_i]:A_2, \dots, \text{attr\_An\_ch}[p_i]:A_n$

602. If there are several parts,  $p_1, p_2, \dots, p_m:P$  then an array channel over indices  $p_1, p_2, \dots, p_m$  is declared for each applicable attribute.

**channel**

602  $\{\text{attr\_A1\_ch}[p_j] \mid p_j:P \bullet p_j \in \{p_1, p_2, \dots, p_m\}\}:A_1,$

602  $\{\text{attr\_A2\_ch}[p_j] \mid p_j:P \bullet p_j \in \{p_1, p_2, \dots, p_m\}\}:A_2,$

602 ...

602  $\{\text{attr\_An\_ch}[p_j] \mid p_j:P \bullet p_j \in \{p_1, p_2, \dots, p_m\}\}:A_n$

### N.4.3.3 The Case Study Attribute Channels

**N.4.3.3.1 ‘Other’ Drones:** ‘Other’ drones have the following not biddable or programmable dynamic channels:

603. dynamics, including velocity, acceleration, orientation and position,  
 $\{\text{attr\_DYN\_ch}[odi]:\text{DYN} \mid odi:\text{ODI} \bullet odi \in \text{odis}\}.$

**channel**

603  $\{\text{attr\_DYN\_ch}[odi]:\text{DYN} \mid odi:\text{ODI} \bullet odi \in \text{odis}\}$

**N.4.3.3.2 Enterprise Drones:** Enterprise drones have the following not biddable or programmable dynamic channels:

604. dynamics, including velocity, acceleration, orientation and position,  
 $\{\text{attr\_DYN\_ch}[edi]:\text{DYN} \mid edi:\text{EDI} \bullet edi \in \text{edis}\}.$

**channel**

604  $\{\text{attr\_DYN\_ch}[odi]:\text{DYN} \mid odi:\text{ODI} \bullet odi \in \text{odis}\}$

**N.4.3.3.3 Geography:** The geography has the following not biddable or programmable dynamic channels:

605. land,  $\text{attr\_L\_ch}[g_i]:L$ , and

606. weather,  $\text{attr\_W\_ch}[g_i]:W$ .

**channel**

605  $\text{attr\_L\_ch}[g_i]:L$

606  $\text{attr\_W\_ch}[g_i]:W$

We do not show any graphics for the attribute channels.

## N.4.4 The Atomic Behaviours

TO BE WRITTEN

### N.4.4.1 Monitor Behaviour

607. The signature of the **monitor** behaviour

- (a) lists the **monitor**’s unique identifier, carries the **monitor**’s mereology, has no static arguments (... maybe ...), has the programmable time-stamped recordings, **dtp**, of all drone positions (present and past) and
- (b) further designates the **input** channel **d\_cm\_ch**[\*.\*] from all **drones** and the channel **output** **cm\_cp\_ch**[cmi,cpi] to the **planner**.

608. The **monitor** [otherwise] behaves as follows:

- (a) All **drones** provide as input, **d\_cm\_ch**[di,cmi]?, their time-stamped positions, **rec**.
- (b) The programmable **mrodd** attribute is updated, **mrodd**’, to reflect the latest time stamped dynamics per drone identifier.
- (c) The updated attribute is provided to the **planner**.
- (d) Then the **monitor** resumes being the **monitor**, forwarding, as the programmable attribute, the time-stamped drone position recording.

```

value
607a monitor: cmi:CMI×cmm:(dis:D1-set×cpi:CPI) → MRoDD →
607b in {d_cm_ch[di,cmi]|di:D1•di∈dis} out cm_cp_ch Unit
608 monitor(mi,(dis,cpi))(mrodd) ≡
608a let rec = {[di → d_cm_ch[di,cmi]?|di:D1•di∈dis]} in
608b let mrodd' = mrodd † [di→(rec(di))^mrodd(di)|di:D1•di∈dis] in
608c cm_cp_ch[cmi,cpi] † mrodd';
608d monitor(cmi,(dis,cpi))(mrodd')
608 end end
608 axiom cmi=cmi∧cpi=cpi

```

We have decided to let the **monitor** maintain the present and past time-stamped drone positions. It is the **monitor** which records these positions. Not the **planner**. But the **monitor** provides these traces, again-and-again, to the **planner**.

#### N.4.4.2 Planner Behaviour

609. The signature of the **planner** behaviour

- (a) lists the **planner**'s unique identifier, carries the planner's mereology, has, perhaps ..., some static arguments, has the programmable planner directories, and
- (b) further designates the single **input** channel **cm\_cp\_ch** and the single **output** channel **cp\_ca\_ch**.

610. The **planner** [otherwise] behaves as follows:

- (a) the **planner** [internal] non-deterministically ("coin-flipping") decides whether to transfer a drone between business swarms, or to calculate flight plans, or ... other.
- (b) Depending on the [outcome of the "coin-flipping"] the **planner**
- (c) either effects a transfer,
  - i. by delegating to an auxiliary function, **transfer**, the necessary modifications of the swarm directory –
  - ii. whereupon the **planner** behaviour resumes;
- (d) or effects a [re-]calculation on drone flights,
  - i. by, again, delegating to an auxiliary function, **flight\_planning**, the necessary calculations –
  - ii. which are communicated to the **actuator**,
  - iii. whereupon the **planner** behaviour resumes;
- (e) or ... other!

```

value
610 planner: cpi:CPI × (cai>CAI×cmi:CMI×gi:GI) × TDIR →
610 in cm_cp_ch[cmi,cpi], g_cp_ch[gi,cpi] out cp_ca_ch[cpi,cai] Unit
609 planner(cpi,(cai,cmi,gi),...)(bdir,sdir,ddir) ≡
610a let cmd = "transfer" † "flight_plan" † ... in
610b cases cmd of
610c "transfer" →
610(c)i let sdir' = transfer(tdir) in
610(c)ii planner(cpi,(cai,cmi,gi),...)(bdir,sdir',ddir) end
610d "flight_plan" →
610(d)i let ddir' = flight_planning(tdir) in
610(d)ii planner(cpi,(cai,cmi,gi),...)(bdir,sdir,ddir') end
610e ...
609 end
609 axiom cpi=cpi∧cai=cai∧cmi=cmi∧gi=gi

```

##### N.4.4.2.1 The Auxiliary transfer Function

611. The **transfer** function has a simpler signature than the **planner** behaviour in that it need not communicate with other behaviours.

- (a) The **transfer** function *internal non-deterministically chooses* a business designator, **bi**;
- (b) from among that business' swarm designators it *internal non-deterministically chooses* two distinct swarm designators, **fsi,tsi**;
- (c) and from the **fsi** entry in **sdir** ( which is set of enterprise drone identifiers), it *internal non-deterministically chooses* an enterprise drone identifier, **di**.

- (d) Given the swarm and drone identifiers *the resulting swarm directory* can now be made to reflect the transfer: reference to *di* is *removed* from the *fsi* entry in *sdir* and that reference instead *inserted* into the *tsi* entry.

**value**

```

611 transfer: TDIR → SDIR
611 transfer(bdir,sdir,ddir) ≡
611a let bi:BI•bi ∈ dom bdir in
611b let fsi,tsi:SI•{fsi,tsi} ⊆ bdir(bi) ∧ fsi ≠ tsi in
611c let di:DI•di ∈ sdir(fsi) in
611d sdir † [fsi → sdir(fsi) \ {di}] † [tsi → sdir(tsi) ∪ {di}]
611 end end end

```

#### N.4.4.2.2 The Auxiliary flight\_planning Function

612. The signature of the *flight\_planning* behaviour needs two elements: the triplet of business, swarm and drone directories, and the planner-to-actuator channel.

- (a) The *flight\_planning* behaviour offers to accept the time-stamped recordings of the most recent drone positions and dynamics as well as all the past such recordings.
- (b) The *flight\_planning* behaviour selects, *internal, non-deterministically* a business, designated by *bi*,
- (c) one of whose swarms, designated by *si*, it has thus decided to perform a flight [re-]calculation for.
- (d) An objective for the new flight plan is chosen.
- (e) The *flight\_plan* is calculated.
- (f) That flight plan is communicated to the *actuator*.
- (g) And the flight plan, appended to the drone directory's (past) flight plans.

**value**

```

612 flight_planning: TDIR → in cm_cp_ch[cmi,cpi], out cp_ca_ch[cpi,cai] DTP
612 flight_planning(bdir,sdir,ddir) ≡
612a let dtp = cm_cp_ch[cpi,cai] ? ,
612b bi:BI • bi ∈ dom bdir
612c let si:SI • si ∈ bdir(bi) in
612d let fp_obj:fp_objective(bi,si) in
612e let flight_plan = calculate_flight_plan(dtp,sdir(si),fp_obj,tdir) in
612f cp_ca_ch[cpi,cai] ! flight_plan ;
612g (flight_pla) ^ ddir
612 end end end end

```

**type**

612d FP\_OBJ

**value**

612d fp\_objective: BI × SI → FP\_OBJ

612d fp\_objective(bi,si) ≡ ...

613. The *calculate\_flight\_plan* function is the absolute focal point of the *planner*.

613 calculate\_flight\_plan: DTP × DI-set × FP-Obj × TDIR → FP

613 calculate\_flight\_plan(dtp,sdir(si),fp\_obj,tdir) ≡ ...

There are many ways of calculating flight plans.

[139, Mehmood et al., Stony Brook, 2018: *Declarative vs Rule-based Control for Flocking Dynamics*] is one such:

TO BE WRITTEN

In [157–159, Craig Reynolds: *OpenSteer, Steering Behaviours for Autonomous Characters*]

TO BE WRITTEN

In [143, Reza Olfati-Saber: *Flocking for Multi-agent Dynamic Systems: Algorithms and Theory*, 2006]

TO BE WRITTEN



The `calculate_flight_plan` function, Item 613 on the facing page, is deliberately provided with all such information that can be gathered and hence can be the only ‘external’<sup>23</sup> data that can be provided to such calculation functions,<sup>24</sup> and is therefore left further unspecified; future work<sup>25</sup> will show whether this assumption holds. If it does, then, OK, and we can proceed. If it does not, we shall revise the present model.

### N.4.4.3 Actuator Behaviour

614. The actuator accepts a **current flight plan**, `cfp:CFP`, i.e., a number of enterprise drone identifier-indexed flight plans, from the planner.
615. The signature of the **actuator** behaviour lists the **actuator’s** unique identifier, carries the actuator’s mereology, has, perhaps ..., some static arguments, has the programmable flight directory, and further designates the **input** channel `cp_ca_ch[cp_i,cai]` and the **output** channel `ca_ed_ch[cai,*]`.
616. The actuator further behaves as follows:
- It offers to accept a current flight plan from the planner.
  - It then proceeds to offer those enterprise drones which are designated in the flight plan their flight plan.
  - Whereupon the actuator resumes being the actuator, now with its programmable flight plan directory updated with the latest such!

**type**

614 `CFP = EDI  $\mapsto$  FP`

**value**

615 `actuator: cai:CAI  $\times$  (cpi:CPI  $\times$  edis:EDI-set)  $\rightarrow$  FDDIR  $\rightarrow$`

615 `in cp_ca_ch[cp_i,cai] out {ca_ed_ch[cai,edi]|edi:EDI•edi  $\in$  edis} Unit`

616 `actuator(cai,(cpi,edis),...)(pfp,pfpl)  $\equiv$`

616a `let cfp = ca_cp_ch[cai,cp_i] ? in comment: fp:EDI  $\mapsto$  FP`

616b `{ca_ed_ch[cai,edi]|!cfp(edi)|edi:EDI•edi  $\in$  dom cfp} ;`

616c `actuator(cai,(cpi,edis),...)(cfp,(pfp)^pfpl)`

614 `end`

615 `axiom cai= $ca_i$   $\wedge$  cpi= $cp_i$`

### N.4.4.4 ‘Other’ Drone Behaviour

617. The signature of the ‘other’ drone behaviour
- lists the ‘other’ drone’s unique identifier, the ‘other’ drone’s mereology, has, perhaps ..., some static arguments; then the programmable attribute of the geography (i.e., the area, the land and the weather) it is moving over and in;
  - then, as **input** channels, the *inert*, *active*, *autonomous* and *biddable* attributes: velocity, acceleration, orientation and position, and, finally
  - further designates the array **input** channel `g_d_ch[*]` from the *geography* and the array **output** channel `d_cm_ch[*]` to the *monitor*.
618. The ‘other’ drone otherwise behaves as follows:
619. internal, non-deterministically the ‘other’ drone chooses to either ..., or “pro”viding to the monitors request for drone “dyn”amics, or ... .
620. If the choice is ... ,
621. If the choice is “provide dynamics” the behaviour `drone_monitor` is invoked, with arguments similar to that of `other_drone`, but “marked” with an additional, “frontal” argument: “other”, and with “tail”, programmable arguments `((),())`.
622. If the choice is ... .

**value**

617 `other_drone: odi:ODI  $\times$  (cmi:CMI  $\times$  gi:GI)  $\times$  ...  $\rightarrow$  (DYN  $\times$  ImG)  $\rightarrow$`

617b `in attr.DYN_ch[odi],g_d_ch[gi,odi] out d_cm_ch[odi,cmi] Unit`

618 `other_drone(odi,(cmi,gi),...)(dyn:(v,a,o,p),img)  $\equiv$`

619 `let mode = "..."  $\parallel$  "pro_dyn"  $\parallel$  "..." in`

<sup>23</sup>Flight plan *objectives* are here referred to as ‘internal’.

<sup>24</sup>Well – better check this!

<sup>25</sup>– for you ShaoFa!

```

619 case mode of
620 "" → ... ,
621 "pro_dyn" → drone_moni(odi,(cmi,gi),...)(dyn:(v,a,o,p),img)
622 "" → ...
619 end
617 end

```

623. If the choice is "provide dynamics"

- (a) then the `drone-monitor` behaviour ascertains its dynamics (velocity, acceleration, orientation and position),
- (b) informs the monitor 'thereof', and
- (c) resumes being the 'other' drone with that updated, programmable dynamics.

**value**

```

623 drone_moni: odi:ODI × (cmi:CMI × gi:GI) × ... → (DYN × ImG) →
623 in attr_DYN_ch[odi],g_d_ch[gi,odi] out d_cm_ch[odi,cmi] Unit
622 drone_moni(odi,(cmi,gi),...)(dyn:(v,a,o,p),img) ≡
623a let (ti,dyn',img') =
623a (time(),
623a (let (v',a',o',p') = attr_DYN[odi]? in
623a (v',a',o',p'),
623a d_g_ch[odi,gi]!p' ; g_d_ch[gi,odi]? end)) in
623b d_cm_ch[odi,cmi] ! (ti,dyn') ;
623c other_drone(cai,(cpi,edis),...)(dyn',img')
623a end

```

#### N.4.4.5 Enterprise Drone Behaviour

624. The enterprise donor lists its enterprise drone's unique identifier, carries it's mereology, has, perhaps ..., some static arguments, the programmable enterprise drone attributes: a pair of the present flight plan, and the past flight plans, and a pair of the most recently observed dynamics and immediate geography, and further designates the single **input** channel and the **output** channel array .

Enterprise drones otherwise behave as follows:

625. internal, non-deterministically an enterprise drone chooses to either "rec"ording the "geo"graphy, i.e., the area, land and weather it is situated in, or "pro"viding to the monitors request for drone "dyn"amics, or "acc"epting the actuators offer of a new "f"light "p"lan, or "move" "on" (i.e., continue to fly), either "follow"ing the "flight plan" most recently received from the actuator, or, "ignor"ing this directive, "just plondering on" !

626. If the choice is "rec\_geo" then the `enterprise_geo` behaviour is invoked,

627. If the choice is "pro\_dyn" (provide dynamics to the *monitor*) then the `enterprise_moni` behaviour is invoked,

628. If the choice is "acc\_fp" then the `enterprise_accept_flight_plan` behaviour is invoked,

629. If the choice is "move\_on" then the enterprise drone decides either to "ignore" the flight plan, or to "follow" it.

(a) If it "ignore"s the flight plan then the `enterprise_ignore` behaviour is invoked,

(b) If the choice is "follow" then the `enterprise_follow` behaviour is invoked.

```

624 enterprise_drone: edi:EDI × (cmi:CMI × cai:CAI × gi:GI) →
624 ((FPL × PFPL) × (DDYN × ImG)) →
624 in attr_DYN_ch[edi],g_d_ch[gi,edi],ca_ed_ch[cai,edi]
624 out d_cm_ch[edi,cmi],d_g_ch[edi,gi] Unit
624 enterprise_drone(edi,(cmi,cai,gi),...)(fpl,pfpl,(ddyn,img)) ≡
625 let mode = "rec_geo" [] "pro_dyn" [] "acc_fp" [] "move_on" in
625 case mode of
626 "rec_geo" → enterprise_geo(edi,(cmi,cai,gi),...)(fpl,pfpl,(ddyn,img))
627 "pro_dyn" → enterprise_moni(edi,(cmi,cai,gi),...)(fpl,pfpl,(ddyn,img))
628 "acc_fp" → enterprise_acc_fl_pl(edi,(cmi,cai,gi),...)(fpl,pfpl,(ddyn,img))
629 "move_on" →
629 let m_o_mode = "ignore" [] "follow" in
629 case m_o_mode of
629a "ignore" → enterprise_ignore(edi,(cmi,cai,gi),...)(fpl,pfpl,(ddyn,img))
629b "follow" → enterprise_follow(edi,(cmi,cai,gi),...)(fpl,pfpl,(ddyn,img))

```

```

635 end
635 end
625 end
625 end
624 axiom cmi=cmi∧cai=cai∧gi=gi

```

630. If the choice is "rec\_geo"

- (a) then dynamics is ascertained so as to obtain a positions;
- (b) that position is used in order to obtain a "fresh" immediate geography;
- (c) with which to resume the enterprise drone behaviour.

```

624 enterprise_geography: edi:EDI×(cmi:CMI×cai:CAI×gi:GI) →
624 ((FPL×PFPL)×(DDYN×ImG)) →
624 in attr_DYN_ch[edi],g_d_ch[gi,edi],ca_ed_ch[cai,edi]
624 out d_cm_ch[edi,cmi],d_g_ch[edi,gi] Unit
624 enterprise_geography(edi,(cmi,cai,gi),...)((fpl,pfpl),(ddyn,img)) ≡
630a let (v,a,o,p) = attr_DYN_ch[edi]? in
630b let img' = d_g_ch[edi,gi]!p;g_d_ch[gi,edi]? in
630c enterprise_drone(edi,(cmi,cai,gi),...)((fpl,pfpl),((v,a,o,p),img'))
630a end end

```

631. If the choice is "pro\_dyn" (provide dynamics to the *monitor*)

- (a) then a triplet is obtained as follows:
- (b) the current time,
- (c) the dynamics (v,a,o,p), and
- (d) the immediate geography of position p,
- (e) such that the *monitor* can be given the current dynamics,
- (f) and the enterprise drone behaviour is resumed with updated dynamics and immediate geography.

```

624 enterprise_monitor: edi:EDI×(cmi:CMI×cai:CAI×gi:GI) →
624 ((FPL×PFPL)×(DDYN×ImG)) →
624 in attr_DYN_ch[edi],g_d_ch[gi,edi],
624 out d_cm_ch[edi,cmi],d_g_ch[edi,gi] Unit
624 enterprise_monitor(edi,(cmi,cai,gi),...)((fpl,pfpl),(ddyn,img)) ≡
631a let (ti,ddyn',img') =
631b (time(),
631c (let (v,a,o,p) = attr_DYN[edi]? in
631c (v,a,o,p),
631d d_g_ch[edi,gi]!p;g_d_ch[gi,edi]? end)) in
631e d_cm_ch[edi,cmi] ! (ti,ddyn') ;
631f enterprise_drone(edi,(cmi,cai,gi),...)((fpl,pfpl),(ddyn',img'))
631a end

```

632. If the choice is "acc\_fp"

- (a) the enterprise drone offers to accept a new flight plan from the *actuator*
- (b) and the enterprise drone behaviour is resumed with that flight plan now becoming the next current flight plan and whatever is left of the hitherto current flight plan appended to the past flight plan list.

```

624 enterprise_acc_fl_pl: edi:EDI×(cmi:CMI×cai:CAI×gi:GI) →
624 ((FPL×PFPL)×(DDYN×ImG)) → in ca_ed_ch[cai,edi] Unit
624 enterprise_axx_fl_pl(edi,(cmi,cai,gi),...)((fpl,pfpl),(ddyn,img)) ≡
632a let fpl' = ca_ed_ch[cmi,edi]? in
632b enterprise_drone(edi,(cmi,cai,gi),...)(fpl',⟨fpl⟩^pfpl,(ddyn,img))
632a end

```

633. If the choice is "move\_on" and the enterprise drone decides to "ignore" the flight plan,

- (a) then it ascertains where it might be moving with the current dynamics
- (b) and then it just keeps moving on till it reaches that dynamics

(c) from about where it resumes the enterprise drone behaviour.

```

624 enterprise_ignore: edi:EDI×(cmi:CMI×cai:CAI×gi:GI) →
624 ((FPL×PFPL)×(DDYN×ImG)) →
624 in attr_DYN_ch[edi] out d_cm_ch[edi,cmi],d_g_ch[edi,gi] Unit
624 enterprise_ignore(edi,(cmi,cai,gi),...)((fpl,pfpl),(ddyn,img)) ≡
633a let (v',a',o',p') = increment(dyn,img) in
633b while let (v'',a'',o'',p'') = attr_DYN_ch[odi]? in
633b ~close(p',p'') end do manoeuvre(dyn,img) ; wait δ t end ;
633c enterprise_drone(cai,(cpi,edis),...)((fpl,pfpl),(attr_DYN_ch[odi]?,img))
633a end

```

634. The manoeuvre behaviour is further unspecified. For a fixed wing aircraft it controls the *yaw*, the *roll* and the *pitch* of the aircraft, hence its flight path, by operating the *elevator*, *aileron*, *ruddr* and the *thrust* of the aircraft based on its current dynamics, weight (including aircraft fuel), meteorological conditions (winds etc.).

**value**

```

634 manoeuvre: DYN × ImG → Unit
634 manoeuvre(dyn,img) ≡ ...

```

The **wait**  $\delta t$  is some drone constant.

635. If the choice is "move\_on" and the enterprise drone decides to "follow" the flight plan,

- (a) then, if the current flight plan has been exhausted, i.e., "used-up" it aborts (**chaos**<sup>26</sup>)
- (b) otherwise it ascertains where it might be moving, i.e., a next dynamics from with the current dynamics.
- (c) So it then "moves along" until it has reached that dynamics –
- (d) from about where it resumes the enterprise drone behaviour.

**value**

```

624 enterprise_follow: edi:EDI×(cmi:CMI×cai:CAI×gi:GI) →
624 ((FPL×PFPL)×(DDYN×ImG)) →
624 in attr_DYN_ch[edi] out d_cm_ch[edi,cmi],d_g_ch[edi,gi] Unit
624 enterprise_follow(edi,(cmi,cai,gi),...)((fpl,pfpl),(ddyn,img)) ≡
635a if fpl = ⟨ ⟩ then chaos else
635b let (v',a',o',p') = increment(dyn,img,hd fpl) in
635c while let (v'',a'',o'',p'') = attr_DYN_ch[odi]? in
635c ~close(p',p'') end do manoeuvre(hd fpl,dyn,img) ; wait δ t end ;
635d enterprise_drone(edi,(cmi,cai,gi),...)((tlfpl,pfpl),(attr_DYN_ch[odi]?,img))
635a end end

```

636. The (overloaded) manoeuvre behaviour is further unspecified. For a fixed wing aircraft it controls the *yaw*, the *roll* and the *pitch* of the aircraft, hence its flight path, by operating the *elevator*, *aileron*, *ruddr* and the *thrust* of the aircraft based on its current dynamics, weight (including aircraft fuel), meteorological conditions (winds etc.).

**value**

```

636 manoeuvre: FPE × DYN × ImG → Unit
636 manoeuvre(fpe,dyn,img) ≡ ...

```

The **wait**  $\delta t$  is some drone constant.

#### N.4.4.6 Geography Behaviour

637. The *geography* behaviour definition

- (a) lists the *geography* behaviour's unique identifier, carries the its mereology, has the static argument of its Euclidean point space, and
- (b) further designates the single input channels *cp\_g\_ch*[*cpi,gi*] from the *planner* and *d\_g\_ch*[\*,*gi*] from the *drones* and the output channels *g\_cp\_ch*[*gi,cpi*] to the *planner* and *g\_d\_ch*[*gi,\**] to the *drones*.

638. The *geography* otherwise behaves as follows:

<sup>26</sup>**chaos** means that we simply decide not to describe what then happens!

- (a) Internal, non-deterministically the geography chooses to either "resp"ond to a request from the "plan"ner.
- (b) If the choice is
- (c) "resp\_plan"
  - i. then the *geography* offers to accept a request from the *planner* for the *immediate geography* of an *area* "around" a *point* and
  - ii. then the *geography* offers that information to the *planner*,
  - iii. whereupon the geography resumes being that;
 else if the choice is
- (d) "resp\_dron"
  - i. then then the *geography* offers to accept a request from the *planner* for the *immediate geography* of an *area* "around" a *point* and
  - ii. then the *geography* offers that information to the *planner*,
  - iii. whereupon the geography resumes being that.

639. The area function takes a pair of a point and a pair of *land* and *weather* and yields an *immediate geography*.

```

value
637 geography: gi:GI × gm:(cpi:CPI×cmi:CMI×dis:DI-set) × EPS →
637a in cp_g_ch[cpi,gi], d_g_ch[*,gi]
637b out g_cp_ch[gi,cpi], g_d_ch[gi,*] Unit
637 geography(gi,(cpi,cmi,dis),eps) ≡
638a let mode = "resp_plan" ∥ "resp_dron" ∥ ... in
638b case mode of
638c "resp_plan" →
638(c)i let p = cp_g_ch[cpi,gi] ? in
638(c)ii g_cp_ch[gi,cpi] ! area(p,(attr_L_ch[gi]?,attr_W_ch[gi]?)) end
638(c)iii geography(gi,(cpi,cmi,dis),eps)
638d "resp_dron" →
638(d)i let (p,di) = ∏{(d_g_ch[di,gi]?,di)|di:DI•di ∈ dis} in
638(d)ii g_cp_ch[di,cpi] ! area(p,(attr_L_ch[gi]?,attr_W_ch[gi]?)) end
638(d)iii geography(gi,(cpi,cmi,dis),eps)
637 end end
axiom
637 gi=gi∧cpi=cpi∧smi=cmi∧dis=dis
value
639 area: P × (L × W) → ImG
639 area(p,(l,w)) ≡ ...

```

## N.5 Conclusion

TO BE WRITTEN



# Appendix O

## Automobile Assembly Lines

### Contents

---

|           |                                                    |     |
|-----------|----------------------------------------------------|-----|
| O.1       | <b>Introduction</b>                                | 419 |
| O.2       | <b>A Domain Analysis &amp; Description</b>         | 419 |
| O.2.1     | <b>An Initial Domain Sketch</b>                    | 419 |
| O.2.2     | <b>Endurants</b>                                   | 420 |
| O.2.2.1   | <b>External Qualities</b>                          | 421 |
| O.2.2.1.1 | <b>Parts</b>                                       | 421 |
| O.2.2.1.2 | <b>On Main Elements</b>                            | 423 |
| O.2.2.1.3 | <b>Automobile Manufacturing: A Wider Context</b>   | 424 |
| O.2.2.1.4 | <b>An Assembly Plant Taxonomy</b>                  | 424 |
| O.2.2.1.5 | <b>Aggregate, Set, Core and Sibling Parts</b>      | 424 |
| O.2.2.1.6 | <b>The Core State</b>                              | 426 |
| O.2.2.1.7 | <b>Invariant: External Qualities</b>               | 428 |
| O.2.2.2   | <b>Internal Qualities</b>                          | 429 |
| O.2.2.2.1 | <b>Unique Identifiers</b>                          | 429 |
| ι 640.    | <b>AP</b> , Assembly Plants                        | 431 |
| ι 641.    | <b>ALA</b> , Assembly Line Aggregates              | 432 |
| ι 642.    | <b>MAL</b> , Main Assembly Lines                   | 432 |
| ι 643.    | <b>SALA</b> , Supply Assembly Line Aggregates      | 433 |
| ι 644.    | <b>SALs=SAL-set</b> , Supply Assembly Line<br>Sets | 433 |
| ι 645.    | <b>SAL</b> , Supply Assembly Lines                 | 433 |
| ι 646.    | <b>SA</b> , Station Aggregates                     | 433 |
| ι 647.    | <b>Ss=S-set</b> , Station Set                      | 434 |
| ι 648.    | <b>S</b> , Stations                                | 434 |
| ι 649.    | <b>ME</b> , Main Elements                          | 434 |
| ι 650.    | <b>RA</b> , Robot Aggregates                       | 434 |
| ι 651.    | <b>Rs=R-set</b> , Robot Sets                       | 435 |
| ι 652.    | <b>R</b> , Robots                                  | 435 |
| ι 653.    | <b>ES</b> , Element Supplies                       | 435 |
| ι 654.    | <b>Es=E-set</b> , Element Supply Sets              | 435 |
| ι 655.    | <b>E</b> , Elements                                | 435 |
| O.2.2.2.2 | <b>Mereology</b>                                   | 435 |
|           | <b>Comments on the Mereology Presentation</b>      | 443 |
|           | <b>Distances of Stations from Outlet</b>           | 444 |

|              |                                                        |            |
|--------------|--------------------------------------------------------|------------|
| O.2.2.2.3    | <b>Attributes</b>                                      | 445        |
| O.2.2.2.3.1  | ↳ <b>640. AP: Assembly Plant:</b>                      | 446        |
| O.2.2.2.3.2  | ↳ <b>641. ALA: Assembly Line Aggregate:</b>            | 446        |
| O.2.2.2.3.3  | ↳ <b>642. MAL: Main Assembly Line:</b>                 | 446        |
| O.2.2.2.3.4  | ↳ <b>643. SALA: Supply Assembly Line Aggregate:</b>    | 446        |
| O.2.2.2.3.5  | ↳ <b>644. SALs: Supply Assembly Line Set:</b>          | 446        |
| O.2.2.2.3.6  | ↳ <b>645. SAL: Supply Assembly Lines:</b>              | 446        |
| O.2.2.2.3.7  | ↳ <b>646. SA: Station Aggregate:</b>                   | 447        |
| O.2.2.2.3.8  | ↳ <b>647. S<sub>s</sub>=S-set: Station Set:</b>        | 447        |
| O.2.2.2.3.9  | ↳ <b>648. S: Station:</b>                              | 447        |
| O.2.2.2.3.10 | ↳ <b>649. ME: Main Element:</b>                        | 448        |
| O.2.2.2.3.11 | ↳ <b>650. RA: Robot Aggregate:</b>                     | 448        |
| O.2.2.2.3.12 | ↳ <b>651. R<sub>s</sub>=R-set: Robot Set:</b>          | 449        |
| O.2.2.2.3.13 | ↳ <b>652. R: Robot:</b>                                | 449        |
| O.2.2.2.3.14 | ↳ <b>653. ES: Element Supply:</b>                      | 449        |
| O.2.2.2.3.15 | ↳ <b>654. E<sub>s</sub>=E-set: Element Supply Set:</b> | 449        |
| O.2.2.2.3.16 | ↳ <b>655. E: Elements:</b>                             | 449        |
| O.2.2.3      | <b>Comments wrt. [77]</b>                              | 449        |
| O.2.3        | <b>Perdurants</b>                                      | 450        |
| O.2.3.1      | <b>From Parts to Behaviours</b>                        | 450        |
| O.2.3.2      | <b>Channels</b>                                        | 451        |
| O.2.3.3      | <b>Actors</b>                                          | 451        |
| O.2.3.3.1    | <b>Actions and Events</b>                              | 451        |
| O.2.3.3.2    | <b>Behaviours</b>                                      | 451        |
| O.2.3.4      | <b>System Initialisation</b>                           | 451        |
| <b>O.3</b>   | <b>Discussion</b>                                      | <b>451</b> |
| <b>O.4</b>   | <b>Conclusion</b>                                      | <b>451</b> |
| O.4.1        | <b>Models and Axioms</b>                               | 451        |
| O.4.2        | <b>Learning Forwards, Practicing In Reverse</b>        | 451        |
| O.4.3        | <b>Diagrammatic Reasoning</b>                          | 452        |
| O.4.4        | <b>The Management of Domain Modeling</b>               | 452        |
| O.4.5        | <b>... one more section ...</b>                        | 453        |
| O.4.6        | <b>... a last section (?) ...</b>                      | 453        |
| O.4.7        | <b>Acknowledgments</b>                                 | 453        |

---

We interpret Sect. 2 of [77]. That is, we present the domain description of a generic, assembly line manufacturing plant, like, for example, an automobile plant. The description is in the style of, i.e., according to the dogma of [58]. It is an aim of this report to (i) classify the various notions of [77] in their relationship to domain analysis & description notions of [58]: endurants and perdurants, external and internal endurant qualities: unique identifiers, mereologies and attributes, as well as domain versus requirements specifications, i.e., descriptions vs. prescriptions.

**Caveat**

The topic of this report is currently being studied and writing progresses accordingly. I have not checked all item (etc.) references, but will, one day I have a printed copy to work from! I have also left many stubs to be resolved. Various sections represent “diverse” modeling attempts. It will be interesting to see which will “survive”! Since this report will be updated on the net daily You may wish to not download-copy it, but to reload it, from day-to-day, if need be.

March 12, 2024: 10:48 am: “Progress”

- Mereology “finished”.
- “Finished” first round of Attributes.



- Speculating on robot tasks.
- Unfinished “business” wrt. parts and robot operations.

**A Development Document**

This report cum paper, may look like a paper. But it is not. It is a report on “*work in progress*”. It expresses, in its current form, the way we would, sequentially, develop an experimental domain model, such as mentioned in Sect. O.4.4 on page 452, in the item labeled **Experiment** on Page 452.

## O.1 Introduction

The current author has put forward a theory and a methodology of domain engineering [51,55,58]. That methodology is the result of 30 years of experimental development of analyses & descriptions of numerous domains. Isolated aspects of the domain of assembly line manufacturing has been a topic of study, also in computing science, for some years. See, for example, [https://en.wikipedia.org/wiki/Cellular\\_manufacturing](https://en.wikipedia.org/wiki/Cellular_manufacturing). These computing science studies have, however, focused, less on overall assembly lines, and more on their individual manufacturing cells – in this report referred to as operators (or stations (?)). So when I heard of and read [77] I was ready to myself tackle the domain analysis & description of an “entire” production line, i.e., a single assembly line complex of a main and possibly several supply assembly lines.

MORE TO COME

## O.2 A Domain Analysis & Description

### O.2.1 An Initial Domain Sketch

We refer to Fig. O.5 on page 421. In this section we shall give an informal sketch of the domain. The domain is that of the generic *assembly line* “core” of a manufacturing plant – think of an automobile factory!<sup>1</sup>

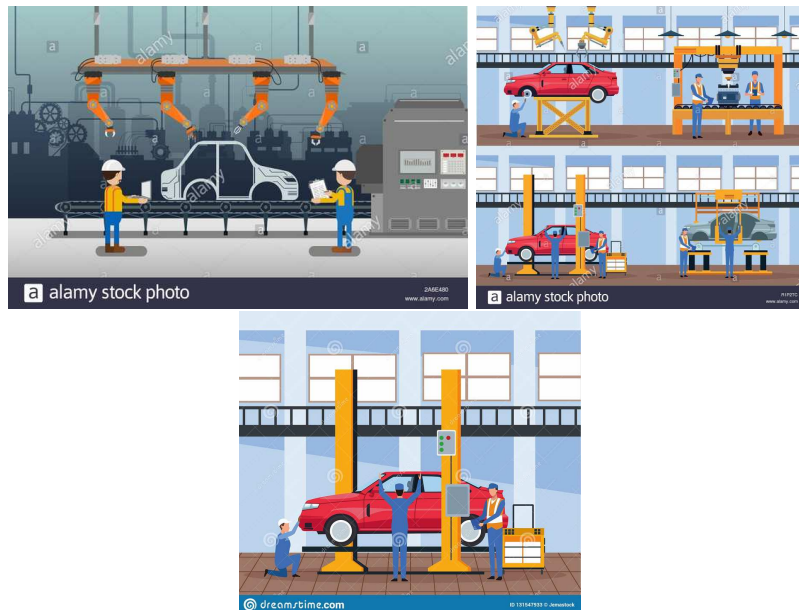


Figure O.1: Aspects of an Automobile Assembly Line, I

<sup>1</sup>For the specific case of automobile factories the assembly line focus thus omits consideration of number of major components: the motor foundry etc., the paint shop, etc.

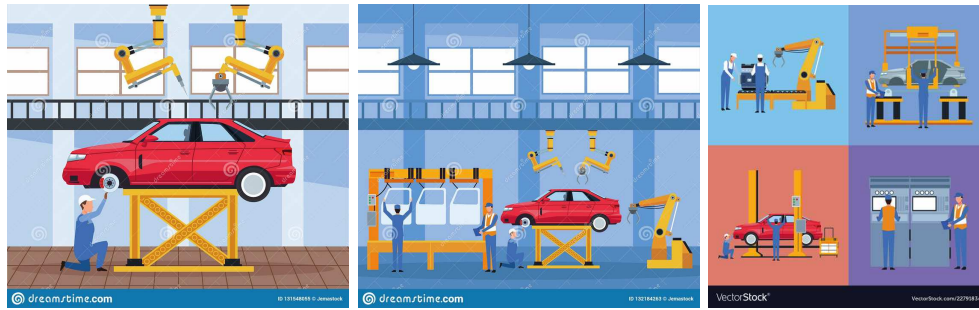


Figure O.2: Aspects of an Automobile Assembly Line, II



Figure O.3: Aspects of an Automobile Assembly Line, III

We thus focus solely on *assembly lines*<sup>2,3</sup>. Figure O.5 shows an idealised layout of an assembly line. It shows one *main assembly line* and three *supply assembly lines*. Assembly lines assemble, as we shall call them, *elements*.<sup>4</sup> Assembly of elements, from other, the constituent, elements are performed by *robots*<sup>5</sup> at stations. *Stations* are linearly ordered within an assembly line. Assembly lines has a *flow* direction, i.e., the direction in which increasingly “bigger” elements “flow”. Each station consists of one or more *robots*. Robots direct their work at a *main element*, and apply their grips to elements supplied from an *element supply*,<sup>6</sup> or to a “larger” assembly “fetched” from a *supply assembly line* incident at that *station*!

## O.2.2 Endurants

The endurant analysis & description is according to the ontology graph of Fig. O.6 on page 422. The analysis & description is otherwise according to either of [51, 55, 58]. It suffices to have studied [55].

<sup>2</sup>[https://en.wikipedia.org/wiki/Assembly\\_line](https://en.wikipedia.org/wiki/Assembly_line): An **assembly line** is a manufacturing process (often called a progressive assembly) in which parts (usually interchangeable parts) are added as the semi-finished assembly moves from workstation to workstation where the parts are added in sequence until the final assembly is produced. By mechanically moving the parts to the assembly work and moving the semi-finished assembly from work station to work station, a finished product can be assembled faster and with less labor than by having workers carry parts to a stationary piece for assembly.

Assembly lines are common methods of assembling complex items such as automobiles and other transportation equipment, household appliances and electronic goods.

<sup>3</sup>Example supply assembly lines are: (i) engine assembly (where the start of such lines are supplied with already prepared engine blocks (from a non-assembly line engine foundry and machining shop), (ii-v) four left and right front and rear door assemblies, (vi-ix) body interior left and right front and rear sofa, and panel assemblies.

<sup>4</sup>Other, perhaps more common terms are: products or parts. The term ‘part’ is used in our domain analysis & description method, [51, 55, 58], for quite other purposes –so that is “out!”

<sup>5</sup>Robots are either humans assisted by various machine tools, as in Charlie Chaplin’s movie: ‘*Modern Times*’ (1936), or are, indeed, robots.

<sup>6</sup>That is, a station local storage of elements that are to be joined, at a station, by the help of robots, to the main element. How the supply elements are introduced to the supply is currently left unspecified.



Figure O.4: Aspects of an Automobile Assembly Line,IV

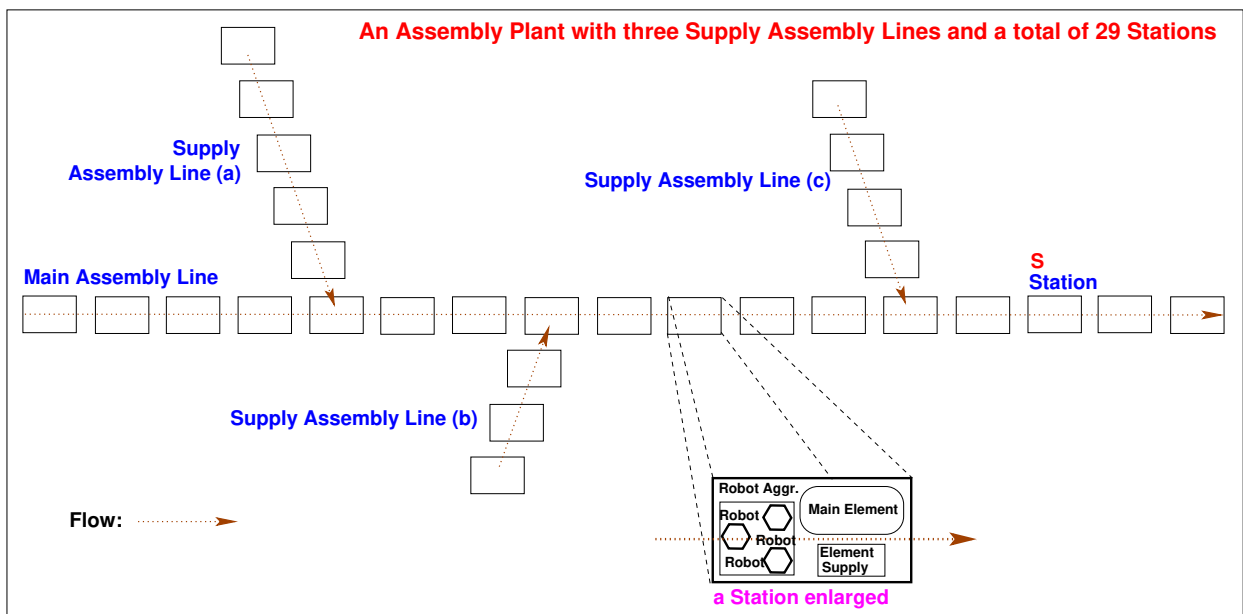


Figure O.5: A simplified *Assembly Plant* diagram

**O.2.2.1 External Qualities**

The *domain analyser cum describer*<sup>7</sup>, who is assumed fully familiar with the domain analysis & description method, [58], starts with analysing and describing external qualities of the domain. In the case of an assembly plant these are the solid durants, or, as they are called in [58], the parts. The domain analyser cum describer, from being familiar with the method, therefore is, all the time, aware that these (described) parts will, in the transition to the analysis & description of perdurants, be transcendently deduced, i.e., “morphed” into behaviours. It is this a-priory knowledge that guides the analyser cum describer’s determination as to whether parts are to be modeled as atomic or as compounds, and the decomposition of compound parts into atomic, aggregate and set parts.

**O.2.2.1.1 Parts** The domain, that is, the universe of discourse, is that of an **assembly plant** – say for automobiles, for machinery or for electronic gadgets.

- 640. In an **assembly plant**, AP, we can observe
- 641. an assembly line aggregate, ALA<sup>8</sup>.
- 642. From an assembly line aggregate one can observe a composite of a main assembly line, MAL,
- 643. and aggregate of supply assembly lines, SALA,

<sup>7</sup>The notion of ‘*domain analyser cum describer*’ covers one, individually (as the author of this paper) working person, or a well-managed group of two or more persons, all “equally” familiar with the method of [58].

<sup>8</sup>We omit observations of *motor works (foundry, machining, etc.), body shop (pressing, etc.), paint shop, etc.*

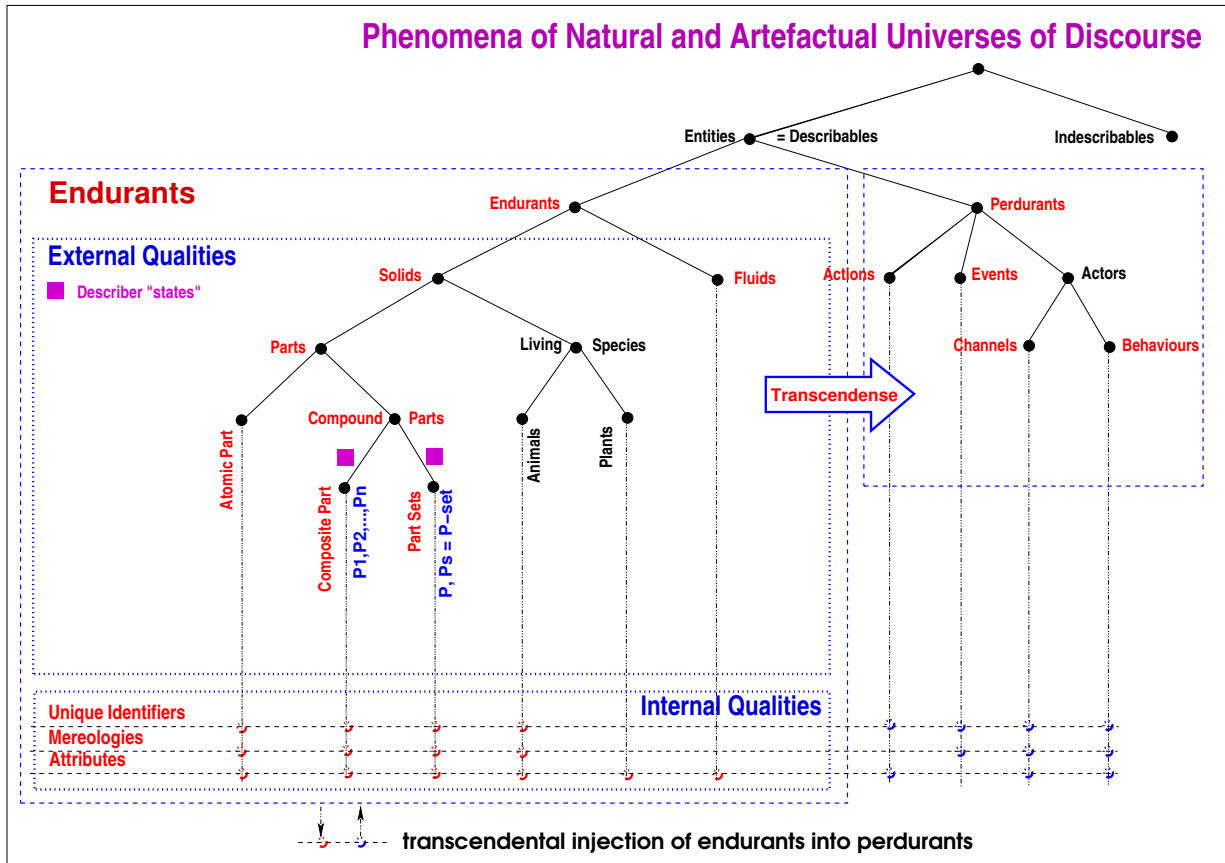


Figure O.6: The simple *Ontology Graph* underlying our *Analysis & Description*

644. with the latter being sets  $SALs = SAL\text{-set}$ .
645. of supply assembly lines,
646. From main and supply assembly lines one can observe aggregates of stations,  $SA$ ,
647. which are sets  $Ss = S\text{-set}$ <sup>9</sup>
648. of two or more stations  $S$ . From a station one can observe
649. a main element,  $ME$ , (an assembly<sup>10</sup>),
650. an aggregate robot,  $RA$ , which is
651. a set,  $Rs = R\text{-set}$ ,<sup>11</sup> of
652. one or more robots,  $R$ , and
653. an aggregate,  $ES$ , of ["supply"] elements<sup>12</sup>,
654. which are sets,  $ESs = E\text{-set}$ ,
655. of manufacturing elements  $E$ .

<sup>9</sup>**Linear Lines:** The mereology of sect. O.2.2.2.2 will order these in a linear sequence

<sup>10</sup><https://www.merriam-webster.com/dictionary/assembly>: **Assembly:** *the fitting together of manufactured elements into a complete machine, structure, or unit of a machine*

<sup>11</sup> [77]: **Our interpretation of 'operator':** robot perform processes which consists of tasks. These are perdurants, that is, an operator will, subsequently, in this report be transcendently "morphed" into a set of one or more concurrent processes. These processes are then subject, in the domain model, to invariants, and in a subsequent requirements "model" into constraints.

<sup>12</sup>These are local storage, usually simple, mostly atomic solid or fluid elements such as bolts & nuts, glue, etc.

| type                | value                                            |
|---------------------|--------------------------------------------------|
| 640. AP             | 655. E                                           |
| 641. ALA            | 641. obs_ALA: AP → ALA                           |
| 642. MAL            | 642. obs_MAL: ALA → MAL                          |
| 643. SALA           | 643. obs_SALA: ALA → SALA                        |
| 644. SALs = SAL-set | 644. obs_SALs: SALA → SALs                       |
| 645. SAL            | 646. obs_SA: (MAL SAL) → SA                      |
| 646. SA             | 647. obs_Ss: SA → Ss                             |
| 647. Ss = S-set     | 649. obs_ME: S → ME                              |
| 648. S              | 650. obs_RA: S → RA                              |
| 649. ME             | 651. obs_Rs: RA → Rs                             |
| 650. RA             | 653. obs_ES: S → ES                              |
| 651. Rs = R-set     | 654. obs_Es: ES → Es                             |
| 652. R              | <b>axiom</b>                                     |
| 653. ES             | 647. $\forall ss:Ss \bullet \text{card } ss > 1$ |
| 654. Es = E-set     | 651. $\forall rs:Rs \bullet \text{card } rs > 0$ |

Figure O.7 repeats Fig. O.5 on page 421, but now marked with the names of composite sorts introduced in Items 640–654.

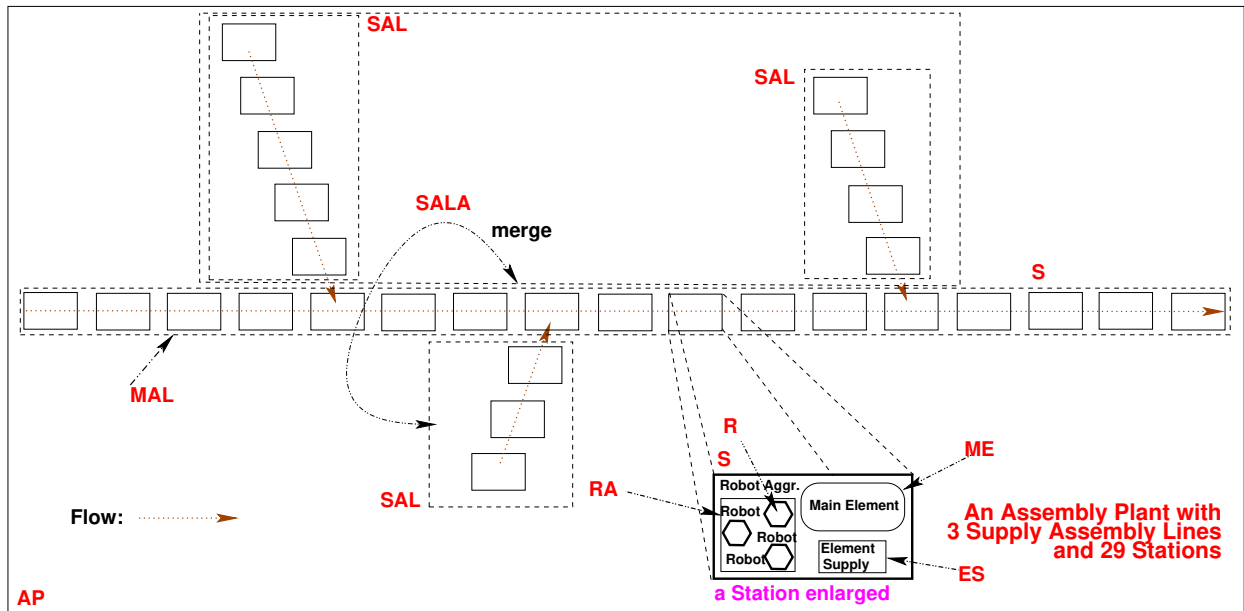


Figure O.7: A simplified *Assembly Plant* diagram marked with composite enduring sort names

**O.2.2.1.2 On Main Elements** This section is an aside.

In this section we shall discuss what is meant here by a main element, that is, “what is in store” – what will/might come up later on.

**General**

The *main element* is here modeled as a solid enduring. It is a “*place-holder*” for “the thing for which the manufacturing plant” is intended. The plan is to endow main elements with an attribute [Sect. O.2.2.2.3.10 on page 448]: That attribute may itself be thought of as being a solid enduring. We shall then use the term *part*<sup>13</sup> Robots, then, perform operations on the main element. These operations are functions, which are attributes of robots. As functions they take the main element [main] part attribute and a set of element supply elements and yield an updated main element part. So You may think of the main element as a “container” for that main part. There may be no contents of the container, in which case the main element’s part attribute is “nil”. Its content is “received” from the main element of the previous station, if there is one, else from an element supply. A content

<sup>13</sup>Not to be confused with the Design Analysis & Description concept of parts, i.e., solid enduring.

that a station can no longer contribute to is “passed” on to the next station, or “fused” in, if from a supply assembly line, as a supply element, to a main assembly line elements, or, to an outside, the “ready product!”

#### Assembly Line Element Types

The type of main elements is a pair: the type that is stroven for, that is, the assembly line type, and the type of the element “currently residing in” the main elements. So each station is particularly typed by its “current” main element type.

**O.2.2.1.3 Automobile Manufacturing: A Wider Context** These are, roughly the principal components of automobile manufacturing<sup>14</sup>:

- **Chassis:** The chassis of the car is the baseline component. All other parts are integrated on, or within the chassis. This is typically a welded frame that’s initially attached to a conveyor that moves along a production line. As the frame progresses, the car is literally “built from the frame up” to create a final product. Parts that are sequentially applied to the chassis include the engine, front and rear suspension, gas tank, rear-end and half-shafts, transmission, drive shaft, gear box, steering box, wheel drums and the brake system.
- **Body:** Once the “running gear” is integrated within the frame, the body is constructed as a secondary process. First, the floor pan is positioned properly, then the left and right quarter panels are positioned and welded to the floor structure. This step is followed by adding the front/rear door pillars, the body side panels, rear deck, hood and roof. The entire process is typically executed by robotic machines.
- **Paint:** Before painting the vehicle, a quality control team inspects the body as it sits. Skilled workers look for dents, abrasives or other deformations that could create a finishing problem when undergoing the painting process. Once this step is completed, the car is automatically “dipped” with primer, followed by a layer of undercoat and dried in a heated paint bay. Once the primer/undercoat process is finished, the car is again “dipped” with the base coat and again dried before moving the assembly to the next stage.
- **Interior:** After the structure is entirely painted, the body is moved to the interior department in the plant. There, all of the internal components are integrated with the body. These components include: instrumentation, wiring systems, dash panels, interior lights, seats, door/trim panels, headliner, radio, speakers, glass, steering column, all weather-striping, brake and gas pedals, carpeting and front/rear fascias.
- **Chassis/Body Mating:** The two central major assemblies are next mated for final setup and roll-out. Again, this process is executed via computer and control machines to ensure speed, and perfect the fit between the body assembly and the chassis. Once the car is rolling on its own, it’s driven to the final quality control point, inspected and placed in a waiting line for transportation to its final dealer destination.

**O.2.2.1.4 An Assembly Plant Taxonomy** Figure O.8 on the next page “graphs” the composition of solid endurants of an assembly plant according to the endurant composition of Items 640– 654 on page 422.

Some diagram explications: (i) the top left dashed triangle shall show an endurant composition as does the main, large, dashed triangle; (ii) the vertical dotted lines “hanging down” from two SALs “hint” at the “tree” emanating down from the “middle” SAL; and (iii) the horizontal dots, . . . , in SAL, S, R and E “lines” hint at any number, 0 or more, of these endurants!

**O.2.2.1.5 Aggregate, Set, Core and Sibling Parts** We review<sup>15</sup>, as an aside, the [58, Monograph] concepts of *atomic*, *compound*, *aggregates*, *sets*, *core* or *root*, and *sibling* parts.

#### Atomic and Compound Parts

Atomic parts are solid endurants whose possibly “internal” composition is ignored. Compound parts are solid endurants which we further analyse into *core* (or, equivalently, *root*) and *sibling* parts.

#### Aggregates and Sets

Compound parts are either *composites* of one or more parts of different sorts, i.e., like Cartesians, but where we avoid modeling a composite as a Cartesian of a definite number of parts – since we may, “later”, wish to “add” additional parts, or are [finite] *sets* of zero, one or more parts of the same sort.

We use the term aggregate to cover both kind of compounds. Usually, however, we use aggregates for composites, and sets for sets!

#### Cores [Roots] and Siblings

With compound parts we distinguish between the core part and the sibling parts.

The core part is understood as follows: It is to be considered a “proper” part although it may sometime be more of an abstraction than a solid! Consider the following example: a car, as seen from the point of view of an automobile plant, is a composite, with a core, the car as a whole, as “somehow” embodied in the overall software that monitors and co-controls various of the car’s siblings; these siblings are then further aggregates, each with their cores and siblings. Immediate car siblings could be the chassis, the motor, the engine train, the body. The chassis, as an aggregate, has, usually, four wheels, etc. The body, as an aggregate, has, perhaps, four doors, a trunk and a hood. And each of these, the *chassis*, *motor*, *engine train*, *body*, etc., has their cores.

We now “formalise” the notion of the core of a compound.

<sup>14</sup>This account is taken, *ad verbatim*, from: <https://itstillruns.com/car-manufacturing-process-5575669.html>.

<sup>15</sup>The treatment of part *cores* (in [58] called *roots*) is here augmented with the  $\ominus$  operation – not mentioned in [58].

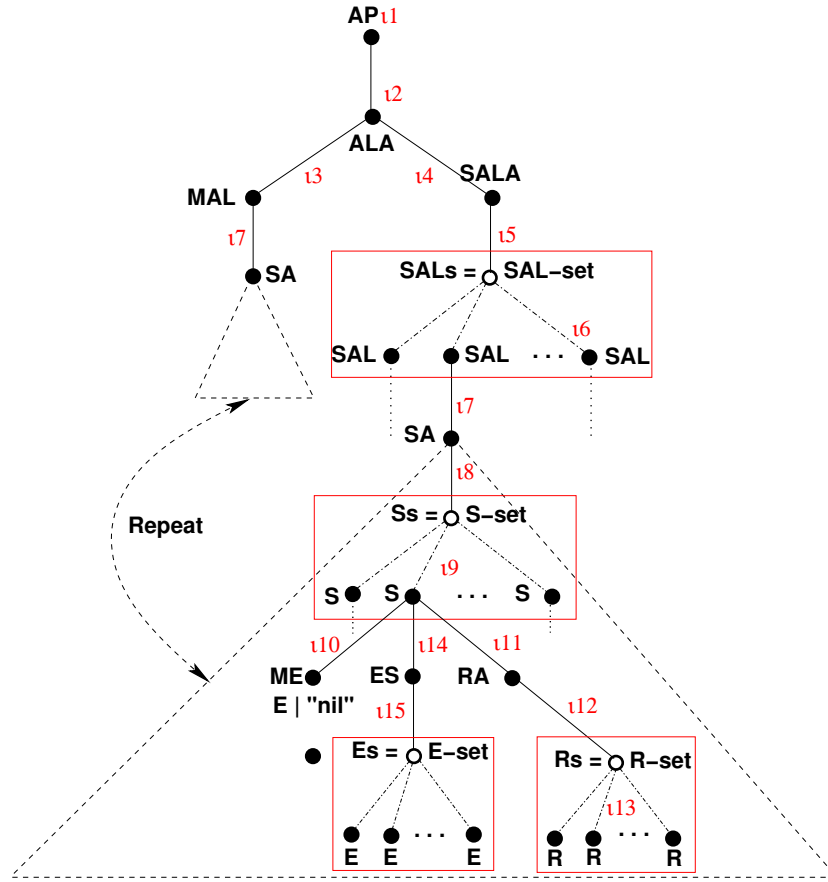


Figure O.8: The Composition of Solid Endurants of an Assembly Plant (AP)  
 Larger red framed boxes designate set parts.  
 $i, i, \pi p$  refer to item  $i$  page 422.

- We do so by introducing an otherwise not further defined [binary or distributive] operator,  $\ominus$ .
- It applies to a pair of parts:
  - one is an aggregate,  $cp$ , and
  - the other,  $sp$ , is
    - \* either a single one of its siblings,  $p_i$ ,
    - \* or a set of these,  $\{p_i, p_j, \dots, p_k\}$ ,
  - i.e.,  $cp \ominus ps$ , “somehow removes”  $ps$  from  $p$ .

We now apply the  $\ominus$  systematically to all components of our domain.

- The ‘core’ of an atomic part is that part.
- The ‘core’ of a composite part is that part “minus” ( $\ominus$ ) its “sibling” parts:
  - Let  $p$  be a composite part,
  - then  $p_1 = \text{obs}_{P_1}(p)$ ,  $p_2 = \text{obs}_{P_2}(p)$ , ...,  $p_m = \text{obs}_{P_m}(p)$  are its sibling parts
  - [where  $\text{obs}_{P_1}$ ,  $\text{obs}_{P_2}$ , ...,  $\text{obs}_{P_m}$  are the observers of parts  $p$  (of type  $P$ )].
  - The ‘core’ of  $p$ , i.e.,  $\text{core}_P(p)$ , is then  $p \ominus \{p_1, p_2, \dots, p_m\} : P_\kappa$ .
- The ‘core’ of a set of parts is that part “minus” ( $\ominus$ ) its “sibling” parts:
  - Let  $ps$  be a set part (of type  $Ps = Q\text{-set}$ ),



- ‘core’ of  $ps$ , i.e.,  $\text{core\_Ps}(ps)$ , is then  $ps \ominus \{q_1, q_2, \dots, q_n\} : Ps_\kappa$ .

Subsequently introduced *unique identifier*, *mereology* and *attribute observers* apply to core parts as they do to non-core parts.

**O.2.2.1.6 The Core State** To encircle the notion of domain core states we need characterise:

- the state narratively, Sect. O.2.2.1.6; and
- the state formally, Sect. O.2.2.1.6 on the facing page.

#### The Order of Assembly Plant Core Parts

The expression  $\mathcal{O}_e(n)$  shall express that  $e_\kappa$  is of the order  $n$ , that is, that  $e_\kappa$  contains “around”  $n$  core parts.

- $\iota$  640  $\pi$  421. [AP] Assembly plants are of  $\mathcal{O}_{ap_\kappa}(1)$ .
- $\iota$  640  $\pi$  421. [ALA] Assembly line aggregates are of  $\mathcal{O}_{ala_\kappa}(1)$  per plant.
- $\iota$  642  $\pi$  421. [MAL] Main assembly line aggregates are of  $\mathcal{O}_{mal_\kappa}(1)$  per plant.
- $\iota$  642  $\pi$  421. [SALA] Supply assembly line aggregates are of  $\mathcal{O}_{sala_\kappa}(10 - 20)$  per plant.
- $\iota$  644  $\pi$  422. [SALS] Supply assembly line sets are of  $\mathcal{O}_{sals_\kappa}(10 - 20)$  per plant.
  - Let the number of an assembly plant’s supply assembly lines be  $n_l$ ,
  - i.e., supply lines  $l_1, l_2, \dots, l_{n_l}$ .
- $\iota$  644  $\pi$  422. [SAL] Supply assembly lines are of  $\mathcal{O}_{sal_\kappa}(1)$ .
- $\iota$  646  $\pi$  422. [SA] Aggregates of stations are of  $\mathcal{O}_{sa_\kappa}(1)$  per line.
- $\iota$  647  $\pi$  422. [Ss] Station sets, for one aggregate, are of  $\mathcal{O}_{ss_\kappa}(50 - 100)$  per line.
  - Let the number of stations of line  $\ell_i$ , for  $0 \leq n_\ell$ , be  $n_{s\ell_i}$ ;
  - $\ell_0$  is the main assembly line and  $\ell_i$ , for  $0 < i \leq n_\ell$ , is a supply assembly line.
  - Thus the total number,  $n_{sss}$ , of stations,  $sss$ , over all lines is  $\sum_{i=0}^{i=n_l} (n_{s\ell_i})$ ;
  - that is, typically,  $n_{sss} = \mathcal{O}_{sss_\kappa}(1000)$
- $\iota$  649  $\pi$  422. [ME] Main elements are of order  $\mathcal{O}_{me_\kappa}(0 - 1)$  per station<sup>16</sup>.
- $\iota$  649  $\pi$  422. [E] Elements are of order  $\mathcal{O}_{me_\kappa}(1)$ .
- $\iota$  651  $\pi$  422. [Rs] Robot aggregates are of order  $\mathcal{O}_{rs_\kappa}(1)$  per station.
- $\iota$  651  $\pi$  422. [R] Robots are of order  $\mathcal{O}_{r_\kappa}(10)$  per station.
- $\iota$  653  $\pi$  422. [ES] Element supplies are of order  $\mathcal{O}_{es_\kappa}(1)$  per station.
- $\iota$  654  $\pi$  422. [ESs] Element supply elements are of order  $\mathcal{O}_{ess_\kappa}(100)$  per element supply.

#### State Narrative

We shall now narrate the assembly plant domain state. We start by referring to Fig. O.9 on the next page.

656. We shall model the assembly plant state,  $\sigma$ , by a set of *core parts* composed as follows:

|                       |                                                                                                     |                                            |
|-----------------------|-----------------------------------------------------------------------------------------------------|--------------------------------------------|
| $\iota$ 640 $\pi$ 421 | <sup>17</sup> (AP) the <i>assembly plant</i> <i>core</i>                                            | $ap_\kappa$ :AP $_\kappa$ ;                |
| $\iota$ 641 $\pi$ 421 | (ALA) the <i>assembly line aggregate</i> <i>core</i>                                                | $ala_\kappa$ :ALA $_\kappa$ ;              |
| $\iota$ 642 $\pi$ 421 | (MAL) the <i>main assembly line</i> <i>core</i> ,                                                   | $mal_\kappa$ :MAL $_\kappa$ ;              |
| $\iota$ 643 $\pi$ 421 | (SALA) the <i>aggregate of supply assembly lines</i> <i>core</i> ,                                  | $sala_\kappa$ :SALA $_\kappa$ ;            |
| $\iota$ 644 $\pi$ 422 | (SALS) the <i>consolidated</i> <sup>18</sup> <i>set of all sets of assembly line</i> <i>cores</i> , | $csals_\kappa$ :ALS $_\kappa$ ;            |
| $\iota$ 646 $\pi$ 422 | (SA) the <i>consolidated</i> <sup>19</sup> <i>set of all station aggregates</i>                     | $cssa_\kappa$ :SA $_\kappa$ - <b>set</b> ; |
| $\iota$ 647 $\pi$ 422 | (Ss) the <i>consolidated set of all assembly lines’ station set</i> <i>cores</i> ,                  | $css_\kappa$ :Ss $_\kappa$ ;               |
| $\iota$ 648 $\pi$ 422 | (S) the <i>consolidated set of all assembly lines’ station</i> <i>cores</i> ,                       | $cs_\kappa$ :S $_\kappa$ ;                 |
| $\iota$ 649 $\pi$ 422 | (ME) the <i>consolidated set of all main element</i> <i>cores</i> ,                                 | $csme_\kappa$ :ME $_\kappa$ ;              |
| $\iota$ 650 $\pi$ 422 | (RA) the <i>consolidated set of all robot aggregates</i> <i>cores</i> ,                             | $csra_\kappa$ :RA $_\kappa$ - <b>set</b> ; |
| $\iota$ 651 $\pi$ 422 | (Rs) the <i>consolidated set of all robot set</i> <i>cores</i> ,                                    | $csrs_\kappa$ :Rs $_\kappa$ ;              |

<sup>16</sup>In this report we do not go into any detail as to how elements are composed.

<sup>17</sup> $\iota$  *item*  $\pi$  *page* labels refers to narrative *item* on *page*; the corresponding formalisation is found on page[s] 423–423.

<sup>18</sup>– from both the main assembly line and from all the supply assembly lines

<sup>19</sup>Henceforth, by consolidated, we mean as in footnote 18.



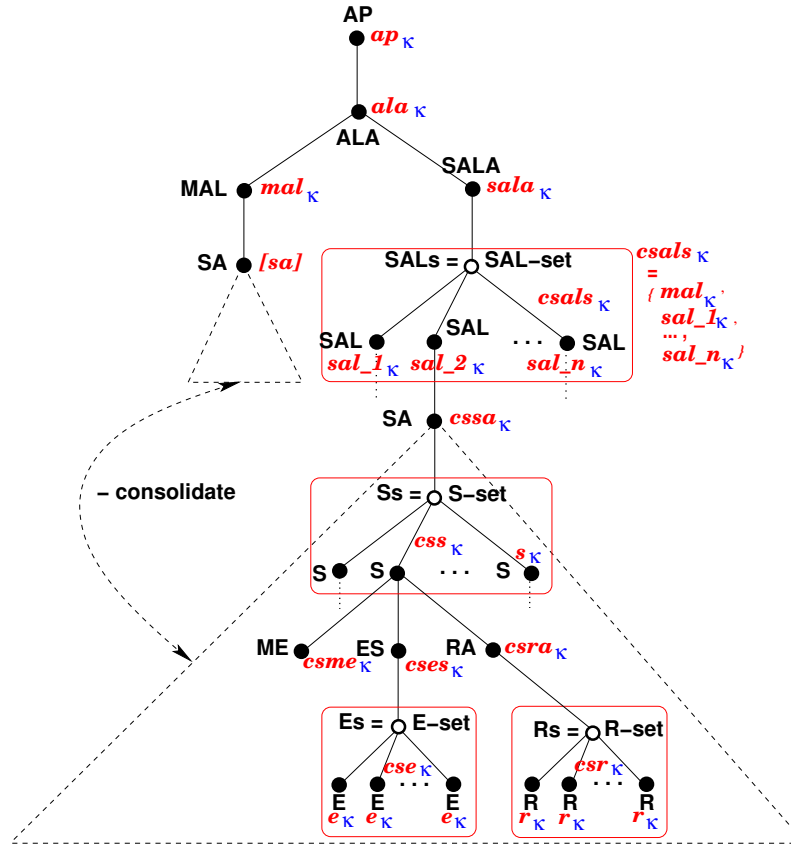


Figure O.9: Red/Blue text labels designate contributions to domain state

- $\iota$  652  $\pi$  422 (R) the consolidated set of all robot  $\kappa$ ores,  $csr_{\kappa}:\mathbf{Rs}_{\kappa};$
- $\iota$  653  $\pi$  422 (ES) the consolidated set of all element supply  $\kappa$ ores,  $cse_{\kappa}:\mathbf{ES}_{\kappa}\text{-set};$
- $\iota$  654  $\pi$  422 (Es) the consolidated set of all  $\kappa$ ore elements,  $cse_{\kappa}:\mathbf{E}_{\kappa}\text{-set};$
- 657. as a set,  $\sigma_{s_{\kappa}}$ .

**Endurant State Formalisation**

- 658. We consolidate the main and the supply assembly lines into one kind of assembly line, AL,
- 659. and their corresponding [consolidated] sets.

**type**

658. AL = MAL | SAL

659. ALs = AL-set

**vaue**

$\iota$  640  $\pi$  421.  $ap:AP$

$\iota$  640  $\pi$  421.  $ap_{\kappa}:AP_{\kappa} = core\_AP(ap)$

$\iota$  641  $\pi$  421.  $ala:ALA = obs\_ALA(ap)$

$\iota$  641  $\pi$  421.  $ala_{\kappa}:ALA_{\kappa} = core\_ALA(ala)$

$\iota$  642  $\pi$  421.  $mal:MAL = obs\_MAL(ala)$

$\iota$  642  $\pi$  421.  $mal_{\kappa}:MAL_{\kappa} = core\_MAL(ala)$

$\iota$  643  $\pi$  421.  $sala:SALA = obs\_SALA(ala)$

- $\iota$  643  $\pi$  421.  $sala_{\kappa}:SALA_{\kappa} = core\_SALA(sala)$   
 $\iota$  644  $\pi$  422.  $csals:AL\text{-set} = \{mal\} \cup obs\_SALs(sala)$   
 $\iota$  644  $\pi$  422.  $csals_{\kappa}:AL\text{-set}_{\kappa} = \cup\{core\_AL(al)|al:AL \bullet al \in csal\}$   
 $\iota$  646  $\pi$  422.  $cssa:SA\text{-set} = \{obs\_SA(al)|al:AL \bullet al \in csal\}$   
 $\iota$  646  $\pi$  422.  $cssa_{\kappa}:SA\text{-set}_{\kappa} = \cup\{core\_SA(sa)|sa:SA \bullet sa \in cssa\}$   
 $\iota$  647  $\pi$  422.  $css:S\text{-set} = \cup\{obs\_Ss(sa)|sa:SA \bullet sa \in cssa\}$   
 $\iota$  647  $\pi$  422.  $css_{\kappa}:S\text{-set}_{\kappa} = \cup\{core\_Ss(obs\_Ss(s))|s:S \bullet s \in css\}$   
 $\iota$  648  $\pi$  422.  $cs:S\text{-set} = \cup css$   
 $\iota$  648  $\pi$  422.  $cs_{\kappa}:S\text{-set}_{\kappa} = \{core\_S(s)|s:S \bullet s \in cs\}$   
 $\iota$  649  $\pi$  422.  $csme:ME\text{-set} = \cup\{obs\_ME(s)|s:S \bullet s \in css\}$   
 $\iota$  649  $\pi$  422.  $csme_{\kappa}:ME\text{-set}_{\kappa} = core\_ME(csme)$   
 $\iota$  650  $\pi$  422.  $csra:RA\text{-set} = \cup\{obs\_RA(s)|s:S \bullet s \in css\}$   
 $\iota$  650  $\pi$  422.  $csra_{\kappa}:RA\text{-set}_{\kappa} = core\_RA(csra)$   
 $\iota$  651  $\pi$  422.  $csrs:R\text{-set} = \cup\{obs\_Rs(ra)|ra:RA \bullet ra \in csra\}$   
 $\iota$  651  $\pi$  422.  $csrs_{\kappa}:R\text{-set}_{\kappa} = core\_R(csrs)$   
 $\iota$  652  $\pi$  422.  $crs:R\text{-set} = \cup\{obs\_Rs(ra)|ra:RA \bullet ra \in csra\}$   
 $\iota$  652  $\pi$  422.  $crs_{\kappa}:R\text{-set}_{\kappa} = core\_R(csrs)$   
 $\iota$  653  $\pi$  422.  $cse:ES = \cup\{obs\_ES(s)|s:S \bullet s \in csra\}$   
 $\iota$  653  $\pi$  422.  $cse_{\kappa}:ES = core\_ES(cse)$   
 $\iota$  654  $\pi$  422.  $cse:E\text{-set} = \cup\{obs\_Es(es)|es:ES \bullet s \in cses\}$   
 $\iota$  654  $\pi$  422.  $cse_{\kappa}:E\text{-set}_{\kappa} = core\_E(cse)$   
657.  $\sigma_s:(AP|ALA|MAL|SALA|SALs|Ss|S|ME|RA|Rs|R|ES|Es|E)\text{-set} =$   
657.  $\{ap_{\kappa}\} \cup \{ala_{\kappa}\} \cup \{mal_{\kappa}\} \cup \{sala_{\kappa}\} \cup csals_{\kappa} \cup cssa_{\kappa} \cup$   
657.  $css_{\kappa} \cup cs_{\kappa} \cup csme_{\kappa} \cup csra_{\kappa} \cup csrs_{\kappa} \cup crs_{\kappa} \cup cses_{\kappa} \cup cse_{\kappa}$

### O.2.2.1.7 Invariant: External Qualities

660. No two assembly lines, whether main or supply, are equal;  
661. no two stations in same or different assembly lines are equal;  
662. no two robots in different stations are equal;  
663. no two main elements are equal;  
664. no two element supplies in different stations are equal;  
665. etc.
660.  $\forall al_i, al_j:AL \bullet \{al_i, al_j\} \subseteq csal \Rightarrow$   
660. to come  
660.  
661.  $\forall s_i, s_j:S \bullet \{s_i, s_j\} \subseteq csal \Rightarrow$   
661. to come  
661.  
662.  $\forall r_i, r_j:R \bullet \{r_i, r_j\} \subseteq csr \Rightarrow$   
662. to come  
662.  
663.  $\forall me_i, me_j:ME \bullet \{me_i, me_j\} \subseteq csme \Rightarrow$   
663. to come  
663.  
664.  $\forall es_i, es_j:ES \bullet \{es_i, es_j\} \subseteq cses \Rightarrow$   
664. to come  
664.

664.  
665. ...

### O.2.2.2 Internal Qualities

External qualities can be said to represent manifestation: that an endurant can be seen and touched. Internal qualities gives “contents” to the manifests in three ways:

- by the obvious endowment of solid endurants with *unique identification* [58, Sect. 5.2],
- by stating relations between solid endurants, whether topological or conceptual, e.g., operational, in the form of *mereologies* [58, Sect. 5.3], and
- by giving “flesh & blood, body & soul” to these endurants in the form of wide ranging *attributes* [58, Sect. 5.4].

**O.2.2.2.1 Unique Identifiers** We shall show that many of the concerns of [77] have their “root” in the unique identification of solid endurants of the domain.

666. All parts, whether compound or atomic, have unique identifiers.

**type**

666. API, ALAI, MALI, SALAI, SALsl, SALI, SAI, Ss, SI, MEI, RAI, Rsl, RI, ESI, Esl, EI

**value**

|                            |                         |                      |
|----------------------------|-------------------------|----------------------|
| 666. uid_AP: AP→API,       | 666. uid_SAL: SAL→SALI, | 666. uid_Rs: Rs→Rsl, |
| 666. uid_ALA: AL→ALAI,     | 666. uid_SA: SA→SAI,    | 666. uid_R: R→RI,    |
| 666. uid_MAL: MAL→MALI,    | 666. uid_Ss: Ss→Ssl,    | 666. uid_ES: ES→EI,  |
| 666. uid_SALA: SALA→SALAI, | 666. uid_S: S→SI,       | 666. uid_E: E→EI     |
| 666. uid_SALs: SALs→SALsl, | 666. uid_ME: ME→MEI,    |                      |
|                            | 666. uid_RA: RA→RAI,    |                      |

#### Common Unique Identifier Observer

667. Given that  $is\_P(p)$  holds if  $p$  is of type  $P$ , and is false otherwise, we can define a common unique identifier observer function for all assembly plant types.

**type**

667.  $P = AP|ALA|MAL|SALA|SALs|SA|Ss|S|ME|RA|Rs|R|ES|Es|E$   
667.  $PI = API|ALAI|MALI|SALAI|SALsl|SAI|Ssl|SI|MEI|RAI|Rsl|RI|ESI|Esl|EI$

**value**

667. uid:  $P \rightarrow PI$

**value**

|                              |                                    |
|------------------------------|------------------------------------|
| 667. uid(p) $\equiv$         | 667. is_S(p)→uid_S(p),             |
| 667. is_AP(p)→uid_AP(p),     | 667. is_ME(p)→uid_ME(p),           |
| 667. is_ALA(p)→uid_ALA(p),   | 667. is_RA(p)→uid_RA(p),           |
| 667. is_MAL(p)→uid_MAL(p),   | 667. is_Rs(p)→uid_Rs(p),           |
| 667. is_SALA(p)→uid_SALA(p), | 667. is_R(p)→uid_R(p),             |
| 667. is_SALs(p)→uid_SALs(p), | 667. is_ES(p)→uid_ES(p),           |
| 667. is_SA(p)→uid_SA(p),     | 667. is_Es(p)→uid_Es(p),           |
| 667. is_Ss(p)→uid_Ss(p),     | 667. is_E(p)→uid_E(p),             |
|                              | 667. $\_ \rightarrow \text{false}$ |

#### The Unique Identifier State

As for endurant parts, cf. Sect. O.2.2.1.6 on page 427, we can define a state of all endurant parts’ unique identifiers. To do so we make use of the uid\_E observers as also being distributive, that is, if uid\_E is applied to a set of solid endurants, say  $\{e_1, e_2, \dots, e_n\}$ , then the result is  $\{\text{uid}_E(e_1), \text{uid}_E(e_2), \dots, \text{uid}_E(e_n)\}$ .

640<sub>uid.</sub>  $uid\_ap = \text{uid\_AP}(ap)$

641<sub>uid.</sub>  $uid\_ala = \text{uid\_ALA}(ala)$

642<sub>uid.</sub>  $uid\_mal = \text{uid\_MAL}(mal)$

643<sub>uid.</sub>  $uid\_sala = \text{uid\_SALA}(sala)$

644<sub>uid.</sub>  $uid\_csal = \cup\{\text{uid\_SAL}(sal)|sal:SAL \bullet sal \in csal\}$

- 645<sub>uid.</sub>  $uid\_csals = \cup\{uid\_SALs(sals)|sals:SALs\bullet sals \in csals\}$   
 646<sub>uid.</sub>  $uid\_cssa = \cup\{uid\_SA(sa)|sa:SA\bullet sa \in cssa\}$   
 647<sub>uid.</sub>  $uid\_css = \cup\{uid\_Ss(ss)|ss:Ss\bullet ss \in css\}$   
 648<sub>uid.</sub>  $uid\_cs = \cup\{uid\_S(s)|s:S\bullet s \in cs\}$   
 649<sub>uid.</sub>  $uid\_csme = \cup\{uid\_ME(me)|me:ME\bullet me \in csme\}$   
 650<sub>uid.</sub>  $uid\_csra = \cup\{uid\_RA(ra)|ra:RA\bullet ra \in csra\}$   
 651<sub>uid.</sub>  $uid\_csrs = \cup\{uid\_Rs(rs)|rs:Rs\bullet rs \in csrs\}$   
 652<sub>uid.</sub>  $uid\_csr = \cup\{uid\_R(R)|r:R\bullet r \in csr\}$   
 653<sub>uid.</sub>  $uid\_cses = \cup\{uid\_ES(es)|es:ES\bullet es \in cses\}$   
 654<sub>uid.</sub>  $uid\_cse = \cup\{uid\_Es(es)|es:Es\bullet es \in cse\}$
- 657<sub>uid.</sub>  $uid\_s\_σ:(AP|AL|MAL|SALA|SALs|SAL|SA|Ss|S|ME|RA|Rs|R|ES|Es)-set =$   
 657<sub>uid.</sub>  $\{uid\_ap\} \cup \{uid\_ala\} \cup \{uid\_mal\} \cup \{uid\_sala\} \cup uid\_csal \cup uid\_cssa \cup$   
 657<sub>uid.</sub>  $uid\_css \cup uid\_csme \cup uid\_csra \cup uid\_csr \cup uid\_cses \cup uid\_cse$

### An Invariant

668. All parts are uniquely identified, cf. Item 657 on page 428 and Item 657<sub>uid</sub> on page 430.

668.  $card\ s\_σ = card\ uid\_s\_σ$

### Part Retrieval

669. From a unique identifier of a domain and the domain endurant state we can obtain the identified endurant.

#### value

669.  $retr\_End: UI \rightarrow P\text{-}set \rightarrow P$

669.  $retr\_End(ui)(σ) \equiv \mathbf{let\ } p\bullet p \in σ \wedge uid(p) = ui \mathbf{\ in\ } p \mathbf{\ end}$

#### axiom

669.  $σ = s\_σ \wedge ui \in uid\_s\_σ \wedge p \in s\_σ$

### The Unique Identifier Indexed Endurant State

We can define a map from unique identifiers of endurant parts to these.

#### value

657.  $σ_{uid} =$

640<sub>σ.</sub>  $[ uid\_ap \mapsto ap,$

641<sub>σ.</sub>  $uid\_ala \mapsto ala,$

642<sub>σ.</sub>  $uid\_mal \mapsto mal,$

643<sub>σ.</sub>  $uid\_sala \mapsto sala,$

644<sub>σ.</sub>  $uid\_csal \mapsto csal,$

645<sub>σ.</sub>  $uid\_csals \mapsto csals,$

646<sub>σ.</sub>  $uid\_cssa \mapsto cssa,$

647<sub>σ.</sub>  $uid\_css \mapsto css,$

648<sub>σ.</sub>  $uid\_cs \mapsto cs,$

649<sub>σ.</sub>  $uid\_csme \mapsto csme,$

650<sub>σ.</sub>  $uid\_csra \mapsto csra,$

651<sub>σ.</sub>  $uid\_csrs \mapsto csrs,$

652<sub>σ.</sub>  $uid\_csr \mapsto csr,$

653<sub>σ.</sub>  $uid\_cses \mapsto cses,$

654<sub>σ.</sub>  $uid\_cse \mapsto cse ]$

We leave it to the reader to state the type of the  $σ_{uid}$  value!

### Taxonomy Map with Unique Identifier Labels

Figure O.10 on the next page<sup>20</sup> repeats Figs. O.8 on page 425 and O.9 on page 427. In Fig. O.10 lines are now labeled with appropriate unique identifiers. This leads up to Fig. O.11 on page 432.

<sup>20</sup>A difference between Fig. O.10 and Figs. O.8–O.9 is that in Fig. O.10 we have “moved” the left **MAL** taxonomy triangle a level down, to “level” with the right **SAL** triangles.

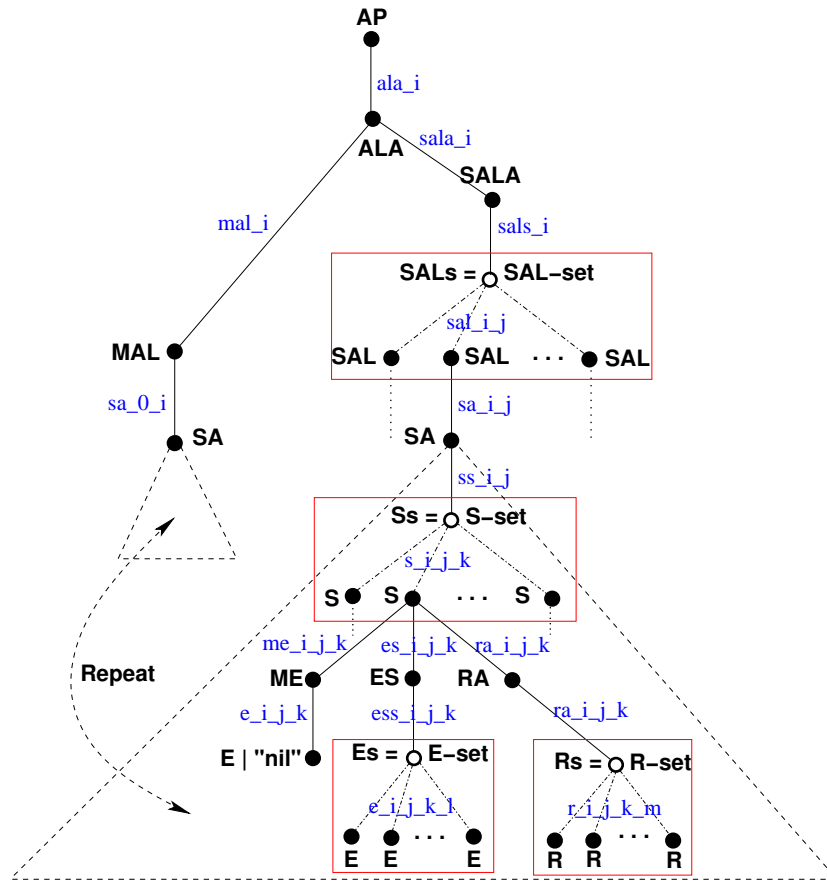


Figure O.10: Taxonomy with Unique Identifier Labels

Figure O.11 on the next page is a first, a graphical, two-dimensional expression. We shall comment on the graphics.

First one may say that Fig. O.11 shows “horizontally” what Figs. O.8–O.10 shows “vertically”.

Then we note that compound composites and compound sets are expressed as maps from unique part identifiers to parts (which include these unique identifiers).

And finally we note that each compound part is expressed as a pair:  $\rho\kappa, \text{map}$ , the  $\rho\kappa$  labels the upper left outside of the map – such that the parentheses of the pair,  $(\rho\kappa, \text{map})$ , is shown just before  $\rho\kappa$  and ends after  $\text{map}$ .

### Unique Identifier State Expressions

We now present the proper *Unique Identifier State Expression* formula sketched in Fig. O.11. It will be defined in terms of the *generate unique identifier state expression* function  $g\_uise$ .

[ 640 on page 421 ] **AP, Assembly Plants**

670. The *unique identifier state expression* for the *assembly plant* is the pair of the assembly plant core,  $ap\kappa$ , and the unique identifier state expression for the *assembly line aggregate*.

**value**

670.  $g\_uise(ap) \equiv (ap\kappa, g\_uise(ala))$

670. **where:**  $ap\kappa = \text{core\_AP}(ap) \wedge ala = \text{obs\_ALA}(ap)$

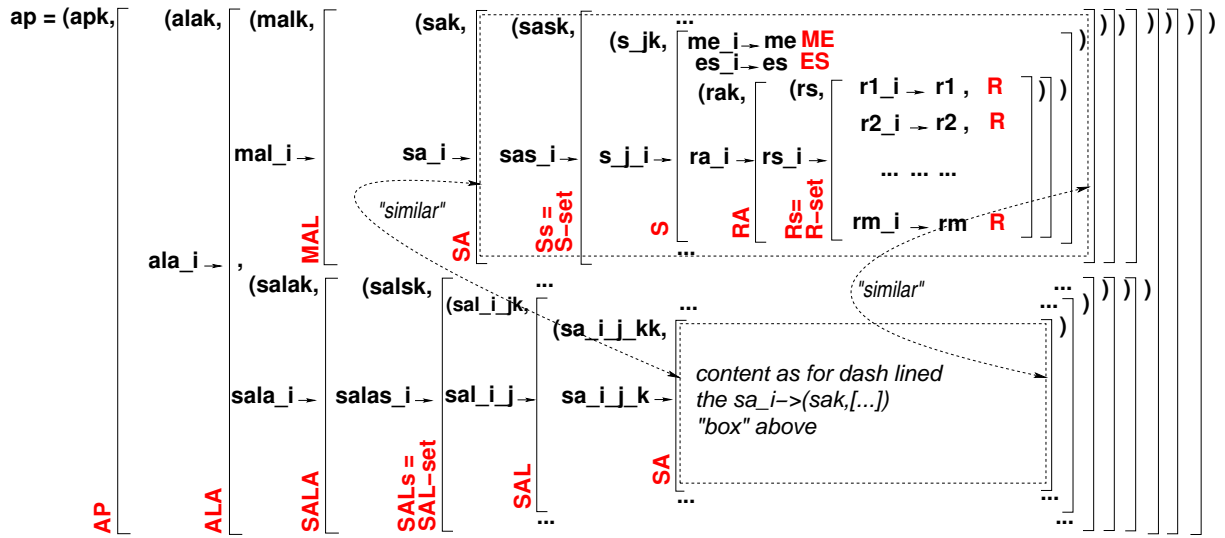


Figure O.11: Unique Identifier State Expression

[ 641 on page 421 ] **ALA, Assembly Line Aggregates**

671. The *unique identifier state expression* for the assembly line aggregate, *ala*, is the pair of the assembly line aggregate core, *alak*, and the singleton map from the unique identifier of the *assembly line aggregate* to that aggregate – expressed as a core-part annotated map.

```

value
671. g_uise(ala) ≡
671.' (alak,
671.' [uid_ALA(ala)↦
671.' [uid_MAL(mal)↦g_uise(mal),
671.' uid_SALA(sala)↦g_uise(sala)]]])

```

The one-liner, Item 671', just above, is too “complex”, better, we think, is the 4 liner just below, i.e., Items 671''–671''''.

```

value
671. g_uise(ala) ≡
671.'' (alak,
671.''' [uid_ALA(ala)
671.''''' ↦ [uid_MAL(mal) ↦ g_uise(mal),
671.''''''' uid_SALA(sala) ↦ g_uise(sala)]]])
671. where: alak=core_AP(ala) ∧ mal = obs_MAL(ala) ∧ sala = obs_SALA(ala)

```

[ 642 on page 421 ] **MAL, Main Assembly Lines**

672. The *unique identifier state expression* for the main assembly line, *mal*, is the pair of the main assembly line core, *malκ*, and the map from the unique identifier of its station assembly, *sa*, and the unique identifier state expression for the station assembly.

**value**

672.  $g\_uise(mal) \equiv$   
 672.  $(mal\kappa,$   
 672.  $[uid\_MAL(mal) \mapsto [uid\_SA(sa) \mapsto g\_uise(sa)]])$   
 672. **where:**  $mal\kappa = core\_MAL(mal) \wedge sa = obs\_SA(mal)$

[ 643 on page 421 ] **SALA, Supply Assembly Line Aggregates**

673. The *unique identifier state expression* for the supply assembly line aggregate, *sala*, is the pair of the supply assembly line aggregate core, *sala* $\kappa$ , and the singleton map from the unique identifier of the *set of assembly lines* to that set – expressed as a core-part annotated map.

**value**

673.  $g\_uise(sala) \equiv$   
 673.  $(sala\kappa,$   
 673.  $[uid\_SALA(sala) \mapsto [uid\_SALs(sals) \mapsto g\_uise(sals)]])$   
 673. **where:**  $sals = obs\_SALs(sala) \wedge$

[ 644 on page 422 ] **SALs=SAL-set, Supply Assembly Line Sets**

674. The *unique identifier state expression* for the supply assembly line set, *sals*, is the pair of the supply assembly line set core, *sals* $\kappa$ , and the map from the unique identifier of each of the *supply assembly lines* to that set – expressed as a core-part annotated map.

**value**

674.  $g\_uise(sals) \equiv$   
 674.  $(sals\kappa,$   
 674.  $[uid\_SAL(sal) \mapsto g\_uise(sal) \mid sal:SAL \bullet sal \in sals])$   
 674. **where:**

[ 645 on page 422 ] **SAL, Supply Assembly Lines**

675. The *unique identifier state expression* for the supply assembly line, *sal*, is the pair of the main assembly line core, *sal* $\kappa$ , and the singleton map from the unique identifier of its station assembly, *sa*, and the unique identifier state expression for that station assembly.

**value**

675.  $g\_uise(sal) \equiv$   
 675.  $(sal\kappa,$   
 675.  $[uid\_SAL(sal) \mapsto [uid\_SA(sa) \mapsto g\_uise(sa)]])$   
 675. **where:**  $sal\kappa = core\_SAL(sal) \wedge sa = obs\_SA(sal)$

[ 646 on page 422 ] **SA, Station Aggregates**

676. The *unique identifier state expression* for the station aggregate, *sa*, is the pair of the station aggregate core, *sa* $\kappa$ , and the map from the unique identifier of each of the *stations* to that set – expressed as a core-part annotated map.

**value**

676.  $g\_uise(sa) \equiv$   
 676.  $(s\kappa ,$   
 676.  $[ uid\_SA(sa) \mapsto g\_uise(ss) ] )$   
 676. **where:**  $s\kappa = core\_SA(sa) \wedge ss = obs\_Ss(sa)$

[ 647 on page 422 ] **Ss=S-set, Station Set**

677. The *unique identifier state expression* for a set of stations,  $ss$ , is the pair of the station set core,  $s\kappa$ , and the singleton map from the unique identifier of each of the *stations* to that set – expressed as a core-part annotated map.

**value**

677.  $g\_uise(ss) \equiv$   
 677.  $(s\kappa ,$   
 677.  $[ uid\_Ss(s) \mapsto g\_uise(s) \mid s:S \cdot s \in ss ] )$   
 677. **where:**

[ 648 on page 422 ] **S, Stations**

678. The *unique identifier state expression* for stations,  $s$ , is the pair of the station core,  $s\kappa$ , and the map from

- the unique identifier of that stations' *main element* to that main element, considered an atomic,
- the unique identifier of that stations' *element supply* to that element supply, here considered an “atomic” (!), and
- the unique identifier of that stations' *robot aggregate* to the unique identifier state expression for that robot aggregate.

**value**

678.  $g\_uise(s) \equiv$   
 678.  $(s\kappa ,$   
 678.  $[ uid\_ME(me) \mapsto me ,$   
 678.  $uid\_ES(es) \mapsto es ,$   
 678.  $uid\_RA(ra) \mapsto g\_uise(ra) ] )$   
 678. **where:**  $s\kappa = core\_S(s) \wedge me = obs\_ME(s) \wedge es = obs\_ES(s) \wedge ra = obs\_RA(s)$

[ 649 on page 422 ] **ME, Main Elements**

Item 678 above expresses that  $g\_uise(me) = me$ .

[ 650 on page 422 ] **RA, Robot Aggregates**

679. The *unique identifier state expression* for robot aggregates,  $ra$ , is a pair of the robot aggregate core,  $ra\kappa$ , and the singleton map from unique identifier of the robot aggregate to the unique identifier state expression for the set of robots,  $rs$ , of that aggregate.



**value**

679.  $g\_uise(ra) \equiv$   
 679.  $(ra\kappa ,$   
 679.  $[ uid\_Rs(rs) \mapsto g\_uise(rs) ] )$   
 679. **where:**  $rs = obs\_Rs(ra)$

[ 651 on page 422 ] **Rs=R-set, Robot Sets**

680. The *unique identifier state expression* for robot sets,  $rs$ , is a pair of the robot set core and the map from the unique identifiers of the robots of the set to these robots.

**value**

680.  $g\_uise(rs) \equiv$   
 680.  $(rs\kappa ,$   
 680.  $[ uid\_R(r) \mapsto r \mid r:R \bullet r \in rs ] )$

[ 652 on page 422 ] **R, Robots**

Item 680 expresses that  $g\_uise(r) = r$ .

[ 653 on page 422 ] **ES, Element Supplies**

Item 678 on the preceding page above expresses that  $g\_uise(es) = es$ .

[ 654 on page 422 ] **Es=E-set, Element Supply Sets**

Item 678 on the preceding page hence expresses that  $g\_uise(ess) = ess$ .

[ 655 on page 422 ] **E, Elements**

Item 678 on the preceding page hence expresses that  $g\_uise(e) = e$ .

**O.2.2.2.2 Mereology** Observation of endurant parts does not itself leave any trace as to their taxonomy, nor does the identification of observed parts.

Mereology is what brings forth the taxonomy structures that is rendered, one way or another, in all the figures shown so far!

We shall show that many of the concerns of [77] have their “root” in mereology-properties of the domain; and we shall show that the topological aspects of the mereology “supports” Microsoft’s *Automated Graph Layout Tool* [142].

We express the mereology properties as relations between the mereology of the endurant being inquired, some or all elements of the mereology of the “ancestor” endurant, and some or all elements of the mereology of the “descendant” endurant(s).

Common to all `mereo_P` observers we “retrieve” the “predecessor” part, from the overall endurant state, and observe its mereology, while also “retrieving” the “descendant” parts, also from the overall endurant state, given their identifiers from the mereology of the part under observation, and then correlate them.

We then end up with a set of mereology types, a set of corresponding mereology observer signatures [not definitions], and a set of corresponding axioms. For any given domain the mereology expresses some property that holds and that property transpires as the fix-point solution to the mutually [but not recursively] – sort-of simultaneous[ly] – expressed axioms [in the form of equations].

The overall property of the mereologies presented here is to secure that no two parts have identical mereologies.

*That should be a provable property of what is presented below.*

- The following numbered paragraphs start with the *item number* of the endurant, whose **name** is given next. The item numbers are formally defined on page 422.

ι **640. AP: Assembly Plant**

681. The mereology of an assembly plant is

- the unique identifier of its assembly line aggregate – such that
  - (a) the successor part's mereology identifies the assembly plant.

**type**

681. AP\_Mer = ALAI

**value**

681. mereo\_AP: AP → AP\_Mer

**axiom**

681. **let** alai = mereo\_AP(ap) **in**

681a. **let** (api, \_) = mereo\_ALA(retr\_ALA(alai)) **in** retr\_AP(api) = ap **end end**

ι **641. ALA: Assembly Line Aggregate**

682. The mereology of an assembly line aggregate is a pair

- of the unique identifier of the main assembly line
- and the unique identifier of the supply assembly line aggregate – such that
  - (a) the [assembly plant's, i.e., the] predecessor's successor is that assembly line aggregate and
  - (b) the two successors' ancestor are likewise.

**type**

682. ALA\_Mer = API × (MALI × SALAI)

**value**

682. mereo\_ALA: ALA → ALA\_Mer

**axiom**

682. **let** (api, (mali, salai)) = mereo\_ALA(ala) **in**

682a. **let** alai = mereo\_AP(retr\_AP(api)),

682b. (alai', \_) = mereo\_MAL(retr\_MAL(mali)),

682b. (alai'', \_) = mereo\_SALA(retr\_SALA(salai)) **in**

682a. alai = uid\_ALA(ala) ∧

682b. alai = alai' = alai'' **end end**

ι **642. MAL: Main Assembly Line**

683. The mereology of a main assembly line aggregate is

- the pair of the unique identifier of an assembly line aggregate and
- the unique identifier of a station aggregate – such that
  - (a) the main assembly line's unique identifier is the same as the [assembly line aggregate] ancestor's successor and
  - (b) [station aggregate] successor's ancestor.

**type**683.  $MAL\_Mer = ALAI \times SAI$ **value**683.  $mereo\_MAL: MAL \rightarrow MAL\_Mer$ **axiom**683. **let** (alai,sai) = mereo\_MAL(mal), mali = uid\_MAL(mal) **in**683a. **let** ( $\_$ , (mali',  $\_$ )) = mereo\_ALA(retr\_ALA(alai')),683b. (mali'',  $\_$ ) = mereo\_SA(retr\_SA(sai)) **in**683a. mali = mali'  $\wedge$ 683b. mali = mali'' **end end****ι 643. SALA: Supply Assembly Line Aggregate**

684. The mereology of a supply assembly line aggregate is

- the unique identifier of an assembly line aggregate and
- a pair of the unique identifier of a supply assembly line set – such that
  - (a) the [assembly line aggregate] predecessor's successor and
  - (b) the [supply line set] successor's predecessor
 supply line aggregate identifiers are the same.

**type**684.  $SALA\_Mer = ALAI \times SALsI$ **value**684.  $mereo\_SALA: SALA \rightarrow SALA\_Mer$ **axiom**684. **let** (alai,salsi) = mereo\_SALA(sala), salai=uid\_SALA(sala) **in**684. **let** ( $\_$ , ( $\_$ , salai')) = mereo\_ALA(retr\_ALA(alai)),684. (salai'',  $\_$ ) = mereo\_SALs(retr\_SALs(salsi)) **in**684a. salai = salai'  $\wedge$ 684b. salai = salai'' **end end****ι 644. SALs=SAL-set: Simple Assembly Line Set**

685. The mereology of a set of simple assembly lines is a pair of

- the unique identifier of a supply assembly line aggregate and
- a set of the unique identifiers of station aggregates – such that
  - (a) the [supply line aggregate] predecessor's successor and
  - (b) each individual simple assembly line's predecessor
 supply line set identifiers are the same.

**type**685.  $SALs\_Mer = SALAI \times SAI\_set$ **value**685.  $mereo\_SALs: SALs \rightarrow SALAI \times SAI\_set$ **axiom**685. **let** (salai,sais) = mereo\_SALs(sals), salsi = uid\_SALs(sals) **in**685. **let** ( $\_$ , salsi') = mereo\_SALA(retr\_SALA(salai)) **in**685a. salsi = salsi'  $\wedge$ 685b.  $\forall$  sai:SAL•sai  $\in$  sais  $\Rightarrow$  **let** (sals'',  $\_$ ) = mereo\_SA(retr\_SA(sai)) **in** salsi=sals'' **end**685. **end end**

### ℓ 645. **SAL: Simple Assembly Lines**

686. The mereology of a simple assembly line is a pair of

- the unique identifier of a [predecessor] supply assembly line set and
- the unique identifier of a [successor] station assembly – such that
  - (a) the [supply assembly line set] predecessor's and
  - (b) the [station assembly] successor's

simple assembly line identifiers are the same and that of the simple assembly line being observed.

#### **type**

686.  $SAL\_Mer = SALsl \times SAI$

#### **value**

686.  $mereo\_SAL: SAL \rightarrow SAL\_Mer$

#### **axiom**

686. **let** (salsi,sai) = mereo\_SAL(sal), sali = uid\_SAL(sal) **in**

686. **let** (\_\_,sali') = mereo\_SALs(retr\_SALs(salsi)), (sali'i,\_\_) = mereo\_SA(retr\_SA(sai)) **in**

686a.  $sali = sali' \wedge$

686b.  $sali = sali''$

686. **end end**

### ℓ 646. **SA: Station Aggregate**

687. The mereology of a station aggregate is a pair of

- the unique identifier of the [simple assembly line] predecessor and
- the unique identifier of the [station set] successor – such that
  - (a) their station aggregate (successor), respectively (predecessor) station aggregate identifiers are the same as that of the station aggregate being observed.

#### **type**

687.  $SA\_Mer = SALI \times Ssl$

#### **value**

687.  $mereo\_SA: SA \rightarrow SA\_Mer$

#### **axiom**

687. **let** (sali,ssi) = mereo\_SA(sa), sai = uid\_SA(sa) **in**

687. **let** (\_\_,sai') = mereo\_SAL(retr\_SAL(sali)), (sai'',\_\_) = mereo\_Ss(retr\_Ss(ssi)) **in**

687a.  $sai = sai' = sai''$  **end end**

### ℓ 647. **Ss = S-set: Station Sets**

688. The mereology of a station set is a pair of

- the unique identifier of a [predecessor] station aggregate and
- a set of unique identifiers of [successor] stations – such that
  - (a) that station aggregate's successor and
  - (b) that each successor station's predecessor

unique identifiers are the same as that of the observed station set.

**type**688.  $Ss\_Mer = SAI \times SI\text{-set}$ **value**688.  $mereo\_Ss: Ss \rightarrow Ss\_Mer$ **axiom**688. **let** (sai, sis) = mereo\_Ss(ss), ssi = uid\_Ss(ss) **in**688a. **let** (\_\_, ssi') = mereo\_(retr\_SA(sai)) **in** ssi = ssi' **end**688b.  $\forall si:SI \bullet si \in ssi \bullet$  **let** (ssi'', \_\_) = mereo\_S(retr\_S(si)) **in** ssi = ssi'' **end end****648. S: Station**

For all but stations the mereologies of solid endurants have modeled the part-hood relation “part of” (in the sense of “sub-part of”). All taxonomy figures<sup>21</sup> show this “sub-part” relation by means of the *lines* connection the  $\bullet$ s. Figure O.5 on page 421 show two additional [topological] part-hood relations: “adjacent to” and “incident upon”. Two stations of a simple assembly line may be adjacent to one another. The last station of a supply assembly line is incident upon a station of a main assembly line. The first station of any assembly line has no predecessor. The last station of a main assembly line has no successor.

689. Thus the mereology of a station,  $s$  identified by  $si$ , is a pair of,

(a) *first* a pair, modeling “part of”:

- i. the unique identifier of a station set,  $ssi$ , the predecessor of  $s$ ,
- ii. the unique identifiers of a triplet  $[(mei, esi, rai)]$  of successors of  $s$ :
  - A. a main element  $mei$ ,
  - B. an element supply,  $esi$ , and
  - C. a robot aggregate  $rai$ ,

and

(b) *then* a pair,  $(nsi, psi)$ , modeling,  $nsi$  “[next] adjacent to”, and,  $psi$  “[previous] incident upon” such that,

- i. for the first of the pair, i.e.,  $nsi$ , is
  - A. either “nil” for the “last” station, the outlet, of a main line,
  - B. or is the next station of a main or supply line,
  - C. or, for  $s$  being the “downstream last” of a supply line station, identifies a mainline station.
- ii. for the second of the pair,  $psi$  is [again] a pair:  $(plsi, lslsi)$ , where
  - A.  $plsi$  is the station identifier of a station of the line to which  $s$  belongs, where
    - $plsi$  is “nil”, if  $s$  is the “first, upstream”, station of its line, or
    - $plsi$  properly identifies an “upstream” immediately previous station,
  - and where  $lslsi$  is
    - B. either “nil”, i.e., station  $s$  is not one incident upon by a supply assembly line,
    - C. or is the proper identifier of a supply assembly line’s “downstream, last” station such that
      - no two main line stations have the same supply assembly line incident upon them, and
      - where the number of supply assembly lines exactly equal the number of main line stations that have supply assembly lined incident upon them.

All of the above must satisfy the following invariants:

<sup>21</sup>Figs. O.6 on page 422, O.7 on page 423, O.8 on page 425, O.9 on page 427 and O.10 on page 431

- (c) the unique identifier,  $si$ , of  $s$ , is in the the set of unique station identifiers of the predecessor,

and such that the unique identifiers of

- (d) the main element's,  
 (e) the element supply's, and  
 (f) the robot aggregate's

predecessors are all the same as that of the station under observation – and such that

- (g) the stations,  $ss$ , of the ancestor station set do indeed form a linear sequence;  
 (h)  $si'$  and  $si''$  [in  $(si', si'')$ ] are indeed station identifiers of that sequence – or one is that of the next-but-last station of a supply assembly line and the other is that of a station of a main assembly line;  
 (i)  $si'$  [in  $(\text{"nil"}, si')$ ] is indeed a station identifier of that sequence; and  
 (j)  $si'$  [in  $(si', \text{"nil"})$ ] is indeed a station identifier of that sequence.

We model the notion of linear sequences [here of stations].

- (k) Let  $ls:LS=S^*$  stand for a linear sequence of two or more stations  $S$ .  
 (l) Let  $ss$  stand for a set of two or more stations, i.e.,  $ss \in Ss = S\text{-set}$ .  
 (m) Then let  $linear\_Ss$  be the function which “converts”  $ss$  to  $ls$ .<sup>22</sup>

### type

689.  $S\_Mer = (Ss1 \times (MEI \times ESI \times RAI)) \times ((opt\_SI \times opt\_SI) \times opt\_SI)$   
 689.  $opt\_SI = (\{\text{"nil"}\} \mid SI)$

### value

689.  $S\_Mer: S \rightarrow S\_Mer$

### axiom

689. **let**  $((ssi, (mei, esi, rai)), ((si\_b, si\_a), si\_sl)) = S\_Mer(s)$ ,  $si = uid\_S(s)$  **in**  
 689. **let**  $(\_, sis) = mereo\_Ss(retr\_Ss(ssi))$ ,  $(si', \_) = mereo\_ME(retr\_ME(mei))$ ,  
 689.  $(si'', \_) = mereo\_ES(retr\_ES(es))$ ,  $(si''', \_) = mereo\_RA(retr\_RA(rai))$  **in**  
 689(b)ii.  $si \in sis \wedge$   
 689d.  $si = si' \wedge$   
 689e.  $si = si'' \wedge$   
 689f.  $si = si'''$  **end end**

689(a)i.  $\forall s:S \bullet \text{let } (\_, (b\_si, a\_si)) = S\_Mer(s) \text{ in}$

689(a)i.  $b\_si \neq \text{"nil"} \wedge a\_si \neq \text{"nil"} \vee$

689(a)i.  $b\_si = \text{"nil"} \wedge a\_si \neq \text{"nil"} \vee$

689(a)i.  $b\_si \neq \text{"nil"} \wedge a\_si = \text{"nil"}$  **end**

689(a)ii.

689(a)iiA.

689(a)iiB.

689(a)iiC.

689(b)i.

689(b)iA.

689(b)iB.

689(b)iC.

<sup>22</sup>It is not a matter of whether or not an  $ss \in Ss = S\text{-set}$  may form a linear sequence. They simply do! An assembly plant's assembly lines simply are linear! Constellations of stations not forming linear sequences do not contribute to a proper assembly plant!

689(b)ii.  
 689(b)iiA.  
 689(b)iiB.

**type**

689k.  $LS = S^*$

**axiom**

689k.  $\forall ls:LS \bullet \text{len } ls > 1$

689k.  $\text{is\_linear}: LS \rightarrow \mathbf{Bool}$

689k.  $\text{is\_linear}(ls) \equiv$

689k.  $\wedge \text{let } (\_,(\text{null},\_)) = \text{mereo\_S}(ls[1]) \text{ in } \text{null} = \text{"nil" end}$

689k.  $\wedge \forall i:\mathbf{Nat} \bullet \{i,i+1\} \subseteq \text{inds } ls \Rightarrow$

689k.  $\quad \text{let } (\_,(\_,\text{si\_a})) = S\_Mer(ls[i]),$

689k.  $\quad (\_,(\text{si\_b},\_)) = S\_Mer(ls[i+1]) \text{ in } \text{si\_a} = \text{uid\_S}(ls[i]) = \text{si\_b end}$

689k.  $\wedge \text{let } (\_,(\text{s\_uid})) = \text{mereo\_S}(ls[\text{len } ls]) \text{ in}$

689k.  $\wedge (\text{s\_uid} = \text{"nil"} \vee \text{is\_MAL\_S}(\text{retr\_S}(\text{s\_uid}))) \text{ end}$

**value**

689k.  $\text{is\_MAL\_S}: S \rightarrow \mathbf{Bool}$

689k.  $\text{is\_MAL\_S}(s) \equiv$

689k.  $\quad \text{let } ((\text{ssi},\_),\_) = \text{mereo\_S}(s) \text{ in}$

689k.  $\quad \text{let } (\text{sai},\_) = \text{mereo\_Ss}(\text{retr\_Ss}(\text{ssi})) \text{ in}$

689k.  $\quad \text{let } \text{ali} = \text{mereo\_SA}(\text{retr\_SA}(\text{ssi})) \text{ in}$

689k.  $\quad \text{is\_MALI}(\text{ali}) \text{ end end end}$

689l.  $\text{ss}:Ss, \text{ axiom card } ss > 1$

689g.  $\text{linear\_Ss}: S\text{-set} \rightarrow S^*$

689g.  $\text{linear\_Ss}(ss) \equiv$

689g.  $\quad \text{let } ls:LS \bullet \text{elems } ls = ss \wedge$

689g.  $\quad \forall i:\mathbf{Nat} \bullet \{i,i+1\} \subseteq \text{inds } ls \Rightarrow$

689g.  $\quad \quad \text{let } (\_,(\_,\text{a\_si})) = \text{mereo\_S}(ls[i])$

689g.  $\quad \quad \text{let } (\_,(\text{b\_si},\_)) = \text{mereo\_S}(ls[i+1]) \text{ in}$

689g.  $\quad \quad \text{a\_si} = \text{b\_si end}$

689g.  $\quad \text{ls end end}$

### ℓ 649. ME: Main Elements

690. The mereology of a main element is a singleton

- of the unique identifier of its predecessor station – such that
  - (a) that station identifies that main element.

**type**

690.  $\text{ME\_Mer} = S1$

**value**

690.  $\text{mereo\_ME}: \text{ME} \rightarrow \text{ME\_Mer}$

**axiom**

690a.  $\text{let } \text{si} = \text{mereo\_ME}(\text{me}), \text{ mei} = \text{uid\_ME}(\text{me}) \text{ in}$

690a.  $\text{let } (\_,(\text{mei}',\_,\_)) = \text{mereo\_S}(\text{retr\_S}(\text{si})) \text{ in}$

690a.  $\text{mei} = \text{mei}' \text{ end end}$

### ℓ 650. RA: Robot Aggregate

691. The mereology of a robot aggregate is

- a pair of the unique identifier of a station (the predecessor) and
- a unique identifier of a robot set (the successors) – such that
  - (a) the station predecessor identifies the robot aggregate, and
  - (b) the identified robot set identifies the same robot aggregate.

**type**

691.  $RA\_Mer = SI \times Rsl$

**value**

691.  $mereo\_RA: RA \rightarrow RA\_Mer$

**axiom**

691. **let**  $(si, rsi) = mereo\_RA(ra)$ ,  $rai = uid\_RA(ra)$  **in**

691. **let**  $(\_, (rai', \_, \_)) = mereo\_S(retr\_S(si))$ ,

691.  $(rai'', \_) = mereo\_Rs(retr\_Rs(rsi))$  **in**

691a.  $rai = rai' \wedge$

691b.  $rai = rai''$  **end end**

ℓ **651. Rs=R-set: Robot Set**

692. The mereology of a robot set is a pair of

- the unique identifier of a robot aggregate and
- a set of unique identifiers of robots – such that
  - (a) the identified robot aggregate identifies the robot set, and
  - (b) all the identified robots also identifies that robot set.

**type**

692.  $Rs\_Mer = RAI \times RI\text{-set}$

**value**

692.  $mereo\_Rs: Rs \rightarrow Rs\_Mer$

**axiom**

692. **let**  $(rai, ris) = mereo\_Rs(rs)$ ,  $rsi = uid\_Rs(rs)$  **in**

692a. **let**  $(\_, rsi') = mereo\_RA(retr\_RA(rai))$  **in**  $rsi = rsi'$  **end**

692a.  $\forall ri: RI \bullet ri \in ris \Rightarrow$  **let**  $rsi'' = mereo\_R(retr\_R(ri))$  **in**  $rsi = rsi''$  **end end**

ℓ **652. R: Robot**

693. The mereology of a robot is

- a singleton of the unique identifier of a robot set – such that.
  - (a) that robot set identifies the robot.

**type**

693.  $R\_Mer = Rsl$

**value**

693.  $mereo\_Rs: Rs \rightarrow Rs\_Mer$

**axiom**

693. **let**  $rsi = mereo\_R(r)$ ,  $ri = uid\_R(r)$  **in**

693a. **let**  $(\_, ris) = mereo\_Rs(retr\_Rs(rsi))$  **in**  $ri \in ris$  **end end**

ℓ **653. ES: Element Supply**



694. The mereology of an element supply is a pair of

- the unique identifier of a station and
- the unique identifier of an element supply set – such that
  - (a) the identified station identifies the element supply, and
  - (b) the identified element supply set identifies the element supply.

**type**

694.  $ES\_Mer = SI \times Esl$

**value**

694.  $mereo\_ES: ES \rightarrow ES\_Mer$

**axiom**

694. **let**  $(si, esi) = mereo\_ES(es)$ ,  $esi = uid\_ES(es)$  **in**

694. **let**  $((\_, (\_, esi', \_)), \_)$   $= mereo\_ES(retr\_ES(es))$ ,  $esi'' = uid\_ES(es)$  **in**

694a.  $esi = esi' \wedge$

694b.  $esi = esi''$  **end end**

ι **654. Es=E-set: Element Supply Set**

695. The mereology of an element supply set is a pair of

- the unique identifier of an element supply aggregate and
- a set of unique identifiers of elements – such that
  - (a) the identified element supply aggregate identifies the element supply set and
  - (b) the all the element identifiers identifies the element supply set.

**type**

695.  $Es\_Mer = ESI \times EI\text{-set}$

**value**

695.  $mereo\_Es: Es \rightarrow Es\_Mer$

**axiom**

695. **let**  $(esi, eis) = mereo\_Es(es)$ ,  $es\_i = uid\_Es(es)$  **in**

695a. **let**  $(\_, es\_j) = mereo\_Es(retr\_Es(es))$  **in**  $es\_i = es\_j$  **end**  $\wedge$

695b.  $\forall ei:EI \bullet ei \in eis \Rightarrow$  **let**  $es\_k = mereo\_E(retr\_E(ei))$  **in**  $es\_i = es\_k$  **end end**

ι **655. E: Elements**

696. The mereology of an element is

- a singleton of the unique identifier of an element supply set – such that
  - (a) this identifier identifies the element's supply set.

**type**

696.  $E\_Mer = Esl$

**value**

696.  $mereo\_E: E \rightarrow ES\_Mer$

**axiom**

696. **let**  $esi = mereo\_E(e)$ ,  $eis = uid\_E(e)$  **in**

696a. **let**  $(\_, eis')$   $= mereo\_Es(retr\_Es(es))$  **in**  $eis = eis'$  **end end**

It is all very tedious: Mereology after mereology – of each and all of the solid endurants. Their narratives and formalisations, expression-wise, all follow the same “pattern”, and the “contents” follow, almost mechanical, from the taxonomy figures<sup>23</sup> and, wrt. stations, from Figs. O.5 on page 421 and O.7 on page 423.

I have not followed a strict narrative for the 16 mereology presentations, and even the formulas differ slightly. Once I get time I will probably device a  $\text{\LaTeX}$  macro so as to generate consistent narratives.

### Distances of Stations from Outlet

#### Paths:

We shall examine an ordering,  $\preceq$ , on stations. To this end we introduce the notion of paths. A path is a sequence of station identifiers such that

- 697. A path is a non-empty sequence of station identifiers such that
- 698. the first identifier is that of the first station of a main assembly line,
- 699. and such that
- 700. adjacent identifiers of a path are those of neighbouring stations,
  - (a) whether of the same assembly line,
  - (b) or of
    - i. the first station of a supply assembly line
    - ii. and of the station of the main assembly line onto which supply assembly line is joined.

#### type

697. Path = SI\*

**axiom** [ paths of an assembly plant ]

697.  $\forall p:\text{Path} \bullet \text{len } p > 0 \wedge$

698. **let**  $\langle si \rangle^{\wedge} p' = p$ ,  $ss:Ss = \text{obs\_Ss}(\text{obs\_SA}(mal))$  **in**

698. **let**  $s:S \bullet s \in ss \wedge$  **let**  $(,(\text{nil})) = \text{mereo\_S}(s)$  **in**  $\text{nil} = "nil" \wedge si = \text{uid\_S}(s)$  **end end**

699.  $\wedge$

700.  $\forall i:\text{Nat} \bullet \{i, i+1\} \subseteq \text{inds } p \Rightarrow$

700. **let**  $(\_, (pi, \_)) = \text{mereo\_S}(\text{retr\_S}(i))$ ,  $(\_, (\_, si)) = \text{mereo\_S}(\text{retr\_S}(i+1))$  **in**

700a.  $(pi = p(i+1) \wedge si = p(i))$

699.  $\vee$

700(b)i.  $($

700(b)i.  $\wedge$

700(b)ii.  $\dots )$

697. **end end**

#### Set of all Paths:

From an assembly plant we can then generate the set of all paths.

701.

702.

703.

704.

<sup>23</sup>Figs. O.6 on page 422, O.7 on page 423, O.8 on page 425, O.9 on page 427 and O.10 on page 431

705.

706.

701.

702.

703.

704.

705.

706.

**Distance:**

Given any station of an assembly plant we can then calculate its distance from the main line outlet.

707.

708.

709.

710.

711.

712.

707.

708.

709.

710.

711.

712.

**The  $\preceq$  Relation:**

713. Given any two stations of an assembly plant we can then express which of the two “precedes” the other wrt. distance from the main line outlet.

713.

**O.2.2.2.3 Attributes****General**

The real action of an assembly line is focused in the stations. The robots apply elements to the contents of the main element. So, in treating now the attributes of assembly lines, we shall in this early version of this project report, focus on the rôle of elements.

**Elements and Parts**

The term ‘*part*’ is a main term of the *domain analysis & description* method [58] that we use. It is not to be confused with the same term, i.e., **part**, used, normally, in connection with machine parts, part assembly, etc. The term **Main Element** is used to name the solid endurant of a station, namely that which, so-to-speak, “holds” the main object of concern: the thing being assembled. We shall think of main elements to be some form of manifest “carrier”. We shall then ascribe

such main elements an attribute, and here we shall switch to the use of the term part, namely a **main part**. So element supplies, which we hitherto explained as containing elements for use in the assembly of main parts, could, as well be called parts. Whereas solid durants such as stations and robot will, later, be “morphed”, i.e., transcendently deduced, into behaviours, we shall not morph main parts into behaviours – not as long, at least, as they stay within the assembly lines. Once a main part has left a main assembly line, “from” its last station, then it may, in some other domain model, attain “life” in the form of a behaviour.<sup>24</sup>

### Relationship to [77]

We shall show that many of the concerns of [77] have their “root” in attribute-properties of the domain.

### Specifics

**O.2.2.2.3.1 / 640. AP: Assembly Plant:** We omit treatment of assembly plant attributes.

**O.2.2.2.3.2 / 641. ALA: Assembly Line Aggregate:** We presently omit treatment of assembly line aggregate attributes.

**O.2.2.2.3.3 / 642. MAL: Main Assembly Line:** With every supply assembly line we associate the attributes

714. that it is a main assembly line, and that

715. the main elements of its stations contain parts of a specific (to be finalised) element type.

#### type

714.  $AL\_Typ = "Main"$

715.  $ME\_Typ = E\_Typ$

#### value

714.  $attr\_AL\_Typ: MAL \rightarrow AL\_Typ$

715.  $attr\_ME\_Typ: MAL \rightarrow ME\_Typ$

**O.2.2.2.3.4 / 643. SALA: Supply Assembly Line Aggregate:** We presently omit treatment of supply assembly line aggregate attributes.

**O.2.2.2.3.5 / 644. SALs: Supply Assembly Line Set:** We presently omit treatment of supply assembly line set attributes.

**O.2.2.2.3.6 / 645. SAL: Supply Assembly Lines:** With every supply assembly line we associate the attributes that

716. it is a supply assembly line<sup>25</sup>,

717. the main elements of its stations contain parts of a specific (to be finalised) element type,

718. it “feeds” into an identified main line station, and that

719. it is either “feeding” into the main line at the “left” or at the “right” !

<sup>24</sup>The main parts leaving the main assembly line of an automobile factory, in an orderly fashion, may then, as an automobile, be able to leave by its own means!

<sup>25</sup>– where main assembly lines are “Main” !

**type**

716. AL\_Typ = "Supply"  
 717. ME\_Typ = E\_Typ  
 719. Feed == "Left" | "Right"

**value**

716. attr\_AL\_Typ: SAL  $\rightarrow$  AL\_Typ  
 717. attr\_ME\_Typ: SAL  $\rightarrow$  ME\_Typ  
 718. attr\_MAL\_S: SAL  $\rightarrow$  SI  
 719. attr\_Feed: SAL  $\rightarrow$  Feed

**O.2.2.2.3.7 / 646. SA: Station Aggregate:** We presently omit treatment of station aggregate attributes.

**O.2.2.2.3.8 / 647. S<sub>s</sub>=S-set: Station Set:** We presently omit treatment of station set attributes.

**O.2.2.2.3.9 / 648. S: Station:** We first discuss some of the rôles played by the robots, main element part and element supply of a station.

- Robots of a station are **capable**, at any one time of performing one of a set of one or more operations. Robots and their operations have names, RN<sub>m</sub> respectively OpN<sub>m</sub>.

So we can attribute a station with

**type**

720. CAP = RN<sub>m</sub>  $\overrightarrow{m}$  OpN<sub>m</sub>-set

We allow two or more robots of any one station to "feature" the same, named operation!

- Operations, OP, are functions from a main element part and
  - either a single part provided by a supply line, if the operation is performed at a main line station, and
    - \* either
    - \* or a set of elements provided by that stations element supply
  - to an updated main element part.

**type**

721. OP = ME\_Part  $\times$  (ME\_Part|E-set)  $\rightarrow$  ME\_Part

- So a Station can be given the following attribute:

**type**

721. OPS = OpN<sub>m</sub>  $\overrightarrow{m}$  OP

Two or more differently named operations may, in fact, designate identical operations!

- Operations have types:

**type**

OpTyp = ME\_Part\_Typ  $\times$  (ME\_Part\_Typ | E\_Typ\*)  $\times$  ME\_Part\_Typ

So we assume that there are (meta-) functions like:

**value**

type\_of:  $E \rightarrow E\text{-Typ}$ , ME\_Part  $\rightarrow$  ME\_Part\_Typ, is\_of\_type:  $E \times E\text{-Typ} \rightarrow \mathbf{Bool}$ , etc.

We now “return” to our attribute “ascription story” proper!

With a station we can associate the following attributes:

- 720. The named operations that can be performed by it robots;
- 721. the catalogue of these operations;
- 722. the area of the assembly floor covered by the station;
- 723. the identified *zones* (sub-areas) into which the station is divided;

**type**

- 720. RNm, OpNm
- 720. CAP = RNm  $\overrightarrow{m}$  OpNm-set
- 721. OPS = OpNm  $\overrightarrow{m}$  OP
- 721. OP = ME\_Part  $\times$  (ME\_Part|E-set)  $\rightarrow$  ME\_Part
- 722. Sta\_Area = AREA
- 723. Zones = ZId  $\overrightarrow{m}$  Zone
- 723. Zone = Zone\_Area
- 723. Zone\_Area = AREA

**value**

- 720. attr\_CAP: S  $\rightarrow$  CAP
- 721. attr OPS: S  $\rightarrow$  OPS
- 722. attr\_StaArea: S  $\rightarrow$  StaArea
- 723. attr\_Zones: S  $\rightarrow$  Zones

**axiom**

- 723. [  $\cup$  of zone areas  $\equiv$  station area ]

**O.2.2.2.3.10**  $\nearrow$  **649. ME: Main Element:** With main elements we associate the following programmable attribute:

- 724. the main part, mp:ME\_Part and
- 725. the types of the main part before, during and after robot operations, i.e., as it enters the station, during its stay at the station, and as it leaves the station.

**type**

- 724. ME\_Part
- 725. ME\_Part\_Types = E\_Typ\*

**value**

- 724. attr\_Part: ME  $\rightarrow$  ME\_Part
- 725. attr\_ME\_Part\_Types ME  $\rightarrow$  ME\_Part\_Types

**Caveat:** The above type model is a bit simplified! Shall/must be reviewed!

**O.2.2.2.3.11**  $\nearrow$  **650. RA: Robot Aggregate: Caveat:** It seems that either stations or robot aggregates must have some form of awareness, expressed in the form of an attribute, of the **tasks** to be collectively, co-operatively performed by the ensemble of robots. **I am currently contemplating such a model!**

**O.2.2.2.3.12** / **651. Rs=R-set: Robot Set:** We presently omit treatment of robot set attributes.

**O.2.2.2.3.13** / **652. R: Robot:** With a robot we can associate the following attributes:

- 726. the zone to which it is allocated;
- 727. the operations it can perform and their type;
- 728. where we leave unspecified these element (i.e., part) types.
- 729. ...

**type**

- 726.  $R\_Zone = Zone$
- 727.  $R\_Ops = OpNm \xrightarrow{m} OpTyp$
- 727.  $OpTyp = ME\_Part\_Typ \times (ME\_Part\_Typ|E\_Typ^*) \times ME\_Part\_Typ$
- 728.  $ME\_Part\_Typ, E\_Typ$
- 729. ...

**value**

- 726.  $attr\_RZone: R \rightarrow RZone$
- 727.  $attr\_ROps: R \rightarrow ROps$
- 728. ...
- 729. ...

**O.2.2.2.3.14** / **653. ES: Element Supply:** An element supply can be characterised by

- 730. a catalogue of element “*quantities on hand*” and their type.

**type**

- 730.  $ES\_QoH\_Typ = E\_Typ \times Nat$

**value**

- 730.  $attr\_ES\_QoH\_Typ: ES \rightarrow ES\_QoH\_Typ$

**O.2.2.2.3.15** / **654. Es=E-set: Element Supply Set:** We presently omit treatment of element set attributes.

**O.2.2.2.3.16** / **655. E: Elements:** An element (i.e., a part) can be characterised by

- 731. its type

**type**

- 731.  $E\_Typ$

**value**

- 731.  $attr\_E\_Typ: E \rightarrow E\_Typ$

**O.2.2.3 Comments wrt. [77]**

We shall now relate the various segments of our model to [77].

TO BE WRITTEN

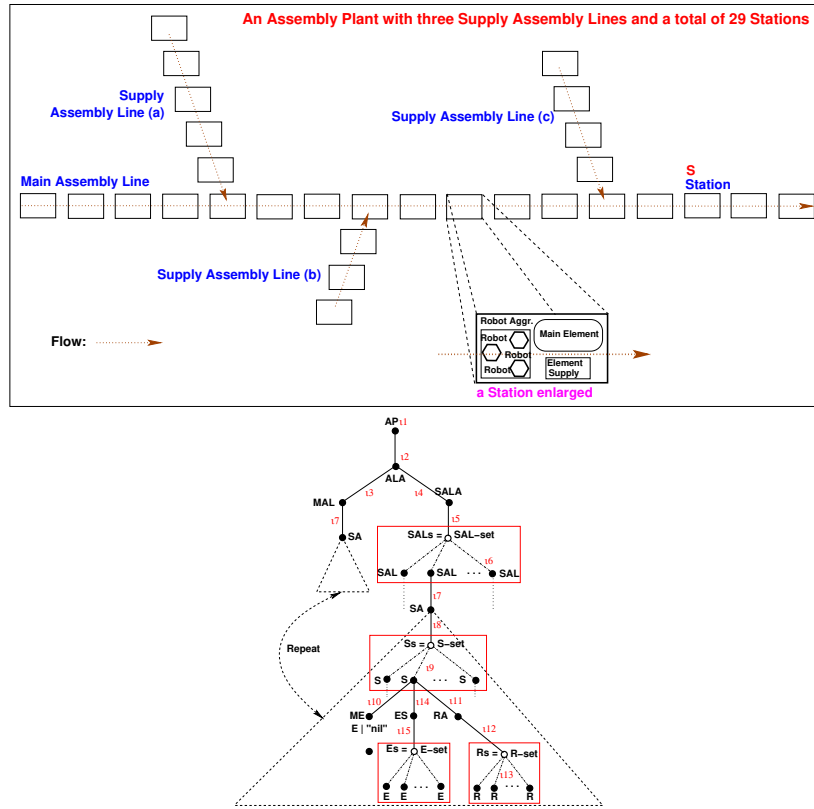


Figure O.12:

### O.2.3 Perdurants

#### O.2.3.1 From Parts to Behaviours

We refer the reader to Figs. O.5 on page 421 and O.8 on page 425 – summarised in Fig. O.12.

- By transcendental deduction, see [58, Chapter 5], we “morph” core parts,  $p_\kappa$ , i.e., including atomic parts, into behaviours  $\beta_p$ .
- Behaviour  $\beta_{ap}$  coordinates behaviour  $\beta_{ala}$  with the rest of the manufacturing plant – remember: the assembly line complex is only one among several factory elements.
- Behaviour  $\beta_{ala}$  coordinates the main assembly line behaviours with that of the behaviour of the supply assembly lines aggregate.
- Behaviour  $\beta_{mal}$  coordinates the main assembly line’s stations.
- Behaviour  $\beta_{sala}$  coordinates the total of all supply lines.
- Behaviours  $\beta_{sal}$  coordinates the specific supply assembly line’s stations.
- Behaviour  $\beta_{sa}$  coordinates the interaction between the stations of an assembly line.
- Behaviour  $\beta_s$  coordinates the specific station’s elements (main element, robots and element supply) as well as that station’s interaction with neighbouring stations.
- Behaviour  $\beta_{me}$  participates in the main elements interaction with its station’s robots.



- Behaviour  $\beta_{es}$  responds to its station's robots' requests for supply elements.
- Behaviour  $\beta_{ra}$  coordinates the specific station's robots.
- Behaviours  $\beta_r$  interacts with its station's main element, its other robots, and its element supply.
- Behaviours  $\beta_e$  – is presently left unspecified.

#### O.2.3.2 Channels

#### O.2.3.3 Actors

##### O.2.3.3.1 Actions and Events

##### O.2.3.3.2 Behaviours

TO BE WRITTEN

#### O.2.3.4 System Initialisation

### O.3 Discussion

We shall relate the model of Sect. O.2 to [77]. To us [77] both describes and prescribes: describes some aspects of the problem domain and prescribes some requirements.

TO BE WRITTEN

## O.4 Conclusion

We shall discuss whether the kind of work reported in [77] could be supported, made easier, made more complete, given that their domain is first properly described.

### O.4.1 Models and Axioms

TO BE WRITTEN

### O.4.2 Learning Forwards, Practicing In Reverse

The Danish philosopher Søren Kierkegaard (1813–1855) is quoted as saying

*Life can only be understood backwards; but it must be lived forwards.*

Now, why do we bringing that quote here?! We do so for the following, slightly, if not radically less “lofty” reason: We learn forward, bit-by-bit, not seeing the overall picture before at the end. Then, when we shall practice what we have been taught, what we have learnt, we apply that knowledge, so-to-speak, backwards, knowing where what we shall end up with from the start of that “doing it”.

When You study [58] You learn the subject forward. But having hopefully understood the domain modeling discipline, You You practice it “sort-of” in reverse.

My reason for bring the Søren Kierkegaard quote is to make You remember “that”!

### O.4.3 Diagrammatic Reasoning

One, of many, observations of this report, are the examples of what I shall refer to as *diagrammatic reasoning*<sup>26</sup>.

One way in which this is manifested, in this compendium is in Figs. O.5 on page 421, O.6 on page 422, O.7 on page 423, O.8 on page 425, O.9 on page 427 and O.10 on page 431. You may think that the number of these figures is a bit high. Very well, but they helped this “seasoned domain engineer” to come to grips with the seeming complexities of the domain being modeled. The internal relationships between these figures is obvious, “*when You look at them!*”, and their “external” relations to the *narration & formalisation* items should also be “obvious”!

### O.4.4 The Management of Domain Modeling

#### A Domain Modeling Development Plan

We outline a plan for the commercial/professional development of a domain model for a “real” [say automobile] assembly plant:

- **Study:**

- A domain is suggested.
- One or two seasoned domain engineers cum scientists , *the initiation team*, make inquiries about the domain:
  - \* Visit one or more such domain sites.
  - \* Search the Internet for reliable accounts on the domain.
  - \* Read technical/scientific papers about the domain.
- At some point the initiation team decides to do one or more experimental domain modeling efforts.

- **Experiment:**

- They follow the dogma of [58] – “strictly”.
- (This report is an example of such an experimental research and engineering development.)
- They may waver along different paths, maybe abandon/abort certain modeling directions, eventually reaching some, usually, incomplete domain analysis & description documentation.
- They may decide to do another, and, perhaps, subsequently yet another experimental research and engineering development.
- Eventually they **either abandon** the attempt to go after a fully complete, professional domain model, **or they conclude that a satisfactory, complete modeling project is professionally and commercial viable.**

- **Apply:**

- The first step in a professional and commercial domain modeling project is that of creating a staff plan:
  - \* An outcome of a final domain modeling experiment is that the main taxonomy of the domain has been settled upon.

<sup>26</sup>

- Gerard Allwein and Jon Barwise (ed.) (1996). *Logical Reasoning with Diagrams*. Oxford University Press.
- [https://en.wikipedia.org/wiki/Diagrammatic\\_reasoning](https://en.wikipedia.org/wiki/Diagrammatic_reasoning).

- \* For each of the main categories of endurants one or two domain engineers [cum scientists] are then to be allocated to the project.
  - \* A development graph<sup>27</sup> is developed.
  - \* A budget is established.
  - \* Negotiations with customer finally establish the financial foundation for the project<sup>28</sup>.
- The commercial development project starts.
  - First the endurant aspects are modeled – with
    - \* external qualities being first modeled [58, Chapter 4], then with
    - \* internal qualities:
      - unique identification [58, Sect. 5.2],
      - mereologies [58, Sect. 5.3], and
      - attributes [58, Sect. 5.4] – including notably *intentional pull* – which has not been illustrated in this report [58, Sect. 5.5].
  - Then perdurants:
    - \* states [58, Sect. 7.2],
    - \* channels [58, Sect. 7.5],
    - \* actor, i.e., action, event and behaviour, signatures [58, Sect. 7.6],
    - \* their definitions [58, Sect. 7.7], and
    - \* system initialisation [58, Sect. 7.8].
  - Etcetera!
  - Each project member either “sticks” to the initially assigned endurant (hence perdurant) area throughout the project, or members have their subject areas “rotated”.

Special circumstances may mandate variations to the above development plan.

For a reasonably “complete”, i.e., covering essential aspects of, say an automobile manufacturing plant’s assembly lines, it is roughly estimated that a group of well-educated domain engineers cum scientists would number 8–10, and that it would take 18–24 months to do the “Apply” phase of a domain modeling development project.

**O.4.5 ... one more section ...**

**O.4.6 ... a last section (?) ...**

**O.4.7 Acknowledgments**

TO BE WRITTEN

<sup>27</sup>For the notion of *Development Graphs* see [16–18].

<sup>28</sup>One cannot assume that the customer explicitly funds the Study and Experiment phases of the project.



# Appendix P

## Nuclear Power Plants

### Contents

---

|             |                                                  |            |
|-------------|--------------------------------------------------|------------|
| <b>P.1</b>  | <b>Introduction</b>                              | <b>457</b> |
| P.1.1       | <b>The Domain</b>                                | 457        |
| P.1.2       | <b>The Domain Description Ontology</b>           | 457        |
| <b>P.2</b>  | <b>Informal Characterisation</b>                 | <b>457</b> |
| <b>P.3</b>  | <b>Sketch of a Conceptual Domain Model</b>       | <b>459</b> |
| P.3.1       | <b>Endurants</b>                                 | 459        |
| P.3.1.1     | <b>External Qualities</b>                        | 459        |
| P.3.1.1.1   | <b>The Parts</b>                                 | 459        |
| P.3.1.1.2   | <b>Liquids</b>                                   | 461        |
| P.3.1.1.3   | <b>States, I</b>                                 | 461        |
| P.3.1.1.4   | <b>A First Review.</b>                           | 461        |
| P.3.1.1.5   | <b>Additional Parts</b>                          | 462        |
| P.3.1.1.6   | <b>A State Update.</b>                           | 463        |
| P.3.1.1.7   | <b>A Second Review.</b>                          | 463        |
| P.3.1.1.8   | <b>The Domain Taxonomy</b>                       | 463        |
| P.3.1.2     | <b>Internal Qualities</b>                        | 463        |
| P.3.1.2.1   | <b>Unique Identifiers</b>                        | 463        |
| P.3.1.2.1.1 | <b>The Unique Identifier Type and Observers.</b> | 463        |
| P.3.1.2.1.2 | <b>A State of Unique Identifiers.</b>            | 465        |
| P.3.1.2.1.3 | <b>An Axiom</b>                                  | 465        |
| P.3.1.2.2   | <b>Mereology</b>                                 | 465        |
| P.3.1.2.3   | <b>Attributes</b>                                | 472        |
| P.3.1.2.3.1 | <b>Common Part Attributes.</b>                   | 472        |
| P.3.1.2.3.2 | <b>Flow Attributes</b>                           | 473        |
| P.3.1.2.3.3 | <b>Pipe Attributes</b>                           | 473        |
| P.3.1.2.3.4 | <b>Specific Part Attributes</b>                  | 474        |
| P.3.1.2.4   | <b>Intentional Pull</b>                          | 479        |
| P.3.2       | <b>Perdurants</b>                                | 480        |
| P.3.2.1     | <b>Channels</b>                                  | 480        |
| P.3.2.2     | <b>Actions</b>                                   | 480        |
| P.3.2.3     | <b>Behaviours</b>                                | 481        |
| P.3.2.3.1   | <b>Behaviour Signatures</b>                      | 481        |
| P.3.2.3.2   | <b>Behaviour Definitions</b>                     | 483        |
| P.3.2.4     | <b>Action Signatures and Definitions</b>         | 488        |
| P.3.2.4.1   | <b>Action Signatures</b>                         | 488        |

|            |           |                                                   |            |
|------------|-----------|---------------------------------------------------|------------|
|            | P.3.2.4.2 | Action Definitions                                | 488        |
|            | P.3.2.5   | Domain Initialization                             | 488        |
|            | P.3.2.6   | Meaning of Domain Models                          | 489        |
| <b>P.4</b> |           | <b>System Domains</b>                             | <b>489</b> |
|            | P.4.1     | Some Preparatory Remarks                          | 490        |
|            | P.4.1.1   | The Triptych Development Dogma                    | 490        |
|            | P.4.1.2   | Simulators [Demos], Monitors and Controllers      | 490        |
|            | P.4.1.3   | Demos or Simulations                              | 491        |
|            | P.4.1.4   | Identity, Microscopic and Macroscopic Simulations | 492        |
|            | P.4.2     | Specific System Domains                           | 493        |
|            | P.4.2.1   | Design                                            | 493        |
|            | P.4.2.2   | Design Feasibility                                | 494        |
|            | P.4.2.3   | Building                                          | 494        |
|            | P.4.2.4   | Commissioning                                     | 494        |
|            | P.4.2.5   | Start-up                                          | 494        |
|            | P.4.2.6   | Steady State Operation                            | 494        |
|            | P.4.2.7   | Emergency Stops                                   | 494        |
|            | P.4.2.8   | Maintenance                                       | 495        |
|            | P.4.2.9   | Repair                                            | 495        |
|            | P.4.2.10  | Renewal of Fuel                                   | 495        |
|            | P.4.2.11  | Decommissioning                                   | 495        |
|            | P.4.2.12  | Removal                                           | 495        |
| <b>P.5</b> |           | <b>Varieties of Generation III-IV Reactors</b>    | <b>495</b> |
| <b>P.6</b> |           | <b>Closing</b>                                    | <b>495</b> |
|            | P.6.1     | What has so far been Achieved ?                   | 495        |
|            | P.6.2     | What is Next ?                                    | 496        |
|            | P.6.3     | Semantic versus Syntactic Reasoning               | 496        |
|            | P.6.3.1   | Petri nets                                        | 496        |
|            | P.6.3.2   | Multilevel Flow Modelling                         | 496        |
|            | P.6.4     | Acknowledgements                                  | 496        |
| <b>P.7</b> |           | <b>Bibliography</b>                               | <b>497</b> |
|            | P.7.1     | Bibliographical Notes                             | 497        |

---

There are different kinds of nuclear power plants. We focus on one kind, the pressurized steam nuclear power plants. In the US there are today almost 100 nuclear power plants. Around the world more than 400 nuclear reactors are installed. The US Nuclear Regulator Commission, U.S.NRC <https://www.nrc.gov/about-nrc.html> has the responsibility for the safe use of radioactive materials for beneficial civilian purposes while protecting people and the environment in the US. The NRC regulates commercial nuclear power plants and other uses of nuclear materials, such as in nuclear medicine, through licensing, inspection and enforcement of its requirements. We sketch the rudiments of a domain model for pressurized steam nuclear power plants. That model is then suggested instantiated to a number of strongly related domain models – covering:

- |                       |                           |                       |
|-----------------------|---------------------------|-----------------------|
| • design,             | • start-up,               | • repair,             |
| • design feasibility, | • steady state operation, | • renewal of fuel,    |
| • building.           | • emergency stops,        | • decommissioning and |
| • commissioning,      | • maintenance,            | • removal.            |

The models could then be a basis for simulators and monitors for regulatory work: inspection etc. The present proposal for modelling nuclear power plants should be seen in the context of, for example, *Multi-level Flow Modelling*, MFM [126]. In MFM it seems that systems are modelled graphically. Our approach is to model systems using simple discrete mathematics and mathematical logic.

**Warning**

This is a rather early draft. The present work began July 4th, 2023. Today is March 12, 2024: 10:48 am, There is no way I can believably fill in the very many most relevant attributes, Sect. P.3.1.2.3, nor the very many most relevant actions and behaviour definitions, Sects. P.3.2.2 and P.3.2.3.2. For that I need to talk with nuclear reactor people: scientists, engineers, and operators.

**References**

This document follows [61] “slavishly”. You may find [updates] to [61] and this document at:

- **Domain Modelling ....**  
<http://www.imm.dtu.dk/~dibj/2023/DomainModelling/DomainModelling23June2023.pdf>
- **Nuclear Power Plants ...**  
<http://www.imm.dtu.dk/~dibj/2023/nupopl/nupopl.pdf>

**Emendations**

Vertical margin bars, as [possibly] shown here, mark text corrected or new text wrt. to an immediately previous version of this document.

## P.1 Introduction

### P.1.1 The Domain

MORE TO COME

We refer to

- Generation II [Fission] Reactors  
[https://en.m.wikipedia.org/wiki/Generation\\_II\\_reactor<sup>1</sup>](https://en.m.wikipedia.org/wiki/Generation_II_reactor<sup>1</sup>),
- Generation III [Fission] Reactors  
[https://en.m.wikipedia.org/wiki/Generation\\_III\\_reactor](https://en.m.wikipedia.org/wiki/Generation_III_reactor),
- Generation IV [Fission] Reactors  
[https://en.m.wikipedia.org/wiki/Generation\\_IV\\_reactor](https://en.m.wikipedia.org/wiki/Generation_IV_reactor) and
- Fusion Reactors  
[https://en.m.wikipedia.org/wiki/Fusion\\_power](https://en.m.wikipedia.org/wiki/Fusion_power)

for general information.

MORE TO COME

### P.1.2 The Domain Description Ontology

Figure P.1 on the following page diagrams the structure of the way in which we analyze and describe domains.

## P.2 Informal Characterisation

We focus, as an example of how to model a nuclear fission power plant on Generation I–III reactors. We shall try explain Fig. P.2 on page 459.

<sup>1</sup>We omit consideration of Generation I Reactors

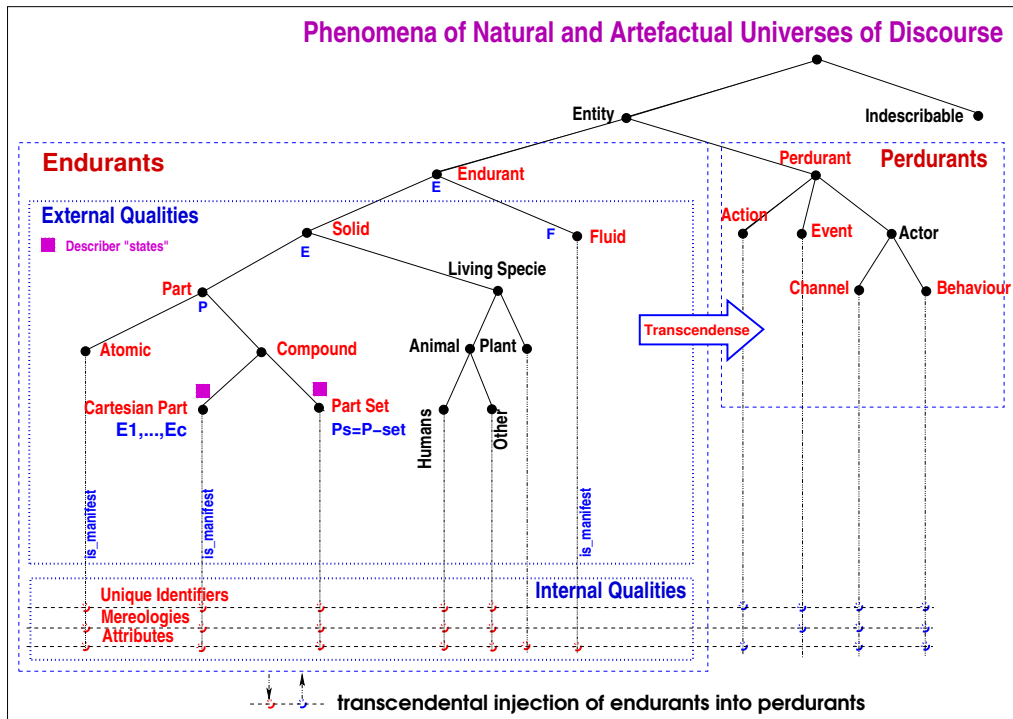


Figure P.1: An Upper Ontology

- The blue signifies cold[er] water.
- The red signifies warm[er] water.
- To the very left is the nuclear reactor, **R**, with its uranium core and control rods.
- To the immediate right of it is the steam generator, **SG**.
- The positioning of the uranium core and the rods is made such as to heat the water.
- The nuclear reactor and the steam generator are contained in a separate highly encased structure, **CS**.
- The cold/warm water of the reactor forms a separate loop – circulating through the steam generator.
- Cold[er] water is taken into the system – shown in the lower right of the figure.
- And is then used to vaporize the warmer water from the condenser, **C**.
- That water forms another, separate loop circulating from intake via the condenser to the cooling tower, **CT**.
- It is used to cool some steam or warm[er] water having been utilized in the turbine above, **T**.
- A third, separate loop of water circulates between the three: the steam generator, **SG**, the turbine, **T**, and the condenser, **C**.
- Figure P.2 on the facing page does not show how the water in the leftmost and middle loops enter and leave the system.



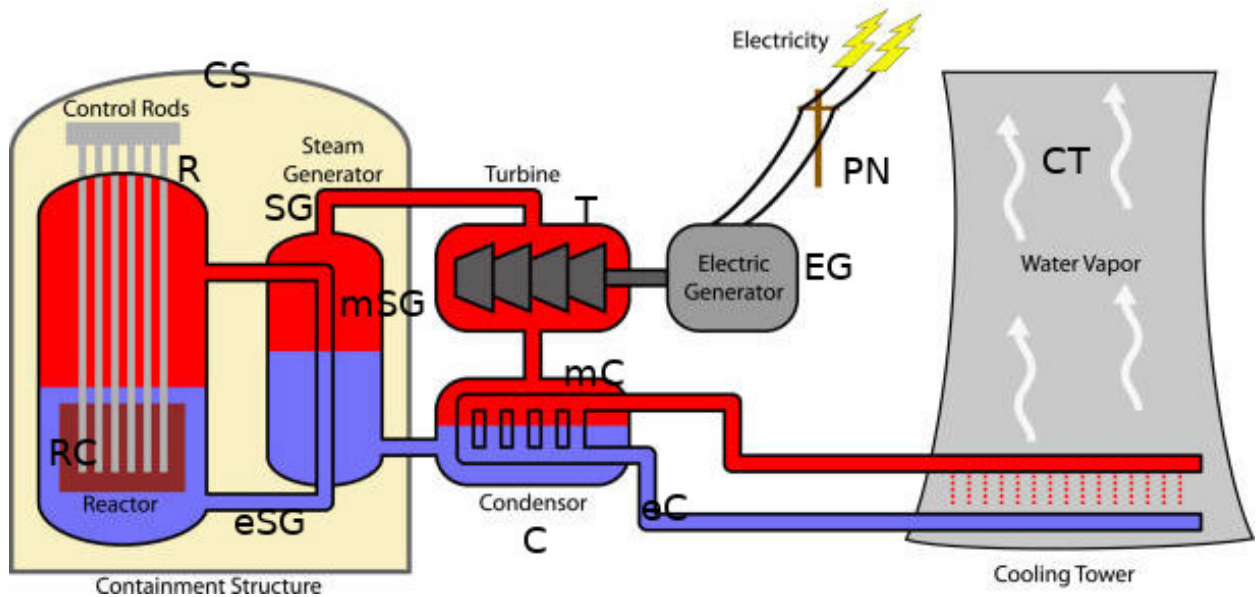


Figure P.2: Components and Flow of a Nuclear Power Plant

- Nor does Fig. P.2 show temperature, radiation, pressure, nor other sensors, pumps, ...**Pump**, and other actuators nor possible pressure tanks, **PT**.<sup>2</sup>
- The water of the leftmost loop is heated up, in the reactor by the uranium rods.
- The the [typically boiling] hot water is used in the steam generator to heat water of the middle, the center loop into steam.
- That steam drives the turbine –
- which generates electricity
- that is transmitted into the power net.

### P.3 Sketch of a Conceptual Domain Model

We follow the sequence of analyzing, describing and presenting a domain model as outlined in [58,61].

#### P.3.1 Endurants

##### P.3.1.1 External Qualities

###### P.3.1.1.1 The Parts

732. The domain is that of pressurized water nuclear power plants.

From a nuclear power plant we can observe

733. a cooler water supply and condenser compound, CWS\_C, compound;

734. a steam generator, turbine and condenser, SG\_T\_C, compound;

<sup>2</sup>Figure P.3 on page 462 shows some of these.

735. a reactor and steam generator containment, R\_SG, compound and

736. an electricity generator and power net, EG\_PN compound.

**type**

732. NPP  
733. CWS\_C  
734. SG\_T\_C  
735. R\_SG  
736. EG\_PN

**value**

733. obs\_CWS\_C: NPP → CWS\_C  
734. obs\_SG\_T\_C: NPP → SG\_T\_C  
735. obs\_R\_SG: NPP → R\_SG  
736. obs\_EG\_PN: NPP → EG\_PN

737. From a cooler water supply and condenser, CWS\_C, compound one can observe its part of the cooling tower, CT, and its likewise embedding in the condenser, eC.

738. From a steam generator, turbine and condenser, SG\_T\_C, compound one can observe its steam generator embedding, eSG, in the containment compound, the turbine compound, TC, and the main condenser compound, mC.

739. From a reactor and steam generator containment compound, R\_SG, one can observe the reactor compound, R, and the [main] steam generator compound, mSG.

740. From the electricity generator and power net, EG\_PN, one can observe the [main] electricity generator compound, mEG, and the power net, PN.

**type**

737. CT, eC  
738. eSG, TC, mC  
739. R, SG  
740. mEG, PN

738. obs\_eSG: SG\_T\_C → eSG

738. obs\_TC: SG\_T\_C → TC

738. obs\_mC: SG\_T\_C → mC

739. obs\_R: R\_SG → R

739. obs\_SG: R\_SG → mSG

740. obs\_mEG: EG\_PN → mEG

740. obs\_PN: EG\_PN → PN

**value**

737. obs\_CT: CWS\_C → CT

737. obs\_eC: CWS\_C → eC

The triplet *steam generator, turbine compound and condenser compound* for the so-called Rankine Cycle<sup>3</sup>

741. From the reactor we can observe the reactor core and the reactor rod compound. [We can also observe the reactor water and the reactor liquid; see below.]

742. The reactor rod compound is a [definite] set of rod-machines.

743. From a rod-machine we can observe a rod and a machine (to move the rod in and out of the reactor core).

Rods and their machines are considered atomic.

**type**

741. RC, RRC, RW, RL  
742. RMs = RM-set, RM  
743. ROD, RMC

741. obs\_RRC: R → RRC

741. obs\_RW: R → RW

741. obs\_RL: R → RL

742. obs\_RMs: RRC → RMs

743. obs\_ROD: RM → ROD

743. obs\_RMC: RM → RMC

**value**

741. obs\_RC: R → RC

<sup>3</sup>The Rankine cycle, also called the Rankine vapor cycle, is a thermodynamic cycle that converts heat into mechanical energy. The Rankine cycle is name after William Johnson Macquorn Rankine, a 19th century Scottish engineer and physicist known for his research in the thermodynamic properties of steam [<https://www.techtarget.com/whatis/definition/Rankine-cycle>].

### P.3.1.1.2 Liquids

744. From the *cooling tower*<sup>4</sup>, *embedded* and *main condenser, turbine compound, embedded* and *main steam generator* and *reactor* we can observe their water [contents].

745. From the *reactor* we can observe the reactor *liquid*.

#### type

744. W

745. L

#### value

744.  $\text{obs\_W}: (\text{CT}|\text{eC}|\text{mC}|\text{TC}|\text{eSG}|\text{mSG}|\text{R}) \rightarrow \text{W}$

745.  $\text{obs\_L}: \text{R} \rightarrow \text{L}$

**P.3.1.1.3 States, I** Based on the above we can define a state notion – as the set of all compound and atomic parts, and, deviating from the “edicts” of [58, 61], also the set of waters contained in the various parts!

#### value

$npp:\text{NPP} = [\text{some } npp:\text{NPP}]$

$cws\_c:\text{CWS\_C} = \text{obs\_CWS\_C}(npp)$

$sg\_t\_c:\text{SG\_T\_C} = \text{obs\_SG\_T\_C}(npp)$

$r\_sg:\text{R\_SG} = \text{obs\_R\_SG}(npp)$

$eg\_pn:\text{EG\_PN} = \text{obs\_EG\_PN}(npp)$

$ct:\text{CT} = \text{obs\_CT}(cws\_c)$

$ec:\text{eC} = \text{obs\_eC}(cws\_c)$

$esg:\text{eSG} = \text{obs\_eSG}(sg\_t\_c)$

$tc:\text{TC} = \text{obs\_TC}(sg\_t\_c)$

$mc:\text{mC} = \text{obs\_mC}(sg\_t\_c)$

$r:\text{R} = \text{obs\_R}(r\_sg)$

$sg:\text{SG} = \text{obs\_SG}(r\_sg)$

$meg:\text{mEG} = \text{obs\_mEG}(eg\_pn)$

$pn:\text{PN} = \text{obs\_PN}(eg\_pn)$

$rc: = \text{obs\_RC}(r)$

$rrc: = \text{obs\_RRC}(r)$

$rms: = \text{obs\_RMs}(r)$

#### value

$(ctw, ecw, mcw, tw, esgw, msgw, rw)$

$= (\text{obs\_W}(ct), \text{obs\_W}(ec), \text{obs\_W}(mc), \text{obs\_W}(tc), \text{obs\_W}(esg), \text{obs\_W}(meg), \text{obs\_W}(r))$

$\sigma' = \{npp, cws\_c, sg\_t\_c, r\_sg, eg\_pn, ct, ec, esg, tc, mc, r, sg, meg, pn, rc, rrc, rw, rms\}$

We do not include the “water states” in the state  $\sigma'$ , as we do not associate unique identifiers [nor mereologies, but only attributes] with water states. That is, we do not model waters as behaviours. We prime the  $\sigma'$  as we shall later need adjust it.

**P.3.1.1.4 A First Review.** It is high time to consider the whole of a pressured water nuclear reactor in further detail, as to its various measuring, i.e., sensor, and control, i.e., actuator devices. See Fig. P.3 on the following page.

We observe then that the various water holding compounds are suitably connected by water pipes, and we observe three water pumps, actuators, for some of these pipes. There are undoubtedly many more.

We can also envisage a number water pressure sensors, water temperature sensors, radiation sensors, et cetera,

We then observe a pressure tank compound.

All of these, and many more, need be included in the **external qualities** description.

<sup>4</sup>We now analyze the waters right-to-left in Fig. P.2 on page 459

## PRESSURIZED WATER REACTOR (PWR)

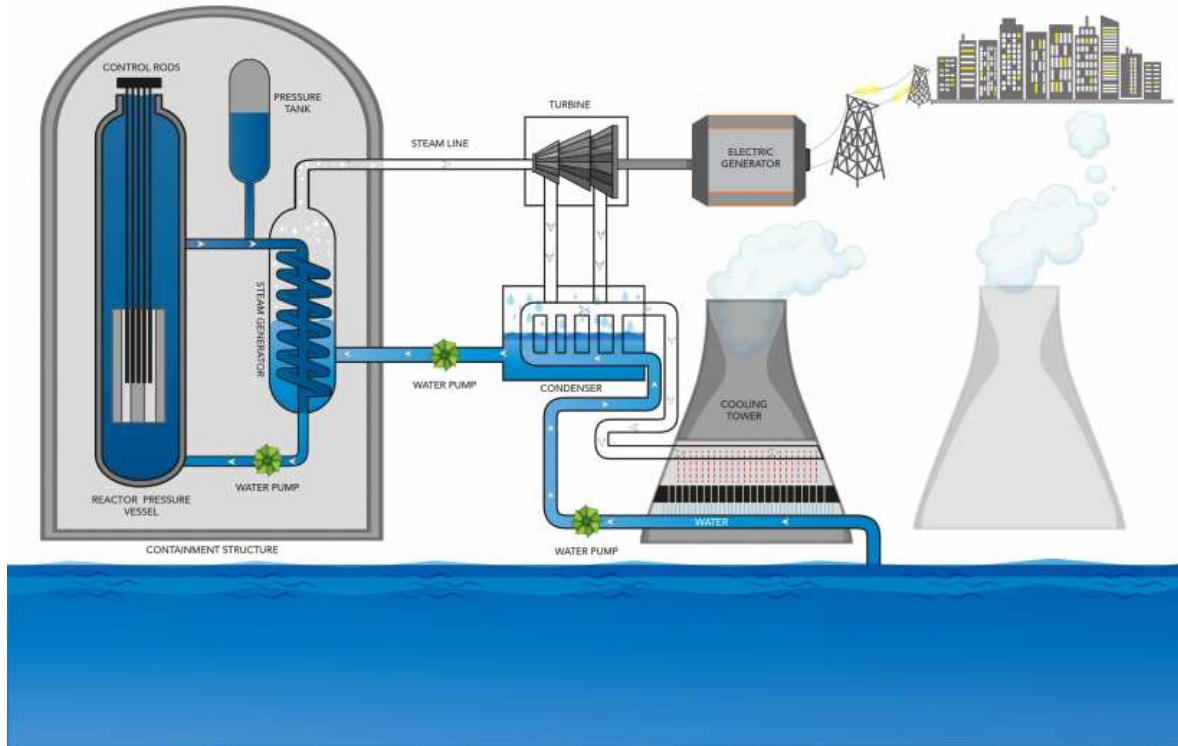


Figure P.3: Components and Flow of a Nuclear Power Plant

**P.3.1.1.5 Additional Parts** We now include some further parts to the domain. We model them on par, i.e., as being on the same “level” as major parts of the nuclear power plant, NPP. We refer to Fig. P.3.

746. From the *Cooler Water Supply and Cooler*, CWS\_C, we observe a pump/valve, CWS\_Pump, placed on the, or a, pipe between the CWS and the external cool water supply (not formalized (!)).<sup>5</sup>
747. From the *Cooling Tower and Condenser*, CT\_C, we observe a pump/valve, CT\_C\_Pump, placed on the, or a, pipe between the *Cooling Tower*, CT, and the *Condenser*, C.<sup>6</sup>
748. From the *Steam Generator, Electric Generator and Condenser* compound we can observe a *pump*, SG\_EG\_C\_Pump, placed on the, or a, pipe between the *condenser*, C, and the *steam generator*, SG.
749. From the *Reactor Steam Generator* compound we can observe a pump, SG\_R\_Pump, placed between the steam generator, SG, and a reactor inlet, R.
750. And from the *Reactor Steam Generator* “pair” we can also observe a *pressure tank*, PT placed on the, or a, pipe between a *reactor* outlet and a *turbine* inlet.

<sup>5</sup>‘Placement’ will be modelled by a suitable mereology.

<sup>6</sup>‘Placement’ will be modelled by a suitable mereology.

**type**

746. CWS\_Pump, CT\_C\_Pump

748. SG\_EG\_C\_Pump

749. SG\_R\_Pump

750. PT

**value**746. obs\_CWS\_Pump: CWS\_C  $\rightarrow$  CWS\_Pump746. obs\_CT\_C\_Pump: CWS\_C  $\rightarrow$  CT\_C\_Pump748. obs\_SG\_EG\_C\_Pump: SG\_EG\_C  $\rightarrow$  SG\_EG\_C\_Pump749. obs\_SG\_R\_Pump: R\_SG  $\rightarrow$  SG\_R\_Pump750. obs\_PT: R\_SG  $\rightarrow$  PT

We do not model thermometers, pressure, radiation nor other meters. Their values are here modelled in terms of attributes, see Items 777– 780 on page 472, of appropriate parts.

**P.3.1.1.6 A State Update.** We adjust the state definition of above.

**value***cws\_pump*: = obs\_CWS\_Pump(*cws\_c*)*ct\_c\_pump*: = obs\_CT\_C\_Pump(*cws\_c*)*sg\_eg\_c\_pump*: = obs\_SG\_EG\_C\_Pump(*sg\_eg\_c*)*sg\_r\_pump*: = obs\_SG\_R\_Pump(*r\_sg*)*press\_tank*: = obs\_PT(*r\_sg*)**value** $\sigma = \sigma' \cup \{cws\_pump, ct\_c\_pump, sg\_eg\_c\_pump, sg\_r\_pump, press\_tank\}$ 

**P.3.1.1.7 A Second Review.** What have we? Really nothing! Nothing that really tells us that we are modelling a nuclear power plant! Just a bunch of names of parts. That they actually model “real” parts of real nuclear power plants is nowhere to be seen! So what?

In order to interpret these many parts in the direction of modelling nuclear power plants we now turn to their **internal qualities**. It is only with the introduction of a suitable number of **part attributes** that the model starts resembling that of a [generic] model of nuclear power plants. To express the **part attributes** properly we first model their **unique identifiers** and **mereologies**.

**P.3.1.1.8 The Domain Taxonomy**

- **Domain Taxonomy** By a domain taxonomy we shall here understand a hierarchical presentation of the parts of a domain: their sub-parts, down to and including atomic parts

■

Figure P.4 on the next page shows a taxonomy for our domain of nuclear power plants. We have not shown atomic sensors and actuators (pumps, etc.).

The taxonomy of a specific domain is a concept quite different from the ontology of a method, cf. Fig. P.1 on page 458, for analyzing and describing that and other domains.

**P.3.1.2 Internal Qualities****P.3.1.2.1 Unique Identifiers****P.3.1.2.1.1 The Unique Identifier Type and Observers.**

751. We define the common type of unique identifiers and

752. the unique identifier observer function for each part type.

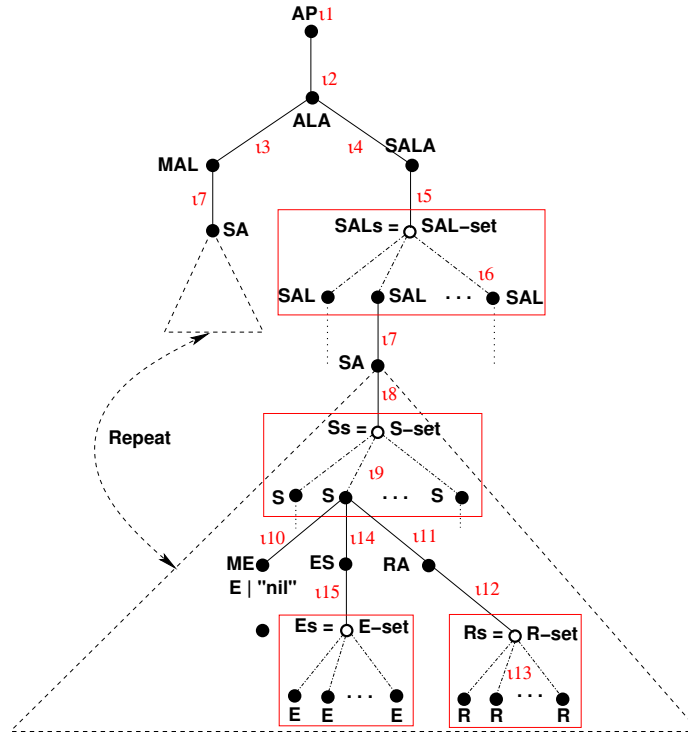


Figure P.4: The Nuclear Power Plant Domain Taxonomy

**type**

- 751. NPP\_UI, CWS\_C\_UI, SG\_T\_C\_UI, R\_SG\_UI, EG\_PN\_UI, CT\_UI, eC\_UI, eSG\_UI,
- 751. TC\_UI, mC\_UI, R\_UI, SG\_UI, mEG\_UI, PN\_UI, RC\_UI, RRC\_UI, ROD\_UI, RMC\_UI,
- 751. CWS\_Pump\_UI, CT\_C\_Pump, SG\_EG\_Pump\_UI, SG\_R\_Pump\_UI, PT\_UI
  
- 751. UI = NPP\_UI | CWS\_C\_UI | SG\_T\_UI | R\_SG\_UI | EG\_PN\_UI | CT\_UI | eC\_UI | eSG\_UI
- 751. | TC\_UI | mC\_UI | R\_UI | SG\_UI | mEG\_UI | PN\_UI | RC\_UI | RRC\_UI | ROD\_UI
- 751. | RMC\_UI | CWS\_Pump\_UI | CT\_C\_Pump | SG\_EG\_Pump\_UI | SG\_R\_Pump\_UI | PT\_UI

**value**

- 752. ui\_NPP: NPP → NPP\_UI
- 752. ui\_CWS\_C: CWS\_C → CWS\_C\_UI
- 752. ui\_SG\_T\_C: SG\_T\_C → SG\_T\_C\_UI
- 752. ui\_R\_SG: R\_SG → R\_SG\_UI
- 752. ui\_EG\_PN: EG\_PN → EG\_PN\_UI
- 752. ui\_CT: CT → CT\_UI
- 752. ui\_eC: eC → eC\_UI
- 752. ui\_eSG: eSG → eSG\_UI
- 752. ui\_TC: TC → TC\_UI
  
- 752. ui\_mC: mC → mC\_UI
- 752. ui\_R: R → R\_UI
- 752. ui\_SG: SG → SG\_UI
- 752. ui\_mEG: mEG → mEG\_UI
- 752. ui\_PN: PN → PN\_UI
- 752. ui\_RC: RC → RC\_UI
- 752. ui\_RRC: RRC → RRC\_UI
- 752. ui\_ROD: ROD → ROD\_UI
- 752. ui\_RMC: RMC → RMC\_UI
  
- 752. ui\_CWS\_Pump: CWS\_Pump → CWS\_Pump\_UI
- 752. ui\_CT\_C\_Pump: CT\_C\_Pump → CT\_C\_UI
- 752. ui\_SG\_EG\_C\_Pump: SG\_EG\_Pump → SG\_EG\_C\_Pump\_UI
- 752. ui\_SG\_R\_Pump: SG\_R\_Pump → SG\_R\_Pump\_UI
- 752. ui\_PT: PT → PT\_UI

**P.3.1.2.1.2 A State of Unique Identifiers.** We form a state of unique identifiers from the sub-states, indicated by *names*, of all parts of the power plant.

**value**

$$\begin{aligned}
 npp_{ui} &= ui\_NPP(npp) & r_{ui} &= ui\_R(r\_sg) \\
 cws\_c_{ui} &= ui\_CWS\_C(npp) & sg_{ui} &= ui\_SG(r\_sg) \\
 sg\_t\_c_{ui} &= ui\_SG\_T\_C(npp) & meg_{ui} &= ui\_mEG(eg\_pn) \\
 r\_sg_{ui} &= ui\_R\_SG(npp) & pn_{ui} &= ui\_PN(eg\_pn) \\
 eg\_pn_{ui} &= ui\_EG\_PN(npp) & & \\
 \\ 
 ct_{ui} &= ui\_CT(cws\_c) & rc_{ui} &= ui\_RC(rc) \\
 ec_{ui} &= ui\_eC(cws\_c) & rrc_{ui} &= ui\_RRC(rrc) \\
 \\ 
 esg_{ui} &= ui\_eSG(sg\_t\_c) & rod_{ui} &= ui\_ROD(r) \\
 tc_{ui} &= ui\_TC(sg\_t\_c) & rmc_{ui} &= ui\_RMC(r) \\
 mc_{ui} &= ui\_mC(sg\_t\_c) & & \\
 \\ 
 cws\_pump_{ui} &= ui\_CWS\_Pump(cws\_pump) \\
 ct\_c\_pump_{ui} &= ui\_CT\_C\_Pump(ct\_c\_pump) \\
 sg\_eg\_c\_pump_{ui} &= ui\_sg\_eg\_c\_pump(sg\_eg\_c\_pump) \\
 sg\_r\_pump_{ui} &= ui\_SG\_EG\_Pump(sg\_r\_pump) \\
 pt_{ui} &= ui\_PT(pt) \\
 \\ 
 rms_{ui} &= \{ ui\_ROD(rod), ui\_RMC(mac) \mid ((rod, mac)):RM \bullet (rod, mac) \in obs\_RMs(r) \}
 \end{aligned}$$

**value**

$$\begin{aligned}
 \sigma_{ui} &= \{ npp_{ui}, cws\_c_{ui}, sg\_t\_c_{ui}, r\_sg_{ui}, eg\_pn_{ui} \} \\
 &\cup \{ ct_{ui}, ec_{ui} \} \\
 &\cup \{ esg_{ui}, tc_{ui}, mc_{ui} \} \\
 &\cup \{ r_{ui}, sg_{ui} \} \\
 &\cup \{ meg_{ui}, pn_{ui} \} \\
 &\cup \{ rc_{ui}, rrc_{ui} \} \\
 &\cup \{ cws\_pump_{ui}, cws\_ct\_pump_{ui}, sg\_eg\_c\_pump_{ui}, sg\_r\_pump_{ui}, rms_{ui}, pt_{ui} \}
 \end{aligned}$$

**P.3.1.2.1.3 An Axiom** The unique identifiers are unique: no two parts have the same unique identifier. That means that the cardinality of state elements equals the cardinality of their unique identifiers.

$$\mathbf{axiom} \quad \mathbf{card} \sigma \equiv \mathbf{card} \sigma_{ui}$$

**P.3.1.2.2 Mereology** Mereology is the study and knowledge of relations between parts and between parts and the “whole”.

We shall model how the various power plant parts relates to one-another. A main cause of ‘relation’ is the topological layout of and [thus, intended] interaction between parts.

The topological layout was chosen by studying Figs. P.2–P.3. It is expected that that “layout” may change as we obtain deeper insight.

The below represents a first attempt at a mereology for all parts of the generic nuclear power plant. It, traditionally, focus on the mereological relations between “ancestors<sup>7</sup>, neighbours<sup>8</sup>” and “immediate sub-components”<sup>9</sup> of parts.

<sup>7</sup>If from a part of type  $A$  one can observe a part of type  $B$ , then  $A$  is an ancestor of  $B$ .

<sup>8</sup>If from a part  $a$  of type  $A$  one can observe a part  $b$  of type  $B$ , and if from  $a : A$  once can further observe parts  $c : C, d : D, \dots, e : E$ , and if parts  $c : C, d : D, \dots, e : E$  interact with  $b$  then  $c : C, d : D, \dots, e : E$  are neighbours<sup>8</sup> of  $b : B$ .

<sup>9</sup>If part  $a : A$  one can observe part of type  $b : B$ , then  $b : B$  is an immediate sub-component of  $a : A$ .

753. The mereology of NPP relates it to

- its “national nuclear regulatory commission”<sup>10</sup>, and
- its “immediate sub-components” CWS\_C, SG\_T, R\_SG and EG\_PN.

**type**

753. Mereology\_NPP = NNRC\_UI × (CWS\_C\_UI × SG\_T\_UI × R\_SG\_UI × EG\_PN\_UI)

**value**

753. mereology\_NPP: NPP → Mereology\_NPP

753. mereology\_NPP(*npp*) as (*nnrc<sub>ui</sub>*, (*cws<sub>c<sub>ui</sub></sub>*, *sg<sub>t<sub>ui</sub></sub>*, *eg<sub>pn<sub>ui</sub></sub>*))

We envisage a nuclear power plant behaviour which monitors and support controls these major components of the domain: receives, i.e., monitors, information from them and support them in the[ir] control of them.

754. The mereology of CWS\_C relates it to

- its “ancestor” NPP,
- its “neighbours” SG\_T, R\_SG and EG\_PN, and
- its “immediate sub-components” CT and eC.

**type**

754. Mereology\_CWS\_C = NPP\_UI × (SG\_TC × R\_SG\_UI × EG\_PN\_UI) × (CT\_UI × eC\_UI)

**value**

754. mereology\_CWS\_C: CWS\_C → Mereology\_CWS\_C

754. mereology\_CWS\_C(*cws<sub>c<sub>ui</sub></sub>*) as (*npp<sub>ui</sub>*, (*sg<sub>tc<sub>ui</sub></sub>*, *r<sub>sg<sub>ui</sub></sub>*, *eg<sub>pn<sub>ui</sub></sub>*), (*ct<sub>ui</sub>*, *ec<sub>ui</sub>*))

We envisage a cooler water supply and condenser behaviour which monitors and support controls its components and “reports back” to the overall monitors and support controller, the NPP.

755. The mereology of SG\_T\_C relates it to

- its “ancestor” NPP,
- its “neighbours” SG\_T, R\_SG and EG\_PN and
- its “immediate sub-components” eSG and TC.

**type**

755. Mereology\_SG\_T\_C = NPP\_UI × (SG\_T\_UI × R\_SG\_UI × EG\_PN\_UI) × (eSG\_UI × TC\_UI)

**value**

755. mereology\_SG\_T\_C: SG\_T → Mereology\_SG\_T\_C

755. mereology\_SG\_T\_C(*sg<sub>t<sub>c<sub>ui</sub></sub></sub>*) as (*npp<sub>ui</sub>*, (*sg<sub>t<sub>ui</sub></sub>*, *r<sub>sg<sub>ui</sub></sub>*, *eg<sub>pn<sub>ui</sub></sub>*), (*esg<sub>ui</sub>*, *tc<sub>ui</sub>*))

We envisage ... etc.<sup>11</sup>

756. R\_SG relates it to

- its “ancestor” NPP,
- its “neighbor” SG\_T\_C, and
- its “immediate sub-components” R and mSG.

<sup>10</sup>– omitted from this document

<sup>11</sup>The reader should, by now, be able to repeat the “we envisage” text of Items 753–754, etc.



**type**756. Mereology:  $\text{Mereo\_R\_SG} = \text{NPP} \times \text{SG\_T\_C\_UI} \times (\text{R\_UI} \times \text{MSG\_UI})$ **value**756. mereology:  $\text{R\_SG} \rightarrow \text{Mereo\_R\_SG}$ 756. mereology(*as*)  $(npp_{ui}, sg\_t\_c_{ui}, (r_{ui}, msg_{ui}))$ 

We envisage ... etc.

757. The mereology of EG\_PN relates it to

- its “*ancestor*” NPP,
- its “*neighbour*” SG\_T\_C and PN\_UI and
- its “*immediate sub-component*” T.

**type**757. Mereology:  $\text{Mereo\_EG\_PN} = \text{NPP\_UI} \times (\text{SG\_T\_C\_UI} \times \text{PN\_UI}) \times \text{T\_UI}$ **value**757. mereology:  $\text{EG\_PN} \rightarrow \text{Mereo\_EG\_PN}$ 757. mereology(*eg-pn*) *as*  $(npp_{ui}, (sg\_t\_c_{ui}, pn_{ui}), t_{ui})$ 

We envisage ... etc.

758. The mereology of CT relates it to

- its “*ancestor*” NPP, and
- its “*neighbour*” CWS\_C.

**type**758. Mereology:  $\text{Mereo\_CT} = \text{NPP\_UI} \times \text{CWS\_C\_UI}$ **value**758. mereology:  $\text{CT} \rightarrow \text{Mereo\_CT}$ 758. mereology(*ct*) *as*  $(npp_{ui}, cws\_c_{ui})$ 

We envisage ... etc.

759. The mereology of eC relates it to

- its “*ancestor*” C, and
- its “*neighbour*” mC.

**type**759. Mereology:  $\text{Mereo\_eC} = \text{C\_UI} \times \text{mC\_UI}$ **value**759. mereology:  $\text{eC} \rightarrow \text{Mereo\_eC}$ 759. mereology(*ec*) *as*  $(c_{ui}, mc_{ui})$ 

We envisage ... etc.

760. The mereology of eSG relates it to

- its “*ancestor*” SG, and
- its “*neighbours*” mSG, and R.

**type**760. Mereoe\_eSG = SG\_UI  $\times$  (mSG\_UI  $\times$  R\_UI)**value**760. mereoe\_eSG: eSG  $\rightarrow$  Mereoe\_eSG760. mereoe\_eSG(*esg*) **as** (*s<sub>gui</sub>*, (*m<sub>s<sub>gui</sub></sub>*, *r<sub>ui</sub>*))

We envisage ... estc.

761. The mereology of TC relates it to

- its “*ancestor*” SG\_T\_C, and
- its “*neighbours*” R and C.

**type**761. Mereoe\_TC = SG\_T\_C\_UI  $\times$  (R\_UI  $\times$  C\_UI)**value**761. mereoe\_TC: TC  $\rightarrow$  Mereoe\_TC761. mereoe\_TC(*tc*) **as** (*s<sub>g-r<sub>c<sub>ui</sub></sub></sub>*, (*r<sub>ui</sub>*, *c<sub>ui</sub>*))

We envisage ... etc.

762. The mereology of mC relates it to

- its “*ancestor*” C,
- its “*neighbours*” mSG, T and CT, and
- its “*immediate sub-components*” mC and eC.

**type**762. Mereoe\_mC = C\_UI  $\times$  (mSG\_UI  $\times$  T\_UI  $\times$  CT\_UI)  $\times$  (mC\_UI  $\times$  eC\_UI)**value**762. mereoe\_mC: mC  $\rightarrow$  Mereoe\_mC762. mereoe\_mC(*mc*) **as** (*c<sub>ui</sub>*, (*m<sub>s<sub>gui</sub></sub>*, *t<sub>ui</sub>*, *c<sub>ui</sub>*), (*m<sub>c<sub>ui</sub></sub>*, *e<sub>c<sub>ui</sub></sub>*))

We envisage ... etc.

763. The mereology of R relates it to

- its “*ancestor*” R\_SG,
- its “*neighbour*” SG, and
- its “*immediate sub-components*” mSG and eSG.

**type**763. Mereoe\_R = R\_SG\_UI  $\times$  SG\_UI  $\times$  (mSG\_UI  $\times$  eSG\_UI)**value**763. mereoe\_R: R  $\rightarrow$  Mereoe\_R763. mereoe\_R(*r*) **as** (*r<sub>s<sub>gui</sub></sub>*, *s<sub>gui</sub>*, (*m<sub>s<sub>gui</sub></sub>*, *e<sub>s<sub>gui</sub></sub>*))

We envisage ... etc.

764. The mereology of SG relates it to

- its “*ancestor*” R\_SG,
- its “*neighbours*” R and T, and

- its “*immediate sub-components*” mSG and eSG.

**type**

764. Mereology\_SG =  $R_{SG\_UI} \times (R_{UI} \times T_{UI}) \times (mSG_{UI} \times eSG_{UI})$

**value**

764. mereology\_SG: SG  $\rightarrow$  Mereology\_SG

764. mereology\_SG(*sg*) **as** (*rsg<sub>ui</sub>*, (*r<sub>ui</sub>*, *t<sub>ui</sub>*), (*msg<sub>ui</sub>*, *esg<sub>ui</sub>*))

We envisage ... etc.

765. The mereology of mEG relates it to

- its “*ancestor*” EG, and
- its “*neighbour*” PN .

**type**

765. Mereology\_mEG =  $EG_{UI} \times PN_{UI}$

**value**

765. mereology\_mEG: mEG  $\rightarrow$  Mereology\_mEG

765. mereology\_mEG(*meg*) **as** (*eg<sub>ui</sub>*, *pn<sub>ui</sub>*)

We envisage ... etc.

766. The mereology of PN relates it to

- its “*ancestor*” EG\_PN, and
- its “*neighbour*” mEG.

**type**

766. Mereology\_PN =  $EG_{PN\_UI} \times mEG_{UI}$

**value**

766. mereology\_PN: mEG  $\rightarrow$  Mereology\_mPN

766. mereology\_PN(*pn*) **as** (*eg-pn<sub>ui</sub>*, *meg<sub>ui</sub>*)

We envisage ... etc.

767. The mereology of RC relates it to

- its “*ancestor*” R, and
- its “*neighbour*” RRC.

**type**

767. Mereology\_RC =  $R_{UI} \times RRC_{UI}$

**value**

767. mereology\_RC: RC  $\rightarrow$  Mereology\_RC

767. mereology\_RC(*rc*) **as** (*r<sub>ui</sub>*, *rrc<sub>ui</sub>*)

We envisage ... etc.

768. The mereology of RRC relates it to

- its “*ancestor*” R, and
- its “*immediate sub-component*” RMs.

**type**768. Mereology:  $\text{RC\_UI} \times \text{RMs\_UI}$ **value**768. mereology:  $\text{RRC} \rightarrow \text{Mereology}$ 768. mereology(*rrc*) **as** (*rr<sub>ui</sub>*, *rms<sub>ui</sub>*)

We envisage ... etc.

769. The mereology of RMCs relates it to

- its “*ancestor*” RRC, and
- its “*immediate sub-components*” – a set of pairs of rods and rod machines: ROD and RMC.
- Such sets (rms’s) are usually order in some form of matrix (or other). We assume, in a matrix of *m* rows and *n* columns.

**type**769. Mereology:  $\text{RRC\_UI} \times ((\text{Nat} \times \text{Nat}) \times ((\text{ROD\_UI} \times \text{RMC\_UI})^*))^*$ **value**769. mereology:  $\text{RMs} \rightarrow \text{Mereology}$ 769. mereology(*rrc*) **as** (*rr<sub>ui</sub>*, (*m*, *n*), *matrix*)769. **post:** *len matrix* = *m*769.  $\wedge \forall \text{row}:(\text{ROD\_UI} \times \text{RMC\_UI})^* \bullet \text{len row} = n$ 

769. ...

where

We envisage ... etc.

770. The mereology of ROD relates it to

- its “*ancestor*” RM, and
- its “*neighbour*” RMC.

**type**770. Mereology:  $\text{RM\_UI} \times \text{RMC\_UI}$ **value**770. mereology:  $\text{ROD} \rightarrow \text{Mereology}$ 770. mereology(*rod*) = (*rm\_{ui}*, *rmc\_{ui}*)770. **pre:** ...

We envisage ... etc.

771. The mereology of RMC relates it to

- its “*ancestor*” ,
- its “*neighbours*” and , and
- its “*immediate sub-components*” and .

**type**771. Mereology:  $\text{RM\_UI} \times \text{ROD\_UI}$ **value**771. mereology:  $\text{RMC} \rightarrow \text{Mereology}$ 771. mereology(*rmc*) = (*rm\_{ui}*, *rod\_{ui}*)771. **pre:** ...

We envisage ... etc.

772. The mereology of CWS\_Pump, cf. [l746,pi462], relates it to

- its “*ancestor*” CWS\_C, and
- its “*neighbours*” CWS and ...

**type**

772. Mereology\_CWS\_Pump = CWS\_UI × ...

**value**

772. mereology\_CWS\_Pump: CWS\_Pump → Mereology\_CWS\_Pump

772. mereology\_CWS\_Pump(*cws\_pump*) **as** (*cws\_cui*, ...)

We envisage ... etc.

773. The mereology of CT\_C\_Pump, cf. [l747,pi462], relates it to

- its “*ancestor*” CWS\_C,
- its “*neighbours*” CWS and CT.

**type**

773. Mereology\_CT\_C\_Pump = CT\_C\_UI × (CT\_UI × C\_UI)

**value**

773. mereology\_CT\_C\_Pump: CT\_C\_Pump → Mereology\_CT\_C\_Pump

773. mereology\_CT\_C\_Pump() **as** (*ct\_cui*, *cui*)

We envisage ... etc.

774. The mereology of SG\_EG\_C\_Pump, cf. [l748,pi462], relates it to

- its “*ancestor*” SG\_EG\_C, and
- its “*neighbours*” C and SG.

**type**

774. Mereology\_SG\_EG\_C\_Pump = SG\_EG\_C\_UI × (C\_UI × SG\_UI)

**value**

774. mereology\_SG\_EG\_C\_Pump: SG\_EG\_C\_Pump → Mereology\_SG\_EG\_C\_Pump

774. mereology\_SG\_EG\_C\_Pump(*sg\_eg\_c*) **as** (*sg\_eg\_cui*, (*cui*, *sgui*))

We envisage ... etc.

775. The mereology of SG\_R\_Pump, cf. [l749,pi462], relates it to

- its “*ancestor*” SG\_R, and
- its “*neighbours*” and .

**type**

775. Mereology\_R\_Pump = SG\_R\_UI × (SG\_UI × R\_UI)

**value**

775. mereology\_SG\_R\_Pump: Mereology\_R\_Pump → Mereology\_SG\_R\_Pump

775. mereology\_SG\_R\_Pump(*sg\_r*) **as** (*sgui*, *ru\_i*)

We envisage ... etc.

776. The mereology of PT, cf. [l750,pi462], relates it to

- its “*ancestor*” SG, and
- its “*neighbours*” R and T.

**type**

776. Mereology\_PT = SG\_UI × (R\_UI × T\_UI)

**value**

776. mereology\_PT: PT → Mereology\_PT

776. mereology\_PT(*pt*) as (*sg<sub>ui</sub>*, (*r<sub>ui</sub>*, *t<sub>ui</sub>*))

We envisage ... etc.

The above mereology definitions represent a first attempt. As we define the various part behaviours, Sect. P.3.2.3.2 we shall be reviewing and using the above mereologies – and then we may have to adjust some.

**P.3.1.2.3 Attributes** Finally we have come to model what really should lend credence to our model as a reasonably relevant model of an, albeit “somewhat” simplified, domain of power plants. Initially it must be said that the domain analyser com describer, me, Dines Bjørner, is, obviously a novice in this area.<sup>12</sup>

**P.3.1.2.3.1 Common Part Attributes.** Common attributes across many different parts are enumerated below.

777. Flow of liquid, water or other. Can be measured at different positions of the liquid containers.<sup>13</sup>

778. Water temperature. Can be measured at different positions of the water containers.

779. Water or steam pressure. Can be measured at different positions of the water and steam containers.

780. Radiation. Can be measured at well-nigh any position in power plant.

781. ...

**type**

777. Flow = m<sup>3</sup>/sec

778. W\_Temp = Real Kelvin

779. W\_Press, S\_Press = Joule/m<sup>3</sup> = kg × m<sup>-1</sup> × sec<sup>-2</sup>

780. Radiation = Beq [Bequerel] or Cu [Curie] [1Beq = 1/sec]

**value**

777. attr\_Flow: ... → Flow

778. attr\_W\_Temp: ... → W\_Temp

779. attr\_W\_Press: ... → W\_Press

779. attr\_S\_Press: ... → S\_Press

780. attr\_Radiation: ... → Radiation

The ... above refers to the union, |, of [applicable] component types.

<sup>12</sup>I did, indeed, study nuclear physics and was around the basic engineering of nuclear power plants in my Bachelor’s study, 1958–1961, at my Alma Mater, the DTU, but from that to claim ..., no!

<sup>13</sup>We shall elaborate on *Flow* attributes in Sect. P.3.1.2.3.2 on the next page.

**P.3.1.2.3.2 Flow Attributes** There are three “isolated loops” of water flow:

- (a.) From the intake of water, CWS\_C, , through the cooling tower, CT, and the [embedded] condenser, eC, and back.
- (b.) from the main steam generator, mSG, through the turbine, T, onto the embedding, i.e., the main condenser, mC, and back to the main steam generator.
- (c.) From the reactor, R, through the [embedded] steam generator, eSG, and back.

We shall now try characterise these cold and hot water and steam flows through a number of attributes. These attributes will be “attached” to, i.e., associated with, the parts in which they occur.<sup>14</sup> These parts are:

- (a.) CWS\_C, CT, eC,
- (b.) mSG, T, mC, and
- (c.) R, eSG.

782. There is the flow, measured in *cubic meter per second*.

783. That flow can be observed at the inlet to and outlet from the eight parts listed above.

784. There is also the pressure of these flows, measured in *Joule*.

**type**

782. Flow = **Real m<sup>3</sup>/sec**

784. Joule = **Real Joule/m<sup>3</sup>** = Real kg×m<sup>-1</sup> × sec<sup>-2</sup>

**value**

783. attr\_(in|out)\_Flow\_(CWS\_C|CT|eC|mSG|mC|R|eSG) → Flow

784. attr\_(in|out)\_Pressure\_(CWS\_C|CT|eC|mSG|mC|R|eSG) → Pressure

MORE TO COME

**P.3.1.2.3.3 Pipe Attributes** To bring the water/steam from part to part these parts have pipe in- and outlets. We have not, till now, modelled these pipes. Now we shall indicate their presence – as attributes of these parts!

785. With each inlets and outlets of the parts itemized in the three •s above (CWS\_C, CT, eC, mSG, T, mC, R, and eSG), we associate a pipe of some length, diameter, and hence volume:

**type**

785. Pipe\_Len = **Real cm**

785. Pipe\_Dia = **Real cm**

785. Pipe\_Vol = **Real m<sup>3</sup>**

**value**

785. attr\_(in|out)\_let\_Pipe: (CWS\_C|CT|eC|mSG|mC|R|eSG)→(Pipe\_Len×Pipe\_Diam×Pipe\_Vol)

MORE TO COME

<sup>14</sup>You may then interpret the parts as also being *containers of water*.

**P.3.1.2.3.4 Specific Part Attributes** We list remaining attribute definitions by part.

**1. The Universe of Discourse, NPP, Items 732 [π459], 751 [π463], 753 [π466]**

786.

787.

788.

789.

786.

787.

788.

789.

**2. Cooler Water Supply - Condenser, CWS\_C, Items 733 [π459], 751 [π463], 754 [π466]**

790.

791.

792.

793.

790.

791.

792.

793.

**3. Steam Generator - Turbine - Condenser, SG\_T\_C, Items 734 [π459], 751 [π463], 755 [π466]**

794.

795.

796.

797.

794.

795.

796.

797.

**4. Reactor - Steam Generator R\_SG, Items 735 [π460], 751 [π463], 756 [π466]**

798.

799.

800.

801.

798.

799.

800.

801.



**5. Electricity Generator - Power Net, EG\_PN, Items 736 [π460], 751 [π463], 757 [π467]**

802.

803.

804.

805.

802.

803.

804.

805.

**6. Cooling Tower, CT, Items 737 [π460], 751 [π463], 758 [π467]**

806.

807.

808.

809.

806.

807.

808.

809.

**7. Embedded Condenser, eC, Items 737 [π460], 751 [π463], 759 [π467]**

810.

811.

812.

813.

810.

811.

812.

813.

**8. Embedded Steam Generator, eSG, Items 738 [π460], 751 [π463], 760 [π467]**

814.

815.

816.

817.

814.

815.

816.

817.

**9. Turbine Compound, TC, Items 738 [π460], 751 [π463], 761 [π468]**

818.

819.

820.

821.

818.

819.

820.

821.

**10. Main Condenser, mC, Items 738 [π460], 751 [π463], 762 [π468]**

822.

823.

824.

825.

822.

823.

824.

825.

**11. Reactor, R, Items 739 [π460], 751 [π463], 763 [π468]**

826.

827.

828.

829.

826.

827.

828.

829.

**12. Steam Generator, SG, Items 739 [π460], 751 [π463], 764 [π468]**

830.

831.

832.

833.

830.

831.

832.

833.

**13. Main Electricity Generator, mEG, Items 739 [π460], 751 [π463], 765 [π469]**

834.

835.

836.

837.

834.

835.

836.

837.

**14. Power Net, PN, Items 739 [π460], 751 [π463], 766 [π469]**

838.

839.

840.

841.

838.

839.

840.

841.

**15. Reactor Core, RC, Items 741 [π460], 751 [π463], 767 [π469]**

842.

843.

844.

845.

842.

843.

844.

845.

**16. Reactor Rod Compound, RRC, Items 741 [π460], 751 [π463], 768 [π469]**

846.

847.

848.

849.

846.

847.

848.

849.

**17. Reactor Rod Machine Set, RMCs, Items 742 [π460], 751 [π463], 769 [π470]**

850.

851.

852.

853.

850.

851.

852.

853.

**18. Rod, ROD, Items 743 [π460], 751 [π463], 770 [π470]**

854.

855.

856.

857.

854.

855.

856.

857.

**19. Rod Machine, RMC, Items 743 [π460], 751 [π463], 771 [π470]**

858.

859.

860.

861.

858.

859.

860.

861.

**20. CWS Pump, CWS\_Pump, Items 743 [π460], 751 [π463], 772 [π471]**

862.

863.

864.

865.

862.

863.

864.

865.

**21. CT\_C Pump, CT\_C\_Pump, Items 743 [π460], 751 [π463], 773 [π471]**

866.

867.

868.

869.

866.

867.

868.

869.

**22. SG\_EG\_C Pump, SG\_EG\_C\_Pump, Items 743 [π460], 751 [π463], 774 [π471]**

870.

871.

872.

873.

870.

871.

872.

873.

**23. Pressure Tank, PT, Items 743 [π460], 751 [π463], 776 [π472]**

874.

875.

876.

877.

874.

875.

876.

877.

**P.3.1.2.4 Intentional Pull** This concept has yet to be investigated. My hunch is that the relation between the reactor, the steam generator and the turbine includes an intentional pull. And also the relation between the positions of the rod with respect to the reactor core and the temperature of the reactor water.

### P.3.2 Perdurants

In this section we shall treat the *methodological phases, stages and steps of analyzing and describing perdurants* from *one specific perspective*, namely that of *interpreting*, i.e., *transcendentally deducing*, endurants into perdurants, where these perdurants reflect on the, as one could claim, *basic intentions* of the domain being modelled.

This may sound cryptic. The main example in this report is that of a [generic, conceptual class of] nuclear power plants. We therefore claim, or postulate, that the *basic intentions* of the modelling so far has been to model the domain of stably operating nuclear power plants.

The perdurants, therefore, of this major section, are therefor to model actions, events and behaviours of normally operating nuclear power plants.

Section P.4 shall then deal with a number of “derivative” transcendental deductions.

#### P.3.2.1 Channels

878. We, somewhat superfluously, but for simplicity let channel indices range over any set of two [thus distinct] unique identifiers of  $\sigma_{uid}$ .

879. These channels communicate messages of type M, where the definition of their contribution to M shall evolve as we identify and define nuclear power plant behaviours.

The mereologies of the individual behaviours will, anyway, identity applicable such unique identifiers.

#### channel

878.  $\{ ch[\{ui,uj\}] \mid ui,ij:UI \bullet \{ui,ij\} \subset \sigma_{ui} \wedge ui \neq uj \} : M$

#### type

879.  $M = \dots$

#### P.3.2.2 Actions

We informally identify and briefly sketch part behaviours.

880. NPP: gather attribute values ...; direct “immediate” NPP components to do one or another action; ... .

881. CWS\_C: gather attribute values ...; inform NPP of [a summary of] these values; direct “immediate” CWS\_C components to do one or another action; ... .

882. SG\_T\_C:

883. R\_SG:

884. EG\_PN:

885. CT:

886. eC:

887. eSG:

888. TC:

889. mC:

890. R:

891. SG:

- 892. mEG:
- 893. PN:
- 894. RC:
- 895. RMs:
- 896. RM:
- 897. ROD:
- 898. RMC: start raising, respectively lowering and stop raising, respectively lowing of rod; ...
- 899. CWS\_Pump: start, respectively stop of rod pump; ...
- 900. CT\_C\_Pump: start, respectively stop of rod pump; ...
- 901. SG\_EG\_C\_Pump: start, respectively stop of rod pump; ...
- 902. PT: start, respectively stop of pressuring; ...

### P.3.2.3 Behaviours

We now associate behaviours with parts.

There are basically two kinds of part behaviours.

**Active Behaviours** are those which, own their own initiative (for example as prompted by behavioural events “of their parts”), or as directed by monitoring behaviours, initiate actions.

**Passive Behaviours** are those which do not initiate actions, but [usually] respond to interaction with [mereology-]connected behaviours.

#### P.3.2.3.1 Behaviour Signatures

- 903.
- 904.
- 905.
- 906.
- 907.
- 908.
- 909.
- 910.
- 911.
- 912.
- 913.
- 914.
- 915.
- 916.
- 917.

918.

919.

920.

921.

922.

923.

924.

925.

926.

927.

**value**

904. behaviour\_NPP: NPP\_UI×Mereo\_NPP×StaV\_NPP×MonV\_NPP  
→ ProV\_NPP → chs\_NPP **Unit**
905. behaviour\_CWS\_C: CWS\_C\_UI×Mereo\_CWS\_C×StaV\_CWS\_C×MonV\_CWS\_C  
→ ProV\_CWS\_C → chs\_CWS\_C **Unit**
906. behaviour\_SG\_T\_C: SG\_T\_C\_UI×Mereo\_SG\_T\_C×StaV\_SG\_T\_C×MonV\_SG\_T\_C  
→ SG\_T\_C\_NPP → chs\_SG\_T\_C **Unit**
907. behaviour\_R\_SG: R\_SG\_UI×Mereo\_R\_SG×StaV\_R\_SG×MonV\_R\_SG  
→ ProV\_R\_SG → chs\_R\_SG **Unit**
908. behaviour\_EG\_PN: EG\_PN\_UI×Mereo\_EG\_PN×StaV\_EG\_PN×MonV\_EG\_PN  
→ ProV\_EG\_PN → chs\_EG\_PN **Unit**
909. behaviour\_CT: \_CT\_UI×Mereo\_CT×StaV\_CT×MonV\_CT  
→ ProV\_CT → chs\_CT **Unit**
910. behaviour\_eC: eC\_UI×Mereo\_eC×StaV\_eC×MonV\_eC  
→ ProV\_eC → chs\_eC **Unit**
911. behaviour\_eSG: eSG\_UI×Mereo\_eSG×StaV\_eSG×MonV\_eSG  
→ ProV\_eSG → chs\_eSG **Unit**
912. behaviour\_TC: TC\_UI×Mereo\_TC×StaV\_TC×MonV\_TC  
→ ProV\_TC → chs\_TC **Unit**
913. behaviour\_mC: mC\_UI×Mereo\_mC×StaV\_mC×MonV\_mC  
→ ProV\_mC → chs\_mC **Unit**
914. behaviour\_R: R\_UI×Mereo\_R×StaV\_R×MonV\_R  
→ ProV\_R → chs\_R **Unit**
915. behaviour\_SG: SG\_UI×Mereo\_SG×StaV\_SG×MonV\_SG  
→ ProV\_SG → chs\_SG **Unit**
916. behaviour\_mEG: mEG\_UI×Mereo\_mEG×StaV\_mEG×MonV\_mEG  
→ ProV\_mEG → chs\_mEG **Unit**
917. behaviour\_PN: PN\_UI×Mereo\_PN×StaV\_PN×MonV\_PN  
→ ProV\_PN → chs\_PN **Unit**
918. behaviour\_RC: \_RC\_UI×Mereo\_RC×StaV\_RC×MonV\_RC  
→ ProV\_RC → chs\_RC **Unit**
919. behaviour\_RRC: RRC\_UI×Mereo\_RRC×StaV\_RRC×MonV\_RRC  
→ ProV\_RRC → chs\_RRC **Unit**
920. behaviour\_RMs: RMs\_UI×Mereo\_RMs×StaV\_RMs×MonV\_RMs  
→ ProV\_RMs → chs\_RMs **Unit**
921. behaviour\_RM: RM\_UI×Mereo\_RM×StaV\_RM×MonV\_RM  
→ ProV\_RM → chs\_RM **Unit**
922. behaviour\_ROD: ROD\_UI×Mereo\_ROD×StaV\_ROD×MonV\_ROD  
→ ProV\_ROD → chs\_ROD **Unit**
923. behaviour\_RMC: RMC\_UI×Mereo\_RMC×StaV\_RMC×MonV\_RMC  
→ ProV\_RMC → chs\_RMC **Unit**
924. behaviour\_CWS\_Pump: CWS\_Pump\_UI×Mereo\_CWS\_Pump×StaV\_CWS\_Pump×MonV\_CWS\_Pump  
→ ProV\_CWS\_Pump → chs\_CWS\_Pump **Unit**
925. behaviour\_CWS\_CT\_Pump: CWS\_CT\_Pump\_UI×Mereo\_CWS\_CT\_Pump×StaV\_CWS\_CT\_Pump×MonV\_CWS\_CT\_Pump  
→ ProV\_CWS\_CT\_Pump → chs\_CWS\_CT\_Pump **Unit**
926. behaviour\_SG\_EG\_Pump: SG\_EG\_Pump\_UI×Mereo\_SG\_EG\_Pump×StaV\_SG\_EG\_Pump×MonV\_SG\_EG\_Pump  
→ ProV\_SG\_EG\_Pump → chs\_SG\_EG\_Pump **Unit**



927. behaviour\_PT: PT\_UI×Mereo\_PT×StaV\_PT×MonV\_PT  
 → ProV\_PT → chs\_PTC Unit

**P.3.2.3.2 Behaviour Definitions** We list all the definitions of behaviours.

**1. The Universe of Discourse, NPP, Items 732 [π459], 751 [π463], 753 [π466]**

928.

929.

930.

931.

928.

929.

930.

931.

**2. Coler Water Supply - Condenser, CWS\_C, Items 733 [π459], 751 [π463], 754 [π466]**

932.

933.

934.

935.

932.

933.

934.

935.

**3. Generator - Turbine - Condenser, SG\_T\_C, Items 734 [π459], 751 [π463], 755 [π466]**

936.

937.

938.

939.

936.

937.

938.

939.

**4. Reactor - Generator R\_SG, Items 735 [π460], 751 [π463], 756 [π466]**

940.

941.

942.

943.

940.  
941.  
942.  
943.

**5. Electricity Generator - Power Net, EG\_PN, Items 736 [π460], 751 [π463], 757 [π467]**

944.  
945.  
946.  
947.

944.  
945.  
946.  
947.

**6. Cooling Tower, CT, Items 737 [π460], 751 [π463], 758 [π467]**

948.  
949.  
950.  
951.

948.  
949.  
950.  
951.

**7. Embedded Condenser, eC, Items 737 [π460], 751 [π463], 759 [π467]**

952.  
953.  
954.  
955.

952.  
953.  
954.  
955.

**8. Embedded Steam Generator, eSG, Items 738 [π460], 751 [π463], 760 [π467]**

956.  
957.  
958.  
959.

- 956.
- 957.
- 958.
- 959.

**9. Turbine Compound, TC, Items 738 [π460], 751 [π463], 761 [π468]**

- 960.
- 961.
- 962.
- 963.

- 960.
- 961.
- 962.
- 963.

**10. Main Condenser, mC, Items 738 [π460], 751 [π463], 762 [π468]**

- 964.
- 965.
- 966.
- 967.

- 964.
- 965.
- 966.
- 967.

**11. Reactor, R, Items 739 [π460], 751 [π463], 763 [π468]**

- 968.
- 969.
- 970.
- 971.

- 968.
- 969.
- 970.
- 971.

**12. Steam Generator, SG, Items 739 [π460], 751 [π463], 764 [π468]**

- 972.
- 973.
- 974.
- 975.

972.  
973.  
974.  
975.

**13. Main Electricity Generator, mEG, Items 739 [π460], 751 [π463], 765 [π469]**

976.  
977.  
978.  
979.

976.  
977.  
978.  
979.

**14. Power Net, PN, Items 739 [π460], 751 [π463], 766 [π469]**

980.  
981.  
982.  
983.

980.  
981.  
982.  
983.

**15. Reactor Core, RC, Items 741 [π460], 751 [π463], 767 [π469]**

984.  
985.  
986.  
987.

984.  
985.  
986.  
987.

**16. Reactor Rod, RRC, Items 741 [π460], 751 [π463], 768 [π469]**

988.  
989.  
990.  
991.

988.  
989.  
990.  
991.

**17. Reactor Rod Machine Set, RMCs, Items 742 [π460], 751 [π463], 769 [π470]**

992.  
993.  
994.  
995.

992.  
993.  
994.  
995.

**18. Rod, ROD, Items 743 [π460], 751 [π463], 770 [π470]**

996.  
997.  
998.  
999.

996.  
997.  
998.  
999.

**19. Rod Machine, RMC, Items 743 [π460], 751 [π463], 771 [π470]**

1000.  
1001.  
1002.  
1003.

1000.  
1001.  
1002.  
1003.

**20. CWS Pump, CWS\_Pump, Items 743 [π460], 751 [π463], 772 [π471]**

1004.  
1005.  
1006.  
1007.

1004.  
1005.  
1006.  
1007.

**21. CT\_C Pump, CT\_C\_Pump, Items 743 [π460], 751 [π463], 773 [π471]**

1008.  
1009.  
1010.  
1011.

1008.  
1009.  
1010.  
1011.

**22. SG\_EG\_C Pump, SG\_EG\_C\_Pump, Items 743 [π460], 751 [π463], 774 [π471]**

1012.  
1013.  
1014.  
1015.

1012.  
1013.  
1014.  
1015.

**23. Pressure Tank, PT, Items 743 [π460], 751 [π463], 776 [π472]**

1016.  
1017.  
1018.  
1019.

1016.  
1017.  
1018.  
1019.

**P.3.2.4 Action Signatures and Definitions**

**P.3.2.4.1 Action Signatures**

**P.3.2.4.2 Action Definitions**

**P.3.2.5 Domain Initialization**

The domain behaviour is the parallel composition of the following behaviours, the

|                                             |                                   |                                |
|---------------------------------------------|-----------------------------------|--------------------------------|
| 1020. nuclear power plant,                  | 1027. embedded steam generator,   | 1035. reactor rod compound,    |
| 1021. cooler water supply condenser,        | 1028. turbine,                    | 1036. reactor rod machine set, |
| 1022. electricity generator turbine cooler, | 1029. main condenser,             | 1037. set of reactor rod,      |
| 1023. reactor generator,                    | 1030. reactor,                    | 1038. set of reactor machines, |
| 1024. generator power net,                  | 1031. steam generator,            | 1039. ...                      |
| 1025. cooling tower,                        | 1032. main electricity generator, | 1040. ...                      |
| 1026. embedded condenser,                   | 1033. power net,                  | 1041. ...                      |
|                                             | 1034. reactor core,               |                                |

In order to express the static, the monitorable and the programmable attributes we make use of the:

- `static_attribute_names` [stavs],
- `monitorable_attribute_names` [monvs] and
- `programmable_attribute_names` [prgvs]

auxiliary domain analysis functions defined in [61, Sect. 5.4.2.5]. We refer to them by the abbreviations given above. in brackets.

|       |                                                                                                                                                   |
|-------|---------------------------------------------------------------------------------------------------------------------------------------------------|
| 1020. | <code>behaviour_NPP(ui_NPP(<i>npp</i>),mereo_NPP(<i>npp</i>),stavs(<i>npp</i>),monvs(<i>npp</i>))(prgvs(<i>npp</i>))</code>                       |
| 1021. | <code>   behaviour_CWS_C(ui_CWS_C(<i>cws_c</i>),mereo_CWS_C(<i>cws_c</i>),stavs(<i>cws_c</i>),monvs(<i>cws_c</i>))(prgvs(<i>cws_c</i>))</code>    |
| 1022. | <code>   behaviour_SG_T_C(ui_SG_T_C(<i>sg_tc</i>),mereo_SG_T_C(<i>sg_tc</i>),stavs(<i>sg_tc</i>),monvs(<i>sg_tc</i>))(prgvs(<i>sg_tc</i>))</code> |
| 1023. | <code>   behaviour_R_SG(ui_R_SG(<i>r_sg</i>),mereo_R_SG(<i>r_sg</i>),stavs(<i>r_sg</i>),monvs(<i>r_sg</i>))(prgvs(<i>r_sg</i>))</code>            |
| 1024. | <code>   behaviour_EG_PN(ui_EG_PN(<i>eg_pn</i>),mereo_EG_PN(<i>eg_pn</i>),stavs(<i>eg_pn</i>),monvs(<i>eg_pn</i>))(prgvs(<i>eg_pn</i>))</code>    |
| 1025. | <code>   behaviour_CT(ui_CT(<i>ct</i>),mereo_CT(<i>ct</i>),stavs(<i>ct</i>),monvs(<i>ct</i>))(prgvs(<i>ct</i>))</code>                            |
| 1026. | <code>   behaviour_eC(ui_eC(<i>ec</i>),mereo_eC(<i>ec</i>),stavs(<i>ec</i>),monvs(<i>ec</i>))(prgvs(<i>ec</i>))</code>                            |
| 1027. | <code>   behaviour_eSG(ui_eSG(<i>esg</i>),mereo_eSG(<i>esg</i>),stavs(<i>esg</i>),monvs(<i>esg</i>))(prgvs(<i>esg</i>))</code>                    |
| 1028. | <code>   behaviour_TC(ui_TC(<i>tc</i>),mereo_TC(<i>tc</i>),stavs(<i>tc</i>),monvs(<i>tc</i>))(prgvs(<i>tc</i>))</code>                            |
| 1029. | <code>   behaviour_mC(ui_mC(<i>mc</i>),mereo_mC(<i>mc</i>),stavs(<i>mc</i>),monvs(<i>mc</i>))(prgvs(<i>mc</i>))</code>                            |
| 1030. | <code>   behaviour_R(ui_R(<i>r</i>),mereo_R(<i>r</i>),stavs(<i>r</i>),monvs(<i>r</i>))(prgvs(<i>r</i>))</code>                                    |
| 1031. | <code>   behaviour_SG(ui_SG(<i>sg</i>),mereo_SG(<i>sg</i>),stavs(<i>sg</i>),monvs(<i>sg</i>))(prgvs(<i>sg</i>))</code>                            |
| 1032. | <code>   behaviour_mEG(ui_mEG(<i>meg</i>),mereo_mEG(<i>meg</i>),stavs(<i>meg</i>),monvs(<i>meg</i>))(prgvs(<i>meg</i>))</code>                    |
| 1033. | <code>   behaviour_PN(ui_PN(<i>pn</i>),mereo_PN(<i>pn</i>),stavs(<i>pn</i>),monvs(<i>pn</i>))(prgvs(<i>pn</i>))</code>                            |
| 1034. | <code>   behaviour_RRC(ui_RRC(<i>rrc</i>),mereo_RRC(<i>rrc</i>),stavs(<i>rrc</i>),monvs(<i>rrc</i>))(prgvs(<i>rrc</i>))</code>                    |
| 1035. | <code>   behaviour_RMs(ui_RMs(<i>rms</i>),mereo_RMs(<i>rms</i>),stavs(<i>rms</i>),monvs(<i>rms</i>))(prgvs(<i>rms</i>))</code>                    |
| 1036. | <code>   { behaviour_ROD(ui_ROD(<i>rod</i>),mereo_ROD(<i>rod</i>),stavs(<i>rod</i>),monvs(<i>rod</i>))(prgvs(<i>rod</i>))</code>                  |
| 1036. | <code>  (<i>rod</i>,_): (ROD × RMC) • (<i>rod</i>,_) ∈ rms }</code>                                                                               |
| 1037. | <code>   { behaviour_RMC(ui_RMC(<i>rmc</i>),mereo_RMC(<i>rmc</i>),stavs(<i>rmc</i>),monvs(<i>rmc</i>))(prgvs(<i>rmc</i>))</code>                  |
| 1037. | <code>  (<i>rmc</i>,_): (ROD × RMC) • (_, <i>rmc</i>) ∈ rms }</code>                                                                              |
| 1038. | <code>   behaviour_CWS_Pump(ui_CWS_Pump(<i>cwspump</i>),mereo_CWS_Pump(<i>cwspump</i>),</code>                                                    |
| 1038. | <code>stavs(<i>cwspump</i>),monvs(<i>cwspump</i>))(prgvs(<i>cwspump</i>))</code>                                                                  |
| 1039. | <code>   behaviour_CT_C_Pump(ui_CT_C_Pump(<i>cwscpump</i>),mereo_CT_C_Pump(<i>cwscpump</i>),</code>                                               |
| 1039. | <code>stavs(<i>cwscpump</i>),monvs(<i>cwscpump</i>))(prgvs(<i>cwscpump</i>))</code>                                                               |
| 1040. | <code>   behaviour_SG_EG_Pump(ui_SG_EG_Pump(<i>sgegpump</i>),mereo_SG_EG_Pump(<i>sgegpump</i>),</code>                                            |
| 1040. | <code>stavs(<i>sgegpump</i>),monvs(<i>sgegpump</i>))(prgvs(<i>sgegpump</i>))</code>                                                               |
| 1041. | <code>   behaviour_PT(ui_PT(<i>pt</i>),mereo_PT(<i>pt</i>),</code>                                                                                |
| 1041. | <code>stavs(<i>pt</i>),monvs(<i>pt</i>))(prgvs(<i>pt</i>))</code>                                                                                 |

### P.3.2.6 Meaning of Domain Models

We have developed and presented a domain model,  $\mathbb{D}$ . We can consider this model from three points of view: (i) as a syntactic object, (ii) as a semantic object, and (iii) as a pragmatic object. (i) From the point of view of the syntactic object we can, for example, ask whether the syntax of  $\mathbb{D}$  is correct. (ii) From the point of view of the semantic object we can, for example, ask what is the meaning of  $\mathbb{D}$ . (iii) From the point of view of the syntactic object we can, for example, ask for relations between the domain  $\mathcal{D}$  and the model  $\mathbb{D}$ .

Without further elaboration we shall postulate a *meaning function*,  $\mathcal{M}$ , and claim that the meaning of  $\mathbb{D}$  is  $\mathcal{M}(\mathbb{D})$ .

## P.4 System Domains

### P.4.1 Some Preparatory Remarks

This section builds on [58, *Chapter 9, 2021*] [which is based on [43, 2011]].

The perdurants of Sect. P.3.2 reflected, we claim, on the intention of the nuclear power plant domain model: the so-called *conceptual domain model*, namely that of describing its intended, normal, state of operation. In this section we shall consider a number of, what we shall refer to as *system domains*. One could claim that these system domains are *derivative*, that is, that they derive from the conceptual domain model, when considering how the domain instances com about: are conceived, designed and built, are started-up, operated in normal, failure, repair or renewal mode, and are closed down and dismantled.

To prepare for such modelling let us first consider the possible uses of a domain model. There are basically two uses. (i) as models of how the domain behave, and (ii) as the basis for the development of software. Use (i) should need no further comments. Use (ii) need the following, long, comment: The domain model serves, now, as a basis for developing one or another kind of *requirements prescription*, from which to then further develop a *software specification*, i.e., “code” to be interpreted by a computing system.

#### P.4.1.1 The Triptych Development Dogma

Figure P.5 intends to show the main phases of the development from that of a domain description, via that of a requirements prescription, for either of three kinds of software, to either of these kinds. The idea behind Fig. P.5 is that the rounded “box” refers to the actual domain, whether existing or contemplated, that the domain description box refers to a description of that domain, that the requirements prescription boxes refer t a prescription for some desired software, and that the three rightmost boxes refer to respective, implemented softwares.

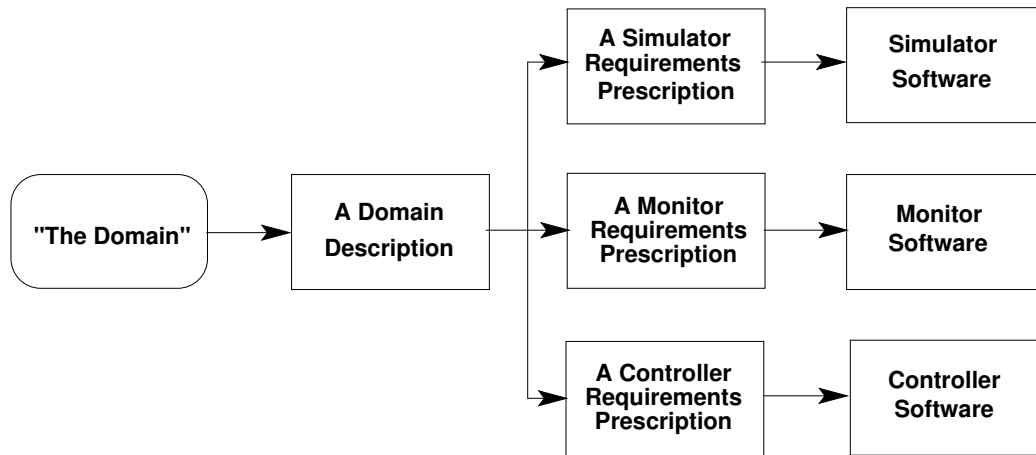


Figure P.5: The Domain, Requirements, Software Development Triptych

How the transitions from description to prescriptions are effected are covered extensively in [58, *Chapter 8, 2021*]. The transitions from prescriptions to software design & code is covered extensively in [26–28, *Software Engineering, vols.1-2-3, 2005-6*].

#### P.4.1.2 Simulators [Demos], Monitors and Controllers

A “meaning” of the concepts of *simulator*, *monitor*, and *controller* software is hinted at in Fig. P.6 on the facing page.

Figure P.6 on the next page shows three pairs, from left-to-right, of relations: a simulator [also, sometimes, called “a demo”], a monitor, and a monitor & controller pair. Each pair shows a time line, a rounded box above the time line, ostensibly referring to the actual domain [being simulated,



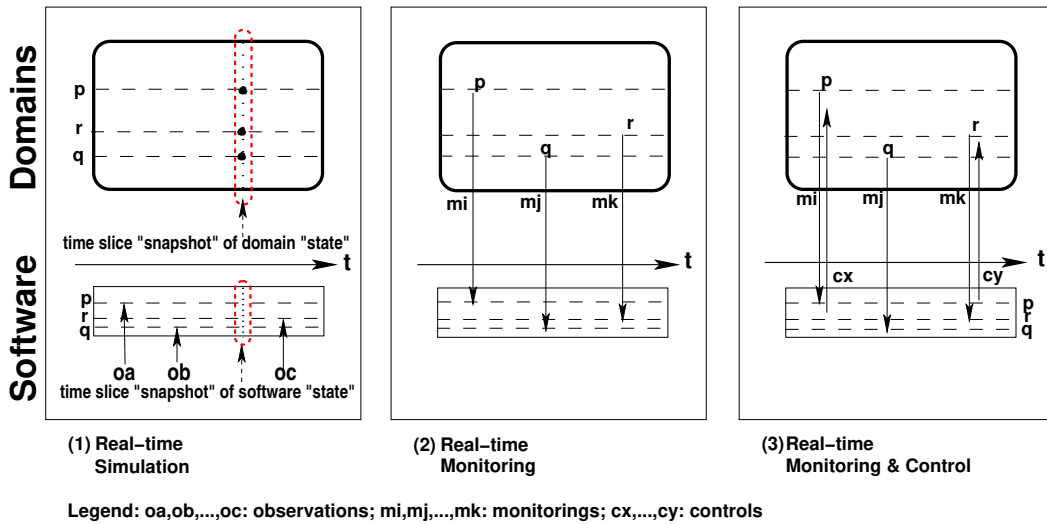


Figure P.6: Simulator, Monitor and Controller Behaviours

or monitored, or controlled], and the software [simulator, monitor, controller] below the line. Both the domain “box” and the software box are meant to show, by a vertical “slice” through these, the state of the domain behaviour, respectively of the software “under execution”, that is, of the various enduring attributes. These latter,  $p, q, r$  named, attributes are shown as horizontal lines.

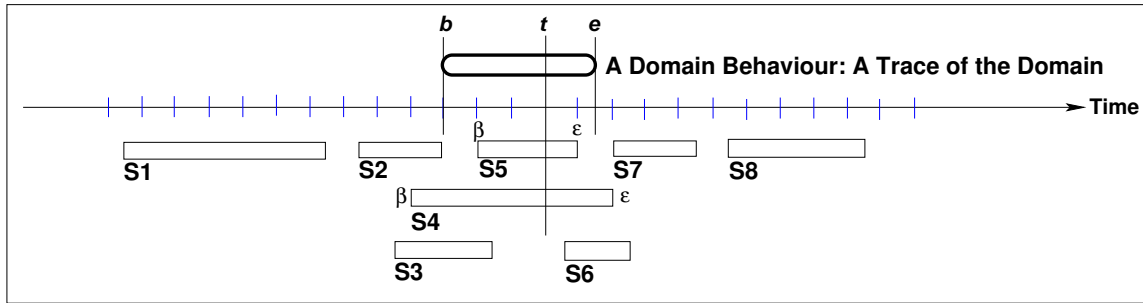
In the leftmost, the simulator, pair the domain behaviour and the software execution do not “communicate”. It is as if neither of them “exists”!

In the center, the monitor, pair the domain behaviour and the software execution the executing software “samples” (monitors) selected domain attributes. The executing software may then do whatever the requirements have otherwise prescribed.

In the rightmost, the [monitor &] controller, pair the domain behaviour and the software execution the executing software “samples” (monitors) selected domain attributes. The executing software may then react, as it is expected that the requirements have prescribed, by attempting to set, i.e., control, selected [biddable or programmable] attributes.

### P.4.1.3 Demos or Simulations

“Simulation is the imitation of some real thing, state of affairs, or process; the act of simulating something generally entails representing certain key characteristics or behaviours of a selected physical or abstract system” [Wikipedia] for the purposes of testing some hypotheses usually stated in terms of the model being simulated and pairs of statistical data and expected outcomes. A trace (whether a domain behaviour or a program execution trace) is a time-stamped sequence of states: domain states, respectively simulator [or demo], monitor and monitor & control states. Figure P.7 on the following page (a) shows a horizontal time axis which basically “divides” that figure into two sub-figures. (b) Above the time axis the “fat” rounded edge rectangle alludes to the time-wise behaviour, a *domain trace*, of “The Domain” (i.e., the actual, or real, domain). (c) Below the time axis there are eight “thin” rectangles. These are labels S1, S2, S3, S4, S5, S6, S7 and S8. (d) Each of these denote a “run”, i.e., a time-stamped “execution”, a *program trace*, of the “Demo”. Their “relationship” to the time axis is this: their execution takes place in the real time as related to that of “The Domain” behaviour. (e) S1 and S2 ‘begins’ and ‘ends’ before the “The Domain” behaviour. (f) S7 and S8 ‘begins’ and ‘ends’ after the “The Domain” behaviour. (g) S3 ‘begins’ before and ‘ends’ during the “The Domain” behaviour. (g) S4 ‘begins’ before and



**S1, S2, S3, S4, S5, S6, S7, S8: "runs" of the Domain Simulation**

Figure P.7: Simulations

'ends' after the "The Domain" behaviour. (g) S5 'begins' after the begin and 'ends' before the end of the "The Domain" behaviour. (h) S6 'begins' during and 'ends' after that of the "The Domain" behaviour.

From Fig. P.7 and the above explication we can conclude that "executions" S4 and S5 each share exactly one time point,  $t$ , at which "The Domain" and "The Simulation" "share" time, that is, the time-stamped execution S4 and S5 reflect a "Simulation" state which at time  $t$  should reflect (some abstraction of) "The Domain" state.

Only if the domain behaviour (i.e., trace) fully "surrounds" that of the simulation trace, or, vice-versa (cf. Fig. P.7[S4,S5]), is there a "shared" time. Only if the 'begin' and 'end' times of the domain behaviour are identical to the 'start' and 'finish' times of the simulation trace, is there an infinity of shared 1-1 times.

In Fig P.6 on the preceding page we show "the same" "Domain Behaviour" (three times) and a (1) simulation, a (2) monitoring and a (3) monitoring & control, all of whose 'begin/start' ( $b/\beta$ ) and 'end/finish' ( $e/\epsilon$ ) times coincide. In such cases the "Demo/Simulation" takes place in real-time throughout the 'begin...end' interval.

Let  $\beta$  and  $\epsilon$  be the 'start' and 'finish' times of either S4 or S5. Then the relationship between  $t, \beta, \epsilon, b$  and  $e$  is  $\frac{t-b}{e-t} = \frac{t-\beta}{\epsilon-t}$  — which leads to a second degree polynomial in  $t$  which can then be solved in the usual, high school manner.

#### P.4.1.4 Identity, Microscopic and Macroscopic Simulations

In explaining Fig. P.7 we did not comment on the time relations wrt.  $b, e, \beta, \epsilon$ . But it was assumed, and now made explicit, that the intervals  $[b : e]$ , respectively  $[\beta : \epsilon]$ , express that the simulation  $[\beta : \epsilon]$ , stands in a one-to-one relation with the domain behaviour  $[b : e]$  [why otherwise show the simulation?]. That is that every distinct time point in interval  $[\beta : \epsilon]$  corresponds to exactly one distinct time point in interval  $[b : e]$

We can distinguish between four kinds of simulations. (i) If the time interval length  $[e - b]$  equals  $[\epsilon - \beta]$  then we say that the simulation is an *identity simulation*. (ii) If the time interval length  $[e - b]$  is larger than  $[\epsilon - \beta]$  then we say that the simulation is a *microscopic simulation*. (iii) If the time interval length  $[e - b]$  is smaller than  $[\epsilon - \beta]$  then we say that the simulation is a *macroscopic simulation*. (iv) If  $b$  is equal to  $\beta$  and  $e$  is equal to  $\epsilon$  then we say that the simulation is a *real-time simulation*.

As a consequence we can only "speak of" *monitors* and *controllers* if the simulation is a *real-time simulation*.

## P.4.2 Specific System Domains

We shall not go further into the above possible variety of simulations but focus on just some *real-time simulations*.<sup>15</sup>

Among possibly interesting such for the case of nuclear power plants we suggest:

- that a *conceptual domain model* be researched and developed for a generic nuclear power plant;
- a model that is then particularized into a number, say 12, further detailed *system domain models* – covering the
 

|                                  |                              |
|----------------------------------|------------------------------|
| – <b>design,</b>                 | – <b>emergency stops,</b>    |
| – <b>design feasibility,</b>     | – <b>maintenance,</b>        |
| – <b>building.</b>               | – <b>repair,</b>             |
| – <b>commissioning,</b>          | – <b>renewal of fuel,</b>    |
| – <b>start-up,</b>               | – <b>decommissioning</b> and |
| – <b>steady state operation,</b> | – <b>removal</b>             |

of nuclear power plants;

- that a monitor version of each of these models be requirements developed, [58, Chapter 9], and
- that each such requirements be implemented in software – for the support of financial backers, designers, builders, nuclear power plant staff and nuclear regulatory agency staff.

The conceptual domain model is what stake-holders of nuclear power plants have in mind when they now turn to model anyone of the system domains.

We now elaborate on each of the 12 nuclear power plant monitors.

### P.4.2.1 Design

The system domain model for the design of a specific, concrete nuclear power plant, takes its departure point in the conceptual domain model of Sect. P.3.

The *domain design model* shall model the the physics and engineering design of the major parts *the nuclear reactor, steam generator, the turbine, and the electricity generator* (R, SG+C+CT, T, and CT) as conceptualized in Sect. P.3.

That is, the domain analyzer cum describer works in very close contact, for months, with physicists and professional engineers from each of these *fields*. These persons shall inform *the system domain modeller* on *system* endurants and perdurants specific to the design process.

The resulting domain design model, DDM, also models the calculations – but does not actually perform these calculations – that must be performed during actual design.

The *input* to the design process (to be modelled) is (i) the professional skills of all relevant physicists and professional engineers  $\mathbb{P}_D$  and (ii) the *conceptual domain model* CDM.

The output, being built during the design process, is ( $\alpha$ ) a the *[formal] description* of [usually considerable set of] *annotated engineering drawings* EDD, ( $\beta$ ) a revised version of the *conceptual domain model*  $r\text{CDM}$ , and ( $\gamma$ ) a *domain design model* DDM:

- **value**

$$\text{DESIGN}: \mathbb{P}_D \times \text{CDM} \rightsquigarrow \text{EDD} \times r\text{CDM} \times \text{DDM}$$

<sup>15</sup>We leave out the obviously interesting one of developing and deploying a *non-real-time simulator* – say for *demo* purposes!

An interpretation of  $\mathcal{DESIGN}$ , with the prescribed arguments, i.e., an actual design process, for which  $\mathbb{DDM}$  is a model, then results in actual, “physical” drawings,  $\mathbb{EDD}_{afd}$ , whether on paper or electronically stored, or both.

The meaning,  $\mathcal{M}_{\mathbb{DDM}}$ ,  $\mathbb{DDM}$ , i.e.,  $\mathcal{M}_{\mathbb{DDM}}(\mathbb{DDM})$ , can then be requirements developed into software, i.e., a monitor that can help the management of designs.

#### P.4.2.2 Design Feasibility

TO BE WRITTEN

#### P.4.2.3 Building

The *domain building model* shall model the building, i.e., the “on-the-ground” actual engineering construction, of the *nuclear power plant* [as outlined in Sect. P.3].

That is, the domain analyzer cum describer works in very close contact, for months, with professional engineers from the design and the construction firms – as well as some of the physicists and professional engineers from the design study. They will inform *the system domain modeller* on *system* endurants and perdurants specific to the building process.

An element of the *domain building model* is a *program evaluation and review technique* [PERT<sup>16</sup>] like annotated and formal network diagram that shows the *critical paths*<sup>17</sup> of construction.

The *input* to the building process (to be modelled) is (i) the professional skills of all construction engineers and workers  $\mathbb{P}_B$ , (ii) the revised *conceptual domain model*  $r\mathbb{CDM}$ , (iii) the *domain design model*  $\mathbb{DDM}$ , (iv) the informally annotated and formal engineering drawings  $\mathbb{EDD}$ <sup>18</sup> and (v) the materials and tools  $\mathbb{MAT}$  that are to go into the construction.

The output, being built during the design process, is ( $\alpha$ ) the actual nuclear power plant, as a physical entity  $\mathbb{NPP}$ , there, on-the-ground, ( $\beta$ ) a set of *revised annotated engineering drawings*  $r\mathbb{EDD}$ , ( $\gamma$ ) a possibly further revised version of the *conceptual domain model*  $fr\mathbb{CDM}$ , ( $\delta$ ) a possibly further revised version of the *domain design model*  $fr\mathbb{DDM}$ , and ( $\epsilon$ ) a *domain building model*  $\mathbb{DBM}$ :

- value

$$BUILDD: \mathbb{P}_B \times r\mathbb{EDD} \times fr\mathbb{CDM} \times r\mathbb{DDM} \times \mathbb{MAT} \rightsquigarrow \mathbb{P} \times r\mathbb{EDD} \times fr\mathbb{CDM} \times fr\mathbb{DDM} \times \mathbb{DBM}$$

The meaning,  $\mathcal{M}_{\mathbb{DBM}}$ ,  $\mathbb{DBM}$ , i.e.,  $\mathcal{M}_{\mathbb{DBM}}(\mathbb{DBM})$ , can then be requirements developed into software, i.e., a monitor that can help the management of construction.

#### P.4.2.4 Commissioning

TO BE WRITTEN

#### P.4.2.5 Start-up

TO BE WRITTEN

#### P.4.2.6 Steady State Operation

TO BE WRITTEN

#### P.4.2.7 Emergency Stops

TO BE WRITTEN

<sup>16</sup>[https://en.wikipedia.org/wiki/Program\\_evaluation\\_and\\_review\\_technique](https://en.wikipedia.org/wiki/Program_evaluation_and_review_technique)

<sup>17</sup>[https://en.wikipedia.org/wiki/Critical\\_path\\_method](https://en.wikipedia.org/wiki/Critical_path_method)

<sup>18</sup>– not the drawings themselves, but a syntax for these

P.4.2.8 Maintenance

TO BE WRITTEN

P.4.2.9 Repair

TO BE WRITTEN

P.4.2.10 Renewal of Fuel

TO BE WRITTEN

P.4.2.11 Decommissioning

TO BE WRITTEN

P.4.2.12 Removal

TO BE WRITTEN

P.5 Varieties of Generation III-IV Reactors

Unlike in automobile construction, where hundreds of thousands of “copies” are produced, annually, of the same model (barring interior leather and fabric and exterior colouring), it seems that very, very few nuclear power plants are constructed according to one-and-the-same design.

Figures P.8, P.9 on the next page, and P.10 on page 497 schematize six such Generation IV designs<sup>19</sup>:

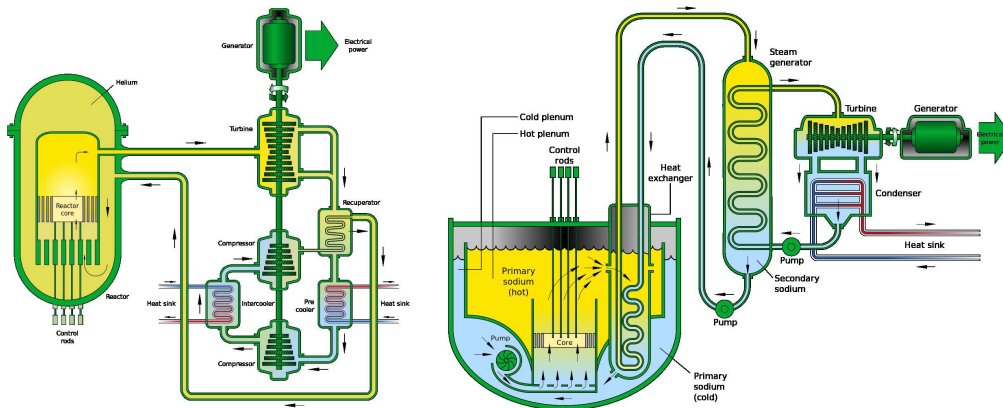


Figure P.8: Gas-Cooled and Sodium-Cooled Fast Reactors

MORE TO COME

P.6 Closing

P.6.1 What has so far been Achieved ?

TO BE WRITTEN

<sup>19</sup>From [https://en.m.wikipedia.org/wiki/Generation\\_IV\\_reactor](https://en.m.wikipedia.org/wiki/Generation_IV_reactor).

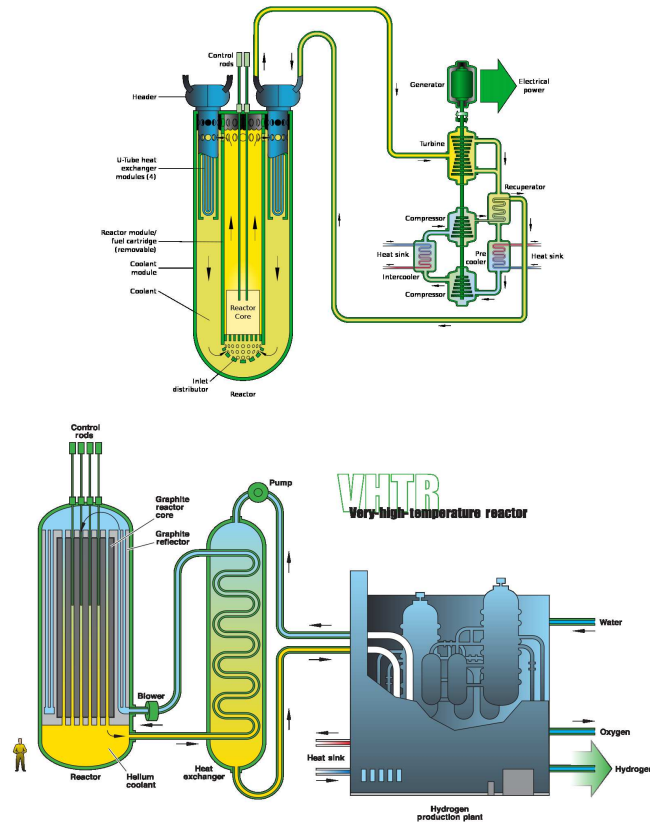


Figure P.9: Experiment Molten Salt and Molten Salt Reactors

## P.6.2 What is Next ?

TO BE WRITTEN

## P.6.3 Semantic versus Syntactic Reasoning

The current author can envisage two kinds of reader-remarks:

- What about using *Petri nets*, [152–156], to model nuclear power plants ?
- What about using *Multilevel Flow Modelling*, [126], to model nuclear power plants ?

### P.6.3.1 Petri nets

TO BE WRITTEN

### P.6.3.2 Multilevel Flow Modelling

TO BE WRITTEN

## P.6.4 Acknowledgements

I thank *Dr. Joseph Kiniry* of Galois, Inc., Portland, Oregon for having induced me onto the subject of this report.

MORE TO COME

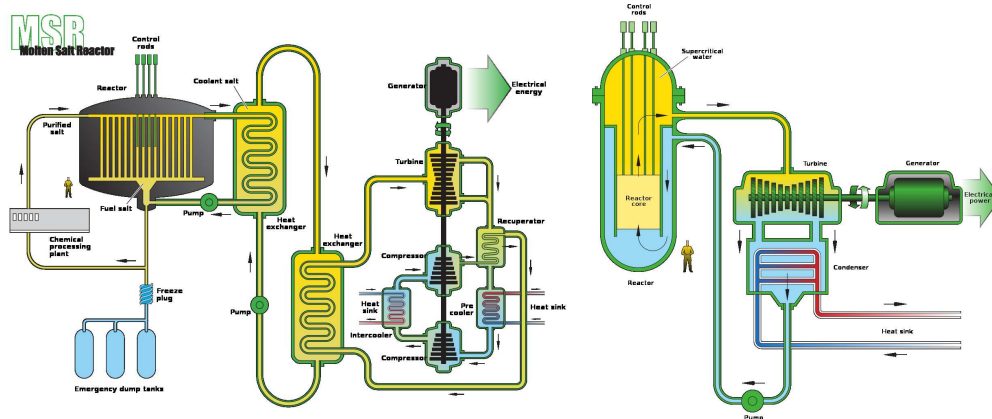


Figure P.10: Molten Salt and Supercritical-Water-Cooled Reactors

## P.7 Bibliography

### P.7.1 Bibliographical Notes

We are aware of the *Multilevel Flow Modelling*, MGM, approach of Morten Lind, [126]<sup>20</sup>. It seems that MFM relates its reasoning to graphical, visual, renditions of domains, not to mathematical logical domain descriptions. But their relationship should be studied.

<sup>20</sup><https://orbit.dtu.dk/en/publications/an-introduction-to-multilevel-flow-modeling>





# Part VI

## System Models

In Chapters Q–T we present four models – not quite of domains, but !

- **Chapter Q: Urban Planning** pages 501–549
- **Chapter R: Weather Systems** pages 551–565
- **Chapter S: A Stock Exchange** pages 567–590
- **Chapter T: An Extensible Virtual Shared Memory** pages 591–606



# Appendix Q

## Urban Planning

|         |     |
|---------|-----|
| Warning | The |
|---------|-----|

present urban planning domain description makes use of a “global” clock!

### Contents

---

|                                                                           |            |
|---------------------------------------------------------------------------|------------|
| <b>Q.1 Structures and Parts</b>                                           | <b>504</b> |
| Q.1.1 The Urban Space, Clock, Analysis & Planning Complex                 | 504        |
| Q.1.2 The Analyser Structure and Named Analysers                          | 505        |
| Q.1.3 The Planner Structure                                               | 505        |
| Q.1.4 Atomic Parts                                                        | 506        |
| Q.1.5 Preview of Structures and Parts                                     | 506        |
| Q.1.6 Planner Names                                                       | 506        |
| Q.1.7 Individual and Sets of Atomic Parts                                 | 506        |
| <b>Q.2 Unique Identifiers</b>                                             | <b>508</b> |
| Q.2.1 Urban Space Unique Identifier                                       | 508        |
| Q.2.2 Analyser Unique Identifiers                                         | 508        |
| Q.2.3 Master Planner Server Unique Identifier                             | 509        |
| Q.2.4 Master Planner Unique Identifier                                    | 509        |
| Q.2.5 Derived Planner Server Unique Identifier                            | 509        |
| Q.2.6 Derived Planner Unique Identifier                                   | 509        |
| Q.2.7 Derived Plan Index Generator Identifier                             | 509        |
| Q.2.8 Plan Repository                                                     | 510        |
| Q.2.9 Uniqueness of Identifiers                                           | 510        |
| Q.2.10 Indices and Index Sets                                             | 510        |
| Q.2.11 Retrieval of Parts from their Identifiers                          | 511        |
| Q.2.12 A Bijection: Derived Planner Names and Derived Planner Identifiers | 511        |
| <b>Q.3 Mereologies</b>                                                    | <b>512</b> |
| Q.3.1 Clock Mereology                                                     | 513        |
| Q.3.2 Urban Space Mereology                                               | 513        |
| Q.3.3 Analyser Mereology                                                  | 513        |
| Q.3.4 Analysis Depository Mereology                                       | 513        |
| Q.3.5 Master Planner Server Mereology                                     | 514        |
| Q.3.6 Master Planner Mereology                                            | 514        |
| Q.3.7 Derived Planner Server Mereology                                    | 514        |
| Q.3.8 Derived Planner Mereology                                           | 514        |
| Q.3.9 Derived Planner Index Generator Mereology                           | 515        |
| Q.3.10 Plan Repository Mereology                                          | 515        |

|            |                                                                                |            |
|------------|--------------------------------------------------------------------------------|------------|
| <b>Q.4</b> | <b>Attributes</b>                                                              | <b>515</b> |
| Q.4.1      | <b>Clock Attribute</b>                                                         | 515        |
| Q.4.1.1    | <b>Time and Time Intervals and their Arithmetic</b>                            | 515        |
| Q.4.1.2    | <b>The Attribute</b>                                                           | 516        |
| Q.4.2      | <b>Urban Space Attributes</b>                                                  | 516        |
| Q.4.2.1    | <b>The Urban Space</b>                                                         | 516        |
| Q.4.2.2    | <b>The Urban Space Attributes</b>                                              | 517        |
| Q.4.2.2.1  | <b>Main Part and Attributes</b>                                                | 517        |
| Q.4.2.2.2  | <b>Urban Space Attributes – Narratives and Formalisation</b>                   | 518        |
| Q.4.2.2.3  | <b>General Form of Attribute Models</b>                                        | 518        |
| Q.4.2.2.4  | <b>Geodetic Attribute[s]</b>                                                   | 518        |
| Q.4.2.2.5  | <b>Cadastral Attribute[s]</b>                                                  | 518        |
| Q.4.2.2.6  | <b>Geotechnical Attribute[s]</b>                                               | 519        |
| Q.4.2.2.7  | <b>Meteorological Attribute[s]</b>                                             | 519        |
| Q.4.2.2.8  | <b>Socio-Economic Attribute[s]</b>                                             | 519        |
| Q.4.2.2.9  | <b>Law Attribute[s]: State, Province, Region, City and District Ordinances</b> | 520        |
| Q.4.2.2.10 | <b>Industry and Business Economics</b>                                         | 520        |
| Q.4.2.2.11 | <b>Etcetera</b>                                                                | 520        |
| Q.4.2.2.12 | <b>The Urban Space Attributes – A Summary</b>                                  | 520        |
| Q.4.2.2.13 | <b>Discussion</b>                                                              | 521        |
| Q.4.3      | <b>Scripts</b>                                                                 | 521        |
| Q.4.4      | <b>Urban Analysis Attributes</b>                                               | 521        |
| Q.4.5      | <b>Analysis Depository Attributes</b>                                          | 521        |
| Q.4.6      | <b>Master Planner Server Attributes</b>                                        | 522        |
| Q.4.7      | <b>Master Planner Attributes</b>                                               | 522        |
| Q.4.8      | <b>Derived Planner Server Attributes</b>                                       | 523        |
| Q.4.9      | <b>Derived Planner Attributes</b>                                              | 523        |
| Q.4.10     | <b>Derived Planner Index Generator Attributes</b>                              | 523        |
| Q.4.11     | <b>Plan Repository Attributes</b>                                              | 524        |
| Q.4.12     | <b>A System Property of Derived Planner Identifiers</b>                        | 524        |
| <b>Q.5</b> | <b>The Structure Translators</b>                                               | <b>525</b> |
| Q.5.1      | <b>A UNIVERSE OF DISCOURSE Translator</b>                                      | 525        |
| Q.5.2      | <b>The ANALYSER STRUCTURE Translator</b>                                       | 525        |
| Q.5.3      | <b>The PLANNER STRUCTURE Translator</b>                                        | 525        |
| Q.5.3.1    | <b>The MASTER PLANNER STRUCTURE Translator</b>                                 | 526        |
| Q.5.3.2    | <b>The DERIVED PLANNER STRUCTURE Translator</b>                                | 526        |
| Q.5.3.3    | <b>The DERIVED PLANNER PAIR STRUCTURE Translator</b>                           | 526        |
| <b>Q.6</b> | <b>Channel Analysis and Channel Declarations</b>                               | <b>526</b> |
| Q.6.1      | <b>The clk_ch Channel</b>                                                      | 526        |
| Q.6.2      | <b>The tus_a_ch Channel</b>                                                    | 527        |
| Q.6.3      | <b>The tus_mps_ch Channel</b>                                                  | 527        |
| Q.6.4      | <b>The a_ad_ch Channel</b>                                                     | 528        |
| Q.6.5      | <b>The ad_s_ch Channel</b>                                                     | 528        |
| Q.6.6      | <b>The mps_mp_ch Channel</b>                                                   | 528        |
| Q.6.7      | <b>The p_pr_ch Channel</b>                                                     | 529        |
| Q.6.8      | <b>The p_dpxg_ch Channel</b>                                                   | 529        |
| Q.6.9      | <b>The pr_s_ch Channel</b>                                                     | 529        |
| Q.6.10     | <b>The dps_dp_ch Channel</b>                                                   | 530        |
| <b>Q.7</b> | <b>The Atomic Part Translators</b>                                             | <b>530</b> |

|           |                                                                                      |     |
|-----------|--------------------------------------------------------------------------------------|-----|
| Q.7.1     | <b>The CLOCK Translator</b>                                                          | 530 |
| Q.7.1.1   | <b>The Translate_CLK Function</b>                                                    | 530 |
| Q.7.1.2   | <b>The clock Behaviour</b>                                                           | 531 |
| Q.7.2     | <b>The URBAN SPACE Translator</b>                                                    | 531 |
| Q.7.2.1   | <b>The Translate_TUS Function</b>                                                    | 531 |
| Q.7.2.2   | <b>The urb_spa Behaviour</b>                                                         | 532 |
| Q.7.3     | <b>The ANALYSER<sub>anm<sub>i</sub></sub>, <math>i:[1:n]</math> Translator</b>       | 533 |
| Q.7.3.1   | <b>The Translate_A<sub>anm<sub>j</sub></sub> Function</b>                            | 533 |
| Q.7.3.2   | <b>The analyser<sub>ui<sub>j</sub></sub> Behaviour</b>                               | 534 |
| Q.7.4     | <b>The ANALYSIS DEPOSITORY Translator</b>                                            | 534 |
| Q.7.4.1   | <b>The Translate_AD Function</b>                                                     | 534 |
| Q.7.4.2   | <b>The ana_dep Behaviour</b>                                                         | 535 |
| Q.7.5     | <b>The DERIVED PLANNER INDEX GENERATOR Translator</b>                                | 535 |
| Q.7.5.1   | <b>The Translate_DPXG(<i>dpxg</i>) Function</b>                                      | 535 |
| Q.7.5.2   | <b>The dpxg Behaviour</b>                                                            | 536 |
| Q.7.6     | <b>The PLAN REPOSITORY Translator</b>                                                | 536 |
| Q.7.6.1   | <b>The Translate_PR Function</b>                                                     | 536 |
| Q.7.6.2   | <b>The plan_rep Behaviour</b>                                                        | 537 |
| Q.7.7     | <b>The MASTER SERVER Translator</b>                                                  | 537 |
| Q.7.7.1   | <b>The Translate_MPS Function</b>                                                    | 537 |
| Q.7.7.2   | <b>The master_server Behaviour</b>                                                   | 538 |
| Q.7.8     | <b>The MASTER PLANNER Translator</b>                                                 | 538 |
| Q.7.8.1   | <b>The Translate_MP Function</b>                                                     | 538 |
| Q.7.8.2   | <b>The Master urban_planning Function</b>                                            | 539 |
| Q.7.8.3   | <b>The master_planner Behaviour</b>                                                  | 540 |
| Q.7.8.4   | <b>The initiate derived servers and derived planners Behaviour</b>                   | 540 |
| Q.7.9     | <b>The DERIVED SERVER<sub>nm<sub>i</sub></sub>, <math>i:[1:p]</math> Translator</b>  | 541 |
| Q.7.9.1   | <b>The Translate_DPS<sub>nm<sub>j</sub></sub> Function</b>                           | 541 |
| Q.7.9.2   | <b>The derived_server Behaviour</b>                                                  | 542 |
| Q.7.10    | <b>The DERIVED PLANNER<sub>nm<sub>i</sub></sub>, <math>i:[1:p]</math> Translator</b> | 542 |
| Q.7.10.1  | <b>The Translate_DP<sub>dp<sub>nm<sub>j</sub></sub></sub> Function</b>               | 543 |
| Q.7.10.2  | <b>The derived_urban_planning Function</b>                                           | 543 |
| Q.7.10.3  | <b>The derived_planner<sub>nm<sub>j</sub></sub> Behaviour</b>                        | 544 |
| Q.8       | <b>Initialisation of The Urban Space Analysis &amp; Planning System</b>              | 545 |
| Q.8.1     | <b>Summary of Parts and Part Names</b>                                               | 545 |
| Q.8.2     | <b>Summary of of Unique Identifiers</b>                                              | 545 |
| Q.8.3     | <b>Summary of Channels</b>                                                           | 546 |
| Q.8.4     | <b>The Initial System</b>                                                            | 546 |
| Q.8.5     | <b>The Derived Planner System</b>                                                    | 546 |
| Q.9       | <b>Further Work</b>                                                                  | 546 |
| Q.9.1     | <b>Reasoning About Deadlock, Starvation, Live-lock and Liveness</b>                  | 546 |
| Q.9.2     | <b>Document Handling</b>                                                             | 547 |
| Q.9.2.1   | <b>Urban Planning Documents</b>                                                      | 547 |
| Q.9.2.2   | <b>A Document Handling System</b>                                                    | 547 |
| Q.9.3     | <b>Validation and Verification (V&amp;V)</b>                                         | 547 |
| Q.9.4     | <b>Urban Planning Project Management</b>                                             | 548 |
| Q.9.4.1   | <b>Urban Planning Projects</b>                                                       | 548 |
| Q.9.4.2   | <b>Strategic, Tactical and Operational Management</b>                                | 548 |
| Q.9.4.2.1 | <b>Project Resources</b>                                                             | 548 |
| Q.9.4.2.2 | <b>Strategic Management</b>                                                          | 549 |

|           |                                  |     |
|-----------|----------------------------------|-----|
| Q.9.4.2.3 | <b>Tactical Management</b>       | 549 |
| Q.9.4.2.4 | <b>Operational Management</b>    | 549 |
| Q.9.4.3   | <b>Urban Planning Management</b> | 549 |

## ENDURANTS

By an *entity* we shall understand a phenomenon, i.e., something that can be observed, i.e., be seen or touched by humans, or that can be conceived as an abstraction of an entity. We further demand that an entity can be objectively described

By an *endurant* we shall understand an entity that can be observed or conceived and described as a “complete thing” at no matter which given snapshot of time. Were we to “freeze” time we would still be able to observe the entire endurant.

By a *discrete endurant* we shall understand an endurant which is separate, individual or distinct in form or concept.

By a *part* we shall understand a discrete endurant which the domain engineer chooses to endow with *internal qualities*<sup>1</sup> such as *unique identification*, *mereology*, and one or more *attributes*. We shall define these three categories in Sects. Q.2, Q.3, respectively Sect. Q.4. We refer in general to [51].

In this, a major section of this report, we shall cover

- Sect. Q.1: Parts,
- Sect. Q.2: Unique Identifiers,
- Sect. Q.3: Mereology, and
- Sect. Q.4: Attributes.

## Q.1 Structures and Parts

From an epistemological<sup>2</sup> point of view a study of the parts of a universe of discourse is often the way to understand “*who the players*” of that domain are. From the point of view of [51] *knowledge about parts lead to knowledge about behaviours*. This is the reason, then, for our interest in parts.

### Q.1.1 The Urban Space, Clock, Analysis & Planning Complex

The domain-of-interest, i.e., the universe of discourse for this report is that of *the urban space analysis & planning complex* – where the ampersand, ‘&’, shall designate that we consider this complex as ‘one’!

1042. The *universe of discourse*, UoD, is here seen as a *structure* of four elements:

- a *clock*, CLK,
- the urban space*, TUS,
- an *analyser aggregate*, AA,
- the *planner aggregate*, PA,

#### type

1042 UoD, CLK, TUS, AAG, PA

#### value

1042a obs\_CLK: UoD → CLK

1042b obs\_TUS: UoD → TUS

<sup>1</sup>– where by *external qualities* of an endurant we mean whether it is discrete or *continuous*, whether it is a part, or a *component* – such as these are defined in [51].

<sup>2</sup>Epistemology is the branch of philosophy concerned with the theory of knowledge.

1042c obs\_AAG: UoD  $\rightarrow$  AAG

1042d obs\_PA: UoD  $\rightarrow$  PA

The clock and the urban space are here considered *atomic*, the *analyser aggregate*, AA, and the *planner aggregate*, PA, are here seen as *structures*.

### Q.1.2 The Analyser Structure and Named Analysers

1043. The *analyser structure* consists of

- (a) a *structure*, AC, which consists of two elements:
  - i. a *structure* of an indexed set, hence named *analysers*,
  - ii.  $A_{anm_1}, A_{anm_2}, \dots, A_{anm_n}$ ,

and

1044. an *atomic analysis depository*, AD.

There is therefore defined a set, ANms, of

1045. *analyser names*:  $\{anm_1, anm_2, \dots, anm_n\}$ , where  $n \geq 0$ .

#### type

1043 AA, AC, A, AD

1043(a)i  $A = A_{anm_1} \mid A_{anm_2} \mid \dots \mid A_{anm_n}$

1045 ANms =  $\{[anm_1, anm_2, \dots, anm_n]\}$

#### value

1043a obs\_AC: AA  $\rightarrow$  AC

1043(a)ii obs\_AC<sub>anm<sub>i</sub></sub>: AC  $\rightarrow$  A<sub>anm<sub>i</sub></sub>,  $i:[1..n]$

1044 obs\_AD: AA  $\rightarrow$  AD

*Analysers* and the *analysis depository* are here seen as atomic parts.

### Q.1.3 The Planner Structure

1046. The composite *planner structure* part, consists of

- (a) a *master planner structure*, MPA, which consists of
  - i. an atomic *master planner server*, MPS, and
  - ii. an atomic *master planner*, MP, and
- (b) a *derived planner structure*, DPA, which consists of
  - i. a *structure* in the form of an indexed set of (hence named) *derived planner structures*, DPC<sub>nm<sub>j</sub></sub>,  $j:[1..p]$ , which each consists of
    - A. a atomic *derived planner servers*, DPS<sub>nm<sub>j</sub></sub>,  $j:[1..p]$ , and
    - B. a atomic *derived planners*, DP<sub>nm<sub>j</sub></sub>,  $j:[1..p]$ ;
- (c) an atomic *plan repository*, PR, and
- (d) an atomic *derived planner index generator*, DPXG.

#### type

1046 PA, MPA, MPS, MP, DPA, DPC<sub>nm<sub>j</sub></sub>, DPS<sub>nm<sub>j</sub></sub>, DP<sub>nm<sub>j</sub></sub>,  $i:[1..p]$

**value**

- 1046a obs\_MPA: PA  $\rightarrow$  MPA  
 1046(a)i obs\_MPS: MPA  $\rightarrow$  MPS  
 1046(a)ii obs\_MP: MPA  $\rightarrow$  MP  
 1046b obs\_DPA: PA  $\rightarrow$  DPA  
 1046(b)i obs\_DPC<sub>nm<sub>j</sub></sub>: DPA  $\rightarrow$  DPC<sub>nm<sub>j</sub></sub>, i:[1..p]  
     1046(b)iA obs\_DPS<sub>nm<sub>j</sub></sub>: DPC<sub>nm<sub>j</sub></sub>  $\rightarrow$  DPS<sub>nm<sub>j</sub></sub>, i:[1..p]  
     1046(b)iB obs\_DP<sub>nm<sub>j</sub></sub>: DPC<sub>nm<sub>j</sub></sub>  $\rightarrow$  DP<sub>nm<sub>j</sub></sub>, i:[1..p]  
 1046c obs\_PR: PA  $\rightarrow$  PR  
 1046d obs\_DPXG:  $\rightarrow$  DPXG

We have chosen to model as *structures* what could have been modeled as *composite* parts. If we were to domain analyse & describe *management & organisation* facets of the urban space analysis & planning domain then we might have chosen to model some of these *structures* instead as *composite parts*.

**Q.1.4 Atomic Parts**

The following are seen as atomic parts:

- *clock*,
- *urban space*,
- *analysis deposit*,
- each *analyser* in the indexed set of *analyser<sub>anm<sub>i</sub>S</sub>*,
- *master planner server*,
- *master planner*,
- each *server* in the indexed set of *derived planner server<sub>nm<sub>j</sub>S</sub>*,
- each *planner* in the indexed set of *derived planner<sub>nm<sub>j</sub>S</sub>*,
- *derived planner index generator*.
- *plan repository* and

We shall return to these atomic part sorts when we explore their properties: *unique identifiers*, *mereologies* and *attributes*.

**Q.1.5 Preview of Structures and Parts**

Let us take a preview of the parts, see Fig. Q.1 on the next page.

**Q.1.6 Planner Names**

1047. There is therefore defined identical sets of *derived planner aggregate names*, *derived planner server names*, and *derived planner names*:  $\{dnm_1, dnm_2, \dots, dnm_p\}$ , where  $g \geq 0$ .

**type**

- 1047 DNms =  $\{dnm_1, dnm_2, \dots, dnm_p\}$

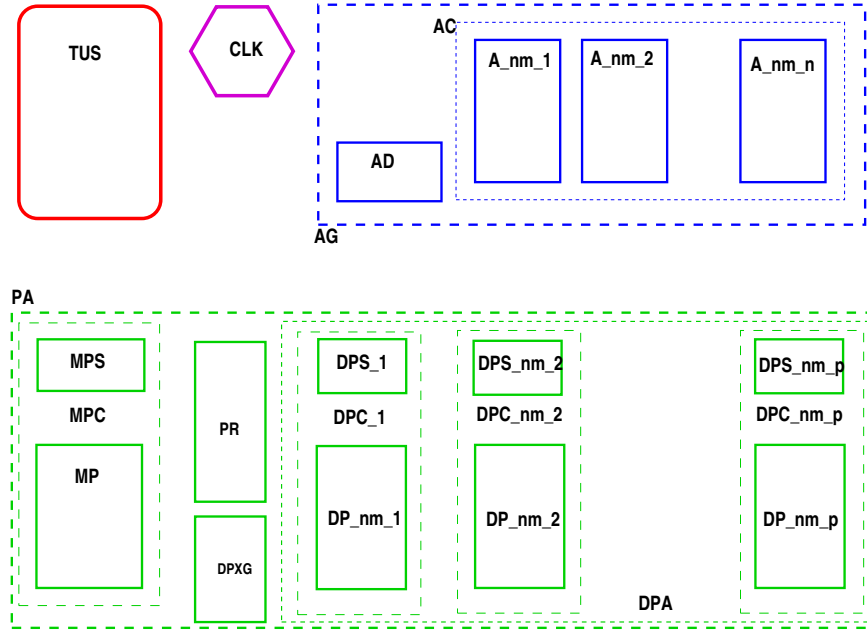
**Q.1.7 Individual and Sets of Atomic Parts**

In this closing section of Sect. Q.1.7 we shall identify individual and sets of atomic parts.

1048. We postulate an arbitrary *universe of discourse*, uod:UoD and let that be a constant value from which we calculate a number of individual and sets of atomic parts.

1049. There is the *clock*, *clk*:CLK,



Figure Q.1: The Urban Analysis and Planning System: **Structures and Atomic Parts**

1050. *the urban space*,  $tus:TUS$ ,
1051. the set of *analysers*,  $a_{anm_i}:A_{anm_i}, i:[1..n]$ ,
1052. the *analysis depository*,  $ad$ ,
1053. the *master planner server*,  $mps:MPS$ ,
1054. the *master planner*,  $mp:MP$ ,
1055. the set of *derived planner servers*,  $\{dps_{nm_i}:DPS_{nm_i} \mid i:[1..p]\}$ ,
1056. the set of *derived planners*,  $\{dp_{nm_i}:DP_{nm_i} \mid i:[1..p]\}$ ,
1057. the *derived plan index generator*,  $dp_xg$ ,
1058. the *plan repository*,  $pr$ , and
1059. the set of pairs of *derived server* and *derived planners*,  $sps$ .

**value**

- 1048  $uod : UoD$
- 1049  $clk : CLK = obs\_CLK(uod)$
- 1050  $tus : TUS = obs\_TUS(uod)$
- 1051  $ans : A_{anm_i}\text{-set}, i:[1..n] =$   
 $\{ obs\_A_{anm_i}(aa) \mid aa \in (obs\_AA(uod)), i:[1..n] \}$
- 1052  $ad : AD = obs\_AD(obs\_AA(uod))$
- 1053  $mps : MPS = obs\_MPS(obs\_MPA(uod))$
- 1054  $mp : MP = obs\_MP(obs\_MPA(uod))$
- 1055  $dps : DPS_{nm_i}\text{-set}, i:[1..p] =$   
 $\{ obs\_DPS_{nm_i}(dpc_{nm_i}) \mid$

```

1055 dpcnmi:DPCnmi • dpcnmi ∈ obs_DPCSnmi(obs_DPA(uod)), i:[1..p] }
1056 dps : DPnmi-set, i:[1..p] =
1056 { obs_DPnmi(dpcnmi) |
1056 dpcnmi:DPCnmi • dpcnmi ∈ obs_DPCSnmi(obs_DPA(uod)), i:[1..p] }
1057 dpxg : DPXG = obs_DPXG(uod)
1058 pr : PR = obs_PR(uod)
1059 spsps : (DPSnmi × DPnmi)-set, i:[1..p] =
1059 { (obs_DPSnmi(dpcnmi), obs_DPnmi(dpcnmi)) |
1059 dpcnmi:DPCnmi • dpcnmi ∈ obs_DPCSnmi(obs_DPA(uod)), i:[1..p] }

```

## Q.2 Unique Identifiers

We introduce a notion of unique identification of parts. We assume (i) that all parts,  $p$ , of any domain  $P$ , have unique identifiers, (ii) that unique identifiers (of parts  $p:P$ ) are abstract values (of the unique identifier,  $\pi$ , sort  $\Pi\_UI$  of parts  $p:P$ ), (iii) such that distinct part sorts,  $P_i$  and  $P_j$ , have distinctly named unique identifier sorts, say  $\Pi\_UI_i$  and  $\Pi\_UI_j$ , (iv) that all  $\pi:\Pi\_UI_i$  and  $\pi_j:\Pi\_UI_j$  are distinct, and (v) that the observer function  $uid\_P$  applied to  $p$  yields the unique identifier, say  $\pi:\Pi\_UI$ , of  $p$ .

The analysis & description of unique identification is a prerequisite for talking about mereologies of universes of discourse, and the analysis & description of mereologies are a means for understanding how parts relate to one another.

Since we model as *structures* what elsewhere might have been modeled as *composite parts* we shall only deal with *unique identifiers* of *atomic parts*.

### Q.2.1 Urban Space Unique Identifier

1060. The urban space has a unique identifier.

```

type
1060 TUS_UI
value
1060 uid_TSU: TSU → TUS_UI

```

### Q.2.2 Analyser Unique Identifiers

1061. Each analyser has a unique identifier.

1062. The analysis depository has a unique identifier.

```

type
1061 A_UI = A_UIanm1 | A_UIanm2 | ... | A_UIanmn
1062 AD_UI
value
1061 uid_A: Anmi → A_UInmi, i : [1..n]
1062 uid_AD: AD → AD_UI
axiom
1061 ∀ anmi:Anmi •
1061 let a_uinmi = uid_A(anmi) in a_uinmi ≈ nmi end

```

The mathematical symbol  $\simeq$  (in this report) denotes *isomorphy*.

**Q.2.3 Master Planner Server Unique Identifier**

1063. The *unique identifier* of the *master planner server*.

**type**

1063 MPS\_UI

**value**

1063 uid\_MPS: MPS  $\rightarrow$  MPS\_UI

**Q.2.4 Master Planner Unique Identifier**

1064. The *unique identifier* of the *master planner*.

**type**

1064 MP\_UI

**value**

1064 uid\_MP: MP  $\rightarrow$  MP\_UI

**Q.2.5 Derived Planner Server Unique Identifier**

1065. The *unique identifiers* of *derived planner servers*.

**type**

1065 DPS\_UI = DPS\_UI<sub>nm<sub>1</sub></sub> | DPS\_UI<sub>nm<sub>2</sub></sub> | ... | DPS\_UI<sub>nm<sub>p</sub></sub>

**value**

1065 uid\_DPS: DPS<sub>nm<sub>i</sub></sub>  $\rightarrow$  DPS\_UI<sub>nm<sub>i</sub></sub>,  $i : [1..p]$

**axiom**

1065  $\forall$  dps<sub>nm<sub>i</sub></sub>:DPS<sub>nm<sub>i</sub></sub> •

1065 **let** dps\_ui<sub>nm<sub>i</sub></sub> = uid\_DPS(dps<sub>nm<sub>i</sub></sub>) **in** dps\_ui<sub>nm<sub>i</sub></sub>  $\simeq$  nm<sub>i</sub> **end**

**Q.2.6 Derived Planner Unique Identifier**

1066. The *unique identifiers* of *derived planners*.

**type**

1066 DP\_UI = DP\_UI<sub>nm<sub>1</sub></sub> | DP\_UI<sub>nm<sub>2</sub></sub> | ... | DP\_UI<sub>nm<sub>p</sub></sub>

**value**

1066 uid\_DP: DP<sub>nm<sub>i</sub></sub>  $\rightarrow$  DP\_UI<sub>nm<sub>i</sub></sub>,  $i : [1..p]$

**axiom**

1066  $\forall$  dp<sub>nm<sub>i</sub></sub>:DP<sub>nm<sub>i</sub></sub> •

1066 **let** dp\_ui<sub>nm<sub>i</sub></sub> = uid\_DP(dp<sub>nm<sub>i</sub></sub>) **in** dp\_ui<sub>nm<sub>i</sub></sub>  $\simeq$  nm<sub>i</sub> **end**

**Q.2.7 Derived Plan Index Generator Identifier**

1067. The *unique identifier* of *derived plan index generator*:

**type**

1067 DPXG\_UI

**value**

1067 uid\_DPXG: DPXG  $\rightarrow$  DPXG\_UI

### Q.2.8 Plan Repository

1068. The *unique identifier* of *plan repository*:

**type**

1068 PR\_UI

**value**

1068 uid\_PR: PR  $\rightarrow$  PR\_UI

### Q.2.9 Uniqueness of Identifiers

1069. The identifiers of all analysers are distinct.

1070. The identifiers of all derived planner servers are distinct.

1071. The identifiers of all derived planners are distinct.

1072. The identifiers of all other atomic parts are distinct.

1073. And the identifiers of all atomic parts are distinct.

1069 **card**  $ans = \mathbf{card} a_{ui}s$

1070 **card**  $dpss = \mathbf{card} dps_{ui}s$

1071 **card**  $dps = \mathbf{card} dp_{ui}s$

1072 **card** $\{clk_{ui}, tus_{ui}, ad_{ui}, mps_{ui}, mp_{ui}, dpxg_{ui}, plas_{ui}\} = 7$

1073  $\cap(ans, dpss, dps, \{clk_{ui}, tus_{ui}, ad_{ui}, mps_{ui}, mp_{ui}, dpxg_{ui}, plas_{ui}\}) = \{\}$

### Q.2.10 Indices and Index Sets

It will turn out to be convenient, in the following, to introduce a number of index sets.

1074. There is the *clock* identifier,  $clk_{ui}$ :CLK\_UI.

1075. There is the *the urban space* identifier,  $tus_{ui}$ :TUS\_UI.

1076. There is the set,  $a_{ui}s$ :A\_UI-**set**, of the identifiers of all *analysers*.

1077. The *analysis depository* identifier,  $ad_{ui}$ .

1078. There is the *master planner server* identifier,  $mps_{ui}$ :MPS\_UI.

1079. There is the *master planner* identifier,  $mp_{ui}$ :MP\_UI.

1080. There is the set,  $dps_{ui}s$ :DPS\_UI-**set**, of the identifiers of all *derived planner servers*.

1081. There is the set,  $dp_{ui}s$ :DP\_UI-**set**, of the identifiers of all derived planners.

1082. There is the *derived plan index generator* identifier,  $dpxg_{ui}$ :DPXG\_UI.

1083. And there is the *plan repository* identifier,  $pr_{ui}$ :PR\_UI.

**value**

1074  $clk_{ui} : \text{CLK\_UI} = \text{uid\_CLK}(uod)$

1075  $tus_{ui} : \text{TUS\_UI} = \text{uid\_TUS}(uod)$

1076  $a_{ui}s : \text{A\_UI-**set}}**$  =  $\{\text{uid\_A}(a) | a : \text{A} \bullet a \in ans\}$

1077  $ad_{ui} : \text{AD\_UI} = \text{uid\_AD}(ad)$

1078  $mps_{ui} : \text{MPS\_UI} = \text{uid\_MPS}(mps)$

1079  $mp_{ui} : \text{MP\_UI} = \text{uid\_MP}(mp)$

1080  $dp_{s_{ui}s} : \text{DPS\_UI-set} = \{\text{uid\_DPS}(dps) \mid dps : \text{DPS} \bullet dps \in dp_{s_{s}}\}$   
 1081  $dp_{p_{ui}s} : \text{DP\_UI-set} = \{\text{uid\_DP}(dp) \mid dp : \text{DP} \bullet dp \in dp_{s}\}$   
 1082  $dp_{xg_{ui}} : \text{DPXG\_UI} = \text{uid\_DPXG}(dp_{xg})$   
 1083  $pr_{ui} : \text{PR\_UI} = \text{uid\_PR}(pr)$

1084. There is also the set of identifiers for all servers:  $ps_{ui}s : (\text{MPS\_UI} \mid \text{DPS\_UI})\text{-set}$ ,
1085. there is then the set of identifiers for all planners:  $ps_{ui}s : (\text{MP\_UI} \mid \text{DP\_UI})\text{-set}$ ,
1086. there is finally the set of pairs of paired *derived planner server* and *derived planner* identifiers.
1087. there is a map from the unique derived server identifiers to their “paired” unique derived planner identifiers, and
1088. there is finally the reverse map from planner to server identifiers.

**value**

1084  $s_{ui}s : (\text{MPS\_UI} \mid \text{DPS\_UI})\text{-set} = \{mp_{s_{ui}}\} \cup dp_{s_{ui}s}$   
 1085  $p_{ui}s : (\text{MP\_UI} \mid \text{DP\_UI})\text{-set} = \{mp_{ui}\} \cup dp_{ui}s$   
 1086  $sips : (\text{DPS\_UI} \times \text{DP\_UI})\text{-set} = \{(\text{uid\_DPS}(dps), \text{uid\_DP}(dp)) \mid (dps, dp) : (\text{DPS} \times \text{DP}) \bullet (dps, dp) \in sps\}$   
 1087  $si\_pi\_m : \text{DPS\_UI} \xrightarrow{\overline{m}} \text{DP\_UI} = [\text{uid\_DPS}(dps) \mapsto \text{uid\_DP}(dp) \mid (dps, dp) : (\text{DPS} \times \text{DP}) \bullet (dps, dp) \in sps]$   
 1088  $pi\_si\_m : \text{DP\_UI} \xrightarrow{\overline{m}} \text{DPS\_UI} = [\text{uid\_DP}(dp) \mapsto \text{uid\_DPS}(dps) \mid (dps, dp) : (\text{DPS} \times \text{DP}) \bullet (dps, dp) \in sps]$

**Q.2.11 Retrieval of Parts from their Identifiers**

1089. Given the global set  $dp_{s_{s}}$ , cf. 1055 on page 507, i.e., the set of all derived servers, and given a unique planner server identifier, we can calculate the derived server with that identifier.
1090. Given the global set  $dp_{s}$ , cf. 1056 on page 507, the set of all derived planners, and given a unique derived planner identifier, we can calculate the derived planner with that identifier.

**value**

1089  $c_s : dp_{s_{s}} \rightarrow \text{DPS\_UI} \rightarrow \text{DPS}$   
 1089  $c_s(dp_{s_{s}})(dps\_ui) \equiv \text{let } dps : \text{DPS} \bullet dps \in dp_{s_{s}} \wedge \text{uid\_DPS}(dps) = dps\_ui \text{ in } dps \text{ end}$   
 1090  $c_p : dp_{s} \rightarrow \text{DP\_UI} \rightarrow \text{DP}$   
 1090  $c_p(dp_{s})(dp\_ui) \equiv \text{let } dp : \text{DP} \bullet dp \in dp_{s} \wedge \text{uid\_DP}(dp) = dp\_ui \text{ in } dp \text{ end}$

**Q.2.12 A Bijection: Derived Planner Names and Derived Planner Identifiers**

We can postulate a unique relation between the names,  $dn : \text{DNm-set}$ , i.e., the names  $dn \in \text{DNms}$ , and the unique identifiers of the named planners:

1091. We can claim that there is a function,  $\text{extr\_DNm}$ , from the unique identifiers of derived planner servers to the names of these unique identifiers.
1092. Similarly can claim that there is a function,  $\text{extr\_DNm}$ , from the unique identifiers of derived planners to the names of these unique identifiers.

**value**

1091  $\text{extr\_Nm} : \text{DPS\_UI} \rightarrow \text{DNm}$   
 1091  $\text{extr\_Nm}(dps\_ui) \equiv \dots$   
 1092  $\text{extr\_Nm} : \text{DP\_UI} \rightarrow \text{DNm}$   
 1092  $\text{extr\_Nm}(dp\_ui) \equiv \dots$

**axiom**

1091  $\forall dps\_ui1, dps\_ui2 : \text{DPS\_ui} \bullet dps\_ui1 \neq dps\_ui2 \Rightarrow \text{extr\_Nm}(dps\_ui1) \neq \text{extr\_Nm}(dps\_ui2)$   
 1092  $\forall dp\_ui1, dp\_ui2 : \text{DP\_ui} \bullet dp\_ui1 \neq dp\_ui2 \Rightarrow \text{extr\_Nm}(dp\_ui1) \neq \text{extr\_Nm}(dp\_ui2)$

1093. Let  $\text{dps\_ui\_dnm}:\text{DPS\_UI\_DNm}$ ,  $\text{dp\_ui\_dnm}:\text{DP\_UI\_DNm}$  stand for maps from derived planner server, respectively derived planner unique identifiers to derived planner names.

1094. Let  $\text{nm\_dp\_ui}:\text{Nm\_DP\_UI}$ ,  $\text{nm\_dp\_ui}:\text{Nm\_DP\_UI}$  stand for the reverse maps.

1095. These maps are bijections.

#### type

1093  $\text{DPS\_UI\_DNm}: \text{DPS\_UI} \xrightarrow{\cong} \text{DP\_Nm}$

1093  $\text{DP\_UI\_DNm}: \text{DP\_UI} \xrightarrow{\cong} \text{DP\_Nm}$

1094  $\text{DNm\_DPS\_UI}: \text{DP\_Nm} \xrightarrow{\cong} \text{DP\_UI}$

1094  $\text{DNm\_DP\_UI}: \text{DP\_Nm} \xrightarrow{\cong} \text{DP\_UI}$

#### axiom

1095  $\forall \text{dps\_ui\_dnm}:\text{DPS\_UI\_DNm} \cdot \text{dps\_ui\_dnm}^{-1} \cdot \text{dps\_ui\_dnm} = \lambda x.x$

1095  $\forall \text{dp\_ui\_dnm}:\text{DP\_UI\_DNm} \cdot \text{dp\_ui\_dnm}^{-1} \cdot \text{dp\_ui\_dnm} = \lambda x.x$

1095  $\forall \text{dnm\_dps\_ui}:\text{DNm\_DPS\_UI} \cdot \text{dnm\_dps\_ui}^{-1} \cdot \text{dnm\_dps\_ui} = \lambda x.x$

1095  $\forall \text{dnm\_dp\_ui}:\text{DNm\_DP\_UI} \cdot \text{dp\_ui\_dnm}^{-1} \cdot \text{dnm\_dp\_ui} = \lambda x.x$

that is:

1095  $\forall \text{dps\_ui\_dnm}:\text{DPS\_UI\_DNm}, \text{dp\_ui\_dnm}:\text{DP\_UI\_DNm}, \text{dps\_ui}:\text{DPS\_UI} \cdot$

1095  $\text{dps\_ui} \in \mathbf{dom} \text{dps\_ui\_dnm} \Rightarrow \text{dp\_ui\_dnm}(\text{dps\_ui\_dnm}(\text{dps\_ui})) = \text{dps\_ui}$

et cetera !

1096. The function  $\text{mk\_DNm\_DUI}$  takes the set of all derived planner servers, respectively derived planners and produces bijective maps,  $\text{dnm\_dps\_ui}$ , respectively  $\text{dnm\_dp\_ui}$ .

1097. Let  $\text{dnm\_dps\_ui}:\text{DNm\_DPS\_UI}$  and

1098.  $\text{dnm\_dp\_ui}:\text{DNm\_DP\_UI}$

stand for such [global] maps.

#### value

1096  $\text{mk\_Nm\_DPS\_UI}: \text{DPS}_{\text{nm}_i}\text{-set} \rightarrow \text{DNm\_DPS\_UI}$

1096  $\text{mk\_Nm\_DPS\_UI}(dps) \equiv [\text{uid\_DPS}(dps) \mapsto \text{extr\_Nm}(\text{uid\_DPS}(dps)) | \text{dps}:\text{DPS} \cdot \text{dps} \in dps]$

1096  $\text{mk\_Nm\_DP\_UI}: \text{DP}_{\text{nm}_i}\text{-set} \rightarrow \text{DNm\_DP\_UI}$

1096  $\text{mk\_Nm\_DP\_UI}(dps) \equiv [\text{uid\_DP}(dp) \mapsto \text{extr\_Nm}(\text{uid\_DP}(dp)) | dp:\text{DP} \cdot \text{dps} \in dps]$

1097  $\text{nm\_dps\_ui}:\text{Nm\_DPS\_UI} = \text{mk\_Nm\_DPS\_UI}(dps)$

1098  $\text{nm\_dp\_ui}:\text{Nm\_DP\_UI} = \text{mk\_Nm\_DP\_UI}(dps)$

## Q.3 Mereologies

Mereology (from the Greek  $\mu\epsilon\rho\omicron\varsigma$  ‘part’) is the *theory of part-hood relations*: of *the relations of part to whole* and *the relations of part to part within a whole*<sup>3</sup>.

Part mereologies inform of how parts relate to other parts. As we shall see in the section on *perdurants*, mereologies are the basis for analysing & describing communicating between part behaviours.

Again: since we model as *structures* what is elsewhere modeled as *composite parts* we shall only consider *mereologies* of *atomic parts*.

<sup>3</sup>Achille Varzi: Mereology, <http://plato.stanford.edu/entries/mereology/> 2009 and [81].

**Q.3.1 Clock Mereology**

1099. The clock is related to all those parts that create information, i.e., documents of interest to other parts. Time is then used to *time-stamp* those documents. These other parts are: the *urban space*, the *analysers*, the *planner servers* and the *planners*.

**type**

1099 CLK\_Mer = TSU\_UI × A\_UI-set × MPS\_UI × MP\_UI × DPS\_UI-set × DP\_UI-set

**value**

1099 mereo\_CLK: CLK → Clk\_Mer

**axiom**

1099 mereo\_CLK(*uod*) = (*tus<sub>ui</sub>*, *a<sub>ui</sub>s*, *mps<sub>ui</sub>*, *mp<sub>ui</sub>*, *dps<sub>ui</sub>s*, *dp<sub>ui</sub>s*)

**Q.3.2 Urban Space Mereology**

The urban space stands in relation to those parts which consume urban space information: the *clock* (in order to time stamp urban space information), the *analysers* and the *master planner server*.

1100. The mereology of the urban space is a triple of the clock identifier, the identifier of the master planner server and the set of all analyser identifiers. all of which are provided with urban space information.

1101. The constraint here is expressed in the ‘the’: for the universe of discourse it must be the master planner aggregate unique identifier and the set of exactly all the analyser unique identifiers for that universe.

**type**

1100 TUS\_Mer = CLK\_UI × A\_UI-set × MPS\_UI

**value**

1100 mereo\_TUS: TUS → TUS\_Mer

**axiom**

1101 mereo\_TUS(*tus*) = (*clk<sub>ui</sub>*, *a<sub>ui</sub>s*, *mps<sub>ui</sub>*)

**Q.3.3 Analyser Mereology**

1102. The mereology of a[ny] *analyser* is that of a triple: the *clock identifier*, the *urban space identifier*, and the *analysis depository identifier*.

**type**

1102 A\_Mer = CLK\_UI × TUS\_UI × AD\_UI

**value**

1102 mereo\_A: A → A\_Mer

**Q.3.4 Analysis Depository Mereology**

1103. The mereology of the *analysis depository* is a triple: the *clock identifier*, the *master planner server identifier*, and the set of *derived planner server identifiers*.

**type**

1103 AD\_Mer = CLK\_UI × MPS\_UI × DPS\_UI-set

**value**

1103 mereo\_AD: AD → AD\_Mer

### Q.3.5 Master Planner Server Mereology

1104. The *master planner server* mereology is a quadruplet of the clock identifier (time is used to time stamp input arguments, prepared by the server, to the planner), the urban space identifier, the analysis depository and the master planner identifier.
1105. And for all universes of discourse these must be exactly those of that universe.

**type**

1104  $MPS\_Mer = CLK\_UI \times TUS\_UI \times AD\_UI \times MP\_UI$

**value**

1104 mereo\_MPS:  $MPS \rightarrow MPS\_Mer$

**axiom**

1105  $mereo\_MPS(mps) = (clk_{ui}, tus_{ui}, ad_{ui}, mp_{ui})$

### Q.3.6 Master Planner Mereology

1106. The mereology of the *master planner* is a triple of: the clock identifier<sup>4</sup>, master server identifier<sup>5</sup>, derived planner index generator identifier<sup>6</sup>, and the plan repository identifier<sup>7</sup>.

**type**

1106  $MP\_Mer = CLK\_UI \times MPS\_UI \times DPXG\_UI \times PR\_UI$

**value**

1106 mereo\_MP:  $MP \rightarrow MP\_Mer$

**axiom**

1106  $mereo\_MP(mp) = (clk_{ui}, mps_{ui}, dpxg_{ui}, pr_{ui})$

### Q.3.7 Derived Planner Server Mereology

1107. The *derived planner server* mereology is a quadruplet of:

the clock identifier<sup>8</sup>, the set of all analyser and the derived planner identifier<sup>11</sup>. identifiers<sup>9</sup>, the plan repository identifier,<sup>10</sup>

**type**

1107  $DPS\_Mer = CLK\_UI \times AD\_UI \times PLAS\_UI \times DP\_UI$

**value**

1107 mereo\_DPS:  $DPS \rightarrow DPS\_Mer$

**axiom**

1107  $\forall (dps, dp): (DPS \times DP) \cdot (dps, dp) \in sps \Rightarrow$

1107  $mereo\_DPS(dps) = (clk_{ui}, ad_{ui}, plas_{ui}, uid\_DP(dp))$

### Q.3.8 Derived Planner Mereology

1108. The *derived planner* mereology is a quadruplet of:

<sup>4</sup>From the clock the planners obtain the time with which they stamp all information assembled by the planner.

<sup>5</sup>from which the master planner obtains essential input arguments

<sup>6</sup>in collaboration with which the master planner obtains a possibly empty set of derived planning indices

<sup>7</sup>with which it posits and from which it obtains summaries of all urban planning plans produced so far.

<sup>8</sup>From the clock the servers obtain the time with which they stamp all information assembled by the servers.

<sup>9</sup>From the analysers the servers obtain analyses.

<sup>10</sup>In collaboration with the plan repository the planners deposit plans etc. and obtains summaries of all urban planning plans produced so far

<sup>11</sup>The server provides its associated planner with appropriate input arguments.



the clock identifier, the derived plan server identifier, and the plan repository identifier. identifier, the derived plan index generator

**type**

1108  $DP\_Mer = CLK\_UI \times DPS\_UI \times DPXG\_UI \times PR\_UI$

**value**

1108  $mereo\_DP: DP \rightarrow DP\_Mer$

**axiom**

1108  $\forall (dps, dp): (DPS \times DP) \cdot (dps, dp) \in sps \Rightarrow$

1108  $mereo\_DP(dp) = (clk_{ui}, uid\_DPS(dps), dp_{xg_{ui}, pr_{ui}})$

**Q.3.9 Derived Planner Index Generator Mereology**

1109. The mereology of the derived planner index generator is the set of all planner identifiers: master and derived.

**type**

1109  $DPXG\_Mer = (MP\_UI | DP\_UI)\text{-set}$

**value**

1109  $mereo\_DPXG: DPXG \rightarrow DPXG\_Mer$

**axiom**

1109  $mereo\_DPXG(dp_{xg}) = ps_{uis}$

**Q.3.10 Plan Repository Mereology**

1110. The plan repository mereology is the set of all planner identifiers: master and derived.

1110  $PR\_Mer = (MP\_UI | DP\_UI)\text{-set}$

**value**

1110  $mereo\_PR: PR \rightarrow PR\_Mer$

**axiom**

1110  $mereo\_PR(pr) = ps_{uis}$

**Q.4 Attributes**

Parts are typically recognised because of their spatial form and are otherwise characterised by their intangible, but measurable attributes. That is, whereas endurants, whether discrete (as are parts and components) or continuous (as are materials), are physical, tangible, in the sense of being spatial (or being abstractions, i.e., concepts, of spatial endurants), attributes are intangible: cannot normally be touched, or seen, but can be objectively measured. Thus, in our quest for describing domains where humans play an active rôle, we rule out subjective “attributes”: feelings, sentiments, moods. Thus we shall abstain, in our domain science also from matters of aesthetics. A formal concept, that is, a type, consists of all the entities which all have the same qualities. Thus removing a quality from an entity makes no sense: the entity of that type either becomes an entity of another type or ceases to exist (i.e., becomes a non-entity)

**Q.4.1 Clock Attribute****Q.4.1.1 Time and Time Intervals and their Arithmetic**

1111. Time is modeled as a continuous entity.

- 1112. One can subtract two times and obtain a time interval.
- 1113. There is an “infinitesimally” smallest time interval,  $\delta t:T$ .
- 1114. Time intervals are likewise modeled as continuous entities.
- 1115. One can add or subtract a time interval to, resp. from a time and obtain a time.
- 1116. One can compare two times, or two time intervals.
- 1117. One can add and subtract time intervals.
- 1118. One can multiply time intervals with real numbers.

**type**

1111 T  
 1112 TI

**value**

1112 sub:  $T \times T \rightarrow TI$   
 1113  $\delta t:TI$   
 1115 add,sub:  $TI \times T \rightarrow T$   
 1116  $<,\leq,=,\geq,>: ((T \times T)|(TI \times TI)) \rightarrow \mathbf{Bool}$   
 1117 add,sub:  $TI \times TI \rightarrow TI$   
 1118 mpy:  $TI \times \mathbf{Real} \rightarrow TI$

**Q.4.1.2 The Attribute**

- 1119. The only attribute of a clock is time. It is a programmable attribute.

**type**

1119 T

**value**

1119 attr\_T:  $CLK \rightarrow T$

**axiom**

1119  $\forall clk:CLK \bullet$   
 1119 **let** (t,t') = (attr\_CLK(clk);attr\_CLK(clk)) **in**  
 1119  $t \leq t'$  **end**

The ‘;’ in an expression (a;b) shall mean that first expression a is evaluated, then expression b.

**Q.4.2 Urban Space Attributes****Q.4.2.1 The Urban Space**

- 1120. We shall assume a notion of *the urban space*,  $tus:TUS$ , from which we can observe the attribute:
- 1121. an infinite, compact Euclidean set of points.
- 1122. By a *point* we shall understand a further undefined atomic notion.
- 1123. By an *area* we shall understand a concept, related to the urban space, that allows us to speak of “a point being in an area” and “an area being equal to or properly within another area”.
- 1124. To an[y] *urban space* we can associate an area; we may think of an area being an *attribute* of the urban space.

**type**

1120 TUS  
 1121 PtS = Pt-infset

**value**

1120 attr\_PtS: TUS → Pt-infset

**type**

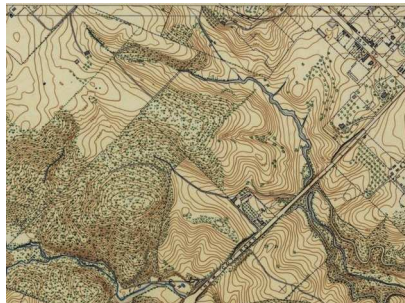
1122 Pt  
 1123 Area

**value**

1124 attr\_Area: TUS → Area  
 1123 is\_Pt\_in\_Area: Pt × (TUS|Area) → **Bool**  
 1123 is\_Area\_within\_Area: Area × (TUS|Area) → **Bool**

**Q.4.2.2 The Urban Space Attributes**

By *urban space attributes* we shall here mean the facts by means of which we can characterize that which is subject to urban planning: the land, what is in and on it: its geodetics, its cadastra<sup>12</sup>, its meteorology, its socio-economics, its rule of law, etc. As such we shall consider ‘the urban space’ to be a *part* in the sense of [51]. And we shall consider the *geodetic, cadastral, geotechnical, meteorological, “the law”* (i.e., *state, province, city and district ordinances*) and *socio-economic* properties as *attributes*.



Left: geodetic map, right: cadastral map.

**Q.4.2.2.1 Main Part and Attributes** One way of observing *the urban space* is presented: to the left, in the framed box, we **narrate** the story; to the right, in the framed box, we **formalise** it.

|                                                                                                                                                                                                                                                                                                                                                                    |                                                                                                                                                                                                                                                                                                                                                                                                                     |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>1125. The Urban Space (TUS) has the following</p> <ul style="list-style-type: none"> <li>(a) PointSpace attributes,</li> <li>(b) Geodetic attributes,</li> <li>(c) Cadastre attributes,</li> <li>(d) Geotechnical attributes,</li> <li>(e) Meteorological attributes,</li> <li>(f) Law attributes,</li> <li>(g) Socio-Economic attributes, etcetera.</li> </ul> | <p><b>type</b></p> <p>1125 TUS, PtS, GeoD, Cada, GeoT, Met, Law, SocEco, ...</p> <p><b>value</b></p> <p>1125a attr_Pts: TUS → PtS<br/>             1125b attr_GeoD: TUS → GeoD<br/>             1125c attr_Cada: TUS → Cada<br/>             1125d attr_GeoT: TUS → GeoT<br/>             1125e attr_Met: TUS → Met<br/>             1125f attr_Law: TUS → Law<br/>             1125g attr_SocEco: TUS → SocEco</p> |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

The  $attr\_A: P \rightarrow A$  is the **signature** of a postulated *attribute (observer) function*. From parts of type P it **observes** attributes of type A.  $attr\_A$  are postulated functions. They express that we can always observe attributes of type A of parts of type P.

<sup>12</sup>Cadastra: A Cadastra is normally a parcel based, and up-to-date land information system containing a record of interests in land (e.g. rights, restrictions and responsibilities). It usually includes a geometric description of land parcels linked to other records describing the nature of the interests, the ownership or control of those interests, and often the value of the parcel and its improvements. See <http://www.fig.net/>

**Q.4.2.2.2 Urban Space Attributes – Narratives and Formalisation** We describe attributes of the domain of urban spaces. As they are, in real life. Not as we may record them or represent them (on paper or within the computer). We can “freely” model that reality as we think it is. If we can talk about and describe it, then it is so! For meteorological attributes it means that we describe precipitation, evaporation, humidity and atmospheric pressure as these physical phenomena “really” are: continuous over time! Similar for all other attributes. Etcetera.

**Q.4.2.2.3 General Form of Attribute Models**

1126. We choose to model the *General Form of Attributes*, such as geodetical, cadastral, geotechnical, meteorological, socio-economic, legal, etcetera, as [continuous] functions from time to maps from points or areas to the specific properties of the attributes.

1127. The points or areas of the properties maps must be in, respectively within, the area of the urban space whose attributes are being specified.

**type**

1126  $GFA = T \rightarrow ((Pt|Area) \rightsquigarrow Properties)$

**value**

1127  $wf\_GFA: GFA \times TUS \rightarrow Bool$

1127  $wf\_GFA(gfa, tus) \equiv$

1127 **let**  $area = attr\_Area(tus)$  **in**

1127  $\forall t:T \bullet t \in \mathcal{D} gfa \Rightarrow$

1127  $\quad \forall pt:Pt \bullet pt \in \mathbf{dom} gfa(t) \Rightarrow is\_Pt\_in\_Area(pt, area)$

1127  $\quad \wedge \forall ar:Area \bullet ar \in \mathbf{dom} gfa(t) \Rightarrow is\_within\_Area(ar, area)$

1127 **end**

$\mathcal{D}$  is a hypothesized function which applies to continuous functions and yield their domain!

**Q.4.2.2.4 Geodetic Attribute[s]**

1128. Geodetic attributes map points to

(a) land elevation and what kind of land it is; and (or) to

(b) normal and current water depths and what kind of water it is.

1129. Geodetic attributes also includes road nets and what kind of roads;

1130. etcetera,

**type**

1128  $GeoD = T \rightarrow (Pt \rightsquigarrow ((Land|Water) \times RoadNet \times \dots))$

1128a  $Land = Elevation \times (Farmland|Urban|Forest|Wilderness|Meadow|Swamp|...)$

1128b  $Water = (NormDepth \times CurrDepth) \times (Spring|Creek|River|Lake|Dam|Sea|Ocean|...)$

1129  $RoadNet = \dots$

1130  $\dots$

**Q.4.2.2.5 Cadastral Attribute[s]** A cadastre is a public register showing details of ownership of the real property in a district, including boundaries and tax assessments.

1131. Cadastral maps shows the boundaries and ownership of land parcels. Some cadastral maps show additional details, such as survey district names, unique identifying numbers for parcels, certificate of title numbers, positions of existing structures, section or lot numbers and their respective areas, adjoining and adjacent street names, selected boundary dimensions and references to prior maps.

1132. Etcetera.

**type**

1131 Cada = T  $\rightarrow$  (Area  $\xrightarrow{m}$  (Owner  $\times$  Value  $\times$  ...))

1132 ...

**Q.4.2.2.6 Geotechnical Attribute[s]**

1133. Geotechnical attributes map points to

- (a) top and lower layer soil etc. composition, by depth levels,
- (b) ground water occurrence, by depth levels,
- (c) gas, oil occurrence, by depth levels,
- (d) etcetera.

**type**

1133 GeoT = (Pt  $\xrightarrow{m}$  Composition)

1133a Composition = VerticalScaleUnit  $\times$  Composite\*

1133b Composite = (Soil|GroundWater|Sand|Gravel|Rock|...|Oil|Gas|...)

1133c Soil,Sand,Gravel,Rock,...,Oil,Gas,... = [chemical analysis]

1133d ...

**Q.4.2.2.7 Meteorological Attribute[s]**

1134. Meteorological information records, for points (of an area) precipitation, evaporation, humidity, etc.;

- (a) precipitation: the amount of rain, snow, hail, etc.; that has fallen at a given place and at the time-stamped moment<sup>13</sup>, expressed, for example, in milimeters of water;
- (b) evaporation: the amount of water evaporated (to the air);
- (c) atmospheric pressure;
- (d) air humidity;
- (e) etcetera.

1134 Met = T  $\rightarrow$  (Pt  $\xrightarrow{m}$  (Precip  $\times$  Evap  $\times$  AtmPress  $\times$  Humid  $\times$  ...))

1134a Precip = MMs [milimeters]

1134b Evap = MMs [milimeters]

1134c AtmPress = MB [milibar]

1134d Humid = Percent

1134e ...

**Q.4.2.2.8 Socio-Economic Attribute[s]**

1135. Socio-economic attributes include time-stamped area sub-attributes:

- (a) income distribution;
- (b) housing situation, by housing category: apt., etc.;
- (c) migration (into, resp. out of the area);

---

<sup>13</sup>– that is within a given time-unit

- (d) social welfare support, by citizen category;
- (e) health status, by citizen category;
- (f) etcetera.

**type**

1135 SocEco = T  $\rightarrow$  (Area  $\overrightarrow{m}$  (Inc $\times$ Hou $\times$ Mig $\times$ SoWe $\times$ Heal $\times$ ...))  
 1135a Inc = ...  
 1135b Hou = ...  
 1135c Mig = {"in", "out"}  $\overrightarrow{m}$  ({"male", "female"}  $\overrightarrow{m}$  (Agegroup  $\times$  Skills  $\times$  HealthSumm  $\times$  ...))  
 1135d SoWe = ...  
 1135e CommHeal = ...  
 1135f ...

**Q.4.2.2.9 Law Attribute[s]: State, Province, Region, City and District Ordinances**

1136. By the law we mean any state, province, region, city, district or other ‘area’ ordinance<sup>14</sup>.

1137. ...

**type**

1136 Law

**value**

1136 attr\_Law: TUS  $\rightarrow$  Law

**type**

1136 Law = Area  $\overrightarrow{m}$  Ordinances

1137 ...

**Q.4.2.2.10 Industry and Business Economics**

TO BE WRITTEN

**Q.4.2.2.11 Etcetera**

TO BE WRITTEN

**Q.4.2.2.12 The Urban Space Attributes – A Summary** Summarising we can model the aggregate of urban space attributes as follows.

1138. Each of these attributes can be given a name.

1139. And the aggregate can be modelled as a map (i.e., a function) from names to appropriately typed attribute values.

**type**

1138 TUS\_Attr\_Nm = {"pts", "ged", "cad", "get", "law", "eco", ...}

1139 TUSm = TUS\_Attr\_Nm  $\overrightarrow{m}$  TUS\_Attr

**axiom**

1139  $\forall$  tusm:TUSm  $\bullet$   $\forall$  nm:TUS\_Attr\_Nm  $\bullet$  nm  $\in$  dom tusm  $\Rightarrow$

1139 **case** (nm, mtusm(nm)) **of**

1139 ("pts", v)  $\rightarrow$  is\_PtS(v), ("ged", v)  $\rightarrow$  is\_GeoD(v), ("cad", v)  $\rightarrow$  is\_CaDa(v),

1139 ("get", v)  $\rightarrow$  is\_GeoT(v), ("law", v)  $\rightarrow$  is\_Law(v), ("eco", v)  $\rightarrow$  is\_Eco(v), ...

1139 **end**

<sup>14</sup>Ordinance: a law set forth by a governmental authority; specifically a municipal regulation: for ex.: A city ordinance forbids construction work to start before 8 a.m.

## Q.4.2.2.13 Discussion

TO BE WRITTEN

## Q.4.3 Scripts

The concept of *scripts* is relevant in the context of *analysers* and *planners*.

By a *script* we shall understand the structured, almost, if not outright, formally expressed, wording of a procedure on how to proceed, one that may have legally binding power, that is, which may be contested in a court of law.

Those who *contract* urban analyses and urban plannings may wish to establish that some procedural steps are taken. Examples are: the vetting of urban space information, the formulation of requirements to what the analysis must contain, the vetting of that and its “quality”, the order of procedural steps, etc. We refer to [53,65].

A[ny] *script*, as implied above, is “like a program”, albeit to be “computed” by humans.

Scripts may typically be expressed in some notation that may include: graphical renditions that, for example, illustrate that two or more independent groups of people, are expected to perform a number of named and more-or-less loosely described actions, expressed in, for example, the technical (i.e., domain) language of urban analysis, respectively urban planning.

The design of urban analysis and of urban planning scripts is an experimental research project with fascinating prospects for further understanding *what urban analysis* and *urban planning* is.

## Q.4.4 Urban Analysis Attributes

1140. Each *analyser* is characterised by a script, and
1141. the set of master and/or derived planner server identifiers – meaning that their “attached” planners might be interested in its analysis results.

**type**

1140 A\_Script = A\_Script<sub>anm<sub>1</sub></sub> | A\_Script<sub>anm<sub>2</sub></sub> | ... | A\_Script<sub>anm<sub>n</sub></sub>

1141 A\_Mer = (MPS.UI|DPS.UI)-set

**value**

1140 attr\_A\_Script: A → A\_Scripts

1141 attr\_A\_Mer: A → A\_Mer

**axiom**

1141  $\forall a:A \bullet a \in ans \Rightarrow attr\_A\_Mer(a) \subseteq ps_{uis}$

## Q.4.5 Analysis Depository Attributes

The purpose of the *analysis depository* is to *accept*, *store* and *distribute* collections of *analyses*; it *accepts* these analysis from the analysers. it *stores* these analyses “locally”; and it *distributes* aggregates of these analyses to *plan servers*.

1142. The *analysis depository* has just one attribute, AHist. It is modeled as a map from *analyser names* to *analysis histories*.
1143. An *analysis history* is a time-ordered sequence, of time stamped analyses, most recent analyses first.

**type**

1142 AHist = ANm  $\xrightarrow{m}$  (s\_T:T × s\_Anal:Anal<sub>anm<sub>i</sub></sub>)\*

**value**

1142 attr\_AHist: AD → AHist

**axiom**

- 1143  $\forall ah:AHist, anm:ANm \bullet anm \in \mathbf{dom} ah \Rightarrow$   
 1143  $\quad \forall i:\mathbf{Nat} \bullet \{i,i+1\} \subseteq \mathbf{inds} ah(anm) \Rightarrow$   
 1143  $\quad \quad s\_T((ah(nm))[i]) > s\_T((ah(nm))[i+1])$

#### Q.4.6 Master Planner Server Attributes

The *planner servers*, whether for *master planners* or *derived planners*, assemble arguments for their associated (i.e., ‘paired’) planners. These arguments include information *auxiliary* to other arguments, such as urban space information for the master planner, and analysis information for all planners; in addition the server also provides *requirements* that are resulting planner plans are expected to satisfy. For every iteration of the planner behaviour the pair of *auxiliary* and *requirements* information is to be renewed and the renewed pairs must somehow “fit” the previously issued pairs.

1144. The *programmable* attributes of the master planner server are those of *aux:AUXiliaries* and *req:REquirements*.
1145. We postulate a predicate function, *fit\_mAux\_mReq*, which takes a pair of pairs auxiliary and requirements arguments, and yields a truth value.

##### type

1144 *mAUX*, *mREQ*

##### value

1144 *attr\_mAUX*: *MPS*  $\rightarrow$  *mAUX*

1144 *attr\_mREQ*: *MPS*  $\rightarrow$  *mREQ*

1145 *fit\_mAUX\_mReq*: (*mAUX* $\times$ *mREQ*) $\times$ (*mAUX* $\times$ *mREQ*)  $\rightarrow$  **Bool**

1145 *fit\_mAUX\_mReq*(*arg\_prev*,*arg\_new*)  $\equiv$  ...

#### Q.4.7 Master Planner Attributes

The *master planner* has the following attributes:

1146. a *master planner script* which is a *static attribute*;
1147. an aggregate of *script “counters”*, a *programmable attribute*; the aggregate designates *pointers* in the *master script* where resumption of *master planning* is to take place in a resumed planning;
1148. a set of *names* of the *analysers* whose analyses the master planner is, or may be interested in, a *static attribute*; and
1149. a set of *identifiers* of the *derived planners* which the master planner may initiate *static attribute*.

##### type

1146 *MP\_Script*

1147 *MP\_Script\_Pt*

1147 *MP\_Script\_Pts* = *MP\_Script\_pt-set*

1148 *ANms* = *ANm-set*

1149 *DPUIs* = *DP\_UI-set*

##### value

1146 *attr\_MP\_Script*: *MP*  $\rightarrow$  *MP\_Script*

1147 *attr\_Script\_Pts*: *MP*  $\rightarrow$  *MP\_Script\_Pts*

1148 *attr\_ANms*: *MP*  $\rightarrow$  *ANms*

1149 *attr\_DPUIs*: *MP*  $\rightarrow$  *DPUIs*

##### axiom

1148 *attr\_ANms*(*mp*)  $\subseteq$  *ANms*

1149 *attr\_DPNms*(*mp*)  $\subseteq$  *DNms*



### Q.4.8 Derived Planner Server Attributes

1150. The *programmable* attributes, of the derived planner servers are those of `aux:AUXiliaries` and `req:REQUIREments`, one each of an indexed set.
1151. We postulate an indexed predicate function, `fit_mAux_mReq`, which takes a pair of pairs auxiliary and requirements arguments, and yields a truth value.

#### type

1144  $dAUX = dAUX_{dnm_1} \mid dAUX_{dnm_2} \mid \dots \mid dAUX_{dnm_p}$

1144  $dREQ = dREQ_{dnm_1} \mid dREQ_{dnm_2} \mid \dots \mid dREQ_{dnm_p}$

#### value

1150  $attr\_dAUX_{dnm_i}: MPS_{dnm_i} \rightarrow dAUX_{dnm_i}$

1150  $attr\_dREQ_{dnm_i}: MPS_{dnm_i} \rightarrow dREQ_{dnm_i}$

1151  $fit\_dAUX\_dReq_{dnm_i\_dReq_{dnm_i}}: (dAUX_{dnm_i} \times dREQ_{dnm_i}) \times (dAUX_{dnm_i} \times dREQ_{dnm_i}) \rightarrow \mathbf{Bool}$

1151  $fit\_dAUX\_dReq_i(arg\_prev_{dnm_i}, arg\_new_{dnm_i}) \equiv \dots$

### Q.4.9 Derived Planner Attributes

1152. a *derived planner script* which is a *static attribute*;
1153. an aggregate of *script "counters"*, a *programmable attribute*; the aggregate designates *points* in the *derived planner script* where resumption of *derived planning* is to take place in a resumed planning;
1154. a set of *identifiers* of the *analysers* whose analyses the master planner is, or may be interested in, a *static attribute*; and
1155. a set of *identifiers* of the *derived planners* which any specific derived planner may initiate, a *static attribute*.

#### type

1152 `DP_Script`

1153 `DP_Script_pt`

1153 `DP_Script_Pts = DP_Script_pt*`

1154 `ANms`

1155 `DNms`

#### value

1152  $attr\_MP\_Script: MP \rightarrow MP\_Script$

1153  $attr\_Script\_Pts: MP \rightarrow Script\_Pts$

1154  $attr\_ANms: MP \rightarrow ANms$

1155  $attr\_DNms: MP \rightarrow DNms$

#### axiom

1154  $attr\_AUIs(mp) \subseteq ANms$

1155  $attr\_DPUIs(mp) \subseteq DNms$

### Q.4.10 Derived Planner Index Generator Attributes

The *derived planner index generator* has two attributes:

1156. the set of all derived planner identifiers (a static attribute), and
1157. a set of already used planner identifiers (a programmable attribute).

#### type

1156 `All_DPUIs = DP_UI-set`

1157 `Used_DPUIs = DP_UI-set`

#### value

1156  $attr\_All\_DPUIs: DPXG \rightarrow All\_DPUIs$

1157  $attr\_Used\_DPUIs: DPXG \rightarrow Used\_DPUIs$

#### axiom

1156  $attr\_All\_DPUIs(dp_xg) = dp_{uis}$

1157  $attr\_Used\_DPUIs(dp_xg) \subseteq dp_{uis}$

### Q.4.11 Plan Repository Attributes

The rôle of the *plan repository* is to keep a record of all master and derived plans. There are two plan repository attributes.

1158. A bijective map between derived planner identifiers and names, and  
 1159. a pair of a list of time-stamped master plans and a map from derived planner names to lists of time-stamped plans, where the lists are sorted in time order, most recent time first.

#### type

1158  $NmUIm = DNm \xrightarrow{\overline{m}} DP\_UI$

1159  $PLANS = ((MP\_UI|DP\_UI) \xrightarrow{\overline{m}} (s\_t:T \times s\_pla:PLA)^*)$

#### value

1158  $attr\_NmUIm: PR \rightarrow NmUIm$

#### axiom

1158  $\forall bm:NmUIm \bullet bm^{-1}(bm) \equiv \lambda x.x$

#### value

1158  $attr\_PLANS: PR \rightarrow PLANS$

#### axiom

1159 **let** plans = attr\_PLANS(*pr*) **in**

1159 **dom** plans  $\subseteq \{mp_{ui}\} \cup dp_{uis}$

1159  $\forall pui:(MP\_UI|DP\_UI) \bullet pui \in \{mp_{ui}\} \cup dp_{uis} \Rightarrow time\_ordered(plans(pui))$

1159 **end**

#### value

1159  $time\_ordered: (s\_t:T \times s\_pla:PLA)^* \rightarrow \mathbf{Bool}$

1159  $time\_ordered(tsl) \equiv \forall i:\mathbf{Nat} \bullet \{i, i+1\} \subseteq inds\ tsl \Rightarrow s\_t(sl(i)) > s\_t(tsl(i+1))$

### Q.4.12 A System Property of Derived Planner Identifiers

Let there be given the set of derived planners *dps*.

1160. The function *reachable identifiers* is the one that calculates all derived planner identifiers reachable from a given such identifier, *dp\_ui:DP\_UI*, in *dps*.
- We calculate the derived planner, *dp:DP*, from *dp\_ui*.
  - We postulate a set of unique identifiers, *uis*, initialised with those that can be in the *attr\_DPUIs(dp)* attribute.
  - Then we recursively calculate the derived planner identifiers that can be reached from any identifier, *ui*, in *uis*.
  - The recursion reaches a fix-point when there are no more identifiers “added” to *uis* in an iteration of the recursion.
1161. A derived planner must not “circularly” refer to itself.

#### value

1160  $reachable\_identifiers: DP\_set \times DP\_UI \rightarrow DP\_UI\_set$

1160  $(dps)(dp\_ui) \equiv$

1160a **let** dp = c\_p(*dps*)(*dp\_ui*) **in**

1160b **let** uis = attr\_DPUIs(dp)  $\cup$

1160c  $\{ui|ui:DP\_UI \bullet ui \in uis \wedge ui \in reachable\_identifiers(dps)(ui)\}$

1160d **in uis end end**

1161  $\forall ui:DP\_UI \bullet ui \in dp_{uis} \Rightarrow ui \notin names(dps)(ui)$

The seeming “endless recursion” ends when an iteration of the *dns* construction and its next does not produce new names for *dns* — a least fix-point has been reached.

## Q.5 The Structure Translators

### Q.5.1 A UNIVERSE OF DISCOURSE Translator

In this section, i.e., all of Sect. Q.5.1, we omit complete typing of behaviours.

1162. The universe of discourse, *uod*, compiles and translates into the of its four elements:

- (a) the translation of the atomic clock, see Item Q.7.1 on page 530,
- (b) the translation of the atomic urban space, see Item Q.7.2 on page 531,
- (c) the compilation of the analyser structure, see Item Q.5.2,
- (d) the compilation of planner structure. see Item Q.5.3,

**value**

```
1162 Translate_UoD(uod) ≡
1162a Translate_CLK(clk),
1162b Translate_TUS(tus),
1162c Translate_AA(obs_AA(uod)),
1162d Translate_PA(obs_PA(uod))
```

The **Translator** apply to, as here, *structures*, or composite parts. The **Translator** apply to atomic parts. In this section, i.e., Sect. Q.5.1, we will explain the obvious meaning of these functions: we will not formalise their type, and we will make some obvious short-cuts.

### Q.5.2 The ANALYSER STRUCTURE Translator

1163. Compiling the analyser structure results in an **AMoL-text** which expresses the separate

- (a) translation of each of its *n* analysers, see Item Q.7.3 on page 533, and
- (b) the translation of the analysis depository, see Item Q.7.4 on page 534.

```
1163 Translate_AA(aa) ≡
1163a { Translate_Aanmi(obs_Aanmi(aa)) | i:[1..n] },
1163b Translate_AD(obs_AD(aa))
```

### Q.5.3 The PLANNER STRUCTURE Translator

1164. The *planner structure*, *pa:PA*, compiles into four elements:

- (a) the compilation of the *master planner structure*, see Item Q.5.3.1 on the following page,
- (b) the translation of the *derived server index generator*, see Item Q.7.5 on page 535,
- (c) the translation of the *plan repository*, see Item Q.7.6 on page 536, and
- (d) the compilation of the *derived server structure*, see Item Q.5.3.2 on the next page.

```
1164 Translate_PA(pa) ≡
1164a Translate_MPA(obs_MPA(pa)),
1164b Translate_DPXG(obs_DPXG(pa)),
1164c Translate_PR(obs_PR(pa)),
1164d Translate_DPA(obs_DPA(pa))
```

### Q.5.3.1 The MASTER PLANNER STRUCTURE Translator

1165. Compiling the *master planner structure* results in an **AMoL-text** which expresses the separate translations of the

- (a) the atomic *master planner server*, see Item Q.7.7 on page 537 and
- (b) the atomic *master planner*, see Item Q.7.8 on page 538.

1165 **Translate\_MPA**(mpa)  $\equiv$   
 1165a **Translate\_MPS**(obs\_MPS(mpa)),  
 1165b **Translate\_MP**(obs\_MP(mpa))

### Q.5.3.2 The DERIVED PLANNER STRUCTURE Translator

1166. The compilation of the *derived planner structure* results in some **AMoL-text** which expresses the set of separate compilations of each of the *derived planner pair structures*, see Item Q.5.3.3.

1166 **Translate\_DPA**(dpa)  $\equiv$  { **Translate**(obs\_DPC<sub>nm<sub>j</sub></sub>(pa)) | j:[1..p] }

### Q.5.3.3 The DERIVED PLANNER PAIR STRUCTURE Translator

1167. The compilation of the *derived planner pair structure* results in some **AMoL-text** which expresses

- (a) the results of translating the *derived planner server*, see Item Q.7.9 on page 541 and
- (b) the results of translating the *derived planner*, see Item Q.7.10 on page 542.

1167 **Translate\_DPC<sub>nm<sub>j</sub></sub>**(dpc<sub>nm<sub>j</sub></sub>), i:[1..p]  $\equiv$   
 1167a **Translate\_DPS<sub>nm<sub>j</sub></sub>**(obs\_DPS<sub>nm<sub>j</sub></sub>(dpc<sub>nm<sub>j</sub></sub>)),  
 1167b **Translate\_DP<sub>nm<sub>j</sub></sub>**(obs\_DP<sub>nm<sub>j</sub></sub>(dpc<sub>nm<sub>j</sub></sub>))

## Q.6 Channel Analysis and Channel Declarations

The *transcendental interpretation* of *parts as behaviours* implies *existence of means of communication & synchronisation* of between and of these behaviours. We refer to Fig. Q.2 on the next page for a summary of the channels of the urban space analysis and urban planning system.

MORE TO COME

### Q.6.1 The clk\_ch Channel

The purpose of the `clk_ch` channel is, for the clock, to propagate Time to such entities who inquire. We refer to Sects. Q.3.1 on page 513, Q.3.2 on page 513, Q.3.3 on page 513, Q.3.5 on page 514, Q.3.6 on page 514, Q.3.7 on page 514 and Q.3.8 on page 514 for the mereologies that help determine the indices for the `clk_ch` channel.

1168. There is declared a (single) channel `clk_ch`

1169. whose messages are of type `CLK_MSG` (for Time).

The `clk_ch` is single. There is no need for enquirers to provide their identification. The clock “freely” dispenses of “its” time.

#### type

1168 `CLK_MSG = T`

#### channel

1169 `clk_ch:CLK_MSG`

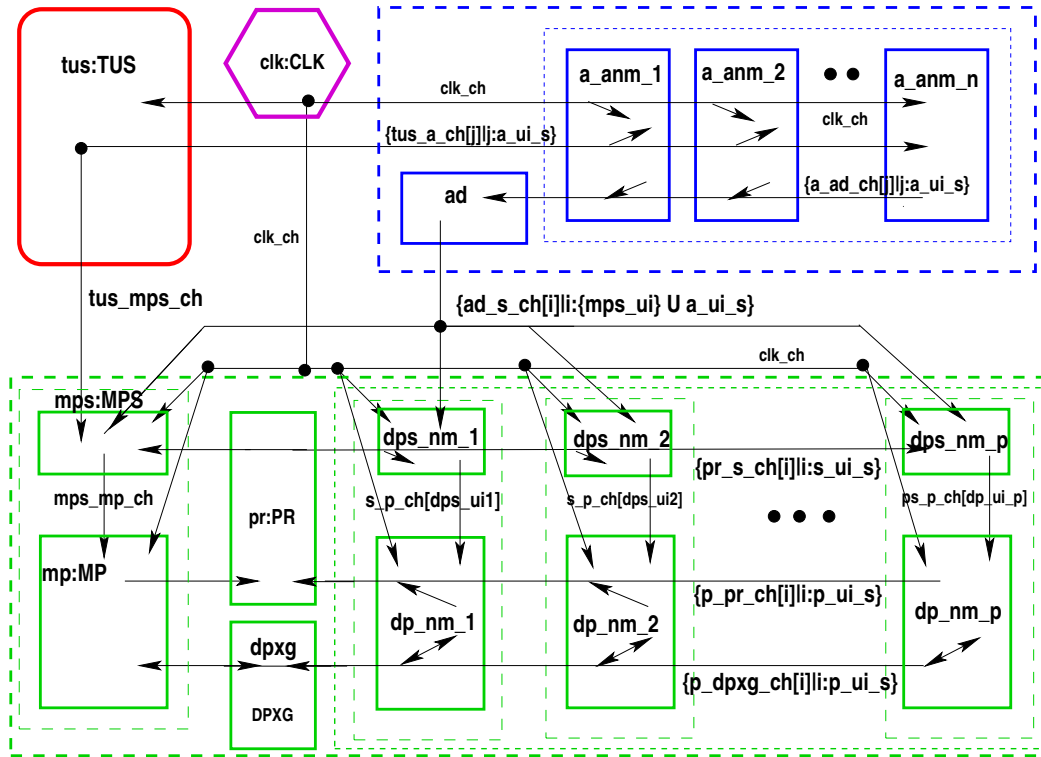


Figure Q.2: The Urban Space and Analysis Channels and Behaviours

### Q.6.2 The tus\_a\_ch Channel

The purpose of the tus\_a\_ch channel is, for the the urban space, to propagate urban space attributes to analysers. We refer to Sects. Q.3.2 and Q.3.3 for the mereologies that help determine the indices for the tus\_a\_ch channel.

1170. There is declared an array channel tus\_a\_ch whose messages are of

1171. type TUS\_MSG (for a *time stamped* aggregate of *urban space attributes*, TUSm, cf. Item 1139 on page 520).

**type**

1171 TUS\_MSG = T × TUSm

**channel**

1170 {tus\_a\_ch[a\_ui]:TUS\_MSG|a\_ui:A-UJ•a\_ui ∈ a<sub>uis</sub>}

The tus\_a\_ch channel is to offer urban space information to all analysers. Hence it is an array channel over indices ANms, cf. Item 1045 on page 505.

### Q.6.3 The tus\_mps\_ch Channel

The purpose of the tus\_mps\_ch channel is, for the the urban space, to propagate urban space attributes to the master planner server. We refer to Sects. Q.3.2 and Q.3.5 for the mereologies that help determine the indices for the tus\_mps\_ch channel.

1172. There is declared a channel tus\_mps\_ch whose messages are of

1171 type TUS\_MSG (for a *time stamped* aggregate of *urban space attributes*, TUSm, cf. Item 1139 on page 520).

**type**

1171 TUS\_MSG = T × TUSm

**channel**

1172 tus\_mps\_ch:TUS\_MSG

The `tus_s_ch` channel is to offer urban space information to just the master server. Hence it is a single channel.

### Q.6.4 The `a_ad_ch` Channel

The purpose of the `a_ad_ch` channel is, for analysers to propagate analysis results to the analysis depository. We refer to Sects. Q.3.3 and Q.3.4 for the mereologies that help determine the indices for the `a_ad_ch` channel.

1173. There is declared a channel `a_ad_ch` whose *time stamped* messages are of

1174. type A\_MSG (for *analysis message*).

**type**

1174 A\_MSG<sub>anm<sub>i</sub></sub> = (s\_T:T × s\_A:Analysis<sub>anm<sub>i</sub></sub>), i:[1:n]

1174 A\_MSG = A\_MSG<sub>anm<sub>1</sub></sub>|A\_MSG<sub>anm<sub>2</sub></sub>|...|A\_MSG<sub>anm<sub>n</sub></sub>

**channel**

1173 {a\_ad\_ch[a\_ui]:A\_MSG|a\_ui:A\_UI•a\_ui ∈ a<sub>uis</sub>}

### Q.6.5 The `ad_s_ch` Channel

The purpose of the `ad_s_ch` channel is, for the analysis depository to propagate histories of analysis results to the server. We refer to Sects. Q.3.4, Q.3.5 and Q.3.7 for the mereologies that help determine the indices for the `ad_s_ch` channel.

1175. There is declared a channel `ad_s_ch` whose messages are of

1176. type AD\_MSG (defined as A\_Hist for a *histories of analyses*), see Item 1142 on page 521.

**type**

1176 AD\_MSG = A\_Hist

**channel**

1175 {ad\_s\_ch[s\_ui]|s\_ui:(MPS\_UI|DPS\_UI)•s\_ui ∈ {mps<sub>ui</sub>} ∪ dps<sub>ui</sub>}:AD\_MSG

The `ad_s_ch` channel is to offer urban space information to the *master* and *derived servers*. Hence it is an array channel.

### Q.6.6 The `mps_mp_ch` Channel

The purpose of the `mps_mp_ch` channel is for the master server to propagate comprehensive master planner input to the master planner. We refer to Sects. Q.3.5 and Q.3.6 for the mereologies that help determine the indices for the `mps_mp_ch` channel.

1177. There is declared a channel `mps_mp_ch` whose messages are of

1178. type MPS\_MSG which are quadruplets of time stamped urban space information, TUS\_MSG, see Item 1171 on the previous page, analysis histories, A\_Hist, see Item 1176, *master planner auxiliary* information, mAUX, and *master plan requirements*, mREQ.

**type**

1178 MPS\_MSG = TUS\_MSG × AD\_MSG × mAUX × mREQ

**channel**

1177 mps\_mp\_ch:MPS\_MSG

The mps\_mp\_ch channel is to offer MPS\_MSG information to just the *master server*. Hence it is a single channel.

**Q.6.7 The p\_pr\_ch Channel**

The purpose of the p\_pr\_ch channel is, for master and derived planners to deposit and retrieve master and derived plans to the plan repository. We refer to Sects. Q.3.6 and Q.3.10 for the mereologies that help determine the indices for the p\_pr\_ch channel.

1179. There is declared a channel p\_pr\_ch whose messages are of

1180. type PLAN\_MSG – for *time stamped master plans*.**type**

1180 PLAN\_MSG = T × PLANS

**channel**1179 {p\_pr\_ch[p\_ui]:PLAN\_MSG|p\_ui:(MP\_UI|DP\_UI)•p\_ui ∈ p<sub>uis</sub>}

The p\_pr\_ch channel is to offer comprehensive records of all current plans to all the the *planners*. Hence it is an array channel.

**Q.6.8 The p\_dpxg\_ch Channel**

The purpose of the p\_dpxg\_ch channel is, for planners to request and obtain derived planner index names of, respectively from the derived planner index generator. We refer to Sects. Q.3.6 and Q.3.9 for the mereologies that help determine the indices for the mp\_dpxg\_ch channel.

1181. There is declared a channel p\_dpxg\_ch whose messages are of

1182. type DPXG\_MSG. DPXG\_MSG messages are

- (a) either *request* from the *planner* to the *index generator* to provide zero, one or more of an indicated set of *derived planner names*,
- (b) or to accept such a (*response*) set from the *index generator*.

**type**

1182 DPXG\_MSG = DPXG\_Req | DPXG\_Rsp

1182a DPXG\_Req :: DNm-set

1182b DPXG\_Rsp :: DNm-set

**channel**1181 {p\_dpxg\_ch[ui]:DPXG\_MSG|ui:(MP\_UI|DP\_UI)•ui ∈ p<sub>uis</sub>}**Q.6.9 The pr\_s\_ch Channel**

The purpose of the pr\_s\_ch channel is, for the plan repository to provide master and derived plans to the derived planner servers. We refer to Sects. Q.3.10 and Q.3.7 for the mereologies that help determine the indices for the pr\_dps\_ch channel.

1183. There is declared a channel pr\_dps\_ch whose messages are of

1184. type PR\_MSGd, defined as PLAp, cf. Item 1159 on page 524.

**type**

1184 PR\_MSG = PLANS

**channel**

1183 {pr\_s\_ch[ui]:PR\_MSGd|ui:(MPS\_UI|DPS\_UI)•ui ∈ *s<sub>uis</sub>*}

### Q.6.10 The dps\_dp\_ch Channel

The purpose of the `dps_dp_ch` channel is, for derived planner servers to provide input to the derived planners. We refer to Sects. Q.3.7 and Q.3.8 for the mereologies that help determine the indices for the `dps_dp_ch` channel.

1185. There is declared a channel `dps_dp_ch[ui_nm_j]`, one for each *derived planner* pair.

1186. The channel messages are of type `DPS_MSGnmj`. These `DPS_MSGnmi` messages are quadruplets of *analysis* aggregates, `AD_MSG`, *urban plan* aggregates, `PLANS`, *derived planner auxiliary information*, `dAUXnmj`, and *derived plan requirements*, `dAUXnmj`.

**type**

1186 `DPS_MSGnmj` = `AD_MSG` × `PLANS` × `dAUXnmj` × `dREQnmj`, *j*: [1..p]

**channel**

1185 {dps\_dp\_ch[ui]:DPS\_MSG<sub>nm<sub>j</sub></sub>|ui:DPS\_UI•ui ∈ *dps<sub>uis</sub>*}

## Q.7 The Atomic Part Translators

### Q.7.1 The CLOCK Translator

We refer to Sect. Q.4.1.2 for the attributes that play a rôle in determining the clock signature.

#### Q.7.1.1 The Translate\_CLK Function

1187. The `Translate_CLK(clk)` results in three text elements:

- (a) the **value** keyword,
- (b) the *signature* of the clock definition,
- (c) and the *body* of that definition.

The clock signature contains the *unique identifier* of the clock; the *mereology* of the clock, cf. Item Q.3.1 on page 513; and the *attributes* of the clock, in some form or another: the programmable time attribute and the channel over which the clock offers the time.

**value**

1187 `Translate_CLK(clk)` ≡

1187a " **value**

1187b clock: T → **out** `clk_ch` **Unit**

1187c `clock(uid_CLK(clk),mereo_CLK(clk))(attr_T(clk))` ≡ ... "



**Q.7.1.2 The clock Behaviour**

The purpose of the clock is to show the time. The “players” that need to know the time are: the urban space when informing requestors of aggregates of urban space attributes, the analysers when submitting analyses to the analysis depository, the planners when submitting plans to the plan repository.

1188. We see the clock as a behaviour.

1189. It takes a programmable input, the *current* time,  $t$ .

1190. It repeatedly emits the some *next* time on channel `clk_ch`.

1191. Each iteration of the clock it non-deterministically, internally increments the *current* time by either nothing or an infinitesimally small time interval  $\delta t_i$ , cf. Item 1113 on page 516.

1192. In each iteration of the clock it either offers this *next* time, or skips doing so;

1193. whereupon the clock resumes being the clock albeit with the new, i.e., *next* time.

**value**

```

1190 $\delta t_i: T_I = \dots$ cf. Item 1113 on page 516
1188 clock: T \rightarrow out clk_ch Unit
1189 clock(uid_clk, mereo_clk)(t) \equiv
1191 let t' = (t + δt_i) \square t in
1192 skip \square clk_ch!t' ;
1193 clock(uid_clk, mereo_clk)(t') end
1193 pre: uid_clk = clk_{ui} \wedge
1193 mereo_clk = ($tus_{ui}, a_{uis}, mps_{ui}, mp_{ui}, dps_{uis}, dp_{uis}$)

```

**Q.7.2 The URBAN SPACE Translator**

We refer to Sect. Q.4.2.2 for the attributes that play a rôle in determining the urban space signature.

**Q.7.2.1 The Translate\_TUS Function**

1194. The `Translate_TUS( $tus$ )` results in three text elements:

- (a) the **value** keyword
- (b) the *signature* of the `urb_spa` definition,
- (c) and the *body* of that definition.

The urban space signature contains the *unique identifier* of the urban space, the *mereology* of the urban space, cf. Item Q.3.2 on page 513, the static point space *attribute*.

**value**

```

1194 Translate_TUS(tus) \equiv
1194a " value
1194b urb_spa: TUS_UI \times TUS_Mer \rightarrow Pts \rightarrow
1194b out ... Unit
1194c urb_spa(uid_TUS(tus), mereo_TUS(tus))(attr_Pts(tus)) \equiv ... "

```

We shall detail the `urb_spa` signature and the `urb_spa` body next.

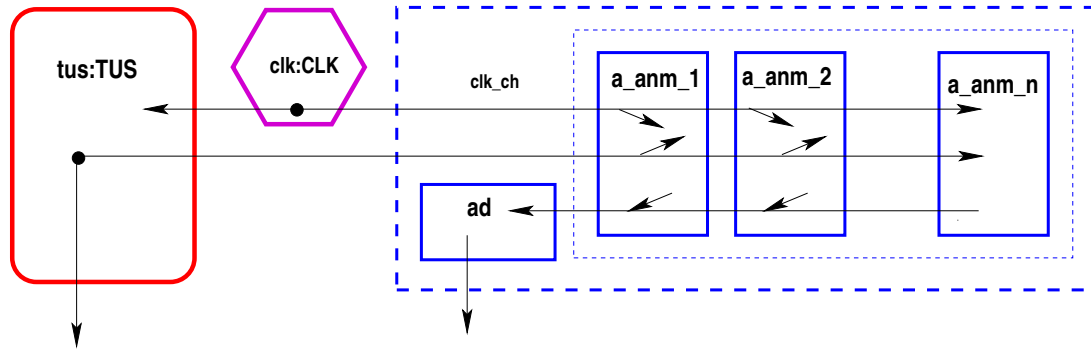


Figure Q.3: The Urban Space and Analysis Behaviours

### Q.7.2.2 The `urb_spa` Behaviour

The urban space can be seen as a behaviour. It is “visualized” as the rounded edge box to the left in Fig. Q.3. It is a “prefix” of Fig. Q.3. In this section we shall refer to many other elements of our evolving specification. To grasp the seeming complexity of the urban space, its analyses and its urban planning functions, we refer to Fig. Q.3.

1195. To every observable part, like `tus:TUS`, there corresponds a behaviour, in this case, the `urb_spa`.
1196. The `urb_spa` behaviour has, for this report, just one static attribute, the point space, `Pts`.
1197. The `urb_spa` behaviour has the following biddable and programmable attributes, the `Cadastral`, the `Law` and the `SocioEconomic` attributes. The biddable and programmable attributes “translate” into behaviour parameters.
1198. The `urb_spa` behaviour has the following dynamic, non-biddable, non-programmable attributes, the `GeoDetic`, `GeoTechnic` and the `Meterological` attributes. The non-biddable, non-programmable dynamic attributes “translate”, in the conversion from parts to behaviours, to input channels etc.

the `urb_spa` behaviour offers its attributes, upon demand,

1199. to `a` urban space analysis behaviours, `tus_ana_i` and one master urban server.
1200. The `urb_spa` otherwise behaves as follows:
- (a) it repeatedly “assembles” a tuple, `tus`, of all attributes;
  - (b) then it external non-deterministically either offers the `tus` tuple
  - (c) to either any of the urban space analysis behaviours,
  - (d) or to the master urban planning behaviour;
  - (e) in these cases it resumes being the `urb_spa` behaviour;
  - (f) or internal-non-deterministically chooses to
  - (g) update the `law`, the `cadastral`, and the `socio-economic` attributes;
  - (h) whereupon it resumes being the `urb_spa` behaviour.

#### channel

- 1198 `attr_Pts_ch:Pts`, `attr_GeoD_ch:GeoD`, `attr_GeoT_ch:GeoT`, `attr_Met_ch:Met`  
 1199 `tus_mps_ch:TUSm`

```

1199 {tus_a_ch[ai]|ai ∈ a_uis}:TUSm
value
1195 urb_spa: TUS_UI × TUS_Mer →
1196 Pts →
1197 (Cada×Law×Soc_Eco×...) →
1198 in attr_Pts_ch, attr_GeoD_ch, attr_GeoT_ch, attr_Met_ch →
1199 out tus_mps_ch, {tus_ana_ch[ai]|ai ∈ [a_1...a_a]} → Unit
1200 urb_spa(pts)(pro) ≡
1200a let geo = ["pts" ↦ attr_Pts_ch?"ged" ↦ attr_GeoD_ch?"cad" ↦ cada,"get" ↦ attr_geoT_ch?,
1200a "met" ↦ attr_Met_ch?"law" ↦ law,"eco" ↦ eco,...] in
1200c (([] {tus_a_ch[ai]|geo|ai ∈ a_uis}
1200b []
1200d tus_mps_ch!geo) ;
1200e urb_spa(pts)(pro) end
1200f []
1200g let pro':(Cada×Law×Soc_Eco×...)*fit_pro(pro,pro') in
1200h urb_spa(pts)(pro') end

1200g fit_pro: (Cada×Law×Soc_Eco×...) × (Cada×Law×Soc_Eco×...) → Bool

```

We leave the *fitness predicate* `fit_pro` further undefined. It is intended to ensure that the biddable and programmable attributes evolve in a commensurate manner.

### Q.7.3 The ANALYSER<sub>anm<sub>i</sub></sub>, *i*:[1 : *n*] Translator

We refer to Sect. Q.4.4 for the attributes that play a rôle in determining the analyser signature.

#### Q.7.3.1 The Translate<sub>A<sub>anm<sub>j</sub></sub></sub> Function

1201. The `TranslateAanmj`(*a<sub>anm<sub>j</sub></sub>*) results in three text elements:

- (a) the **value** keyword,
- (b) the *signature* of the analyser<sub>*a<sub>anm<sub>j</sub></sub>*</sub> definition,
- (c) and the *body* of that definition.

The analyser<sub>*a<sub>anm<sub>j</sub></sub>*</sub> signature contains the *unique identifier* of the analyser, the *mereology* of the analyser, cf. Item Q.3.3 on page 513, and the *attributes*, here just the programmable attribute of the most recent analysis<sub>*a<sub>anm<sub>j</sub></sub>*</sub> performed by the analyser<sub>*a<sub>anm<sub>j</sub></sub>*</sub>.

#### type

1201 Analysis = Analysis<sub>*nm<sub>1</sub>*</sub>|Analysis<sub>*nm<sub>2</sub>*</sub>|...|Analysis<sub>*nm<sub>n</sub>*</sub>

#### value

```

1201 TranslateAnmi(anmi):
1201 " value
1201 analysernmi: (uid_A×mereo_A) →
1201 Analysisnmi →
1201 in tus_a_ch[uid_A(anmi)]
1201 out a_ad_ch[uid_A(anmi)]
1201 analyseruij(uid_A(anmi),mereo_A(anmi))(ananmi) ≡ ... "
```

### Q.7.3.2 The analyser<sub>uij</sub> Behaviour

Analyses, or various kinds, of the urban space, is an important prerequisite for urban planning. We therefore introduce a number,  $n$ , of urban space analysis behaviours,  $\text{analysis}_{anm_i}$  (for  $anm_i$  in the set  $\{anm_1, \dots, anm_a\}$ ). The indexing designates that each  $\text{analysis}_{anm_i}$  caters for a distinct kind of urban space analysis, each analysis with respect to, i.e., across existing urban areas: ...,  $(a_i)$  traffic statistics,  $(a_j)$  income distribution, ...,  $(a_k)$  health statistics,  $(a_\ell)$  power consumption, ...,  $(a_a)$  ... . We shall model, by an indexed set of behaviours,  $\text{ana}_i$ , the urban [space] analyses that are an indispensable prerequisite for urban planning.

1202. Urban [space] analyser,  $\text{tus\_ana}_i$ , for  $a_i \in [a_1 \dots a_a]$ , performs analysis of an urban space whose attributes, except for its point set, it obtains from that urban space – via channel  $\text{tus\_ana\_ch}$  and
1203. offers analysis results to the  $\text{mp\_beh}$  and the  $n$  derived behaviours.
1204. Urban analyser,  $\text{ana}_{a_i}$ , otherwise behaves as follows:
- The analyser obtains, from the urban space, its most recent set of attributes.
  - The analyser then proceeds to perform the specific analysis as “determined” by its index  $a_i$ .
  - The result,  $\text{tus\_ana}_{a_i}$ , is communicated whichever urban, the master or the derived, planning behaviour inquires.
  - Whereupon the analyser resumes being the analyser, improving and/or extending its analysis.

#### type

1201  $\text{Analysis} = \text{Analysis}_{anm_1} | \text{Analysis}_{anm_2} | \dots | \text{Analysis}_{anm_n}$

#### value

```

1204 analysernmi(a_ui, a_mer)(analysisnmi) ≡
1204a let tsm = tus_a_ch[a_ui] ? in
1204b let analysis'nmi = perform_analysisnmi(tsm)(analysis) in
1204c [] a_ad_ch[a_ui] ! (clk_ck?, analysis'nmi) ;
1204d analyseri(a_ui, a_mer)(analysis'nmi) end end

1204b perform_analysisanm_i: TUSm → Analysisanm_i → Analysisanm_i
1204b perform_analysisanm_i(tsm)(analysisanm_i) ≡ ...

```

## Q.7.4 The ANALYSIS DEPOSITORY Translator

We refer to Sect. Q.4.5 for the attributes that play a rôle in determining the analysis depository signature.

### Q.7.4.1 The Translate\_AD Function

1205. The  $\text{Translate\_AD}(ad)$  results in three text elements:
- the **value** keyword
  - the *signature* of the  $\text{ana\_dep}$  definition,
  - and the *body* of that definition.

The  $\text{ana\_dep}$  signature essentially contains the *unique identifier* of the analyser, the *mereology* of the analyser, cf. Item Q.3.4 on page 513, and the *attributes*, in one form or another: the programmable attribute,  $\text{a\_hist}$ , see Item 1142 on page 521, the channels over which  $\text{ana\_dep}$  either accepts time stamped *analyses*,  $\text{Analysis}_{a_{ui}}$ , from  $\text{analyser}_{anm_i}$ , or offers  $\text{a\_hists}$  to either the *master planner server* or the *derived planner servers*.

```

value
1205 Translate_AD(ad) ≡
1205a " value
1205b ana_dep: (A_UI × A_Mer) → AHist →
1205b in {a_ad_ch[i]|i:A_UI•i ∈ a_uis}
1205b out {ad_s_ch[i]|i:A_UI•i ∈ s_uis} Unit
1205c ana_dep(ui_A(ad),mereo_A(ad))(attr_AHist(ad)) ≡ ... "
```

#### Q.7.4.2 The **ana\_dep** Behaviour

The definition of the *analysis depository* is as follows.

1206. The behaviour of **ana\_dep** is as follows: non-deterministically, externally ( $\square$ ), **ana\_dep**
1207. either ( $\square$ , line 1209) offers to accept a time stamped analysis *from some* analyser ( $\square\{ \dots | \dots \}$ ),
- (a) receiving such an analyses it “updates” its history,
  - (b) and resumes being the **ana\_dep** behaviour with that updated history;
1208. or offers the analysis history *to the* master planner server and resumes being the **ana\_dep** behaviour;
1209. or offers the analysis history
- (a) *to whichever* ( $\square\{ \dots | \dots \}$ ) planner server offers to accept a history
  - (b) and resumes being the **ana\_dep** behaviour with that updated history.

```

value
1206 ana_dep(a_ui,a_mer)(ahist) ≡
1207 \square { (let ana = a_ad_ch[i] ? in
1207a let ahist' = ahist†[i→⟨ana⟩^(ahist(i))] in
1207b ana_dep(a_ui,a_mer)(ahist') end end
1207b | i:A_UI•i ∈ a_uis }
1208 \square (ad_mps_ch!ahist ; ana_dep(a_ui,a_mer)(ahist))
1209 \square
1209a ({ ad_s_ch[j]!ahist
1209a | j:(MPS_UI|DPS_UI)•j ∈ s_uis};
1209b ana_dep(a_ui,a_mer)(ahist))
```

### Q.7.5 The DERIVED PLANNER INDEX GENERATOR Translator

We refer to Sect. Q.4.10 for the attributes that play a rôle in determining the derived planner index generator signature.

#### Q.7.5.1 The **Translate\_DPXG**(*dpxg*) Function

1210. The **Translate\_DPXG**(*dpxg*) results in three text elements:
- (a) the **value** keyword
  - (b) the *signature* of the *dpxg* behaviour definition,
  - (c) and the *body* of that definition.

The signature of the `dpxg` behaviour definition has many elements: the *unique identifier* of the `dpxg` behaviour, the *mereology* of the `dpxg` behaviour, cf. Item Q.3.9 on page 515, and the *attributes* in some form or another: the *unique identifier*, the *mereology*, and the *attributes*, in some form or another: the programmable attribute `All_DPUIs`, cf. Item 1156 on page 523, the programmable attribute `Used_DPUIs`, cf. Item 1157 on page 523, the `mp_dpxg_ch` input/output channel, and the `dp_dpxg_ch` input/output array channel.

**value**

```

1210 Translate_DPXG(dpxg) ≡
1210a " value
1210b dpxg_beh: (DPXG_UI × DPXG_Mer) →
1210b (All_DPUIs × Used_DPUIs) →
1210b in,out {p_dpxg_ch[i]|i:(MP_UI|DP_UI)•i∈puis} Unit
1210c dpxg_beh(uid_DPXG(dpxg),mereo_DPXG(dpxg))(all_dpui,used_dpui) ≡ ... "
```

**Q.7.5.2 The dpxg Behaviour**

1211. The index generator otherwise behaves as follows:

- (a) It non-deterministically, externally, offers to accept requests from any planner, whether master or server. The request suggests the names, `req`, of some derived planners.
- (b) The index generator then selects a suitable subset, `sel_dpui`, of these suggested derived planners from those that are yet to be started.
- (c) It then offers these to the requesting planner.
- (d) Finally the index generator resumes being an index generator, now with an updated `used_dpui` programmable attribute.

**value**

```

1211 dpxg: (DPXG_UI × DPXG_Mer) → (All_DPUIs × Used_DPUIs) →
1211 in,out mp_dpxg_ch,
1211 {p_dpxg_ch[j]|j:(MP_UI|DP_UI)•j∈{puis}} Unit
1211 dpxg(dpxg_ui,dpxg_mer)(all_dpui,used_dpui) ≡
1211a [] { let req = p_dpxg_c[j] ? in
1211b let sel_dpui = all_dpui \ used_dpui • sel_dpui ⊆ req_dpui in
1211c dp_dpxg_ch[j] ! sel_dpui ;
1211d dpxg(dpxg_ui,dpxg_mer)(all_dpui,used_dpui|sel_dpui) end end
1211 | j:(MP_UI|DP_UI)•j∈puis }
```

**Q.7.6 The PLAN REPOSITORY Translator**

We refer to Sect. Q.4.11 for the attributes that play a rôle in determining the plan repository signature.

**Q.7.6.1 The Translate\_PR Function**

1212. The `Translate_PR(pr)` results in three text elements:

- (a) the **value** keyword,
- (b) the *signature* of the plan repository definition,
- (c) and the *body* of that definition.

The plan repository signature contains the *unique identifier* of the plan repository, the *mereology* of the plan repository, cf. Item Q.3.10 on page 515, and the *attributes*: the *programmable* plans, cf. 1159 on page 524, and the *input/output channel* `p_pr_ch`.

**value**

```

1212 Translate_PR(pr) ≡
1212a " value
1212b plan_rep: PLANS →
1212b in {p_pr_ch[i]|i:(MP_UI|DP_UI)•i∈puis}
1212b out {s_pr_ch[i]|i:(MP_UI|DP_UI)•i∈suis} Unit
1212c plan_rep(plans)(attr_AllDPUIs(pr),attr_UsedDPUIs(pr)) ≡ ... "
```

### Q.7.6.2 The plan\_rep Behaviour

1213. The plan repository behaviour is otherwise as follows:

- (a) The plan repository non-deterministically, externally chooses between
  - i. offering to accept time-stamped plans from a planner,  $p_{ui}$ , either the master planner or anyone of the derived planners,
  - ii. from whichever planner so offers,
  - iii. inserting these plans appropriately, i.e., at  $p_{ui}$ , as the new head of the list of “there”,
  - iv. and then resuming being the plan repository behaviour appropriately updating its programmable attribute;
- (b) or
  - i. offering to provide a full copy of its plan repository map
  - ii. to whichever server requests so,
  - iii. and then resuming being the plan repository behaviour.

**value**

```

1213a plan_rep(pr_ui,ps_uis)(plans) ≡
1213(a)i [] { let (t,plan) = p_pr_ch[i] ? in assert: i ∈ dom plans
1213(a)iii let plans' = plans † [i→((t,plan))^plans(i)] in
1213(a)iv plan_rep(pr_ui,ps_uis)(plans') end end
1213(a)ii | i:(MP_UI|DP_UI)•i∈puis }
1213b []
1213(b)i [] { s_pr_ch[i] ! plans ; assert: i ∈ dom plans
1213(b)iii plan_rep(pr_ui,ps_uis)(plans)
1213(b)ii | i:(MP_UI|DP_UI)•i∈puis }
```

## Q.7.7 The MASTER SERVER Translator

We refer to Sect. Q.4.6 for the attributes that play a rôle in determining the master server signature.

### Q.7.7.1 The Translate\_MPS Function

1214. The `Translate_MPS(mps)` results in three text elements:

- (a) the **value** keyword,
- (b) the *signature* of the `master_server` definition,
- (c) and the *body* of that definition.

The `master_server` signature contains the *unique identifier* of the master server, the *mereology* of the master server, cf. Item Q.3.5 on page 514, and the *dynamic attributes* of the master server: the most recently, previously produced *auxiliary* information, the most recently, previously produced plan *requirements* information, the clock channel, the urban space channel, the analysis depository channel, and the master planner channel.

**value**

```

1214 Translate_MPS(mps) ≡
1214a " value
1214b master_server: (mAUX×mREQ) →
1214b in clk_ch, tus_m_ch, ad_s_ch[uid_MPS(mps)]
1214b out mps_mp_ch Unit
1214c master_server(uid_MPS(mps),mereo_MPS(mps))(attr_mAUX(mps),attr_mREQ(mps)) ≡ ... "
```

**Q.7.7.2 The master\_server Behaviour**

1215. The `master_server` obtains time from the clock, see Item 1216c, information from the urban space, and the most recent analysis history, assembles these together with “locally produced”

- (a) *auxiliary* planner information and
- (b) plan *requirements*

as input, `MP_ARG`, to the master planner.

1216. The master server otherwise behaves as follows:

- (a) it obtains latest urban space information and latest analysis history, and
- (b) then produces auxiliary planning and plan requirements commensurate, i.e., fit, with the most recently, i.e., previously produced such information;
- (c) it then offers a time stamped compound of these kinds of information to the master planner,
- (d) whereupon the master server resumes being the master server, albeit with updated programmable attributes.

**type**

1215a mAUX

1215b mREQ

1215 mARG = (T × ((mAUX × mREQ) × (TUSm × AHist)))

**value**

1216 master\_server(uid,mereo)(aux,req) ≡

1216a let tusm = tus\_m\_ch ? , ahist = ad\_s\_ch[mps\_ui] ? ,

1216b maux:mAUX, mreq:mREQ • fit\_AuxReq((aux,req),(maux,mreq)) in

1216c s\_p\_ch[uid] ! (clk\_ch?,((maux,mreq),(tusm,ahist))) ;

1216d master\_server(uid,mereo)(maux,mreq)

1216 end

1216b fitAuxReq: (mAUX×mREQ)×(mAUX×mREQ) → Bool

1216b fitAuxReq((aux,req),(maux,mreq)) ≡ ...

**Q.7.8 The MASTER PLANNER Translator**

We refer to Sect. Q.4.7 for the attributes that play a rôle in determining the master planner signature.

**Q.7.8.1 The Translate\_MP Function**

1217. The `Translate_MP(mp)` results in three text elements:

- (a) the **value** keyword,



- (b) the *signature* of the `master_planner` definition,
- (c) and the *body* of that definition.

The `master_planner` signature contains the *unique identifier* of the master planner, the *mereology* of the master planner, cf. Item Q.3.6 on page 514, and the *attributes* of the master planner: the *script*, cf. Sect. Q.4.3 on page 521 and Item 1140 on page 521, a set of *script pointers*, cf. Item 1147 on page 522, a set of *analyser names*, cf. Item 1148 on page 522, a set of *planner identifiers*, cf. Item 1149 on page 522, and the channels as implied by the master planner mereology.

**value**

```

1217 Translate_MP(mp) ≡
1217a " value
1217b master_planner: $M_{mp_{ui}}:P_UI \times MP_Mer \times (Script \times ANms \times DPUIs) \rightarrow$
1217b Script_Pts \rightarrow
1217b in clk_ch, mps_mp_ch, ad_ps_ch[mpui]
1217b out p_pr_ch[mpui]
1217b in,out p_dp_xg_ch[mpui] Unit
1217c master_planner(uid_MP(mp), mereo_MP(mp),
1217c (attr_Script(mp), attr_ANms(mp), attr_DPUIs(mp)))(attr_Script_Ptrs(mp)) ≡ ... "
```

**Q.7.8.2 The Master urban\_planning Function**

1218. The core of the `master_planner` behaviour is the `master_urban_planning` function.
1219. It takes as arguments: the *script*, a set of *analyser names*, a set of *derived planner identifiers*, a set of *script pointers*, and the time-stamped master planner argument, cf. Item 1215 on the facing page;
1220. and delivers, i.e., yields, a set of “remaining” *derived planner identifiers*, an updated set of *script pointers*, and a *master result*: `M_RES`, i.e., a *master plan*, `mp`: `M_PLAN` together with the time stamped master argument from which the plan was constructed.
1221. The `master_urban_planning` function is not defined by other than a predicate:
- (a) the “remaining” *derived planner identifiers* is a subset of the arguments *derived planner identifiers*;
  - (b) the “resulting” *master argument* is the same as the input *master argument*, i.e., it is “carried forward”;
  - (c) the arguments: the *script*, the *analyser names*, the *derived planner identifiers*, the set of *script pointers*, the time-stamped *master planner argument*, and the result plan otherwise satisfies a predicate  $\mathcal{P}(\text{script}, \text{anms}, \text{dpuis}, \text{ptrs}, \text{marg})(\text{mplan})$  expressing that the result `mplan` is an appropriate plan in view of the other arguments.

**type**

```

1220 M_PLAN
1220 M_RES = M_PLAN × DPUI-set × M_ARG
```

**value**

```

1219 master_urban_planning:
1219 Script × ANm-set × DP_UI-set × Script_Ptr-set × M_ARG
1220 → (DP_UI-set × Script_Ptr-set) × M_RES
1218 master_urban_planning(script, anms, dpuis, ptrs, marg)
1221a as ((dpuis', ptrs'), (mplan, marg'))
1221a dpuis' ⊆ dpuis
1221b ∧ marg' = marg
```

```

1221c $\wedge \mathcal{P}(\text{script}, \text{anms}, \text{dpuis}, \text{ptrs}, \text{marg})(\text{mplan})$
1218 $\mathcal{P}: ((\text{Script} \times \text{ANM-set} \times \text{DP_UI-set} \times \text{Script_Ptr-set} \times \text{M_ARG} \times \text{MPLAN} \times \text{Script_Ptr-set})$
1218 $\times (\text{DP_UI-set} \times \text{Script_Ptr-set} \times \text{M_ARG} \times \text{MPLAN})) \rightarrow \mathbf{Bool}$
1218 $\mathcal{P}((\text{script}, \text{anms}, \text{dpuis}, \text{ptrs}, \text{marg}, \text{mplan}, \text{ptrs}), (\text{dpuis}', \text{ptrs}', \text{marg}, \text{mplan})) \equiv \dots$

```

### Q.7.8.3 The master\_planner Behaviour

1222. The master\_planner behaviours is otherwise as follows:

- (a) The master\_planner obtains, from the master server, its time stamped master argument, cf. Item 1215 on page 538;
- (b) it then invokes the master urban planning function;
- (c) the time-stamped result is offered to the plan repository;
- (d) if the result is OK as a final result,
- (e) then the behaviour is stopped;
- (f) otherwise
  - i. the master planner inquires the derived planner index generator as for such derived planner identifiers which are not used;
  - ii. the master planner behaviour is resumed with the appropriately updated programmable script pointer attribute, in parallel with
  - iii. the distributed parallel composition of the parallel behaviours of the derived servers
  - iv. and the derived planners
  - v. designated by the derived planner identifiers transcribed into (*nm\_dps\_ui*) derived server, respectively into (*nm\_dp\_ui*) derived planner names. For these transcription maps we refer to Sect. Q.2.12 on page 511, Item 1097 on page 512.

#### value

```

1222 master_planner(uid, mereo, (script, anms, puis))(ptrs) \equiv
1222a let (t, ((maux, mreq), (tasm, ahist))) = mps_mp_ch ? in
1222b let ((dpuis', ptrs'), mres) = master_urban_planning(script, anms, dpuis, ptrs) in
1222c p_pr_ch[uid] ! mres ;
1222d if completed(mres) assert: ptrs' = {}
1222e then init_der_serv_planrs(uid, dpuis')
1222f else
1222(f)i init_der_serv_planrs(ui, dpuis)
1222(f)ii || master_planner(uid, mereo, (script, anms, puis))(ptrs')
1222 end end end

```

### Q.7.8.4 The initiate derived servers and derived planners Behaviour

The init\_der\_serv\_planrs behaviour plays a central rôle. The outcome of the urban planning functions, whether for master or derived planners, result in a possibly empty set of derived planner identifiers, dpuis. If empty then that shall mean that the planner, in the iteration, of the planner behaviour is suggesting that no derived server/derived planner pairs are initiated. If dpuis is not empty, say consists of the set  $\{dp_{ui_i}, dp_{ui_j}, \dots, dp_{ui_k}\}$  then the planner behaviour is suggesting that derived server/derived planner pairs whose planner element has one of these unique identifiers, be appropriately initiated.

1223. The init\_der\_serv\_planrs behaviour takes the unique identifier, uid, of the “initiate issuing” planner and a suggested set of derived planner identifiers, dpuis.

1224. It then obtains, from the *derived planner index generator*, `dp_xg`, a subset, `dpuis'`, that may be equal to `dpuis`.

It then proceeds with the parallel initiation of

1225. derived servers (whose names are extracted, `extr_Nm`, from their identifiers, cf. Item 1091 on page 511),

1226. and planners (whose names are extracted, `extr_Nm`, from their identifiers, cf. Item 1092 on page 511)

1227. for every `dp_ui` in the set `dpuis'`.

However, we must first express the selection of appropriate arguments for these server and planner behaviours.

1228. The selection of the server and planner parts, making use of the identifier to part mapping `nms_dp_ui` and `nm_dp_ui`, cf. Items 1097– 1098 on page 512;

1229. the selection of respective identifiers,

1230. mereologies, and

1231. auxiliary and

1232. requirements attributes.

**value**

```

1223 init_der_serv_planrs: uid:(DP_UI|MP_UI) × DP_UI-set → in,out pr_dp_xg[uid] Unit
1223 init_der_serv_planrs(uid,dpuis) ≡
1224 let dpuis' = (pr_dp_xg.ch[uid] ! dpuis ; pr_dp_xg.ch[uid] ?) in
1228 || { let p = c_p(dp_ui), s = c_s(nms_dp_ui(dp_ui)) in
1229 let ui_p = uid_DP(p), ui_s = uid_DPS(s),
1230 me_p = mereo_DP(p), me_s = mereo_DPS(s),
1231 aux_p = attr_sAUX(p), aux_s = attr_sAUX(s),
1232 req_p = attr_sREQ(p), req_s = attr_sREQ(s) in
1225 derived_server_extr_Nm(dp_ui)(ui_s,me_s,(aux_s,req_s)) ||
1226 derived_planner_extr_Nm(dp_ui)(ui_p,me_p,(aux_p,req_p))
1227 | dp_ui:DP_UI•dpui ∈ dpuis' end end }
1223 end

```

### Q.7.9 The DERIVED SERVER<sub>nm<sub>i</sub></sub>, *i*:[1 : *p*] Translator

We refer to Sect. Q.4.8 for the attributes that play a rôle in determining the derived server signature.

#### Q.7.9.1 The Translate\_DPS<sub>nm<sub>j</sub></sub> Function

1233. The `Translate_DPS(dpsnmj)` results in three text elements:

- (a) the **value** keyword,
- (b) the *signature* of the `derived_server` definition,
- (c) and the *body* of that definition.

The `derived_servernmj` signature of the derived server contains the *unique identifier*; the *mereology*, cf. Item Q.3.7 on page 514 – used in determining channels: the dynamic clock identifier, the analysis depository identifier, the derived planner identifier; and the *attributes* which are: the auxiliary, `dAUXnmj` and the plan requirements, `dREQnmj`.

**value**

```

1233 Translate_DPS(dps_{nm_j}) \equiv
1233a " value
1233b derived_server $_{nm_j}$:
1233b DPS_UI $_{nm_j}$ \times DPS_Mer $_{nm_j}$ \rightarrow (DAUX $_{nm_j}$ \times dREQ $_{nm_j}$) \rightarrow
1233b in clk_ch, ad_s_ch[uid_DPS(dps_{nm_j})]
1233b out s_p_ch[uid_DPS(dps_{nm_j})] Unit
1233c derived_server $_{nm_j}$
1233c (uid_DPS(dps_{nm_j}), mereo_DPS(dps_{nm_j}), (attr_dAUX(dps_{nm_j}), attr_dREQ(dps_{nm_j}))) \equiv ... "
```

**Q.7.9.2 The derived\_server Behaviour**

The `derived_server` is almost identical to the master server, cf. Sect. Q.7.7.2, except that *plans* replace *urban space* information.

1234. The `derived_server` obtains time from the clock, see Item 1235c, , and the most recent analysis history, assembles these together with “locally produced”

- (a) *auxiliary* planner information and
- (b) plan *requirements*

as input, `MP_ARG`, to the master planner.

1235. The master server otherwise behaves as follows:

- (a) it obtains latest plans and latest analysis history, and
- (b) then produces auxiliary planning and plan requirements commensurate, i.e., fit, with the most recently, i.e., previously produced such information;
- (c) it then offers a time stamped compound of these kinds of information to the derived planner,
- (d) whereupon the derived server resumes being the derived server, albeit with updated programmable attributes.

**type**

```

1234a dAUX $_{nm_j}$
1234b dREQ $_{nm_j}$
1234 dARG $_{nm_j}$ = (T \times ((dAUX $_{nm_j}$ \times dREQ $_{nm_j}$) \times (PLANS \times AHist)))
```

**value**

```

1235 derived_server $_{nm_j}$ (uid, mereo)(aux, req) \equiv
1235a let plans = ps_pr_ch[uid] ?, ahist = ad_s_ch[uid] ?,
1235b daux:dAUX, dreq:dREQ • fit_AuxReq $_{nm_j}$ ((aux, req), (daux, dreq)) in
1235c s_p_ch[uid] ! (clk_ch?, ((maux, mreq), (plans, ahist))) ;
1235d derived_server $_{nm_j}$ (uid, mereo)(daux, dreq)
1235 end
```

```

1235b fitAuxReq $_{nm_j}$: (dAUX $_{nm_j}$ \times dREQ $_{nm_j}$) \times (dAUX $_{nm_j}$ \times dREQ $_{nm_j}$) \rightarrow Bool
```

```

1235b fitAuxReq $_{nm_j}$ ((aux, req), (daux, dreq)) \equiv ...
```

You may wish to compare formula Items 1234–1235d above with those of formula Items 1215–1216d of Sect. Q.7.7.2 on page 538.

**Q.7.10 The DERIVED PLANNER $_{nm_i}$ ,  $i$ :[1 :  $p$ ] Translator**

We refer to Sect. Q.4.9 for the attributes that play a rôle in determining the derived planner signature.

**Q.7.10.1 The Translate<sub>DP</sub> $dp_{nm_j}$  Function**

This function is an “almost carbon copy” of the **Translate<sub>MP</sub> $dp_{nm_j}$**  function. Thus Items 1236–1236c are “almost the same” as Items 1217–1217c on page 539.

1236. The **Translate<sub>DP</sub> $(nm_j)$**  results in three text elements:

- (a) the **value** keyword,
- (b) the *signature* of the **derived<sub>planner</sub> $nm_j$**  definition,
- (c) and the *body* of that definition.

The **derived<sub>planner</sub> $nm_j$**  signature of the derived planner contains the *unique identifier*, the *mereology*, cf. Item Q.3.8 on page 514 and the *attributes*: the *script*, cf. Sect. Q.4.3 on page 521 and Item 1140 on page 521, a set of *script pointers*, cf. Item 1153 on page 523, a set of *analyser names*, cf. Item 1154 on page 523, a set of *planner identifiers*, cf. Item 1155 on page 523, and the channels as implied by the master planner mereology.

**value**

```

1236 TranslateDP (dp) \equiv
1236a " value
1236b derivedplanner: $dp_{ui}:\text{DP_UI} \times \text{DP_Mer} \times (\text{Script} \times \text{ANms} \times \text{DPUIs}) \rightarrow \text{Script_Pts} \rightarrow$
1236b in s_p_ch[dp_{ui}], clk_ch, ad_ps_ch[dp_{ui}]
1236b out p_pr_ch[dp_{ui}]
1236b in,out p_dp_xg_ch[dp_{ui}] Unit
1236c derivedplanner(uid_DP(dp),mereo_DP(dp),
1236c (attr_Script(dp),attr_ANms(dp),attr_DPUIs(dp)))(attr_Script_Ptrs(dp)) $\equiv \dots$ "
```

**Q.7.10.2 The derived<sub>urban\_planning</sub> Function**

This function is an “almost carbon copy” of the **master<sub>urban\_planning</sub>** function. Thus Items 1237–1240c are “almost the same” as Items 1218–1221c on page 539.

1237. The core of the **derived<sub>planner</sub>** behaviour is the **derived<sub>urban\_planning</sub>** function.

1238. It takes as arguments: the *script*, a set of *analyser names*, a set of *derived planner identifiers*, a set of *script pointers*, and the time-stamped *derived planner argument*, cf. Item 1215 on page 538;

1239. and delivers, i.e., yields, a set of “remaining” *derived planner identifiers*, an updated set of *script pointers*, and a *master result*, **M\_RES**, i.e., a *master plan*, **mp:M\_PLAN** together with the time stamped *master argument* from which the plan was constructed.

1240. The **master<sub>urban\_planning</sub>** function is not defined by other than a predicate:

- (a) the “remaining” *derived planner identifiers* is a subset of the arguments *derived planner identifiers*;
- (b) the “resulting” *master argument* is the same as the input *master argument*, i.e., it is “carried forward”;
- (c) the arguments: the *script*, the *analyser names*, the *derived planner identifiers*, the set of *script pointers*, the time-stamped *master planner argument*, and the result plan otherwise satisfies a predicate  $\mathcal{P}_{dnm_i}(\text{script}_{dnm_i}, \text{anms}, \text{dpuis}, \text{ptrs}, \text{marg}_{dnm_i})(\text{dplan}_{dnm_i})$  expressing that the result **mplan** is an appropriate plan in view of the other arguments.

**type**1239  $D\_PLAN_{dnm_i}$ 1239  $D\_RES_{dnm_i} = D\_PLAN_{dnm_i} \times DP\_UI\_set \times D\_ARG_{dnm_i}$ **value**1238  $derived\_urban\_planning_{dnm_i}$ :1238  $Script_{dnm_i} \times ANm\_set \times DP\_UI\_set \times Script\_Ptr\_set \times D\_ARG_{dnm_i}$ 1239  $\rightarrow (DP\_UI\_set \times Script\_Ptr\_set) \times D\_RES_{dnm_i}$ 1237  $derived\_urban\_planning_{dnm_i}(script,anms,dpuis,ptrs,darg)$ 1240a **as**  $((dpuis',ptrs'),(dplan,ptrs'darg'))$ 1240a  $dpuis' \subseteq dpuis$ 1240b  $\wedge darg' = darg$ 1240c  $\wedge \mathcal{P}_{dnm_i}(script,anms,dpuis,ptrs,darg),((dpuis',ptrs'),(dplan,ptrs'darg'))$ 1237  $\mathcal{P}_{dnm_i} : ((Script_{dnm_i} \times ANM\_set \times DP\_UI\_set \times Script\_Ptr\_set \times D\_ARG_{dnm_i})$ 1237  $\times (DP\_UI\_set \times Script_{dnm_i}\_Ptr\_set \times D\_RES_{dnm_i})) \rightarrow \mathbf{Bool}$ 1237  $\mathcal{P}_{dnm_i}((script_{dnm_i},anms,dpuis,ptrs,darg_{dnm_i}),(dp\_uis',ptrs',dres)) \equiv \dots$ **Q.7.10.3 The derived\_planner<sub>nm<sub>j</sub></sub> Behaviour**

This behaviour is an “almost carbon copy” of the **derived\_planner<sub>nm<sub>j</sub></sub>** behaviour. Thus Items 1241–1241k are “almost the same” as Items 1222– 1222(f)v on page 540.

1241. The **derived\_planner** behaviour is otherwise as follows:

- (a) The **derived\_planner** obtains, from the derived server, its time stamped master argument, cf. Item 1215 on page 538;
- (b) it then invokes the derived urban planning function;
- (c) the time-stamped result is offered to the plan repository;
- (d) if the result is OK as a final result,
- (e) then the behaviour is stopped;
- (f) otherwise
- (g) the derived planner inquires the derived planner index generator as for such derived planner identifiers which are not used;
- (h) the derived planner behaviour is the resumed with the appropriately updated programmable script pointer attribute, in parallel with
- (i) the distributed parallel composition of the parallel behaviours of the derived servers
- (j) and the derived planners
- (k) designated by the derived planner identifiers transcribed into  $(nm\_dps\_ui)$  derived server, respectively into  $(nm\_dp\_ui)$  derived planner names. For these transcription maps we refer to Sect. Q.2.12 on page 511, Item 1097 on page 512.

**value**1222  $derived\_planner_{dnm_i}(uid,merco,(script_{dnm_i},anms,puis))(ptrs) \equiv$ 1222a **let**  $(t,((dau_{dnm_i},dreq_{dnm_i}),(plans,ahist))) = s\_p\_ch[uid] ? \mathbf{in}$ 1222b **let**  $((dpuis',ptrs'),dres_{dnm_i}) = derived\_urban\_planning_{dnm_i}(script_{dnm_i},anms,dpuis,ptrs) \mathbf{in}$ 1222c  $p\_pr\_ch[uid] ! dres_{dnm_i} ;$ 1222d **if**  $completed(dres_{dnm_i})$ 1222e **then**  $init\_der\_serv\_planrs(uid,dpuis') \mathbf{assert:} ptrs' = \{\}$ 1222f **else**1222(f)i  $init\_der\_serv\_plans(uid,dpuis')$ 1222(f)ii  $\parallel derived\_planner(uid,merco,(script_{dnm_i},anms,puis))(ptrs')$ 1222 **end end end**

## Q.8 Initialisation of The Urban Space Analysis & Planning System

Section Q.5 presents a *compiler* from *structures* and *parts* to *behaviours*. This section presents an initialisation of some of the behaviours. First we postulate a global *universe of discourse*, *uod*. Then we summarise the global values of *parts* and *part names*. This is followed by a summaries of *part qualities* – in four subsections: a summary of the global values of unique identifiers; a summary of channel declarations; the system as it is initialised; and the system of derived servers and planners as they evolve.

### Q.8.1 Summary of Parts and Part Names

#### value

|                  |                                                                                                                                  |
|------------------|----------------------------------------------------------------------------------------------------------------------------------|
| 1048 on page 506 | $uod : \text{UoD}$                                                                                                               |
| 1049 on page 506 | $clk : \text{CLK} = \text{obs\_CLK}(uod)$                                                                                        |
| 1050 on page 507 | $tus : \text{TUS} = \text{obs\_TUS}(uod)$                                                                                        |
| 1051 on page 507 | $ans : \text{A}_{anm_i}\text{-set}, i:[1..n] = \{ \text{obs\_A}_{anm_i}(aa) \mid aa \in (\text{obs\_AA}(uod)), i:[1..n] \}$      |
| 1052 on page 507 | $ad : \text{AD} = \text{obs\_AD}(\text{obs\_AA}(uod))$                                                                           |
| 1053 on page 507 | $mps : \text{MPS} = \text{obs\_MPS}(\text{obs\_MPA}(uod))$                                                                       |
| 1054 on page 507 | $mp : \text{MP} = \text{obs\_MP}(\text{obs\_MPA}(uod))$                                                                          |
| 1055 on page 507 | $dpss : \text{DPS}_{nm_i}\text{-set}, i:[1..p] =$                                                                                |
| 1055 on page 507 | $\{ \text{obs\_DPS}_{nm_i}(\text{dpc}_{nm_i}) \mid$                                                                              |
| 1055 on page 507 | $\text{dpc}_{nm_i} : \text{DPC}_{nm_i} \bullet \text{dpc}_{nm_i} \in \text{obs\_DPCS}_{nm_i}(\text{obs\_DPA}(uod)), i:[1..p] \}$ |
| 1056 on page 507 | $dps : \text{DP}_{nm_i}\text{-set}, i:[1..p] =$                                                                                  |
| 1056 on page 507 | $\{ \text{obs\_DP}_{nm_i}(\text{dpc}_{nm_i}) \mid$                                                                               |
| 1056 on page 507 | $\text{dpc}_{nm_i} : \text{DPC}_{nm_i} \bullet \text{dpc}_{nm_i} \in \text{obs\_DPCS}_{nm_i}(\text{obs\_DPA}(uod)), i:[1..p] \}$ |
| 1057 on page 507 | $dp_xg : \text{DPXG} = \text{obs\_DPXG}(uod)$                                                                                    |
| 1058 on page 507 | $pr : \text{PR} = \text{obs\_PR}(uod)$                                                                                           |
| 1059 on page 507 | $spsps : (\text{DPS}_{nm_i} \times \text{DP}_{nm_i})\text{-set}, i:[1..p] =$                                                     |
| 1059 on page 507 | $\{ (\text{obs\_DPS}_{nm_i}(\text{dpc}_{nm_i}), \text{obs\_DP}_{nm_i}(\text{dpc}_{nm_i})) \mid$                                  |
| 1059 on page 507 | $\text{dpc}_{nm_i} : \text{DPC}_{nm_i} \bullet \text{dpc}_{nm_i} \in \text{obs\_DPCS}_{nm_i}(\text{obs\_DPA}(uod)), i:[1..p] \}$ |

### Q.8.2 Summary of Unique Identifiers

#### value

|                  |                                                                                                                                                                                                                       |
|------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1074 on page 510 | $clk_{ui} : \text{CLK\_UI} = \text{uid\_CLK}(uod)$                                                                                                                                                                    |
| 1075 on page 510 | $tus_{ui} : \text{TUS\_UI} = \text{uid\_TUS}(uod)$                                                                                                                                                                    |
| 1076 on page 510 | $a_{uis} : \text{A\_UI-set} = \{ \text{uid\_A}(a) \mid a : \text{A} \bullet a \in ans \}$                                                                                                                             |
| 1077 on page 510 | $ad_{ui} : \text{AD\_UI} = \text{uid\_AD}(ad)$                                                                                                                                                                        |
| 1078 on page 510 | $mps_{ui} : \text{MPS\_UI} = \text{uid\_MPS}(mps)$                                                                                                                                                                    |
| 1079 on page 510 | $mp_{ui} : \text{MP\_UI} = \text{uid\_MP}(mp)$                                                                                                                                                                        |
| 1080 on page 510 | $dps_{uis} : \text{DPS\_UI-set} = \{ \text{uid\_DPS}(dps) \mid \text{dps} : \text{DPS} \bullet \text{dps} \in dpss \}$                                                                                                |
| 1081 on page 510 | $dp_{uis} : \text{DP\_UI-set} = \{ \text{uid\_DP}(dp) \mid \text{dp} : \text{DP} \bullet \text{dp} \in dps \}$                                                                                                        |
| 1082 on page 510 | $dp_xg_{ui} : \text{DPXG\_UI} = \text{uid\_DPXG}(dp_xg)$                                                                                                                                                              |
| 1083 on page 510 | $pr_{ui} : \text{PR\_UI} = \text{uid\_PR}(pr)$                                                                                                                                                                        |
| 1083 on page 510 | $s_{uis} : (\text{MPS\_UI} \mid \text{DPS\_UI})\text{-set} = \{ mps_{uis} \} \cup dps_{uis}$                                                                                                                          |
| 1085 on page 511 | $p_{uis} : (\text{MP\_UI} \mid \text{DP\_UI})\text{-set} = \{ mp_{uis} \} \cup dp_{uis}$                                                                                                                              |
| 1086 on page 511 | $sips : (\text{DPS\_UI} \times \text{DP\_UI})\text{-set} = \{ (\text{uid\_DPS}(dps), \text{uid\_DP}(dp)) \mid (\text{dps}, \text{dp}) : (\text{DPS} \times \text{DP}) \bullet (\text{dps}, \text{dp}) \in sps \}$     |
| 1087 on page 511 | $si\_pi\_m : \text{DPS\_UI} \xrightarrow{m} \text{DP\_UI} = [ \text{uid\_DPS}(dps) \mapsto \text{uid\_DP}(dp) \mid (\text{dps}, \text{dp}) : (\text{DPS} \times \text{DP}) \bullet (\text{dps}, \text{dp}) \in sps ]$ |
| 1088 on page 511 | $pi\_si\_m : \text{DP\_UI} \xrightarrow{m} \text{DPS\_UI} = [ \text{uid\_DP}(dp) \mapsto \text{uid\_DPS}(dps) \mid (\text{dps}, \text{dp}) : (\text{DPS} \times \text{DP}) \bullet (\text{dps}, \text{dp}) \in sps ]$ |

### Q.8.3 Summary of Channels

#### channel

|                  |                                                                                                                                                 |
|------------------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| 1169 on page 526 | $\text{clk\_ch:CLK\_MSG}$                                                                                                                       |
| 1170 on page 527 | $\{\text{tus\_a\_ch}[a\_ui]:\text{TUS\_MSG} a\_ui:A\_UI \bullet a\_ui \in a\_uis\}$                                                             |
| 1172 on page 527 | $\text{tus\_mps\_ch:TUS\_MSG}$                                                                                                                  |
| 1173 on page 528 | $\{\text{a\_ad\_ch}[a\_ui]:A\_MSG a\_ui:A\_UI \bullet a\_ui \in a\_uis\}$                                                                       |
| 1175 on page 528 | $\{\text{ad\_s\_ch}[s\_ui] s\_ui:(\text{MPS\_UI} \text{DPS\_UI}) \bullet s\_ui \in \{\text{mps\_ui}\} \cup \{\text{dps\_ui}\}\}:\text{AD\_MSG}$ |
| 1177 on page 528 | $\text{mps\_mp\_ch:MPS\_MSG}$                                                                                                                   |
| 1179 on page 529 | $\{\text{p\_pr\_ch}[p\_ui]:\text{PLAN\_MSG} p\_ui:(\text{MP\_UI} \text{DP\_UI}) \bullet p\_ui \in p\_uis\}$                                     |
| 1181 on page 529 | $\{\text{p\_dpxg\_ch}[ui]:\text{DPXG\_MSG} ui:(\text{MP\_UI} \text{DP\_UI}) \bullet ui \in p\_uis\}$                                            |
| 1183 on page 529 | $\{\text{pr\_s\_ch}[ui]:\text{PR\_MSGd} ui:(\text{MPS\_UI} \text{DPS\_UI}) \bullet ui \in s\_uis\}$                                             |
| 1185 on page 530 | $\{\text{dps\_dp\_ch}[ui]:\text{DPS\_MSG}_{nm_j} ui:\text{DPS\_UI} \bullet ui \in \text{dps\_uis}\}$                                            |

### Q.8.4 The Initial System

|                   |                                                                                                                                                           |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1194c on page 531 | $\text{urb\_spa}(\text{uid\_TUS}(tus), \text{mereo\_TUS}(tus))(\text{attr\_Pts}(tus))$                                                                    |
| 1187c on page 530 | $\text{clock}(\text{uid\_CLK}(clk), \text{mereo\_CLK}(clk))(\text{attr\_T}(clk))$                                                                         |
| 1201 on page 533  | $\parallel \{ \text{analyser}_{ui_i}(\text{uid\_A}(a_{ui_i}), \text{mereo\_A}(a_{ui_i}))(\text{ana}_{anm_i}) \mid ui_i:A\_UID \bullet ui_i \in a\_uis \}$ |
| 1187c on page 530 | $\text{ana\_dep}(\text{ui\_A}(ad), \text{mereo\_A}(ad))(\text{attr\_AHist}(ad))$                                                                          |
| 1212c on page 536 | $\text{plan\_rep}(\text{plans})(\text{attr\_AllDPUIs}(pr), \text{attr\_UsedDPUIs}(pr))$                                                                   |
| 1210c on page 535 | $\text{dpxg\_beh}(\text{uid\_DPXG}(dpxg), \text{mereo\_DPXG}(dpxg))(\text{all\_dpuis}, \text{used\_dpuis})$                                               |
| 1214c on page 537 | $\text{master\_server}(\text{uid\_MPS}(mps), \text{mereo\_MPS}(mps))(\text{attr\_mAUX}(mps), \text{attr\_mREQ}(mps))$                                     |
| 1217c on page 539 | $\text{master\_planner}(\text{uid\_MP}(mp), \text{mereo\_MP}(mp),$                                                                                        |
| 1217c on page 539 | $\quad (\text{attr\_Script}(mp), \text{attr\_ANms}(mp), \text{attr\_DPUIs}(mp)))(\text{attr\_Script\_Ptrs}(mp))$                                          |

### Q.8.5 The Derived Planner System

|                   |                                                                                                                                    |
|-------------------|------------------------------------------------------------------------------------------------------------------------------------|
| 1233c on page 541 | $\{ \text{derived\_server}_{dps_{nm_j}}$                                                                                           |
| 1233c on page 541 | $\quad (\text{uid\_DPS}(dps_{nm_j}), \text{mereo\_DPS}(dps_{nm_j}))(\text{attr\_dAUX}(dps_{nm_j}), \text{attr\_dREQ}(dps_{nm_j}))$ |
| 1236c on page 543 | $\parallel$                                                                                                                        |
| 1236c on page 543 | $\quad \text{derived\_planner}(\text{uid\_DP}(dp_{nm_j}), \text{mereo\_DP}(dp_{nm_j}),$                                            |
| 1236c on page 543 | $\quad (\text{attr\_Script}(dp_{nm_j}), \text{attr\_ANms}(dp_{nm_j}), \text{attr\_DPUIs}(dp_{nm_j})))$                             |
| 1236c on page 543 | $\mid j:[1..p] \}$                                                                                                                 |

## Q.9 Further Work

### Q.9.1 Reasoning About Deadlock, Starvation, Live-lock and Liveness

The current author is quite unhappy about the way in which he has defined the urban planning, oracle and repository behaviours. Such issues as which invariants are maintained across behaviours are not addressed. In fact, it seems to be good practice, following Dijkstra, Lamport and others, to formulate appropriate such invariants and only then “derive” behaviour definitions accordingly. In a rewrite of this research note, if ever, into a proper paper, the current author hopes to follow proper practices. He hopes to find younger talent to co-author this effort.



## Q.9.2 Document Handling

I may appear odd to the reader that I now turn to document handling. One central aspect of urban planning, strange, perhaps, to the reader, is that of handling the “zillions upon zillions” of documents that enter into and accrue from urban planning. If handling of these documents is not done properly a true nightmare will occur. So we shall briefly examine the urban planning document situation! From that we conclude that we must first try understand:

- **What do we mean by a document?**

### Q.9.2.1 Urban Planning Documents

The urban planning functions and the urban planning behaviours, including both the **base** and the *n* derived variants, rely on documents. These documents are **created**, **edited**, **read**, **copied**, and, eventually, **shredded** by urban-planners. Editing documents result in new versions of “the same” document. While a document is being **edited** or **read** we think of it as not being **accessible** to other urban-planners. If urban-planners need to read a latest version of a document while that version is subject to editing by another urban planner, copies must first be made, before editing, one for each “needy” reader. Once, editing has and readings have finished, the “reader” copies need, or can, be shredded.

### Q.9.2.2 A Document Handling System

In Chapter M we sketch[ed] a document handling system domain.<sup>15</sup> That is, not a document handling software system, not even requirements for a document handling software system, but just a description which, in essence, models documents and urban planners’ actions on documents. (The urban planners are referred to as document handlers.) The description further models two ‘aggregate’ notions: one of ‘handler management’, and one of ‘document archive’. Both seem necessary in order to “sort out” the granting of document access rights (that is, permissions to perform operations on documents), and the creation and shredding of documents, and in order to avoid dead-locks in access to and handling of documents.

## Q.9.3 Validation and Verification (V&V)

By **validation** of a document we shall mean: the primarily informal and social process of checking that the document description meets customer expectations.

Validation serves to get the right product.

By **verification** of a document we shall mean: the primarily formal, i.e., mathematical process of checking, testing and formal proof that the model, which the document description entails, satisfies a number of properties.

Verification serves to get the product right.

By **validation of the urban planning model** of this document we shall understand the social process of explaining the model to urban planning stakeholders, to obtain their reaction, and to possibly change the model according to stakeholder objections.

By **verification of the urban planning model** of this document we shall understand the formal process, based on formalisations of the argument and result types of the description, of testing, model checking and formally proving properties of the model.

MORE TO COME

<sup>15</sup>I had, over the years, since mid 1990s, reflected upon the idea of “*what is a document?*”. A most recent version, as I saw it in 2017, was “documented” in Chapter 7 [66]. But, preparing for my work, at TongJi University, Shanghai, September 2017, I reworked my earlier notes [66] into what is now Chapter M.

### Q.9.4 Urban Planning Project Management

In this research note we have focused on the urban planning project behaviours, their interactions, and their information “passing”. Usually publications about urban planning: research papers, technical papers, survey papers, etcetera, focus on specific “functions”. In this research note we do not. We focus instead on what we can say about the domain of urban planning: the fact, or the possibility, that an initial, a core, here referred to as a base, urban planning effort (i.e., project, hence behaviour) can “spew off”, generate, a number of (derived, i.e., in some sense subsidiary), more specialised, urban planning projects.

#### Q.9.4.1 Urban Planning Projects

We think of a comprehensive urban planning project as carried out by urban planners. As is evident from the model the project consists of one **base** urban planning project and up to  $n$  derived urban planning projects. The urban planners involved in these projects are professionals in the areas of planning:

- master urban planning issues:
  - geodesy,
  - geotechniques,
  - meteorology,
- master urban plans:
  - cartography,
  - cadestral matters,
  - zoning;
- derived urban planning issues:
  - industries,
- residential and shopping,
- apartment buildings,
- villas,
- recreational,
- etcetera;
- technological infrastructures:
  - transport,
  - electricity,
  - telecommunications,
  - gas,
- water,
- waste,
- etcetera;
- societal infrastructures:
  - health care,
  - schools,
  - police,
  - fire brigades,
  - etcetera;
- etcetera, etcetera, etcetera !

To anyone with any experience in getting such diverse groups and individuals of highly skilled professionals to work together it is obvious that some form of management is required. The term ‘comprehensive’ was mentioned above. It is meant to express that the comprehensive urban planning project is the only one “dealing” with a given geographic area, and that no other urban planning projects “infringe” upon it, that is, “deal” with sub-areas of that given geographic area.

#### Q.9.4.2 Strategic, Tactical and Operational Management

We can distinguish between

- strategic,
- tactical and
- operational

management.

**Q.9.4.2.1 Project Resources** But first we need take a look at the **resources** that management is charged with:

- the urban planners, i.e., humans,
- time,
- finances,
- office space,
- support technologies: computing etc.,
- etcetera.

**Q.9.4.2.2 Strategic Management** By **strategic management** we shall understand the analysis and decisions of, and concerning, scarce resources: people (skills), time, monies: their deployment and trade-offs.

**Q.9.4.2.3 Tactical Management** By **tactical management** we shall understand the analysis and decisions with respect to budget and time plans, and the monitoring and control of serially reusable resources: office space, computing.

**Q.9.4.2.4 Operational Management** By **operational management** we shall understand the monitoring and control of the enactment, progress and completion of individual deliverables, i.e., documents, the quality (adherence to “standards”, fulfillment of expectations, etc.) of these documents, and the day-to-day human relations.

**Q.9.4.3 Urban Planning Management**

The above (*strategic, tactical & operational management*) translates, in the context of *urban planning*, into:

TO BE WRITTEN



# Appendix R

## Weather Systems

### Contents

---

|            |                                                    |            |
|------------|----------------------------------------------------|------------|
| <b>R.1</b> | <b>On Weather Information Systems</b>              | <b>552</b> |
| R.1.1      | On a Base Terminology                              | 552        |
| R.1.2      | Some Illustrations                                 | 553        |
| R.1.2.1    | Weather Stations                                   | 553        |
| R.1.2.2    | Weather Forecasts                                  | 553        |
| R.1.2.3    | Forecast Consumers                                 | 553        |
| <b>R.2</b> | <b>Major Parts of a Weather Information System</b> | <b>553</b> |
| <b>R.3</b> | <b>Endurants</b>                                   | <b>554</b> |
| R.3.1      | Parts and Materials                                | 554        |
| R.3.2      | Unique Identifiers                                 | 555        |
| R.3.3      | Mereologies                                        | 556        |
| R.3.4      | Attributes                                         | 556        |
| R.3.4.1    | Clock, Time and Time-intervals                     | 556        |
| R.3.4.2    | Locations                                          | 557        |
| R.3.4.3    | Weather                                            | 557        |
| R.3.4.4    | Weather Stations                                   | 558        |
| R.3.4.5    | Weather Data Interpreter                           | 558        |
| R.3.4.6    | Weather Forecasts                                  | 559        |
| R.3.4.7    | Weather Forecast Consumer                          | 559        |
| <b>R.4</b> | <b>Perdurants</b>                                  | <b>559</b> |
| R.4.1      | A WIS Context                                      | 559        |
| R.4.2      | Channels                                           | 560        |
| R.4.3      | WIS Behaviours                                     | 560        |
| R.4.4      | Clock                                              | 561        |
| R.4.5      | Weather Station                                    | 561        |
| R.4.6      | Weather Data Interpreter                           | 562        |
| R.4.6.1    | collect_wd                                         | 562        |
| R.4.6.2    | calculate_wf                                       | 562        |
| R.4.6.3    | disseminate_wf                                     | 563        |
| R.4.7      | Weather Forecast Consumer                          | 564        |
| <b>R.5</b> | <b>Conclusion</b>                                  | <b>565</b> |
| R.5.1      | Reference to Similar Work                          | 565        |
| R.5.2      | What Have We Achieved?                             | 565        |
| R.5.3      | What Needs to be Done Next?                        | 565        |
| R.5.4      | Acknowledgments                                    | 565        |

---

This document reports a class exercise from a PhD course at the University of Bergen, Norway, November 2016.<sup>1</sup> We show an example domain description. It is developed and presented as outlined in [51]. The domain being described is that of a generic weather information system. Four main endurants (i.e., aspects) of a generic weather information system are those of the weather, weather stations (collecting weather data), weather data interpretation (i.e., meteorological institute[s]), and weather forecast consumers. There are, correspondingly, two kinds of weather information: the weather data, and the weather forecasts. These forms of weather information are acted upon: the weather data interpreter (i.e., a meteorological institute) is gathering weather data; based on such interpretations the meteorological institute is “calculating” weather forecasts; and weather forecast consumers are requesting and further “interpreting” (i.e., rendering) such forecasts. Thus weather data is communicated from weather stations to the weather data interpreter; and weather forecasts are communicated from the weather data interpreter to the weather forecast consumers. It is the dual purpose of this technical report to present a domain description of the essence of generic weather information systems, and to add to the “pile” [39,40,44–46,48–50] of technical reports that illustrate the use[fulness] of the principles, techniques and tools of [51].

## R.1 On Weather Information Systems

### R.1.1 On a Base Terminology

From Wikipedia:

1242. **Weather** is the state of the atmosphere, to the degree that it is hot or cold, wet or dry, calm or stormy, clear or cloudy, atmospheric (barometric) pressure: high or low.
1243. So weather is characterized by **temperature, humidity** (incl. **rain, wind** (direction, velocity, center, incl. its possible mobility), **atmospheric pressure**, etcetera.
1244. By **weather information** we mean
- either weather data that characterizes the weather as defined above (Item 1242),
  - or weather forecast, i.e., a prediction of the state of the atmosphere for a given location and time or time interval.
1245. Weather data are collected by **weather stations**. We shall here not be concerned with technical means of weather data collection.
1246. **Weather forecasts** are used by forecast consumers, anyone: you and me.
1247. Weather data interpretation (i.e., **forecasting**) is the science and technology of creating weather forecasts based on **time-** or **time interval-stamped weather data** and **locations**. Weather data interpretation is amongst the charges of meteorological institutes.
1248. **Meteorology** is the interdisciplinary scientific study of the atmosphere.
1249. An **atmosphere** (from Greek  $\alpha\tau\mu\omicron\zeta$  (atmos), meaning “vapour”, and  $\sigma\phi\alpha\iota\rho\alpha$  (sphaira), meaning “sphere”) is a layer of gases surrounding a planet or other material body, that is held in place by the gravity of that body.
1250. Meteorological institutes work together with the World Meteorological Organization (WMO). Besides weather forecasting, meteorological institutes (and hence WMO) are concerned also with aviation, agricultural, nuclear, maritime, military and environmental meteorology, hydrometeorology and renewable energy.

---

<sup>1</sup>I thank my host, Prof. Magne Haveræen for the invitation. The occasion was that of a visit by Mme. Dooren Tuheirwe from Makerere University, Uganda, and her work with the university and the Norwegian Meteorological Institute on a joint project on a Weather Information System for Uganda.

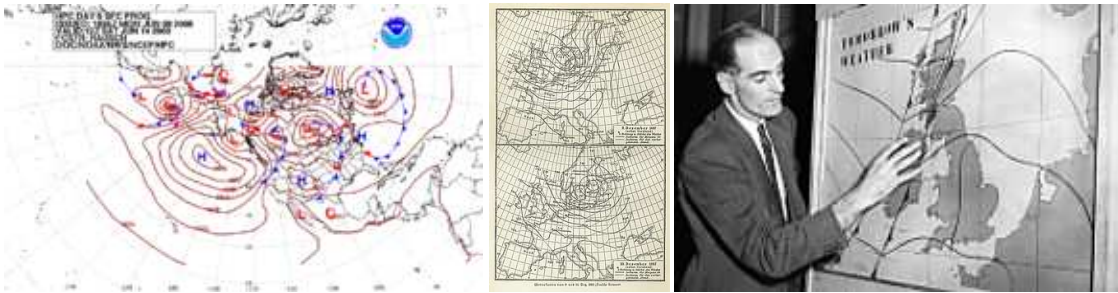
1251. Agricultural meteorologists, soil scientists, agricultural hydrologists, and agronomists are persons concerned with studying the effects of weather and climate on plant distribution, crop yield, water-use efficiency, phenology of plant and animal development, and the energy balance of managed and natural ecosystems. Conversely, they are interested in the rôle of vegetation on climate and weather.

R.1.2 Some Illustrations

R.1.2.1 Weather Stations



R.1.2.2 Weather Forecasts



R.1.2.3 Forecast Consumers



R.2 Major Parts of a Weather Information System

We think of the following parts as being of concern in the kind of weather information systems that we shall analyse and describe: Figure R.1 on the following page shows one **weather** (dashed rounded corner all embracing rectangle), one central **weather data interpreter** (cum meteorological institute) seven **weather stations** (rounded corner squares), nineteen **weather forecast consumers**, and one global **clock**. All are distributed, as hinted at, in some geographical space. Figure R.2 shows “an orderly diagram” of “the same” weather information system as Figure R.1. The lines between pairs of the various parts shall indicate means communication between the pairs of (thus) connected parts. Dashed lines “crossing” bundles of these communication lines are labeled  $ch_{xy}$ . These labels,  $ch_{xy}$ , designated CSP-like channels. An input, by a weather station (wsi), of weather data from the weather (wi), is designated by the CSP expression  $ch_{ws}[wi, wsi] ?$ .

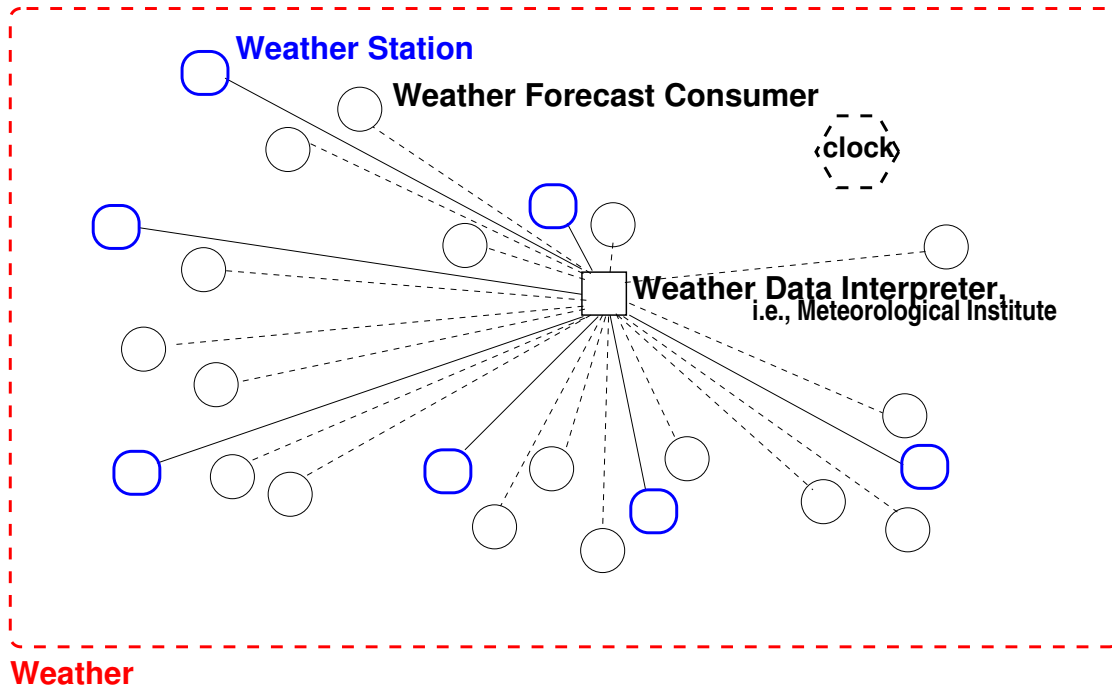


Figure R.1: A Weather Information System

An output, say from the weather data interpreter (*wdi*) to a weather forecast consumer (*fci*), of a forecast *f*, is designated by  $ch\_ic[w_{dii}, f_{ci}] ! f$

## R.3 Endurants

### R.3.1 Parts and Materials

1252. The WIS domain contains a number of sub-domains:

- the weather, *W*, which we consider a material,
- the weather stations sub-domain, *WSS* (a composite part),
- the weather data interpretation sub-domain, *WDIS* (an atomic part),
- the weather forecast consumers sub-domain, *WFCS* (a composite part), and
- the (“global”) clock (an atomic part).

#### type

1252 WIS  
 1252a W  
 1252b WSS  
 1252c WDIS  
 1252d WFCS  
 1252e CLK

#### value

1252a *obs\_material\_W*: WIS → W  
 1252b *obs\_part\_WSS*: WIS → WSS  
 1252c *obs\_part\_WDIS*: WIS → WDIS  
 1252d *obs\_part\_WFCS*: WIS → WFCS  
 1252e *obs\_part\_CLK*: WIS → CLK



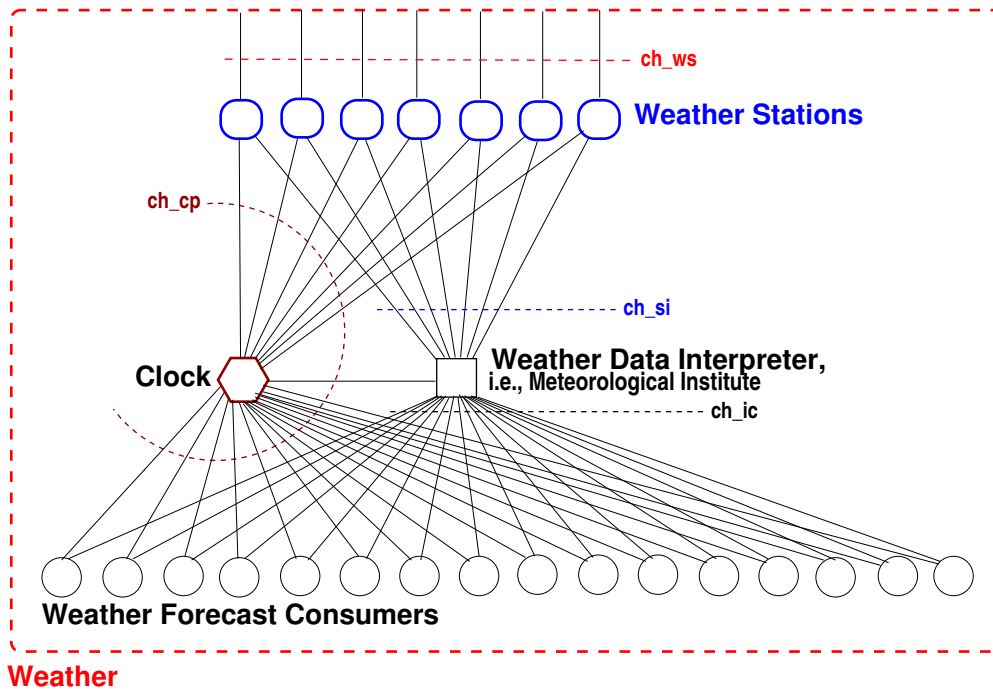


Figure R.2: A Weather Information System Diagram

1253. The weather station sub-domain, WSS, consists of a set, WSs,  
 1254. of atomic weather stations, WS.  
 1255. The weather forecast consumers sub-domain, WFCS, consists of a set, WFCs,  
 1256. of atomic weather forecast consumers, WFC.

**type**

1253 WSs = WS-set

1254 WS

1255 WFCs = WFC-set

1256 WFC

**value**1253 obs\_part\_WSs: WSS  $\rightarrow$  WSs1255 obs\_part\_WFCs: WFCS  $\rightarrow$  WFCs**R.3.2 Unique Identifiers**

We shall consider only atomic parts.

1257. Every single weather station has a unique identifier.  
 1258. The weather data interpretation (i.e., the weather forecast “creator”) has a unique identifier.  
 1259. Every single weather forecast consumer has a unique identifier.  
 1260. The global clock has a unique identifier.

**type**

1257 WSI  
 1258 WDII  
 1259 WFCI  
 1260 CLKI

**value**

1257 uid\_WSI: WS  $\rightarrow$  WSI  
 1258 uid\_WDII: WDIS  $\rightarrow$  WDII  
 1259 uid\_WFCI: WFC  $\rightarrow$  WFCI  
 1259 uid\_CLKI: CLK  $\rightarrow$  CLKI

**R.3.3 Mereologies**

We shall restrict ourselves to consider the mereologies only of the atomic parts.

1261. The mereology of weather stations is the pair of the unique clock identifier and the unique identifier of the weather data interpreter.
1262. The mereology of weather data interpreter is the triple of the unique clock identifier, set of unique identifiers of all the weather stations and the set of unique identifiers of all the weather forecast consumers.
1263. The mereology of weather forecast consumer is the the pair of the unique clock identifier and the unique identifier of the weather data interpreter.
1264. The mereology of the global clock is the triple of the set of all the unique identifiers of weather stations, the unique identifier of the weather data interpreter, and the set of all the unique identifiers of weather forecast consumers.

**type**

1261 WSM = CLKI  $\times$  WDII  
 1262 WDIM = CLKI  $\times$  WSI-**set**  $\times$  WFCI-**set**  
 1263 WFCM = CLKI  $\times$  WDII  
 1264 CLKM = CLKI  $\times$  WDGI-**set**  $\times$  WDII  $\times$  WFCI-**set**

**value**

1261 mereo\_WSM: WS  $\rightarrow$  WSM  
 1262 mereo\_WDI: WDI  $\rightarrow$  WDIM  
 1263 mereo\_WFC: WFC  $\rightarrow$  WFCM  
 1264 mereo\_CLK: CLK  $\rightarrow$  CLKM

**R.3.4 Attributes****R.3.4.1 Clock, Time and Time-intervals**

1265. The global clock has an autonomous time attribute.
1266. Time values are further undefined, but times are considered absolute in the sense as representing some intervals since “the birth of time”, an example, concrete time could be MARCH 12, 2024: 10:48 AM.
1267. Time intervals are further undefined, but time intervals can be considered relative in the sense of representing a quantity elapsed between two times, examples are: 1 day 2 hours and 3 minutes, etc. When a time interval,  $ti$ , is specified it is always to be understood to designate the times from now, or from a specified time,  $t$ , until the time  $t + ti$ .

1268. We postulate  $\oplus$ ,  $\ominus$ , and can postulate further “arithmetic” operators, and

1269. we can postulate relational operators.

**type**

1265 TIME

1266 TI

**value**

1265 attr\_TIME: CLK  $\rightarrow$  TIME

1268  $\oplus$ : TIME  $\times$  TI  $\rightarrow$  TIME, TI  $\times$  TI  $\rightarrow$  TI

1268  $\ominus$ : TIME  $\times$  TI  $\rightarrow$  TIME, TIME  $\times$  TIME  $\rightarrow$  TI

1269 =,  $\neq$ , <,  $\leq$ ,  $\geq$ , >: TIME  $\times$  TIME  $\rightarrow$  **Bool**, TI  $\times$  TI  $\rightarrow$  **Bool**, ...

We do not here define these operations and relations.

**R.3.4.2 Locations**

1270. Locations are metric, topological spaces and can thus be considered dense spaces of three dimensional points.

1271. We can speak of one location properly contained ( $\subset$ ) within, or contained or equal ( $\subseteq$ ), or equal ( $=$ ), or not equal ( $\neq$ ) to another location.

**type**

1270. LOC

**value**

1271.  $\subset$ ,  $\subseteq$ , =,  $\neq$ : LOC  $\times$  LOC  $\rightarrow$  **Bool**

**R.3.4.3 Weather**

1272. The weather material is considered a dense, infinite set of weather point volumes WP. Some dense, infinite subsets (still proper volumes) of such points may be liquid, i.e., rain, water in rivers, lakes and oceans. Other dense, infinite subsets (still proper volumes) of such points may be gaseous, i.e., the air, or atmosphere. These two forms of proper volumes “border” along infinite subsets (curved planes, surfaces) of weather points.

1273. From the material weather one can observe its location.

**type**

1272 W = WP-**inset**

1272 WP

**value**

1273 attr\_LOC: W  $\rightarrow$  LOC

1274. Some meteorological quantities are:

- |                          |                                  |
|--------------------------|----------------------------------|
| (a) <i>Humidity</i> ,    | (c) <i>Wind</i> and              |
| (b) <i>Temperature</i> , | (d) <i>Barometric pressure</i> . |

1275. The weather has an indefinite number of attributes at any one time.

- (a) Humidity distribution, at level (above sea) and by location,
- (b) Temperature distribution, at level (above sea) and by location,

- (c) Wind direction, velocity and mobility of wind center, and by location,
- (d) Barometric pressure, and by location,
- (e) etc., etc.

**type**

- 1274a Hu
- 1274b Te
- 1274c Wi
- 1274d Ba
- 1275a HDL = LOC  $\xrightarrow{m}$  Hu
- 1275b TDL = LOC  $\xrightarrow{m}$  Te
- 1275c WDL = LOC  $\xrightarrow{m}$  Wi
- 1275d BPL = LOC  $\xrightarrow{m}$  Ba
- 1275e ...

**value**

- 1275a attr\_HDL: W  $\rightarrow$  HDL
- 1275b attr\_TDL: W  $\rightarrow$  TDL
- 1275c attr\_WDL: W  $\rightarrow$  WDL
- 1275d attr\_APL: W  $\rightarrow$  BPL
- 1275e ...

**R.3.4.4 Weather Stations**

- 1276. Weather stations have static location attributes.
- 1277. Weather stations sample the weather gathering humidity, temperature, wind, barometric pressure, and possibly other data, into time and location stamped weather data.

**value**

- 1276 attr\_LOC: WS  $\rightarrow$  LOC

**type**

- 1277 WD :: mkWD((TIME $\times$ LOC) $\times$ (TDL $\times$ HDL $\times$ WDL $\times$ BPL $\times$ ...))

**R.3.4.5 Weather Data Interpreter**

- 1278. There is a programmable attribute: weather data repository, wdr:WDR, of weather data, wd:WD, collected from weather stations.
- 1279. And there is programmable attribute: weather forecast repository, wfr:WFR, of forecasts, wf:WF, disseminate-able to weather forecast consumers.

These repositories are updated when

- 1280. received from the weather stations, respectively when
- 1281. calculated by the weather data interpreter.

**type**

- 1278 WDR
- 1279 WFR

**value**

- 1280 update\_wdr: TIME  $\times$  WD  $\rightarrow$  WDR  $\rightarrow$  WDR
- 1281 update\_wfr: TIME  $\times$  WF  $\rightarrow$  WFR  $\rightarrow$  WFR

It is a standard exercise to define these two functions (say algebraically).

**R.3.4.6 Weather Forecasts**

1282. Weather forecasts are weather forecast format-, time- and location-stamped quantities, the latter referred to as wefo:WeFo.
1283. There are a definite number ( $n \geq 1$ ) of weather forecast formats.
1284. We do not presently define these various weather forecast formats.
1285. They are here thought of as being requested, mkWFReq, by weather forecast consumers.

**type**

- 1282  $WF = WFF \times (TIME \times TI) \times LOC \times WeFo$   
 1283  $WFF = WFF1 \mid WFF2 \mid \dots \mid WFFn$   
 1284  $WFF1, WFF2, \dots, WFFn$   
 1285  $WFReq :: mkWFReq(s\_wff:WFF, s\_ti:(TIME \times TI), s\_loc:LOC)$

**R.3.4.7 Weather Forecast Consumer**

1286. There is a programmable attribute, d:D, D for display (!).
1287. Displays can be rendered (RND): visualized, tabularised, made audible, translated (between languages and language dialects, ...), etc.
1288. A rendered display can be “abstracted back” into its basic form.
1289. Any abstracted rendered display is identical to its abstracted form.

**type**

- 1286  $D$   
 1287  $RND$

**value**

- 1286  $attr\_D: WFC \rightarrow D$

- 1287  $rndr\_D: RND \times D \rightarrow D$

- 1288  $abs\_D: D \rightarrow D$

**axiom**

- 1289  $\forall d:D, r:RND \bullet abs\_D(rndr(r,d)) = d$

**R.4 Perdurants****R.4.1 A WIS Context**

1290. We postulate a given system, wis:WIS.  
 That system is characterized by
1291. a dynamic weather  
 1292. and its unique identifier,  
 1293. a set of weather stations  
 1294. and their unique identifiers,  
 1295. a single weather data interpreter  
 1296. and its unique identifier,
1297. a set of weather forecast consumers  
 1298. and their unique identifiers, and  
 1299. a single clock  
 1300. and its unique identifier.
1301. Given any specific wis:WIS there is [therefore] a full set of part identifiers, is, of weather, clock, all weather stations, the weather data interpreter and all weather forecast consumers.

We list the above-mentioned values. They will be referenced by the channel declarations and the behaviour definitions of this section.

**value**

- 1290 wis:WIS
- 1291 w:W = obs\_material\_W(wis)
- 1292 wi:WI = uid\_WI(w)
- 1293 wss:WSs = obs\_part\_WSs(obs\_part\_WSS(wis))
- 1294 wsis:WDGI-set = {uid\_WSI(ws)|ws:WS•ws ∈ wss}
- 1295 wdi:WDI = obs\_part\_WDIS(wis)
- 1296 wdii:WDII = uid\_WDII(wdi)
- 1297 wfcis:WFCs = obs\_part\_WFCs(obs\_part\_WFCS(wis))
- 1298 wfcis:WFI-set = {uid\_WFCI(wfc)|wfc:WFC•wfc ∈ wfcis}
- 1299 clk:CLK = obs\_part\_CLK(wis)
- 1300 clki:CLKI = uid\_CLKI(clk)
- 1301 is:(WI|WSI|WDII|WFCI)-set = {wi} ∪ wsis ∪ {wdii} ∪ wfcis

**R.4.2 Channels**

- 1302. Weather stations share weather data, WD, with the weather data interpreter — so there is a set of channels, one each, “connecting” weather stations to the weather data interpreter.
- 1303. The weather data interpreter shares weather forecast requests, WFReq, and interpreted weather data (i.e., forecasts), WF, with each and every forecast consumer — so there is a set of channels, one each, “connecting” the weather data interpreter to the interpreted weather data (i.e., forecast) consumers.
- 1304. The clock offers its current time value to each and every part, except the weather, of the WIS system.

**channel**

- 1302 { ch\_si[ wsi,wdii ]:WD | wsi:WSI•wsi ∈ wsis }
- 1303 { ch\_ic[ wdii,fci ]:(WFReq|WF) | fci:Fci•fci ∈ fcis }
- 1304 { ch\_cp[ clki,i ]:TIME | i:(WI|CLKI|WSI|WDII|WFCI)•i ∈ is }

**R.4.3 WIS Behaviours**

- 1305. WIS behaviour, wis\_beh, is the
- 1306. parallel composition of all the weather station behaviours, in parallel with the
- 1307. weather data interpreter behaviour, in parallel with the
- 1308. parallel composition of all the weather forecast consumer behaviours, in parallel with the
- 1309. clock behaviour.

**value**

- 1305 wis\_beh: **Unit** → **Unit**
- 1305 wis\_beh() ≡
- 1306 || { ws\_beh(uid\_WSI(ws),mereo\_WS(ws),...) | ws:WS•ws ∈ wss } ||
- 1307 || wdi\_beh(uid\_WDI(wdi),mereo\_WDI(wdi),...)(wd\_rep,wf\_rep) ||
- 1308 || { wfc\_beh(uid\_WFCI(wfc),mereo\_WDG(wfc),...) | wfc:WFC•wfc ∈ wfcis } ||
- 1309 clk\_beh(uid\_CLKI(clk),mereo\_CLK(clk),...)(**"March 12, 2024: 10:48 am"**)

**R.4.4 Clock**

1310. The clock behaviour has a programmable attribute,  $t$ .

1311. It repeatedly offers its current time to any part of the WIS system.

It nondeterministically internally “cycles” between

1312. retaining its current time, or

1313. increment that time with a “small” time interval,  $\delta$ , or

1314. offering the current time to a requesting part.

**value**

1310.  $\text{clk\_beh}: \text{clki}:\text{CLKI} \times \text{clkm}:\text{CLKM} \rightarrow \text{TIME} \rightarrow$

1311. **out**  $\{ \text{ch\_cp}[\text{clki},i] \mid i:(\text{WSI}|\text{WDII}|\text{WFCI}) \bullet i \in \text{wsi} \cup \{ \text{wdii} \} \cup \text{wfcis} \}$  **Unit**

1310.  $\text{clk\_beh}(\text{clki},\text{is})(t) \equiv$

1312.  $\text{clk\_beh}(\text{clki},\text{is})(t)$

1313.  $\sqcap \text{clk\_beh}(\text{clki},\text{is})(t \oplus \delta)$

1314.  $\sqcap ( \sqcap \{ \text{ch\_cp}[\text{clki},i] \mid t \mid i:(\text{WSI}|\text{WDII}|\text{WFCI}) \bullet i \in \text{is} \} ; \text{clk\_beh}(\text{clki},\text{is})(t) )$

**R.4.5 Weather Station**

1315. The weather station behaviour communicates with the global clock and the weather data interpreter.

1316. The weather station behaviour simply “cycles” between sampling the weather, reporting its findings to the weather data interpreter and resume being that overall behaviour.

1317. The weather station time-stamp “sample’ the weather (i.e., meteorological information).

1318. The meteorological information obtained is analysed with respect to temperature (distribution etc.),

1319. humidity (distribution etc.),

1320. wind (distribution etc.),

1321. barometric pressure (distribution etc.), etcetera,

1322. and this is time-stamp and location aggregated (mkWD) and “sent” to the (central ?) weather data interpreter,

1323. whereupon the weather data generator behaviour resumes.

**value**

1315  $\text{ws\_beh}: \text{wsi}:\text{WSI} \times (\text{clki},\text{wi},\text{wdii}):\text{WDGM} \times (\text{LOC} \times \dots) \rightarrow$

1315 **in**  $\text{ch\_cp}[\text{clki},\text{wsi}]$  **out**  $\text{ch\_gi}[\text{wsi},\text{wdii}]$  **Unit**

1316  $\text{ws\_beh}(\text{wsi},(\text{clki},\text{wi},\text{wdii}),(\text{loc},\dots)) \equiv$

1318 **let**  $\text{tdl} = \text{attr\_TDL}(w)$ ,

1319  $\text{hdl} = \text{attr\_HDL}(w)$ ,

1320  $\text{wdl} = \text{attr\_WDL}(w)$ ,

1321  $\text{bpl} = \text{attr\_BPL}(w)$ , ... **in**

1322  $\text{ch\_gi}[\text{wsi},\text{wdii}] \mid \text{mkWD}((\text{ch\_cp}[\text{clki},\text{wsi}] \text{ ?}, \text{loc}),(\text{tdl},\text{hdl},\text{wdl},\text{bpl},\dots))$  **end** ;

1323  $\text{wdg\_beh}(\text{wsi},(\text{clki},\text{wi},\text{wdii}),(\text{loc},\dots))$

### R.4.6 Weather Data Interpreter

1324. The weather data interpreter behaviour communicates with the global clock, all the weather stations and all the weather forecast consumers.
1325. The weather data interpreter behaviour non-deterministically internally ( $\square$ ) chooses to
1326. either collect weather data,
1327. or calculate some weather forecast,
1328. or disseminate a weather forecast.

#### value

```

1324 wdi_beh: wdii:WDII × (clki, wsis, wfcis):WDIM × ... → (WD_Rep × WF_Rep) →
1324 in ch_cp[clki, wdii], { ch_si[wsi, wdii] | wsi:WSI • wsi ∈ wsis },
1324 out { ch_ic[wdii, wfcis] | wfcis:WFCI • wfcis ∈ wfcis } Unit
1324 wdi_beh(wdii, (clki, wsis, wfcis), ...) (wd_rep, wf_rep) ≡
1326 collect_wd(wdii, (clki, wsis, wfcis), ...) (wd_rep, wf_rep)
1325 □
1327 calculate_wf(wdii, (clki, wsis, wfcis), ...) (wd_rep, wf_rep)
1325 □
1328 disseminate_wf(wdii, (clki, wsis, wfcis), ...) (wd_rep, wf_rep)

```

#### R.4.6.1 collect\_wd

1329. The collect weather data behaviour communicates with the global clock and all the weather stations – but “passes-on” the capability to communicate with all of the weather forecast consumers.
1330. The collect weather data behaviour
1331. non-deterministically externally offers to accept weather data from some weather station,
1332. updates the weather data repository with a time-stamped version of that weather data,
1333. and resumes being a weather data interpreter behaviour, now with an updated weather data repository.

#### value

```

1329 collect_wd: wdii:WDII × (clki, wsis, wfcis):WDIM × ...
1329 → (WD_Rep × WF_Rep) →
1329 in ch_cp[clki, wdii], { ch_si[wsi, wdii] | wsi:WSI • wsi ∈ wsis },
1329 out { ch_ic[wdii, wfcis] | wfcis:WFCI • wfcis ∈ wfcis } Unit
1330 collect_wd(wdii, (clki, wsis, wfcis), ...) (wd_rep, wf_rep) ≡
1331 let ((ti, loc), (hdl, tdl, wdl, bpl, ...)) = □ { wsi[wsi, wdii]? | wsi:WSI • wsi ∈ wsis } in
1332 let wd_rep' = update_wdr(ch_cp[clki, wdii]?, ((ti, loc), (hdl, tdl, wdl, bpl, ...))) (wd_rep) in
1333 wdi_beh(wdii, (clki, wsis, wfcis), ...) (wd_rep', wf_rep) end end

```

#### R.4.6.2 calculate\_wf

1334. The calculate forecast behaviour communicates with the global clock – but “passes-on” the capability to communicate with all of weather stations and the weather forecast consumers.
1335. The calculate forecast behaviour
1336. non-deterministically internally chooses a forecast type from among a indefinite set of such,



1337. and a current or “future” time-interval,  
 1338. whereupon it calculates the weather forecast and updates the weather forecast repository,  
 1339. and then resumes being a weather data interpreter behaviour now with the weather forecast repository updated with the calculated forecast.

**value**

```

1334 calculate_wf: wdii:WDII × (clki, wsis, wfcis):WDIM × ... → (WD_Rep × WF_Rep) →
1334 in ch_cp[clki, wdii], { ch_si[wsi, wdii] | wsi:WSI • wsi ∈ wsis },
1334 out { ch_ic[wdii, wfcis] | wfcis:WFCI • wfcis ∈ wfcis } Unit
1335 calculate_wf(wdii, (clki, wsis, wfcis), ...)(wd_rep, wf_rep) ≡
1336 let tf:WWF = ft1 [] ft2 [] ... [] ftn,
1337 ti:(TIME × TIVAL) • toti ≥ ch_cp[clki, wdii] ? in
1338 let wf_rep' = update_wfr(calc_wf(tf, ti)(wf_rep)) in
1339 wdi_beh(wdii, (clki, wsis, wfcis), ...)(wd_rep, wf_rep') end end

```

1340. The calculate\_weather forecast function is, at present, further undefined.

**value**

```

1340. calc_wf: WFF × (TIME × TI) → WFRep → WF
1340. calc_wf(tf, ti)(wf_rep) ≡ ...

```

**R.4.6.3 disseminate\_wf**

1341. The disseminate weather forecast behaviour communicates with the global clock and all the weather forecast consumers – but “passes-on” the capability to communicate with all of weather stations.  
 1342. The disseminate weather forecast behaviour non-deterministically externally offers to received a weather forecast request from any of the weather forecast consumers, wfcis, that request is for a specific format forecast, tf, and either for a specific time or for a time-interval, toti, as well as for a specific location, loc.  
 1343. The disseminate weather forecast behaviour retrieves an appropriate forecast and  
 1344. sends it to the requesting consumer –  
 1345. whereupon the disseminate weather forecast behaviour resumes being a weather data interpreter behaviour

**value**

```

1341 disseminate_wf: wdii:WDII × (clki, wsis, wfcis):WDIM × ... → (WD_Rep × WF_Rep) →
1341 in ch_cp[clki, wdii] in, out { ch_ic[wdii, wfcis] | wfcis:WFCI • wfcis ∈ wfcis } Unit
1341 disseminate_wf(wdii, (clki, wsis, wfcis), ...)(wd_rep, wf_rep) ≡
1342 let mkReqWF((tf, toti, loc), wfcis) = [] { ch_ic[wdii, wfcis] ? | wfcis:WFCI • wfcis ∈ wfcis } in
1343 let wf = retr_WF((tf, toti, loc), wf_rep) in
1344 ch_ic[wdii, wfcis] ! wf ;
1345 disseminate_wf(wdii, (clki, wsis, wfcis), ...)(wd_rep, wf_rep) end end

```

1346. The retr\_WF((tf, toti, loc), wf\_rep) function invocation retrieves the weather forecast from the weather forecast repository most “closely” matching the format, tf, time, toti, and location of the request received from the weather forecast consumer. We do not define this function.

1346.  $\text{retr\_WF}: (\text{WFF} \times (\text{TIME} \times \text{TI}) \times \text{LOC}) \times \text{WFRep} \rightarrow \text{WF}$   
 1346.  $\text{retr\_WF}((\text{tf}, \text{toti}, \text{loc}), \text{wf\_rep}) \equiv \dots$

We could have included, in our model, the time-stamping of receipt (formula Item 1342) of requests, and the time-stamping of delivery of requested forecast in which case we would insert  $\text{ch\_cp}[\text{clki}, \text{wdii}]?$  at respective points in formula Items 1342 and 1344.

### R.4.7 Weather Forecast Consumer

1347. The weather forecast consumer communicates with the global clock and the weather data interpreter.
1348. The weather forecast consumer behaviour
1349. nondeterministically internally either
1350. selects a suitable weather cast format,  $\text{tf}$ ,
1351. selects a suitable location,  $\text{loc}'$ , and
1352. selects,  $\text{toti}$ , a suitable time (past, present or future) or a time interval (that is supposed to start when forecast request is received by the weather data interpreter.
1353. With a suitable formatting of this triple,  $\text{mkReqWF}(\text{tf}, \text{loc}', \text{toti})$ , the weather forecast consumer behaviour “outputs” a request for a forecast to the weather data interpreter (first “half” of formula Item 1352) whereupon it awaits (;) its response (last “half” of formula Item 1352) which is a weather forecast,  $\text{wf}$ ,
1354. whereupon the weather forecast consumer behaviour resumes being that behaviour with it programmable attribute,  $d$ , being replaced by the received forecast suitably annotated;
- 1349 or the weather forecast consumer behaviour
1355. edits a display
1356. and resumes being a weather forecast consumer behaviour with the edited programmable attribute,  $d'$ .

#### value

```

1347 wfc_beh: wfc:WFCI \times (clki,wdii):WFCM \times (LOC \times ...) \rightarrow D \rightarrow
1347 in ch_cp[clki,wfc],
1347 in,out { ch_ic[wdii,wfc] | wfc:WFCI*wfc \in wfcis } Unit
1348 wfc_beh(wfc,(clki,wdii),(loc,...))(d) \equiv
1350 let tf = tf1 \sqcap tf2 \sqcap ... \sqcap tfn,
1351 loc':LOC \bullet loc'= $\text{loc} \vee \text{loc}' \neq \text{loc}$,
1352 (t,ti):(TIME \times TI) \bullet ti \geq 0 in
1353 let wf = (ch_ic[wdii,wfc] ! mkReqWF(tf,loc',(t,ti))) ; ch_ic[wdii,wfc] ? in
1354 wfc_beh(wfc,(clki,wdii),(loc,...))((tf,loc',(t,ti)),wf) end end
1349 \sqcap
1355 let d':D { \EQ } rndr_D(d,{ \DOTDOTDOT }) in
1356 wfc_beh(wfc,(clki,wdii),(loc,...))(d') end

```

The choice of location may be that of the weather forecast consumer location, or it may be one different from that. The choice of time and time-interval is likewise a non-deterministic internal choice.

## R.5 Conclusion

### R.5.1 Reference to Similar Work

As far as I know there are no published literature nor, to our knowledge, institutional or private works on the subject of modelling weather data collection, interpretation and weather forecast delivery systems.

### R.5.2 What Have We Achieved ?

TO BE WRITTEN

### R.5.3 What Needs to be Done Next ?

TO BE WRITTEN

### R.5.4 Acknowledgments

This technical cum experimental research report was begun in Bergen, Wednesday, November 9, 2016 – inspired by a presentation by Ms. Doreen Tuheirwe, Makerere University, Kampala, Uganda. I thank her, and Profs. Magne Haveraaen and Jaakko Järvi of BLDL: the Bergen Language Design Laboratory, Dept. of Informatics, University of Bergen (Norway), for their early comments, and Prof. Haveraaen for inviting me to give PhD lectures there in the week of Nov. 6–12, 2016.



# Appendix S

## The Tokyo Stock Exchange, 2009

I thank Prof. Tetsuo Tamai, Tokyo University, for commenting on an early version of this chapter: clarifying issues and identifying mistakes and typos.

This chapter was begun on January 24. It was released, first time, January 28.

### Contents

---

|            |                                                     |            |
|------------|-----------------------------------------------------|------------|
| <b>S.1</b> | <b>Introduction</b>                                 | <b>568</b> |
| <b>S.2</b> | <b>The Problem</b>                                  | <b>568</b> |
| <b>S.3</b> | <b>A Domain Description</b>                         | <b>568</b> |
| S.3.1      | <b>Market and Limit Offers and Bids</b>             | 568        |
| S.3.1.1    | <b>Order Books</b>                                  | 569        |
| S.3.1.2    | <b>Aggregate Offers</b>                             | 570        |
| S.3.1.3    | <b>The TSE Itayose “Algorithm”</b>                  | 571        |
| <b>S.4</b> | <b>Conclusion</b>                                   | <b>573</b> |
| S.4.1      | <b>Match Executions</b>                             | 574        |
| S.4.2      | <b>Order Handling</b>                               | 574        |
| S.4.3      | <b>Conclusion</b>                                   | 574        |
| <b>S.5</b> | <b>Tetsuo Tamai’s Paper</b>                         | <b>575</b> |
| <b>S.6</b> | <b>Tokyo Stock Exchange arrowhead Announcements</b> | <b>583</b> |
| S.6.1      | <b>Change of trading rules</b>                      | 583        |
| S.6.2      | <b>Points to note when placing orders</b>           | 587        |

---

The reason why these notes are written is the appearance of [177]. I have taken the liberty of including that paper in this document, cf. Appendix S.5. I had the good fortune of visiting Prof. Tetsuo Tamai, Tokyo Univ., 8–Dec.8, 2009.

I read [177] late November.

I then had wished that Tetsuo had given it to me upon my arrival. I was, obviously ignorant of its publication some five months earlier.

I have now reread [177] (late January 2010).

I mentioned to Tetsuo that I would try my hand on a formalisation.

A description, both by a narrative, and by related formulas.

What you see here, in Chap. S, is a first attempt<sup>1</sup>.

---

<sup>1</sup>Earlier versions of this document will have Chap. S being very incomplete.

## S.1 Introduction

This chapter shall try describe: narrate and formalise some facets of the (now “old”<sup>2</sup>) stock trading system of the TSE: Tokyo Stock Exchange (especially the ‘matching’ aspects).

## S.2 The Problem

The reason that I try tackle a description (albeit of the “old” system) is that Prof. Tetsuo Tamai published a delightful paper [177, IEEE Computer Journal, June 2009 (vol. 42 no. 6) pp. 58-65)], *Social Impact of Information Systems*, in which a rather sad story is unfolded: a human error key input: an offer for selling stocks, although “ridiculous” in its input data (“*sell 610 thousand stocks, each at one (1) Japanese Yen*”, whereas one stock at 610,000 JPY was meant), and although several immediate — within seconds — attempts to cancel this “order”, could not be cancelled ! This lead to a loss for the selling broker at around 42 Billion Yen, at today’s exchange rate, 26 Jan. 2010, 469 million US \$s !<sup>3</sup> Prof. Tetsuo Tamai’s paper gives a, to me, chilling account of what I judge as an extremely sloppy and irresponsible design process by TSE and Fujitsu. It also leaves, I think, a strong impression of arrogance on the part of TSE. This arrogance, I claim, is still there in the documents listed in Footnote 2.

So the problem is a threefold one of

- **Proper Requirements:** How does one (in this case a stock exchange) prescribe (to the software developer) what is required by an appropriate hardware/software system for, as in this case, stock handling: acceptance of buy bids and sell offers, the possible withdrawal (or cancellation) of such submitted offers, and their matching (i.e., the actual trade whereby buy bids are marched in an appropriate, clear and transparent manner).
- **Correctness of Implementation:** does one make sure that the software/hardware system meets customers’ expectations.
- **Proper Explanation to Lay Users:** How does one explain, to the individual and institutional customers of the stock exchange, those offering stocks for sale of bids for buying stocks – how does one explain – in a clear and transparent manner the applicable rules governing stock handling.<sup>4</sup>

I shall only try contribute, in this document, to a solution to the first of these sub-problems.

## S.3 A Domain Description

### S.3.1 Market and Limit Offers and Bids

1. A market sell offer or buy bid specifies
  - (a) the unique identification of the stock,

---

<sup>2</sup> We write “old” since, as of January 4, 2010, that ‘old’ stock trading system has been replaced by the so-called arrowhead system. We refer to the following documents:

- <http://www.tse.or.jp/english/rules/equities/arrowhead/pamphlet.html>
- <http://www.tse.or.jp/english/rules/equities/arrowhead/pamphlet-e.pdf>
- <http://www.tse.or.jp/english/rules/equities/arrowhead/pamphlet1e.pdf>
- <http://www.tse.or.jp/english/rules/equities/arrowhead/pamphlet2e.html>

<sup>3</sup>So far three years of law court case hearing etc., has, on Dec. 4, 2009, resulted in complainant being awarded 10.7 billion Yen in damages. See <http://www.ft.com/cms/s/0/e9d89050-e0d7-11de-9f58-00144feab49a.html>.

<sup>4</sup>The rules as explained in the Footnote 2 listed documents are far from clear and transparent: they are full of references to fast computers, overlapping processing, etc., etc.: matters with which these buying and selling customers should not be concerned — so, at least, thinks this author !

- (b) the number of stocks to be sold or bought, and
  - (c) the unique name of the seller.
2. A limit sell offer or buy bid specifies the same information as a market sell offer or buy bid (i.e., Items 1a–1c), and
    - (d) the price at which the identified stock is to be sold or bought.
  3. A trade order is either a (mkMkt marked) market order or (mkLim marked) a limit order.
  4. A trading command is either a sell order or a buy bid.
  5. The sell orders are made unique by the mkSell “make” function.
  6. The buy orders are made unique by the mkBuy “make” function.

**type**

- √ 1 Market = Stock\_id × Number\_of\_Stocks × Name\_of\_Customer
  - 1a Stock\_id
  - √ 1b Number\_of\_Stocks =  $\{|n \cdot n: \mathbf{Nat} \wedge n \geq 1|\}$
  - 1c Name\_of\_Customer
  - 2 Limit = Market × Price
  - √ 2d Price =  $\{|n \cdot n: \mathbf{Nat} \wedge n \geq 1|\}$
  - 3 Trade == mkMkt(m:Market) | mkLim(l:Limit)
  - 4 Trading\_Command = Sell\_Order | Buy\_Bid
  - 5 Sell\_Order == mkSell(t:Trade)
  - 6 Buy\_Bid == mkBuy(t:Trade)

**S.3.1.1 Order Books**

7. We introduce a concept of linear, discrete time.
8. For each stock the stock exchange keeps an order book.
9. An order book for stock  $s_{id} : SI$  keeps track of limit buy bids and limit sell offers (for the identified stock,  $s_{id}$ ), as well as the market buy bids and sell offers; that is, for each price
  - (d) the number stocks, by unique order number, offered for sale at that price, that is, limit sell orders, and
  - (e) the number of stocks, by unique order number, bid for buying at that price, that is, bid orders;
  - (f) if an offer is a market sell offer, then the number of stocks to be sold is recorded, and if an offer is a market buy bid (also an offer), then the number of stocks to be bought is recorded,
10. Over time the stock exchange displays a series of full order books.
11. A trade unit is a pair of a unique order number and an amount (a number larger than 0) of stocks.
12. An amount designates a number of one or more stocks.

**type**

- √ 7 T, On [Time, Order number]
- 8 All\_Stocks\_Order\_Book = Stock\_Id  $\overrightarrow{m}$  Stock\_Order\_Book
- 9 Stock\_Order\_Book = (Price  $\overrightarrow{m}$  Orders) × Market\_Offers
- 9 Orders:: so:Sell\_Orders × bo:Buy\_Bids

```

9d Sell_Orders = On \overrightarrow{m} Amount
9e Buy_Bids = On \overrightarrow{m} Amount
9f Market_Offers :: mkSell(n:Nat) \times mkBuy(n:Nat)
10 TSE = T \overrightarrow{m} All_Stocks_Order_Book
11 TU = On \times Amount
12 Amount = $\{|n \cdot \text{Nat} \wedge n \geq 1|\}$

```

### S.3.1.2 Aggregate Offers

13. We introduce the concepts of aggregate sell and buy orders for a given stock at a given price (and at a given time).
14. The aggregate sell orders for a given stock at a given price is
  - (g) the stocks being market sell offered and
  - (h) the number of stocks being limit offered for sale at that price or lower
15. The aggregate bids for a given stock at a given price is
  - (i) including the stocks being market bid offered and
  - (j) the number of stocks being limit bid for buying at that price or higher

#### value

```

14 aggr_sell: All_Stocks_Order_Book \times Stock_Id \times Price \rightarrow Nat
14 aggr_sell(asob,sid,p) \equiv
14 let ((sos,_) , (mkSell(ns),_)) = asob(sid) in
14g ns +
14h all_sell_summation(sos,p) end
15 aggr_buy: All_Stocks_Order_Book \times Stock_Id \times Price \rightarrow Nat
15 aggr_buy(asob,sid,p) \equiv
15 let ((_,bbs), (_,mkBuy(nb))) = asob(sid) in
15i nb +
15j nb + all_buy_summation(bbs,p) end

```

```

all_sell_summation: Sell_Orders \times Price \rightarrow Nat
all_sell_summation(sos,p) \equiv
 let ps = $\{p' | p': \text{Prices} \cdot p' \in \text{dom sos} \wedge p' \geq p\}$ in accumulate(sos,ps)(0) end

```

```

all_buy_summation: Buy_Bids \times Price \rightarrow Nat
all_buy_summation(bbs,p) \equiv
 let ps = $\{p' | p': \text{Prices} \cdot p' \in \text{dom bos} \wedge p' \leq p\}$ in accumulate(bbs,ps)(0) end

```

The auxiliary `accumulate` function is shared between the `all_sell_summation` and the `all_buy_summation` functions. It sums the amounts of limit stocks in the price range of the `accumulate` function argument `ps`. The auxiliary `sum` function sums the amounts of limit stocks — “peeling off” the their unique order numbers.

#### value

```

accumulate: (Price \overrightarrow{m} Orders) \times Price-set \rightarrow Nat \rightarrow Nat
accumulate(pos,ps)(n) \equiv
 case ps of $\{\}$ \rightarrow n, $\{p\} \cup ps' \rightarrow$ accumulate(pos,ps')(n+sum(pos(p)) $\{\text{dom pos}(p)\}$) end

sum: (Sell_Orders|Buy_Bids) \rightarrow On-set \rightarrow Nat
sum(ords)(ns) \equiv
 case ns of $\{\}$ \rightarrow 0, $\{n\} \cup ns' \rightarrow$ ords(n)+sum(ords)(ns') end

```



To handle the `sub_limit_sells` and `sub_limit_buys` indicated by Item 17c of the `Itayose` “algorithm” we need the corresponding `sub_sell_summation` and `sub_buy_summation` functions:

**value**

```
sub_sell_summation: Stock_Order_Book × Price → Nat
sub_sell_summation(((sos,_) , (ns,_)), p) ≡ ns +
 let ps = {p'|p':Prices • p' ∈ dom sos ∧ p' > p} in accumulate(sos,ps)(0) end

sub_buy_summation: Stock_Order_Book × Price → Nat
sub_buy_summation(((_,bbs) , (_,nb)), p) ≡ nb +
 let ps = {p'|p':Prices • p' ∈ dom bos ∧ p' < p} in accumulate(bbs,ps)(0) end
```

### S.3.1.3 The TSE Itayose “Algorithm”

16. The TSE practices the so-called `Itayose` “algorithm” to decide on opening and closing prices<sup>5</sup>. That is, the `Itayose` “algorithm” determines a single so-called ‘execution’ price, one that matches sell and buy orders<sup>6</sup>:

17. The “matching sell and buy orders” rules:

- (a) *All market orders must be ‘executed’<sup>7</sup>.*
- (b) *All limit orders to sell/buy at prices<sup>8</sup> than the ‘execution price’<sup>9</sup> must be executed.*
- (c) *The following amount of limit orders to sell or buy at the execution prices must be executed: the entire amount of either all sell or all buy orders, and at least one ‘trading unit’<sup>10</sup> from ‘the opposite side of the order book’<sup>11</sup>.*

- The 28 January 2010 version had lines
  - 17c<sub>3</sub> name `some_priced_buys`, should have been, as now, `some_priced_sells` and
  - 17c<sub>4</sub> name `all_priced_buys`, should have been, as now, `all_priced_sells`.
- My current understanding of *and assumptions* about the TSE is
  - that each buy bid or sell order concerns a number, *n*, of one or more of the same kind of stocks (i.e. `sid`).
  - that each buy bid or sell order when being accepted by the TSE is assigned a unique order number *on*, and
  - that this is reflected in some `Sell_Orders` or `Market_Bids` entry being augmented.<sup>12</sup>
- For current (Monday 22 Feb., 2010) lack of a better abstraction<sup>13</sup>, I have structured the `Itayose` “Algorithm” as follows:
  - (17’) either a `match` can be made based on
    - \* all `buys` and

<sup>5</sup> [177, pp 59, col. 1, lines 4-3 from bottom, cf. Page 576]

<sup>6</sup> [177, pp 59, col. 2, lines 1-3 and Items 1.-3. after yellow, four line ‘insert’, cf. Page 576] These items 1.-3. are reproduced as “our” Items 17a-17c.

<sup>7</sup>To execute an order:

<sup>8</sup>

<sup>9</sup>Execution price:

<sup>10</sup>Trading unit:

<sup>11</sup>The opposite side of the order book:

<sup>12</sup>The present, 22.2.2010, model “lumps” all market orders. This simplification must be corrected, as for the `Sell_Orders` and `Market_Bids`, the `Market_Offers` must be modelled as are `Orders`.

<sup>13</sup>One that I am presently contemplating is based on another set of **pre/post** conditions.

- \* some sells,
- (17'<sub>∨</sub>) or
- (17'') a match can be made based on
  - \* some buys and
  - \* all sells.

**value**

17 match: All\_Stocks\_Order\_Book × Stock\_Id → Price-set

17 match(asob,sid) as ps

17 **pre:** sid ∈ dom asob

17 **post:** ∃ p':Price • p' ∈ ps ⇒

17' all\_buys\_some\_sells(p',ason,sid,ps) ∨

17'<sub>∨</sub> ∨

17'' some\_buys\_all\_sells(p',ason,sid,ps)

- (17') The all\_buys\_some\_sells part of the above disjunction “calculates” as follows:

- The all\_buys... part includes
  - \* all the market\_buys
  - \* all the buys properly below the stated price, and
  - \* all the buys at that price.
- The ...some\_sells part includes
  - \* all the market\_sells
  - \* all the sells properly below the stated price, and
  - \* some of the buys at that price.

17' all\_buys\_some\_sells(p',ason,sid,ps) ≡

17' ∃ os:On-set •

17a' all\_market\_buys(asob(sid))

17b' + all\_sub\_limit\_buys(asob(sid))(p')

17c' + all\_priced\_buys(asob(sid))(p')

17a' = all\_market\_sells(asob(sid))

17b' + all\_sub\_limit\_sells(asob(sid))(p')

17c'<sub>∃</sub> + some\_priced\_sells(asob(sid))(p')(os)

- (17'') As for the above, only “versed”.

17'' some\_buys\_all\_sells(p',ason,sid,ps) ≡

17'' ∃ os:On-set •

17a'' all\_market\_buys(asob(sid))

17b'' + all\_sub\_limit\_buys(asob(sid))(p')

17c'' + some\_priced\_buys(asob(sid))(p')(os)

17a'' = all\_market\_sells(asob(sid))

17b'' + all\_sub\_limit\_sells(asob(sid))(p')

17c''<sub>∨</sub> + all\_priced\_sells(asob(sid))(p') ∨

The match function calculates a set of prices for each of which a match can be made. The set may be empty: there is no price which satisfies the match rules (cf. Items 17a–17c below). The set may be a singleton set: there is a unique price which satisfies match rules Items 17a–17c. The set may contain more than one price: there is not a unique price which satisfies match rules Items 17a–17c.

The single (') and the double (") quoted (17a–17c) group of lines, in the **match** formulas above, correspond to the Itayose “algorithm”s Item 17c ‘*opposite sides of the order book*’ description. The existential quantification of a set of order numbers of lines 17' and 17" correspond to that “algorithms” (still Item 17c) point of *at least one ‘trading unit’*. It may be that the **post** condition predicate is only fulfilled for all trading units – so be it.

**value**

all\_market\_buys: Stock\_Order\_Book  $\rightarrow$  Amount  
 all\_market\_buys((\_,(mkBuys(nb))),p)  $\equiv$  nb

all\_market\_sells: Stock\_Order\_Book  $\rightarrow$  Amount  
 all\_market\_sells((\_,(mkSells(ns),\_)),p)  $\equiv$  ns

all\_sub\_limit\_buys: Stock\_Order\_Book  $\rightarrow$  Price  $\rightarrow$  Amount  
 all\_sub\_limit\_buys(((bbs,\_)))(p)  $\equiv$  sub\_buy\_summation(bbs,p)

all\_sub\_limit\_sells: Stock\_Order\_Book  $\rightarrow$  Price  $\rightarrow$  Amount  
 all\_sub\_limit\_sells((sos,\_))(p)  $\equiv$  sub\_sell\_summation(sos,p)

all\_priced\_buys: Stock\_Order\_Book  $\rightarrow$  Price  $\rightarrow$  Amount  
 all\_priced\_buys((\_,bbs),\_)(p)  $\equiv$  sum(bbs(p))

all\_priced\_sells: Stock\_Order\_Book  $\rightarrow$  Price  $\rightarrow$  Amount  
 all\_priced\_sells((sos,\_),\_)(p)  $\equiv$  sum(sos(p))

some\_priced\_buys: Stock\_Order\_Book  $\rightarrow$  Price  $\rightarrow$  On-set  $\rightarrow$  Amount  
 some\_priced\_buys((\_,bbs),\_)(p)(os)  $\equiv$   
**let** tbs = bbs(p) **in if** {} $\neq$ os $\wedge$ os $\subseteq$ dom tbs **then** sum(tbs)(os) **else 0 end end**

some\_priced\_sells: Stock\_Order\_Book  $\rightarrow$  Price  $\rightarrow$  On-set  $\rightarrow$  Amount  
 some\_priced\_sells((sos,\_),\_)(p)(os)  $\equiv$   
**let** tss = sos(p) **in if** {} $\neq$ os $\wedge$ os $\subseteq$ dom tss **then** sum(tss)(os) **else 0 end end**

The formalisation of the Itayise “algorithm”, as well as that “algorithm” [itself], does not guarantee a match where a match “ought” be possible. The “stumbling block” seems to be the Itayose “algorithm”s Item 17c. There it says: ‘*at least one trading unit*’. We suggest that a match could be made in which some of the stocks of a candidate trading unit be matched with the remaining stocks also being traded, but now with the stock exchange being the buyer and with the stock exchange immediately “turning around” and posting those remaining stocks as a TSE marked trading unit for sale.

18. It seems to me that the Tetsuo Tamai paper does not really handle

- (a) the issue of order numbers,
- (b) therefore also not the issue of the number of stocks to be sold or bought per order number.

19. Therefore the Tetsuo Tamai paper does not really handle

- (a) the situation where a match “only matches” part of a buy or a sell order.

## S.4 Conclusion

Much more to come: essentially I have only modelled column 2, rightmost column, Page 59 of [177, Tetsuo Tamai, “TSE”]. Next to be modelled is column 1, leftmost column, Page 60 of [177]. See these same page numbers of the present document !

**S.4.1 Match Executions**

TO BE WRITTEN

**S.4.2 Order Handling**

TO BE WRITTEN


**S.4.3 Conclusion**

TO BE WRITTEN

## S.5 Tetsuo Tamai's Paper

For private, limited circulation only, I take the liberty of enclosing Tetsuo Tamai's IEEE Computer Journal paper.

COVER FEATURE



# SOCIAL IMPACT OF INFORMATION SYSTEM FAILURES

*Tetsuo Tamai, University of Tokyo*

The social impact of information systems becomes visible when serious system failures occur. A case of mistyping in entering a stock order by Mizuho Securities and the following lawsuit between Mizuho and the Tokyo Stock Exchange sheds light on the critical role of software in society.

A

lmost daily, we hear news of system failures that have had a serious impact on society. The ACM Risks Forum moderated by Peter Neumann is an informative source that compiles various reported instances of computer-related risks (<http://catless.ncl.ac.uk/risks>).

One of journalism's shortcomings is that it makes a loud outcry when trouble occurs with a computer-based system, but it remains silent when nothing goes wrong. This gives the general public the wrong impression that computer systems are highly unreliable. Indeed, as software is invisible and not easy for ordinary people to understand, they generally perceive software to be something unfathomable and undependable.

Another problem is that when a system failure occurs, news sources offer no technical details. Reporters usually

don't have the knowledge about software and information systems needed to report technically significant facts, and the stakeholders are generally reluctant to disclose details. The London Ambulance Service failure case is often cited in software engineering literature because its detailed inquiry report is open to the public, which only emphasizes how rare such cases are ([www.cs.ucl.ac.uk/staff/a.finkelstein/las/lascase09.pdf](http://www.cs.ucl.ac.uk/staff/a.finkelstein/las/lascase09.pdf)).

MIZUHO SECURITIES VERSUS THE TOKYO STOCK EXCHANGE

The case of Mizuho Securities versus the Tokyo Stock Exchange (TSE) is archived in the 12 December 2005 issue of the *Risks Digest* (<http://catless.ncl.ac.uk/risks/24.12.html>), and additional information can be obtained from sources such as the *Times* ([www.timesonline.co.uk/tol/news/world/asia/article755598.ece](http://www.timesonline.co.uk/tol/news/world/asia/article755598.ece)) and the *New York Times* ([www.nytimes.com/2005/12/13/business/worldbusiness/13gjitc.html?\\_r=1](http://www.nytimes.com/2005/12/13/business/worldbusiness/13gjitc.html?_r=1)), among others.

The incident started with the mistyping of an order to sell a share of J-Com, a start-up recruiting company, on the day its shares were first offered to the public. An employee at Mizuho Securities, intending to sell one share at 610,000 yen, mistakenly typed an order to sell 610,000 shares at 1 yen.

What happened after that was beyond imagination. The order went through and was accepted by the Tokyo Stock Exchange Order System. Mizuho noticed the blunder and tried to withdraw the order, but the cancel command failed repeatedly. Thus, it was obliged to start buying back the shares itself to cut the loss. In the end, Mizuho's total loss amounted to 40 billion yen (\$225 million). Four days later, TSE called a news conference and admitted that the cancel command issued by Mizuho failed because of a program error in the TSE system. Mizuho demanded compensation for the loss, but TSE refused. Then, Mizuho sued TSE for damages.

When such a case goes to court, we can gain access to documents presented as evidence, which provides a rare opportunity to obtain information about the technical details behind system failures. Still, requesting and acquiring documents from the court requires considerable effort by the third party. As it happened, Mizuho contacted me to give an expert opinion, thus I had access to all materials presented to the court. Admittedly, there is always the possibility of bias, but as a scientist, I have endeavored to report this case as impartially as possible.

Another reason for examining this case is that it involved several typical and interesting software engineering issues including human interface design, fail-safety issues, design anomalies, error injection by fixing code, ambiguous requirements specification, insufficient regression testing, subcontracting, product liability, and corporate governance.

### WHAT HAPPENED

J-Com was initially offered on the Tokyo Stock Exchange Mother Index on 8 December 2005. On that day, a Mizuho employee got a call from a client telling him to sell a single share of J-Com at 610,000 yen. At 9:27 a.m., the employee entered an order to sell 610,000 shares at 1 yen through a Fidessa (Mizuho's securities ordering system) terminal. Although a "Beyond price limit" warning appeared on the screen, he ignored it (pushing the Enter key twice meant "ignore warning" by the specification), and the order was sent to the TSE Stock Order System. J-Com's outstanding shares totaled 14,500, which means the erroneous order was to sell 42 times the total number of shares.

At 9:28 a.m., this order was displayed on the TSE system board, and the initial price was set at 672,000 yen.

### Price determination mechanism

TSE stock prices are determined by two methods: *Itayose* (matching on the board) and *Zaraba* (regular market). The *Itayose* method is mainly used to decide opening and closing prices; the *Zaraba* method is used during continuous auction trading for the rest of the trading session. In the

J-Com case, the *Itayose* method was used as it was the first day of determining the J-Com stock price.

There are two order types for selling or buying stocks: *market orders* and *limit orders*. Market orders do not specify the price to buy or sell and accept the price the market determines, while limit orders specify the price. When sell and buy orders are matched to execute trading, market orders of both sell and buy are always given the first priority.

Market participants generally want to buy low and sell high. But when the *Itayose* method is applied, there is no current market price to refer to, and thus there can be a variety of sell/buy orders, resulting in a wide range of

**An employee at Mizuho Securities, intending to sell one share at 610,000 yen, mistakenly typed an order to sell 610,000 shares at 1 yen.**

prices. With the *Itayose* method, a single execution price is determined that matches sell and buy orders by satisfying the following rules:

1. All market orders must be executed.
2. All limit orders to sell/buy at prices lower/higher than the execution price must be executed.
3. The following amount of limit orders to sell or buy at the execution price must be executed: the entire amount of either all sell or all buy orders, and at least one trading unit from the opposite side of the order book.

The third rule is complicated but functions as a tie-breaker when the first two rules do not determine a unique price. Looking at an example helps to understand how the rules work.

Table 1 represents an instance of the order book. The center column gives the prices. The left center column shows the volume of sell offers at the corresponding price, while the right center column shows the volume of the buy bids. The volume of the market sell orders and the market buy orders is displayed at the bottom line and at the top line, respectively. The leftmost column shows the aggregate volume of sell offers (working from the bottom to the top in the order of priority), and the rightmost column gives the aggregate volume of buy bids (working from the top to the bottom in the order of priority).

We start by focusing on rules (1) and (2) to determine the opening price. First, the price level is searched where the amounts of the aggregated sell and the aggregated buy cross over. In this case, the line is between 500 yen and 499

## COVER FEATURE

Table 1. Order book example illustrating Itayose method.

| Sell offer            |                       | Price (yen) | Buy bid           |                      |
|-----------------------|-----------------------|-------------|-------------------|----------------------|
| Aggregate sell orders | Shares offered at bid |             | Buy offers at bid | Aggregate buy orders |
|                       |                       | Market      | 4,000             |                      |
| 48,000                | 8,000                 | 502         | 0                 | 4,000                |
| 40,000                | 20,000                | 501         | 2,000             | 6,000                |
| 20,000                | 5,000                 | 500         | 3,000             | 9,000                |
| 15,000                | 6,000                 | 499         | 15,000            | 24,000               |
| 9,000                 | 3,000                 | 498         | 8,000             | 32,000               |
| 6,000                 | 0                     | 497         | 20,000            | 52,000               |
|                       | 6,000                 | Market      |                   |                      |

yen. These two prices satisfy conditions (1) and (2), so they are the opening price candidates. Then, applying rule (3), the price is finally determined as 499 yen.

Of course, this algorithm does not always determine the price. For example, if the orders are all buy and no sell, there is no solution that satisfies all three rules. An additional mechanism that holds back transactions even if the matching price is found by the Itayose method is a measure to prevent sudden price leaps or drops. On the TSE, an immediate execution only takes place if the next execution price is within a certain range from the previous execution price. The price level determines the range. For example, if the most recently executed price was 500 yen, the next execution price must be within the range of 490-510 yen. In other words, it can only fluctuate up to 10 yen in either direction.

Suppose the matched price is beyond this range—for example, 550 yen when the previous price was 500 yen. Then, execution does not take place; instead a special bid quote of 510 yen is indicated to call for offers at this price. If no offers at this price are received, the special bid quote will be raised to 520 yen after 5 minutes, and so on until equilibrium is achieved. This mechanism is intended to make a smooth transition between widely divergent prices.

But on the morning of 8 December, J-Com had no previous price. In such cases, the publicly assessed value is used in place of the previous price, which was 610,000 yen. Because the matched price was much higher, a special bid quote of 610,000 yen was shown at 9:00 a.m., then raised to 641,000 yen at 9:10 a.m., which means the range was  $\pm 31,000$  yen, and raised again to 672,000 yen at 9:20 a.m. Table 2 shows the order book at that moment, when the 1-yen sell offer came in.

#### Initial price determination

The term “reverse special quote” denotes this particularly rare event. It means that when a special buy bid quote is displayed, a sell order of low price with a significant

amount that reverses the situation to a special sell offer quote comes in (or conversely a special sell offer quote is reversed to a special buy bid quote). TSE has another rule that applies to such a case. This rule stipulates that the previous special quote is fixed as the execution price, and the transaction proceeds. Thus, the initial price of J-Com was now determined to be 672,000 yen. In addition to the step price range set for reducing sudden price change, there is also a price limit range for a day. The upper and lower limits of the price for each stock are defined based on the initial price of the day. In the J-Com case, the limits were defined at the moment when the initial price was determined: The upper limit was 772,000 yen, and the lower limit was 572,000 yen.

In regular trading, the price limits are fixed at the start of the market day, and orders with prices exceeding the limit (either upper or lower) are rejected. But when the initial price is determined during the market time, as in the J-Com case, orders received before the price limits are set are not ignored. Instead, the price of an order exceeding the upper limit is adjusted to the upper price limit, and an order under the lower limit is adjusted to the lower price limit. Thus, the 1-yen order by Mizuho was adjusted to 572,000 yen.

Noticing the mistake, Mizuho entered a cancel command through a Fidessa terminal at 9:29:21, but it failed. Between 9:35:17 and 9:35:40, Mizuho tried to cancel the order several times through TSE system terminals that are installed at the Mizuho site, but the cancellations failed. Mizuho called TSE asking for a cancellation on the TSE side, but the answer was no.

At 9:35:33, Mizuho started to buy back J-Com shares. In the end, it could only buy back 510,000 shares; nearly 100,000 shares were bought by others and never restored.

#### Aftermath

On 12 December, four days after the incident, TSE president Takuo Tsurushima held a press conference and admitted that the order cancellation by Mizuho failed because of a defect in the TSE Stock Order System.



Table 2. Order book for J-Com stock at 9:20 a.m.

| Sell offer |        | Price (yen) | Buy bid |           |
|------------|--------|-------------|---------|-----------|
| Aggregate  | Amount |             | Amount  | Aggregate |
|            |        | Market      | 253     |           |
| 1,432      | 695    | OVR*        | 1,479   | 1,732     |
| 737        |        | 6,750       | 4       | 1,736     |
| 737        |        | 6,740       | 6       | 1,742     |
| 737        |        | 6,730       | 6       | 1,748     |
| 737        | 3      | 6,720**     | 28      | 1,776     |
| 734        |        | 6,710       | 2       | 1,778     |
| 734        |        | 6,700       | 3       | 1,781     |
| 734        | 1      | 6,690       | 1       | 1,782     |
| 733        |        | 6,680       | 1       | 1,783     |
| 733        | 114    | UDR***      | 120     | 1,903     |
|            | 619    | Market      |         |           |

\* More than 675,000 yen \*\* Special buy quote \*\*\* Less than 668,000 yen

Mizuho could not buy back 96,236 shares, and it was impossible for Mizuho to deliver real shares to those who had bought them. An exceptional measure was taken to settle trading by paying 912,000 yen per share in cash. The result was a 30-billion-yen loss to Mizuho. Mizuho had already suffered a loss of 10 billion yen by buying back 510,000 shares, thus the total loss amounted to 40 billion yen.

Mizuho and TSE started negotiations on compensation for damages in March 2006, but they failed to reach an agreement. Mizuho sent a formal letter to TSE in August 2006 requesting compensation, which TSE declined by sending a letter of refusal.

Mizuho filed a suit against TSE in the Tokyo District Court on 27 October 2006, demanding compensation of 41.5 billion yen. The first oral pleadings took place on 15 December 2006, and trials were held 13 times in two years, the last on 19 December 2008. The court's decision in that trial was scheduled to be given on 27 February 2009, but the court decided to postpone the decision.

In the contract between TSE and each user of the TSE Stock Order System, including Mizuho, there is a clause on exemption from responsibility on the TSE side except when a serious mistake is attributed to TSE. The crucial issue was whether the damage caused by the system defect was due to a serious mistake beyond the range of exemption. TSE also argued that as the incident started with a mistake on the Mizuho side, the mistakes and the resulting damages should be canceled out.

#### PROBABLE CAUSE

The TSE system unduly rejected the Mizuho order cancellation because the module for processing order cancellation erroneously judged that the J-Com target

sell order had been completely executed, thus leaving no transactions to be canceled. This bug had been hiding for five years.

Fujitsu developed the system under contract with TSE and released it for use in May 2000. An evidence document submitted to the court reported that a similar error was found during integration testing in February 2000 and that the current fault occurred as a result of fixing that error.

But there are several mysteries surrounding this apparently simple failure case. Initially, TSE maintained that the target cancellation order could not be found because its price had been changed from 1 yen to the adjusted price of 572,000 yen, whereas the designated cancel price command was the original 1 yen. This explanation is bizarre as it implies that the order data is searched in the database using price as a key when it is obvious that price cannot be a key because there can be multiple orders with the same price. In addition, as this case shows, the price of the same order can be modified during the transaction. This explanation turned out to be wrong, but it came from the fact that there was indeed a logic in the procedure that partly used price to search order data. TSE also maintained that if buy orders did not flow in continuously and thus the target sell orders were not always being matched to buy orders, the order cancel module would not have been invoked within the order matching module but instead invoked in the order entry module, and then the cancellation would have succeeded. However, this explanation implies that different cancel modules are called or the same module behaves differently according to when it is invoked.

The third question, and probably the most crucial one with respect to the direct cause of the error, is how data handling identifies orders causing a reverse special quote. That information is written into a database containing



## COVER FEATURE

the order book data, but once the information is used in determining the execution price, it is immediately cleared. The rationale behind this design decision is mysterious. The programmer who was charged with fixing the February 2000 bug intended to use this data to judge the type of order to be canceled but he did not know that the data no longer existed.

TSE and Fujitsu claimed that this incident occurred in a highly exceptional situation when the following seven conditions held at the same time:

1. The daily price limits have not been determined.
2. The special quote is displayed.
3. The reverse special quote occurs.
4. The price of the order that has caused the reverse special quote is out of the newly defined daily price limits.
5. The target order of cancellation caused the reverse special quote.
6. The target cancellation order is in the process of sell and buy matching, which forces the cancellation process to wait.
7. The target order is continually being matched.

### The order cancellation module appears to have insufficient cohesion as different functions are overloaded.

A general procedure for the order cancellation module would be as follows:

1. Find the order to be canceled.
2. Determine if the order satisfies conditions for cancellation.
3. Execute cancellation if the conditions are met.

Because each order has a few simple attributes—stock name, sell or buy, remaining number of shares to be processed (if 0, the order is completed), and price—the condition that an order can be canceled is straightforward: “the remaining number of shares to be processed is greater than zero.” There is only one other condition that cannot be determined by the order attribute data but can be determined by its execution state: If the target order is in the process of matching, the cancel process must wait.

A remarkable point to note is that factors such as undefined limit price, display of special quote, reversing special quote, price adjustment to the limit, and so forth have no influence on the cancellation judgment. Thinking in this way, it seems that the system design artificially introduced the seven complicated conditions listed by TSE and Fujitsu.

### DESIGN ANOMALY

Figure 1 shows a flowchart of the module that handles order cancellation. Because order cancellation and order change are processed in the same way, the two functions are overloaded in this same module, but for the sake of simplicity I only deal with order cancellation.

The flowchart is not shown to provide details but to illustrate the kind of documents presented to the court. It is extracted and modified from a document submitted as evidence by the defendant, which was an analysis of the error reported by a TSE system engineer. The plaintiff required the defendant to provide the entire design specification and source code, but the defendant refused and the judge did not force the issue, being reluctant to go into technical details in court.

Part A of the flowchart deals with the logic of price adjustment to limit if necessary. The decision logic is as follows:

- if the order to cancel is sell and the price is lower than the lower limit, it is adjusted to the lower limit; and
- if the order to cancel is buy and the price is higher than the upper limit, it is adjusted to the upper limit.

Part B of the flowchart is the logic inserted in February 2000 when an error was found during testing and caused a failure in December 2005. Its logic is as follows: If called in the order matching process; and limit prices are already set; and the order to cancel is a buy over the limit price or a sell under the limit price and is not a reverse special quote order, then a cancellation is infeasible because all shares are already executed. Although this logic is unduly complicated, it is sound only if all the if-conditions are correctly judged. Unfortunately, the judgment on “if not a reverse special quote” gave a wrong answer of “true” in this Mizuho case, and the decision erroneously judged that the cancellation was infeasible.

Insufficient information is available to allow capturing details of the system design, but from what is available we can infer the following design flaws.

### Problems in database design

Three databases are related to the problem in this case: Order DB, Sell/Buy Price DB, and Stock Brand DB. The Order DB stores data of all entered orders. This database should include the current attributes of each order, including those necessary for judging whether the designated order can be canceled. For example, because there is a record field for the executed shares in this database, determining if all the shares of the order have been executed or not should be a trivial process. However, due to the time gap between usage and update of the data, the process is much more complicated. If the principle of database integrity is respected, the logic would be much clearer, but performance seems to be given higher priority than integrity.

Part A of the flowchart in Figure 1 calculates price adjustment within the cancellation handling module, which implies that the price data in the Order DB does not reflect the current status.

The Sell/Buy Price DB sorts sell/buy orders by price for each stock brand. This is by nature a secondary database constructed from the Order DB. The secondary index is price, but identifying an order uniquely in the database requires the order ID. The explanation that price is used to search the database must refer to search in this database, and the price adjustment logic embedded in the order cancel module should be related to it. The data handling over the Price DB and the Order DB appears to be unduly complicated.

The Stock Brand DB corresponds to a physical order book for each stock, but its substantial data is stored in the Sell/Buy Price DB and only some specific data for each stock brand is kept here. However, to implement a rule that an order that has caused a reverse special quote has an exceptional priority in matching—lower than the regular case—the customer ID and order ID of such a stock is written in this database, and they are cleared as soon as the matching is done. This kind of temporary usage of a database goes against the general principle that a database should save persistent data accessed by multiple modules.

**Problems in module design.**

The part of the system that handles order cancellation appears to have low modularity. The logic in part B of the flowchart made a wrong judgment because the information telling it that the target order had induced the reverse special quote had been temporarily written on the Stock Brand DB by the order matching module and had already been cleared. This implies an accidental module coupling between the order matching and order cancelling modules.

The order cancellation module appears to have insufficient cohesion as different functions are overloaded. It is not clear how the tasks of searching the target order to be canceled, determining cancellability, executing cancellation, and updating the database are this module's responsibility.

**LESSONS LEARNED**

In addition to the insights into the associated software design problems, this case provides lessons learned with regard to software engineering technologies, processes, and social aspects.

**Safety and human interface**

If the order entry system on either the Mizuho or TSE side had been equipped with more elaborate safety measures, the accident could have been avoided. It was not the first time that the mistyping of a stock order resulted in a big loss. For instance, in December 2001, a trader at UBS

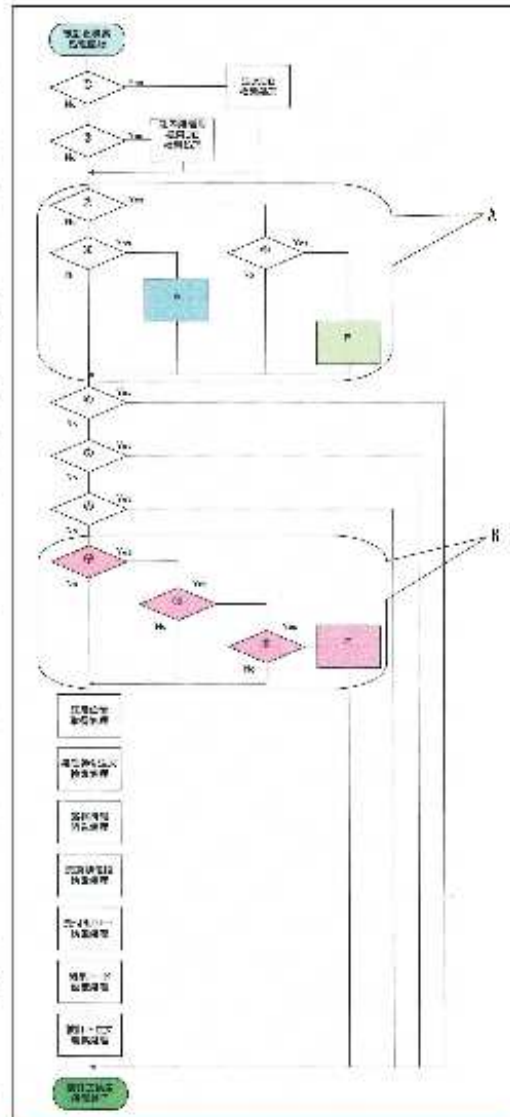


Figure 1. Flowchart of the order cancellation module.

Warburg, the Swiss investment bank, lost more than 10 billion yen while trying to sell 16 shares of the Japanese advertising company Dentsu at 610,000 yen each. He sold 610,000 shares at six yen each. (The similarity between these two cases, including the common figure of 610,000, is remarkable.)

## COVER FEATURE

Table 3. Associations between the Software Engineering Code of Ethics and Professional Practice and the TSE-Mizuho case.

| Engineering Issue          | Applicable ACM/IEEE-CS Principle | Ethics clause                                                                                                                                                                                                                                                            |
|----------------------------|----------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Design anomaly             | 3.01                             | Strive for high quality, acceptable cost and a reasonable schedule, ensuring significant tradeoffs are clear to and accepted by the employer and the client, and are available for consideration by the user and the public.                                             |
|                            | 3.14                             | Maintain the integrity of data, being sensitive to outdated or flawed occurrences.                                                                                                                                                                                       |
| Safety and human interface | 1.03                             | Approve software only if they have a well-founded belief that it is safe, meets specifications, passes appropriate tests, and does not diminish quality of life, diminish privacy or harm the environment. The ultimate effect of the work should be to the public good. |
|                            | 3.07                             | Strive to fully understand the specifications for software on which they work.                                                                                                                                                                                           |
| Requirements specification | 3.08                             | Ensure that specifications for software on which they work have been well documented, satisfy the users' requirements and have the appropriate approvals.                                                                                                                |
|                            | 3.10                             | Ensure adequate testing, debugging, and review of software and related documents on which they work.                                                                                                                                                                     |
| Role of user and developer | 4.02                             | Only endorse documents either prepared under their supervision or within their areas of competence and with which they are in agreement.                                                                                                                                 |
|                            | 5.01                             | Ensure good management for any project on which they work, including effective procedures for promotion of quality and reduction of risk.                                                                                                                                |
| Chain of subcontracting    | 2.01                             | Provide service in their areas of competence, being honest and forthright about any limitations of their experience and education.                                                                                                                                       |
|                            | 3.04                             | Ensure that they are qualified for any project on which they work or propose to work by an appropriate combination of education and training, and experience.                                                                                                            |

The habit of ignoring warning messages is common, but it was a critical factor in these cases. It raises the question of how to design a safe—but not clumsy—human interface.

#### Requirements specification

Development of the current TSE Stock Order System started with the request for proposal (RFP) that TSE presented to the software industry in January 1998. Two companies submitted proposals, and TSE selected Fujitsu as the vendor with which to contract. After several discussions between TSE and Fujitsu, Fujitsu wrote the requirements specification, which TSE approved.

With respect to the order cancellation requirement, it is only mentioned as a function to “Cancel order” in the RFP, and no further details are given there. In the requirements specification, six conditions are listed when cancel (or change) orders are not allowed, but none of them fit the Mizuho case. The document also states that “in all the other cases, change/cancel condition checking should be

the same as the current system.” Here, “current system” refers to the prior version of the TSE Stock Order System, also developed by Fujitsu, which had been in use until May 2000.

The phrase “the same as the current system” frequently appears in this requirements specification, which was criticized by software experts after the Mizuho incident was publicized. The phrase may be acceptable if there is a consensus between the user and the developer on what it means in each context, but when things go wrong, the question arises whether the specification descriptions were adequate.

#### Verification and validation

The fact that an error was injected while fixing a bug found in testing is so typical that every textbook on testing warns about this possibility. It is obvious that regression testing was not properly done. It is perhaps too easy to criticize this oversight, but it would be worthwhile to study why it happened in this particular case. So far, not many details have been disclosed.



### Role of user and developer

It is conceivable that communication between the user and the developer was inadequate during the TSE system development. The user, TSE, basically did not participate in the process of design and implementation. More involvement of the user during the entire development process would have promoted deeper understanding of the requirements by the developer, and the defect injected during testing might have been avoided.

### Subcontracting chain

As in many large-scale information system development projects, the TSE system project was organized in a hierarchical subcontracting structure. The engineer who was in charge of fixing the code in question had a low position in the subcontracting chain. This organizational structure was the likely cause of the misunderstanding about database usage. Such a subcontract structure has often been studied from the industry and labor problem point of view, but it is also important to examine it from the engineering point of view.

### Product liability

The extent to which software is regarded as a product amenable to product liability laws may depend on legal and cultural boundaries, but there is a general worldwide trend demanding stricter liability for software. More lawsuits are being filed, and thus software engineers must be more knowledgeable about software product liability issues.

### ETHICAL ISSUES

This case raises several questions about professional ethics. However, we should be careful in relating ethical issues and legal matters. Illegal conduct and unethical conduct are of course not equivalent. Moreover, the Mizuho incident is a civil case, not a criminal case.

The Software Engineering Code of Ethics and Professional Practice developed by an ACM and IEEE Computer Society joint task force provides a good framework for discussing ethical issues. The Code comprises eight principles, and each clause is numbered by its principle category and the sequence within the principle. The principles are numbered as 1: Public, 2: Client and Employer, 3: Product, 4: Judgment, 5: Management, 6: Profession, 7: Colleagues, and 8: Self.

As Table 3 shows, some clauses in the Code have relatively strong associations with various aspects of the TSE-Mizuho case. However, this discussion is by no means intended to blame the software engineers who participated in planning, soliciting requirements, designing, implementing, testing, or maintaining the TSE system or other related activities, or to suggest negligence of ethical obligations. First, the Code was not intended to be used in this fashion. Second, the collected facts and disclosed materials are insufficient to precisely judge what kind of specific

conduct caused the unfortunate result. However, linking the problems in this case with plausibly related ethical obligation clauses as shown in Table 3 can provide a basis for considering the ethical aspects of this incident and other similar cases.

In addition to individual ethical conduct, the Mizuho-TSE case raises issues pertaining to corporate governance. Why did such an erroneous order by a trader go through unnoticed at Mizuho? Did the TSE staff respond appropriately when they were consulted about the order cancellation? How did Fujitsu manage subcontractors? Corporate governance is another domain where software engineering must deal with social and ethical issues.

If we can learn valuable lessons from this unfortunate incident, it would be beneficial. We should also encourage people who have access to information about similar system failures having significant social impact to analyze and report those cases. ■

*Tetsuo Tamai is a professor in the Graduate School of Arts and Sciences at the University of Tokyo. His research interests include requirements engineering and formal and informal approaches to domain modeling. He received a DrS in mathematical engineering from the University of Tokyo. He is a member of the IEEE Computer Society, the ACM, the Information Processing Society of Japan, and the Japan Society for Software Science and Engineering. Contact him at tamai@graco.e.u-tokyo.ac.jp.*

**COMPUTING THEN**

Learn about computing history and the people who shaped it.

<http://computingnow.computer.org/ct>

## S.6 Tokyo Stock Exchange arrowhead Announcements

### S.6.1 Change of trading rules



## 3. Change of trading rules

» Planned changes to several trading rules are planned with arrowhead launch.

|                                                                   | Until Wed., Dec. 30, 2009<br>(current system)                                                                                                                                                                                                                                                                                                                | From Mon., Jan. 4, 2010                                                                                                                                                                                                                                                                   |                                        |
|-------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------|
|                                                                   |                                                                                                                                                                                                                                                                                                                                                              | arrowhead operations                                                                                                                                                                                                                                                                      | Reversal to current system*            |
| Rule for allocation of simultaneous orders (execution by Itayose) | * After totaling for each trading participant,<br>① 5 cycles of 1 unit each,<br>② 1/3 of remaining units placed,<br>③ 1/2 of remaining units placed,<br>④ remaining units placed. (For stop allocation at daily limit price, pro rata ratio applies from step ② onwards.)                                                                                    | * After totaling for each trading participant, allocate 1 unit to each in turn.                                                                                                                                                                                                           | * Revert to current rule.              |
| Half-day trading session                                          | * Full-day trading session, including Wed., Dec. 30, 2009.                                                                                                                                                                                                                                                                                                   | * Full-day trading session                                                                                                                                                                                                                                                                | * Full-day trading session             |
| Tick size                                                         | (see next page)                                                                                                                                                                                                                                                                                                                                              | * Finer-tuned tick size structure in consideration of overall balance and simplicity.<br>* E.g., Tick size of JPY1 for issues whose prices are in the JPY2,000 range. (see next page)                                                                                                     | * Implement new rule.                  |
| Daily price limits, special quote renewal price intervals, etc.   | (see next page)                                                                                                                                                                                                                                                                                                                                              | * Slight expansion in consideration of overall balance and simplicity.<br>(No change in time intervals to renew special quotes)<br>* E.g., Daily limit price range of issues with price more than or equal to JPY700 but less than JPY1,000 will be changed to JPY150 (currently JPY100). | * Implement new rule.                  |
| Sequential trade quote                                            | * No existing rule                                                                                                                                                                                                                                                                                                                                           | * When sequential execution of a single buy/sell order causes the price to exceed the last execution price plus/minus twice the daily price limit, a sequential trade quote will be displayed for 1 minute, and then the order is matched by Itayose method.                              | * Revert to current rule.<br>(No rule) |
| Matching condition during Itayose / stop allocation               | * ① All market orders and limit orders at better prices<br>② All limit orders at the matching price on one side of order book<br>③ At least one trading unit on the other side at the matching price of the order book is executed.<br>During stop allocation, there is no execution if each trading participant is not allocated at least one trading unit. | * Abolish condition ③.<br>E.g., In the case of unfulfilled Itayose condition, an order will be matched at a price near the last execution price. During stop allocation, an execution will occur if there is at least one trading unit on the other side of the order book.               | * Revert to current rule.              |

\* How operations will continue after Jan. 4, 2010 in the current system if the transition to arrowhead is deemed impossible and reversion to the current system is decided.

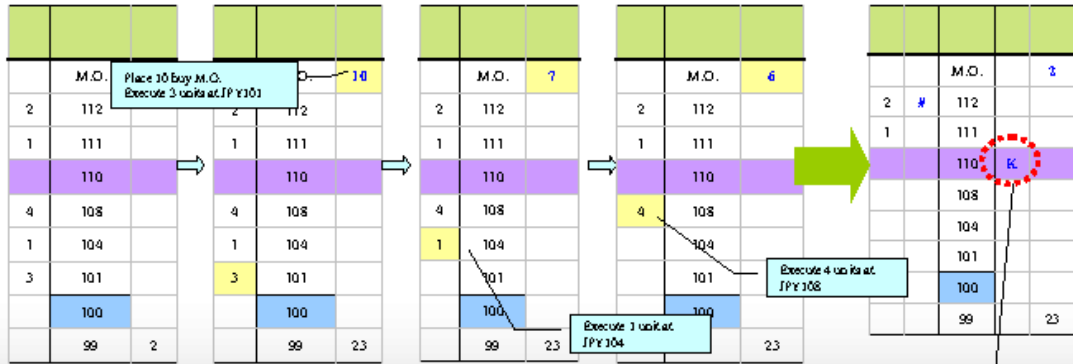


### 3. Change of trading rules

#### » Sequential Trade Quote

New rule to be introduced with arrowhead launch.

If a single order that causes a series of executions arrives in the order book, and such executions cause the price to exceed twice the special quote renewal price interval from the last execution price, a sequential trade quote will be displayed at this price for 1 minute and conduct matching using Itayose.



- (Assumptions on the above chart)
- Zaraba
  - Last execution price: JPY 100
  - Special quote renewal price interval: JPY 5
  - Sequential trade quote: JPY 10\*
  - \* Sequential trade quote is the last execution price plus twice the special quote renewal price interval.

You might expect that 1 unit will be executed at JPY 111. However, because this price exceeds the sequential trade quote (JPY 110), a sequential trade quote (K) is displayed for 1 minute.



### 3. Change of trading rules

» Revision of tick sizes

| Stock price (JPY) |            |   |               | Current (JPY) | Revised (JPY) |
|-------------------|------------|---|---------------|---------------|---------------|
|                   |            | ~ | 2,000 or less | 1             | 1             |
| above             | 2,000      | ~ | 3,000         | 5             | <u>1</u>      |
| "                 | 3,000      | ~ | 5,000         | 10            | <u>5</u>      |
| "                 | 5,000      | ~ | 30,000        | 10            | 10            |
| "                 | 30,000     | ~ | 50,000        | 50            | 50            |
| "                 | 50,000     | ~ | 300,000       | 100           | 100           |
| "                 | 300,000    | ~ | 500,000       | 1,000         | <u>500</u>    |
| "                 | 500,000    | ~ | 3,000,000     | 1,000         | 1,000         |
| "                 | 3,000,000  | ~ | 5,000,000     | 10,000        | <u>5,000</u>  |
| "                 | 5,000,000  | ~ | 20,000,000    | 10,000        | 10,000        |
| "                 | 20,000,000 | ~ | 30,000,000    | 50,000        | <u>10,000</u> |
| "                 | 30,000,000 | ~ | 50,000,000    | 100,000       | <u>50,000</u> |
| "                 | 50,000,000 |   |               | 100,000       | 100,000       |



### 3. Change of trading rules

» Revision of daily price limits and special quote renewal price intervals

| Price (JPY)    | Price limit |         | Renewal price interval |         | Price (JPY)         | Price limit |            | Renewal price interval |           |
|----------------|-------------|---------|------------------------|---------|---------------------|-------------|------------|------------------------|-----------|
|                | Current     | Revised | Current                | Revised |                     | Current     | Revised    | Current                | Revised   |
| 100            | 30          | 30      | 5                      | 5       | 100,000             | 30,000      | 30,000     | 2,000                  | 2,000     |
| 500            | 50          | 50      | 5                      | 5       | 500,000             | 50,000      | 50,000     | 3,000                  | 3,000     |
| 1,000          | 80          | 80      | 5                      | 8       | 1,000,000           | 80,000      | 80,000     | 4,000                  | 5,000     |
| 5,000          | 100         | 100     | 10                     | 10      | 5,000,000           | 50,000      | 70,000     | 5,000                  | 7,000     |
| 10,000         | 100         | 150     | 10                     | 15      | 10,000,000          | 100,000     | 100,000    | 10,000                 | 10,000    |
| 50,000         | 200         | 300     | 20                     | 30      | 50,000,000          | 100,000     | 150,000    | 10,000                 | 15,000    |
| 100,000        | 300         | 400     | 30                     | 40      | 1,000,000,000       | 200,000     | 300,000    | 20,000                 | 30,000    |
| 500,000        | 400         | 500     | 40                     | 50      | 5,000,000,000       | 300,000     | 400,000    | 30,000                 | 40,000    |
| 1,000,000      | 500         | 700     | 50                     | 70      | 10,000,000,000      | 400,000     | 500,000    | 40,000                 | 50,000    |
| 5,000,000      | 1,000       | 1,000   | 100                    | 100     | 50,000,000,000      | 500,000     | 700,000    | 50,000                 | 70,000    |
| 10,000,000     | 1,000       | 1,500   | 100                    | 150     | 100,000,000,000     | 1,000,000   | 1,000,000  | 100,000                | 100,000   |
| 50,000,000     | 2,000       | 3,000   | 200                    | 300     | 500,000,000,000     | 1,000,000   | 1,500,000  | 100,000                | 150,000   |
| 100,000,000    | 2,000       | 3,000   | 200                    | 300     | 1,000,000,000,000   | 2,000,000   | 3,000,000  | 200,000                | 300,000   |
| 500,000,000    | 3,000       | 5,000   | 300                    | 500     | 5,000,000,000,000   | 3,000,000   | 5,000,000  | 300,000                | 500,000   |
| 1,000,000,000  | 4,000       | 7,000   | 400                    | 700     | 10,000,000,000,000  | 4,000,000   | 7,000,000  | 400,000                | 700,000   |
| 5,000,000,000  | 5,000       | 10,000  | 500                    | 1,000   | 50,000,000,000,000  | 5,000,000   | 10,000,000 | 500,000                | 1,000,000 |
| 10,000,000,000 | 10,000      | 15,000  | 1,000                  | 1,500   | 100,000,000,000,000 | 10,000,000  | 10,000,000 | 1,000,000              | 1,000,000 |



## S.6.2 Points to note when placing orders



### 5. Points to note when placing orders

- » With high speed execution processing, there may be sharp fluctuations in stock prices. During the time from order placement by a customer after visual confirmation of the order book to entry of the order into the order book after processing in the trading system, executions of a large volume of orders could already occur, possibly in the scale of hundreds of times especially when there are frequent price movements.
- » Caution is advised, in particular, when placing market orders, as executions may occur outside a price range you initially expected.



## 5. Points to note when placing orders

(Case) Current price in the order book is JPY100.  
An order for 10 units of market buy orders is placed.

| Cumulative Sell | Sell | Price      | Buy                        | Cumulative Buy |
|-----------------|------|------------|----------------------------|----------------|
| 7               |      | M.O.       | 10                         | 10             |
| 7               | 2    | 102        | Place 10 units of buy M.O. |                |
| 5               | 5    | 101        |                            | 10             |
|                 |      | <u>100</u> | 1                          | 11             |
|                 |      | 99         | 4                          | 15             |
|                 |      | 98         | 3                          | 18             |

**What is the difference in execution speed between the current system and arrowhead?**



### 5. Points to note when placing orders

■ How the order book looks like in the current system

| Sell | Price      | Buy | Sell | Price      | Buy |
|------|------------|-----|------|------------|-----|
|      | M.O.       | 10  |      | M.O.       | 5   |
| 2    | 102        |     | 2    | <u>102</u> |     |
| 5    | <u>101</u> |     |      | 101        |     |
|      | 100        | 1   |      | 100        | 1   |
|      | 99         | 4   |      | 99         | 4   |
|      | 98         | 3   |      | 98         | 3   |

*Annotations:*  
 - Orange arrow: "A few seconds after order placement" points to the Sell order at price 101.  
 - Orange arrow: "After another few seconds" points to the Sell order at price 102.  
 - Purple callout: "Execute 5 units at JPY101" points to the Buy order at price 101.  
 - Purple callout: "Execute 2 units at JPY102" points to the Sell order at price 102.

Matching (execution) is performed every few seconds, so placing orders while monitoring the order book was feasible.



## 5. Points to note when placing orders

### ■ How the order book looks like in arrowhead

| Sell | Price      | Buy | Sell       | Price | Buy |
|------|------------|-----|------------|-------|-----|
|      | M.O.       | 10  |            | M.O.  |     |
| 2    | 104        |     | <u>104</u> |       |     |
| 1    | 103        |     | 103        |       |     |
| 2    | 102        |     | 102        |       |     |
| 5    | 101        |     | 101        |       |     |
|      | <u>100</u> | 1   |            |       |     |

Immediate execution

Immediate execution of 10 units of buy M.O. against sell orders between JPY101-104

The price looks as if it has jumped instantaneously to JPY104

Matching (execution) is immediate, and the price in the order book may jump instantaneously from JPY100 to JPY104. You are advised to consider this risk when placing orders while monitoring the order book.

# Appendix T

## XVSM: An Extensible Virtual Shared Memory

### Contents

---

|            |                                                            |            |
|------------|------------------------------------------------------------|------------|
| <b>T.1</b> | <b>Introduction</b>                                        | <b>592</b> |
| T.1.1      | On Targets of Formal Specification                         | 592        |
| T.1.2      | Why Specify Software Concepts Formally                     | 593        |
| T.1.3      | An XVSM Type System                                        | 593        |
| T.1.3.1    | Type Systems                                               | 594        |
| T.1.3.2    | Static and Dynamic Type Systems                            | 594        |
| T.1.3.3    | Why Type Systems                                           | 594        |
| T.1.4      | Words of Caution                                           | 594        |
| <b>T.2</b> | <b>XVSM Trees</b>                                          | <b>595</b> |
| T.2.1      | XTree Rules                                                | 595        |
| T.2.2      | XTree Types                                                | 595        |
| T.2.3      | XTree Type Designator Wellformedness                       | 595        |
| T.2.4      | XTree Type Functions                                       | 596        |
| T.2.5      | XTree Wellformedness                                       | 596        |
| T.2.6      | XTree Subtypes                                             | 597        |
| <b>T.3</b> | <b>XTree Operations</b>                                    | <b>598</b> |
| T.3.1      | XTree Multiset Union                                       | 598        |
| T.3.2      | Commensurate Multiset Arguments                            | 598        |
| T.3.3      | Type “Prediction”                                          | 599        |
| T.3.4      | A Theorem: Correctness of Type “Prediction”                | 599        |
| T.3.5      | XTree Multiset Equality                                    | 599        |
| T.3.6      | XTree Multiset Subset                                      | 599        |
| T.3.7      | Property Multiset Membership                               | 600        |
| T.3.8      | XTree Multiset Membership                                  | 600        |
| T.3.9      | XTree Multiset Cardinality                                 | 600        |
| T.3.10     | Arbitrary Selection of XTrees or Properties from Multisets | 600        |
| T.3.11     | XTree Multiset Difference                                  | 601        |
| T.3.12     | XTree List Concatenation                                   | 601        |
| T.3.13     | XTree List Equality                                        | 601        |
| T.3.14     | XTree List Property Membership                             | 601        |
| T.3.15     | XTree List XTree Membership                                | 602        |
| T.3.16     | XTree List Length                                          | 602        |

|            |                                                         |            |
|------------|---------------------------------------------------------|------------|
| T.3.17     | <b>XTree List Head</b> . . . . .                        | 602        |
| T.3.18     | <b>XTree List Tail</b> . . . . .                        | 602        |
| T.3.19     | <b>XTree List Nth Element</b> . . . . .                 | 602        |
| <b>T.4</b> | <b>Indexing</b> . . . . .                               | <b>603</b> |
| T.4.1      | <b>Paths and Indexes</b> . . . . .                      | 603        |
| T.4.2      | <b>Proper Index</b> . . . . .                           | 603        |
| T.4.3      | <b>Index Selecting</b> . . . . .                        | 603        |
| T.4.4      | <b>Path Indexing</b> . . . . .                          | 604        |
| <b>T.5</b> | <b>Queries</b> . . . . .                                | <b>604</b> |
| T.5.1      | <b>Generally on Semantics</b> . . . . .                 | 604        |
| T.5.2      | <b>Syntax: Simple XVSM Queries</b> . . . . .            | 605        |
| T.5.2.1    | <b>Syntax: Predefined Selector Queries</b> . . . . .    | 605        |
| T.5.2.2    | <b>Semantics: Predefined Selector Queries</b> . . . . . | 605        |
| T.5.2.2.1  | <b>Count</b> . . . . .                                  | 605        |
| T.5.2.2.2  | <b>Sort Up</b> . . . . .                                | 606        |
| <b>T.6</b> | <b>Conclusion</b> . . . . .                             | <b>606</b> |

---

This document presents work in progress. The document constitutes a technical note. It reports on an attempt to formalise XVSM: the Extensible Virtual Shared Memory as reported in the Dipl.Ing. thesis by Stefan Craß: *A Formal Model of the Extensible Virtual Shared Memory (XVSM) and its Implementation in Haskell – Design and Specification*. Technische Universität Wien, 05.02.2010 [84].

## T.1 Introduction

XVSM, the Extensible Virtual Shared Memory concept, has been described in a number of conference proceeding publications: [13, 85, 122, 123]. The MSc Thesis [84] claims to present a formal model, but what is presented is not a proper formal model. To be a proper formal model there must be an abstract presentation in some formal, that is, mathematically well defined specification language and there must be a formal proof system for that language. Usually a formal semantics is also an abstract specification. Haskell, although a commendable programming language, is not suited for the specification of a semantics of XVSM, and [84] presents a Haskell implementation of XVSM and not an abstraction. A reasonably precise, even readable (and also executable), definition of XVSM could have been done in Haskell. Such a definition would carefully build up definitions, in Haskell, of the syntax of XVSM XTrees, of XVSM queries, etc. We shall present a formal definition of XVSM in RSL, the Raise Specification Language [100, 101].

### T.1.1 On Targets of Formal Specification

Formalisation of software concepts started in the 1960s. The most notable example was that of the formal (operational semantics) description of the PL/I programming language [8–10]. The ULD notation emerged (ULD I, ULD II, ULD III -- ULD for Universal Language Description). This name of notation was later renamed into VDL (Vienna Language Description) by J.A.N. Lee [125]. Peter Lucas (sometimes with Kurt Walk) reviewed the [128–134] semantics descriptions of notably ULD III and the background for VDM (the Vienna Development Method).

As a result of the VDL (research and experimental development) work the IBM Vienna Laboratory undertook, in 1973, to develop, for the IBM market a new PL/I compiler for a new IBM computer (code named FSM: Future Systems machine). The US and European IBM laboratories' development of this computer was, eventually, curtailed, in February 1974. Nevertheless, the IBM Vienna Laboratory, was able to complete the work on a formal (denotational semantics-like) description of PL/I [5]. This work led to VDM [71, 73, 98, 118–120] – which later led to RAISE [100, 101] (1990). All the other now available formal specification languages came after VDM: Alloy [115] (2000), B, Event B [2] (1990, 2000) and Z [183] (1980).

First with VDM and now, as here, with RSL, formal specification has been used – other than for the semantics description of programming languages – first for formalising software designs, then for formalising requirements for general software, and for formalising (their) domain descriptions.

In this technical note we apply, not for the first time, formal specification to what the proposers of XVSM refers to as *middleware*: computer software that connects software components or applications. The software consists of a set of services that allows multiple processes running on one or more machines to interact (including sharing data). *Middleware* technology evolved to provide for interoperability in support of the move to coherent distributed architectures, which are most often used to support and simplify complex distributed applications. It includes web servers, application servers, and similar tools that support application development and delivery. *Middleware* is especially integral to modern information technology based on XML, SOAP, Web services, and service-oriented architecture.

### T.1.2 Why Specify Software Concepts Formally

A number of independent reasons can be given for why one might wish to formally specify a software concept<sup>1</sup>. We itemize some of these:

- **As a design aid:** In researching and experimentally developing the design of a software concept, experiments with formal models of the software concept, or just some of its sub-concepts, have shown to help clarify and simplify many design issues<sup>2</sup>.
- **As a communication document:** A suitably narrated and formalised specification, such as the present technical note lays a ground for (but is not yet), can be used as a, or the, ‘semantics’ specification for XVSM. It can serve as a standards document.
- **As a basis for implementation:** A suitably narrated and formalised specification, such as the present technical note, can serve as a basis for (thus provably) correct implementations of proper XVSM middleware.
- **As a basis for teaching & training:** An XVSM communication document can serve as the basis for instruction in the use (i.e., ‘programming’) of XVSM-dependent applications.
- **As a basis for proving properties of XVSM:** The formal specification of XVSM, such as attempted, or at least begun, with the present technical note, can be referred to in formal proofs of properties of XVSM and its applications.

### T.1.3 An XVSM Type System

One of the great contributions of computing science to mathematics has been the studies made of type systems. And one of the great advances of software engineering from the middle 1950s till today has been the use of suitable, usually static, type systems.

The current author has (therefore) been quite surprised when discovering, that a language such as the XVSM query language and the *Core Application Programming Interface Languages*<sup>3</sup> (such as CAPI-1, CAPI-2, and CAPI-3) has not been endowed by a type system. Instead of erroneous

<sup>1</sup>By a software concept we mean such concepts as (the semantics of) programming languages, database models or database management systems, operating systems, specific application systems [such as for air traffic, banking, manufacturing, transportation, or other], their requirements, their underlying domains, etc.

<sup>2</sup>The current author offers the following observation (i) and advice (ii): (i) it seems that formalisation was not used in the conceptualisation of XVSM; and (ii) further extensions of XVSM should preferably be based on the present – or similar, reworked – formalisation and should itself use formal modelling. In reading publications about XVSM an experienced reader of precise descriptions too easily resolves that there are simply far too many ambiguous, inconsistent and incomplete description points: they may not be so, but the current english texts leaves such an experienced reader of precise descriptions to resolve so.

<sup>3</sup>An application programming interface (API) is an interface implemented by a software program which enables it to interact with other software.

query and transaction results (here modelled by **chaos**) an *XVSM* programme could use these type testing facilities to secure provably correct uses of *XVSM*.

We shall, here and there, ‘divert’ from a straight line reformulation of [84], and present components of an *XVSM* Type System (*XVSM/TS*).

### T.1.3.1 Type Systems

Many kinds of type systems can be proposed for *XVSM*. Defining a type system may imply that only correctly typed data, i.e., *XTrees*, and only arguments to operations: queries and actions, that, in some weak or strong sense, satisfy the signature (that is, the type) of the operation are allowed. (We then speak of a weakly, respectively strongly type language.) We shall, through the judicious use of concepts of sub, commensurate- and super-types, suggest one (of several possible) *XVSM* type systems. It is important to emphasize this: that either one of several *XVSM* type systems are possible. The one presented here may not be the best for a number of contemplated applications of *XVSM*, but it is probably a sensible one! Recommendable monographs cum textbooks on type systems and programming languages are [146, 163]. Further foundational studies of type systems are provided in the monographs [1, 102].

### T.1.3.2 Static and Dynamic Type Systems

A programming language is said to use static typing when type checking can be performed during compile-time as opposed to run-time.

A programming language is said to be dynamically typed when the majority of its type checking can only be performed at run-time as opposed to at compile-time. In dynamic typing, values have types but variables do not (necessarily); that is, a variable can refer to a value of any type. Whether one can speak of *XVSM* variables is not known.

We shall anyway think of the type system that we shall put forward for *XVSM* as being a dynamic type system.

### T.1.3.3 Why Type Systems

Reasons for endowing *XVSM* with a type system can be itemized:

- **Safety:** Checking, before execution, that an operation, with the types of its argument and the types of the space-based data, that is, *XTrees*, satisfy the type rules helps avoid otherwise meaningless operations.
- **Optimisation:** Static type-checking may provide useful compile-time information. Dynamic type-checking may provide useful run-time information.
- **Documentation:** In expressive type systems, types can serve as a form of documentation, since they can illustrate the intent of the programmer.
- **Abstraction (Modularity):** Types allow programmers to think about programs at a higher level than the bit or byte, not bothering with low-level implementation.

Any one chosen type system will have been devised so as to satisfy at least one of the above reasons.

## T.1.4 Words of Caution

The type system proposed here for *XVSM* is just an example. I am not quite sure that my particular design choices are the right ones for a system like *XVSM*. A perhaps more proper *XVSM* type system should evolve as the result of close, concentrated discussions and work, in Vienna, not over the Internet, between the leading authors of [13, 84, 85, 122, 123] and Dines Bjørner. But what I am rather sure of is that for *XVSM* to be considered a serious contender for so-called space-based



computing XVSM must be endowed with a type system and with a suitable set of type system (run-time) operations.

## T.2 XVSM Trees

### T.2.1 XTree Rules

20. There are labels and labels are further unspecified quantities.
21. Properties are pairs of labels and XTrees, that is, a property is such a pair.
22. An XTree is either an XTree value or an XTree multiset or an XTree sequence (an XTree list).
  - (a) An XTree value is either some XTree text or is some XTree integer.
  - (b) An XTree multiset consists of a multiset of properties.
  - (c) An XTree sequence consists of a list of properties.

#### type

```

20 L
21 P = L × XT
22 XT = XV | XL | XS
22a XV == mk_ST(sel_txt:Text) | mk_IN(sel_i:Int)
22b XS == mk_XS(sel_xs:(P \overrightarrow{m} Nat))
22c XL == mk_XL(sel_xl:P*)

```

### T.2.2 XTree Types

23. An XTree type is either
  - (a) an integer type, or
  - (b) a text type, or
  - (c) a multiset type which maps its entry labels into corresponding XTree type, or
  - (d) a sequence type which is a sequence of labelled XTree types.

#### type

```

23 XTTy = IntTy | TxtTy | MulTy | SeqTy
23a IntTy == mkITy
23b TxtTy == mkTTy
23c MulTy == mk_MTy(m:(L \overrightarrow{m} XTTy))
23d SeqTy == mk_STy(m:(L × XTTy)*

```

XTTy are type designators.

### T.2.3 XTree Type Designator Wellformedness

24. A type designator, i.e., any XTTy is wellformed if it satisfies the following conditions:
  - (a) Integer and text type designators are wellformed.
  - (b) Multiset type designators are wellformed if the type designators for any label are wellformed.

- (c) Sequence type designators are wellformed if all labelled type designators are wellformed and if the type designators for identically labelled entries are the same type.<sup>4</sup>

**value**

24.  $\text{wf\_XTTy}: \text{XTTy} \rightarrow \mathbf{Bool}$   
 24.  $\text{wf\_XTTy}(t) \equiv$   
 24. **case t of**  
 24a.  $\text{mkITy} \rightarrow \mathbf{true}$ ,  
 24a.  $\text{mkTTy} \rightarrow \mathbf{true}$ ,  
 24b.  $\text{mk\_MTy}(\text{tym}) \rightarrow \forall t': \text{XXTy} \bullet t' \in \mathbf{rng\ ty m} \Rightarrow \text{wf\_XTTy}(t')$   
 24c.  $\text{mk\_STy}(\text{tyl}) \rightarrow$   
 24c.  $\forall (l', t') : (L \times \text{XTTy}) \bullet (l', t') \in \mathbf{elems\ tyl} \Rightarrow \text{wf\_XTTy}(t') \wedge$   
 24c.  $\forall (l'', t'') : (L \times \text{XTTy}) \bullet (l'', t'') \in \mathbf{elems\ tyl} \Rightarrow \text{xtr\_type}(t') = \text{xtr\_type}(t'') \mathbf{end}$

**T.2.4 XTree Type Functions**

25. Given an *XTree* one can “extract” its type:
- (a) The type of an integer value is  $\text{mkITy}$ .
  - (b) The type of a text value is  $\text{mkTTy}$ .
  - (c) The type of an *XTree* multiset,  $\text{ms}$ , is  $\text{mk\_MTy}(\text{tym})$  where  $\text{tym}$  is a mapping from the labels of  $\text{ms}$  to the *XTree* type of the corresponding values.<sup>5</sup>
  - (d) The type of an *XTree* sequence,  $\text{sq}$ , is  $\text{mk\_STy}(\text{tys})$  where  $\text{tys}$  is a sequence of labelled *XTree* types of the indexed (and labelled) *XTree* values of the sequence.

**value**

25.  $\text{xtr\_type}: \text{XT} \xrightarrow{\sim} \text{XTTy}$   
 25.  $\text{xtr\_type}(xt) \equiv$   
 25. **case xt of**  
 25a.  $\text{mk\_IN}(\text{intg}) \rightarrow \text{mkITy}$ ,  
 25b.  $\text{mk\_ST}(\text{text}) \rightarrow \text{mkTTy}$ ,  
 25c.  $\text{mk\_XS}(xs) \rightarrow \text{mk\_MTy}([\text{!} \mapsto \text{xtr\_type}(xt') \mid \text{!} : L \bullet \text{!} \in \mathbf{dom\ xs} \wedge xt' \in \text{xs}(\text{!})])$ ,  
 25d.  $\text{mk\_XL}(xl) \rightarrow \text{mk\_STy}(\langle \langle \text{!}, \text{xtr\_type}(xt') \rangle \mid \text{!} : \mathbf{Nat} \bullet \text{!} \in \mathbf{inds\ xl} \wedge \text{xl}(\text{!}) = (\text{!}, xt') \rangle \rangle)$   
 25. **end**  
 25. **pre:**  $\text{type\_conform}(xt)$

**T.2.5 XTree Wellformedness**

26. An *XTree* is  $\text{type\_conformant}$  if
- (a) it is an integer, or
  - (b) it is a text, or
  - (c) it is a multiset all of whose *XTrees* are  $\text{type\_conformant}$  and all identically labelled *XTrees* have the same type, or
  - (d) it is a sequence all of whose *XTrees* are  $\text{type\_conformant}$  and all of whose identically labelled *XTrees* have the same type.

<sup>4</sup>**Note:** This constraints is in line with the constraint of Item 23c on the preceding page.

<sup>5</sup>**Note:** Thus we constrain two or more properties with the same label to be of the same type – or, as we shall see, subtypes of such a type. This is a consequence of Item 23c on the previous page.

**value**

```

26. type_conform: XT → Bool
26a. type_conform(xt) ≡
26. case xt of
26a. mk_IN(intg) → true,
26b. mk_ST(text) → true,
26c. mk_XS(xs) →
26c. ∀ (l',xt'),(l'',xt''):(L×XT)•{(l',xt'),(l'',xt'')} ⊆ dom xs ∧
26c. type_conform(xt') ∧
26c.a. l'=l'' ⇒ xtr_type(xt') = xtr_type(xt''),
26d. mk_XL(xl) →
26d. ∀ (l',xt'),(l'',xt''):(L×XT)•{(l',xt'),(l'',xt'')} ⊆ elems xl ∧
26d. type_conform(xt') ∧
26d.a. l'=l'' ⇒ xtr_type(xt')=xtr_type(xt'')
26. end

```

**Discussion:** Whether, in formula lines 26c.a and 26d.a, to insist on equality of types or to allow one type to be a subtype of the other (whichever way) is a question to be considered.

**T.2.6 XTree Subtypes**

27. We define a subtype relation as a relation between a pair of type designators:

- (a) The **XTree** integer type is (i.e., designates) a subtype of itself.
- (b) The **XTree** text type is (i.e., designates) a subtype of itself.
- (c) Let two multiset type designators be **mk\_MTy**(tym') and **mk\_MTy**(tym'').  
**mk\_MTy**(tym') is (i.e., designates) a subtype of **mk\_MTy**(tym'')
  - i. if the definition set of labels of **mk\_MTy**(tym') is a subset of the definition set of labels of **mk\_MTy**(tym''),
  - ii. and, if for identical labels,  $\ell$ , in **mk\_MTy**(tym') and **mk\_MTy**(tym''( $\ell$ )) is (i.e., designates) a subtype of **mk\_MTy**(tym''( $\ell$ )).
- (d) Let two sequence type designators be **mk\_STy**(tyl') and **mk\_STy**(tyl'').  
**mk\_STy**(tyl') is (i.e., designates) a subtype of **mk\_STy**(tyl'')
  - i. if the length of tyl' is less than or equal to the length of tyl'',<sup>6</sup>
  - ii. if for index positions,  $i$ , of tyl' the labels of the indexed properties tyl'(i) (= (l',t')) and tyl''(i) (= (l'',t'')) are the same (l'=l'') and
  - iii. type designator  $t'$  is (i.e., designates) a subtype of  $t''$ .
- (e) Only such pairs of types as implied by the above can possibly enjoy a subtype relation.

**value**

```

27. is_subtype: XXTy × XXTy → Bool
27. is_subtype(ta,tb) ≡
27. case (ta,tb) of
27a. (mkITy,mkITy) → true,
27b. (mkTTY,mkTTY) → true,
27c. (mk_MTy(tym'),mk_MTy(tym'')) →
27(c)i. dom tym' ⊆ tym'' ∧
27(c)ii. ∀ l:L•l ∈ dom tym' ⇒ is_subtype(tym'(l),tym''(l)),
27d. (mk_STy(tyl'),mk_STy(tyl'')) →
27(d)i. len tyl' ≤ tyl'' ∧

```

<sup>6</sup>We could, instead of this “prefix” subtype property, have defined an “embedded” subtype property: that tyl' is a subtype of a properly embedded sequence of tyl''

```

27(d)ii. $\forall i:\mathbf{Nat} \cdot 1 \leq i \leq \mathbf{len} \ \mathbf{tyl}' \Rightarrow$
27(d)ii. let $((l',t'),(l'',t''))=(\mathbf{tyl}'(i),\mathbf{tyl}''(i))$ in
27(d)ii. $l'=l'' \wedge$
27(d)iii. $\mathbf{is_subtype}(t',t'')$ end,
27e. $_ \rightarrow \mathbf{false}$
27. end

```

Please note that if  $td'$  and  $td''$  are type designators, then either  $td'$  denotes a subtype of  $td''$  or  $td''$  denotes a subtype of  $td'$  or neither denotes a subtype of the other.

## T.3 XTree Operations

### T.3.1 XTree Multiset Union

28. By the union of two multisets we understand their bag (i.e., multiset) union.

- (a) For any property which is common to both multisets the multiset union maps the property into the sum of its number of occurrences in the two argument multisets.
- (b) For any property which is only in one of the multisets the multiset union contains that property with the number of occurrences designated by that multiset.
- (c) Shared label values must be of comparable\_types.

**value**

```

28 XUnion: XS \times XS \rightarrow XS
28a XUnion(mk_XS(xs1),mk_XS(xs2)) \equiv
28a mk_XS([$p \mapsto \mathbf{xs1}(p) + \mathbf{xs2}(p) \mid p \in \mathbf{dom} \ \mathbf{xs1} \cap \mathbf{dom} \ \mathbf{xs2}$])
28b $\cup \ \mathbf{xs1} \setminus \mathbf{dom} \ \mathbf{xs2} \cup \ \mathbf{xs2} \setminus \mathbf{dom} \ \mathbf{xs1}$)
28c pre: comparable_types(xtr_type(mk_XS(xs1)),xtr_type(mk_XS(xs2)))

```

### T.3.2 Commensurate Multiset Arguments

- 29. Two multiset values (types) are comparable
- 30. if for identical (i.e., shared) labels have identical types (are equal);
- 31. or maybe we should just ask for an appropriate subtype relation.

**value**

```

29. comparable_values: XS \times XS \rightarrow Bool
29. comparable_values(mk_XS(lmt'),mk_XS(lmt'')) \equiv
30. $\forall l:L \cdot l \in \mathbf{dom} \ \mathbf{lmt}' \cap \mathbf{lmt}'' \Rightarrow$
30. $(\mathbf{xtr_type}(\mathbf{lmt}'(l)) = \mathbf{xtr_type}(\mathbf{lmt}''(l))) \vee$
31. $\mathbf{is_subtype}(\mathbf{xtr_type}(\mathbf{lmt}'(l)),\mathbf{xtr_type}(\mathbf{lmt}''(l))) \vee$
31. $\mathbf{is_subtype}(\mathbf{xtr_type}(\mathbf{lmt}''(l)),\mathbf{xtr_type}(\mathbf{lmt}'(l)))$

```

**value**

```

29. comparable_types: XTTy \times XTTy \rightarrow Bool
29. comparable_types(mk_XT(lmt'),mk_XT(lmt'')) \equiv
30. $\forall l:L \cdot l \in \mathbf{dom} \ \mathbf{lmt}' \cap \mathbf{lmt}'' \Rightarrow$
30. $(\mathbf{lmt}'(l) = \mathbf{lmt}''(l)) \vee$
31. $\mathbf{is_subtype}(\mathbf{lmt}'(l),\mathbf{lmt}''(l)) \vee \mathbf{is_subtype}(\mathbf{lmt}''(l),\mathbf{lmt}'(l))$

```

**T.3.3 Type “Prediction”**

32. One can calculate the type of the result of a multiset union from its two arguments:

- (a)
- (b)
- (c)
- (d)

32.  
32a.  
32b.  
32c.  
32d.

**T.3.4 A Theorem: Correctness of Type “Prediction”**

33. One can prove the following theorem:

- (a)
- (b)
- (c)
- (d)
- (e)

33.  
33a.  
33b.  
33c.  
33d.  
33e.

**T.3.5 XTree Multiset Equality**

34. Multiset equality is bag equality of the multisets.

**value**

34 XSequal:  $XS \times XS \rightarrow \mathbf{Bool}$

34 XSequal( $\mathbf{mk\_XS}(xs1), \mathbf{mk\_XS}(xs2)$ )  $\equiv xs1 = xs2$

**T.3.6 XTree Multiset Subset**

35. One multiset is a subset of another multiset

- (a) if the first has a subset of the properties of the latter and
- (b) and, for each property of the first its number of occurrences in the former is equal to or smaller than its number of occurrences in the latter.

**value**

35 XSsubset:  $XS \times XS \rightarrow \mathbf{Bool}$

35 XSsubset( $\mathbf{mk\_XS}(xs1), \mathbf{mk\_XS}(xs2)$ )  $\equiv$

35a  $\mathbf{dom} \ xs1 \subseteq \mathbf{dom} \ xs2 \wedge$

35b  $\forall p:P \cdot p \in \mathbf{dom} \ xs1 \Rightarrow xs1(p) \leq xs2(p)$

### T.3.7 Property Multiset Membership

36. A property,  $p=(l,xt)$ , is in a multiset if it occurs in the multiset with a cardinality higher than 0.

#### value

36 XSmember:  $P \times XS \rightarrow \mathbf{Bool}$

36 XSmember( $p, \mathbf{mk\_XS}(xs)$ )  $\equiv p \in \mathbf{dom} \text{ } xs \wedge xs(p) > 0$

### T.3.8 XTree Multiset Membership

37. An XTree,  $xt$ , is a member of a multiset,  $xs$ , if there exists a label,  $\ell$  such that the property  $(\ell, xt)$  is a member of  $xs$ .

#### value

37 XSmember:  $XT \times XS \rightarrow \mathbf{Bool}$

37 XSmember( $xt, \mathbf{mk\_XS}(xs)$ )  $\equiv \exists l:L \bullet \text{XSmember}((l,xt), \mathbf{mk\_XS}(xs))$

### T.3.9 XTree Multiset Cardinality

38. The cardinality of a multiset is the sum total of all the XTrees of distinct properties of that multiset.

#### value

38 XScard( $\mathbf{mk\_XS}(xs)$ )  $\equiv$

38 **if**  $xs = []$  **then** 0

38 **else**

38 **let**  $(l,xt):P \bullet (l,xt) \in \mathbf{dom} \text{ } xs$  **in**

38  $xs(l,xt) + \text{XScard}(\mathbf{mk\_XS}(xs \setminus \{(l,xt)\}))$  **end end**

### T.3.10 Arbitrary Selection of XTrees or Properties from Multisets

39. To select an XTree of a multiset

- (a) is undefined if the multiset is empty.
- (b) If it is not empty then an arbitrary property is chosen from the (definition set of the) multiset and the XTree of that property is yielded.
- (c) To select a property of a multiset basically follows the above description.

#### value

39 XSselectXT:  $XS \rightsquigarrow XT$

39 XSselectXT( $\mathbf{mk\_XS}(xs)$ )  $\equiv$

39a **if**  $xs = []$

39a **then chaos**

39b **else let**  $(l,xt):P \bullet (l,xt) \in \mathbf{dom} \text{ } xs$  **in**  $xt$  **end**

39b **end**

39 XSselectP:  $XS \rightsquigarrow P$

39 XSselectP( $\mathbf{mk\_XS}(xs)$ )  $\equiv$

39a **if**  $xs = []$

39a **then chaos**

39c **else let**  $p:P \bullet p \in \mathbf{dom} \text{ } xs$  **in**  $p$  **end**

39c **end**

**T.3.11 XTree Multiset Difference**

40. The multiset difference of two multisets,  $xs1$  and  $xs2$ ,
- (a) is the multiset where properties that are in both  $xs1$  and  $xs2$  occur in the result with their number of occurrences being their difference, if larger than 0,
  - (b) to which is joined the multiset of  $xs1$  whose properties are not in  $xs2$ .

**value**

40 XTreeDiff:  $XS \times XS \rightarrow XS$

40 XTreeDiff(**mk\_XS**( $xs1$ ),**mk\_XS**( $xs2$ ))  $\equiv$

40a  $\text{mkXS}(\text{rm0}([\text{p} \mapsto \text{xs1}(\text{p}) - \text{xs2}(\text{p}) \mid \text{p}: \text{P} \bullet \text{p} \in \text{dom } xs1 \cap \text{dom } xs2])$

40b  $\cup xs1 \setminus \text{dom } xs2)$

$\text{rm0}: (\text{P} \overrightarrow{\text{m}} \text{Int}) \rightarrow (\text{P} \overrightarrow{\text{m}} \text{Nat})$

$\text{rm0}(\text{pmn}) \equiv [\text{p} \mapsto \text{pmn}(\text{p}) \mid \text{p}: \text{P} \bullet \text{p} \in \text{dom } \text{pmn} \wedge \text{pmn}(\text{p}) > 0]$

**T.3.12 XTree List Concatenation**

41. The concatenation of two XTree lists is the usual concatenation of lists.
42. Labels,  $\ell$ , common to the two XTree lists must designate XTree,  $xt1$  and  $xt2$  (i.e., properties  $(\ell, xt1)$  and  $(\ell, xt2)$ ) where one is a subtype of the other (i.e., including “vice versa”).

**value**

41 XTreeListConc:  $XL \times XL \rightarrow XL$

41 XTreeListConc(**mk\_XL**( $x1$ ),**mk\_XL**( $x2$ ))  $\equiv$  **mk\_XL**( $x1 \hat{\ } x2$ )

42 **pre**  $\forall (l1, xt1), (l2, xt2): \text{P} \bullet (l1, xt1) \in \text{elems } x1 \wedge (l2, xt2) \in \text{elems } x2 \wedge l1 = l2 \Rightarrow$

42  $\text{subtype}(xt1, xt2) \vee \text{subtype}(xt2, xt1)$

**T.3.13 XTree List Equality**

43. The equality of two XTree lists is the usual equality of lists.

**value**

43 XTreeListEqual:  $XL \times XL \rightarrow \text{Bool}$

43 XTreeListEqual(**mk\_XL**( $x1$ ),**mk\_XL**( $x2$ ))  $\equiv x1 = x2$

**T.3.14 XTree List Property Membership**

44. A property is a member of an XTree list
45. if there is an index into the list which identifies that property.

**value**

44 XMbrTreeList:  $\text{P} \times XL \rightarrow \text{Bool}$

45 XMbrTreeList( $\text{p}, \text{mk\_XL}(x)$ )  $\equiv \exists i: \text{Nat} \bullet i \in \text{inds } x \wedge \text{p} = x(i)$

**T.3.15 XTree List XTree Membership**

46. An `XTree` is a member of an `XTree` list
47. if there is an index into the list which identifies that `XTree`.

**value**

- 46 `XMbrTreeList`:  $\text{XT} \times \text{XL} \rightarrow \text{Bool}$
- 47 `XMbrTreeList`(`xt`,`mk_XL`(`xl`))  $\equiv \exists i:\text{Nat},l:\text{Label} \bullet i \in \text{inds } xl \wedge (l,\text{xt})=\text{xl}(i)$

**T.3.16 XTree List Length**

48. The length of an `XTree` list
- (a) is the length of the list it contains.

**value**

- 48 `XTreeListLength`:  $\text{XL} \rightarrow \text{Nat}$
- 48a `XTreeListLength`(`mk_XL`(`xl`))  $\equiv \text{len } xl$

**T.3.17 XTree List Head**

49. The head, or first, element of an `XTree` list
- (a) is the head property of the list it contains.

**value**

- 49 `XTreeListHead`:  $\text{XL} \rightarrow \text{P}$
- 49a `XTreeListHead`(`mk_XL`(`xl`))  $\equiv \text{if } xl=\langle \rangle \text{ then chaos else hd } xl \text{ end}$

**T.3.18 XTree List Tail**

50. The tail, or rest, of an `XTree` list
- (a) is the tail of the list it contains.

**value**

- 50 `XTreeListTail`:  $\text{XL} \rightarrow \text{XL}$
- 50a `XTreeListTail`(`mk_XL`(`xl`))  $\equiv \text{if } xl=\langle \rangle \text{ then chaos else mk\_XL}(\text{tl } xl) \text{ end}$

**T.3.19 XTree List Nth Element**

51. The  $n$ th element of a list
- (a) if  $n$  is an index of the list then it is the property indexed by  $n$  else it is undefined.

**value**

- 51 `NthXTreeListElem`:  $\text{Nat} \times \text{XL} \xrightarrow{\sim} \text{P}$
- 51a `NthXTreeListElem`( $n$ ,`mk_XL`(`xl`))  $\equiv \text{if } 0 < n \leq \text{len } xl \text{ then } xl(n) \text{ else chaos end}$



## T.4 Indexing

### T.4.1 Paths and Indexes

???

52. An index is either a label or a wildcard or a  
 53. non-zero natural number.  
 54. A path is a finite sequence of zero, one or more indexes.

**type**

52 Index == **mk\_L**(l:L) | **mk\_WldCrd** | **mk\_Nat**(i:Nat1)

53 Nat1 = { |n:Nat•n>0| }

54 Path = Index\*

### T.4.2 Proper Index

55. We define an **is\_Index** predicate over indexes and **Xtrees**.

- (a) If there is a property,  $(\ell, xt)$ , which is in a multiset **mk\_XS**(xs) then  $\ell$  is an index of that **mk\_XS**(xs).  
 (b) If there is an index,  $j$ , into the list,  $xl$ , of an **XTree** list, **mk\_XL**(xl), then  $j$  is an index of that **mk\_XL**(xl);  
 (c) if, furthermore, there is the property,  $(\ell, xt)$  at list  $xl$  position  $j$ , then  $\ell$  is an index into **mk\_XL**(xl); and  
 (d) **mk\_WldCrd** is (always) an index.

**value**

**is\_Index**: Index  $\times$  XT  $\rightarrow$  **Bool**

**is\_Index**(i,xt)  $\equiv$

**case** (i,xt) **of**

55a (**mk\_L**(l),**mk\_XS**(xs))  $\rightarrow \equiv \exists xt':XT \bullet (l,xt') \in \mathbf{dom} \text{ xs},$

55b (**mk\_Nat**(j),**mk\_XL**(xl))  $\rightarrow j \in \mathbf{inds} \text{ xl},$

55c (**mk\_L**(l),**mk\_XL**(xl))  $\rightarrow \exists j:Nat1, xt':XT \bullet j \in \mathbf{inds} \text{ xl} \wedge xl(j) = (l,xt'),$

55d (**mk\_WldCrd**,\_)  $\rightarrow \mathbf{true},$

\_  $\rightarrow \mathbf{false}$

**end**

### T.4.3 Index Selecting

56. Given an index it thus may or may not identify an **XTree**,  $xt'$ , or a property,  $p:P$ , of an argument **XTree**,  $xt$ . The definition follows those of Items 55a–55c.

**value**

56 **Identify**: Index  $\times$  XT  $\rightsquigarrow$  (XT|P)

56 **Identify**(i,xt)  $\equiv$

56 **case** (i,xt) **of**

55a (**mk\_L**(l),**mk\_XS**(xs))  $\rightarrow \mathbf{let} \text{ } xt':XT \bullet (l,xt') \in \mathbf{dom} \text{ xs} \mathbf{ in} \text{ } xt' \mathbf{ end},$

55b (**mk\_Nat**1(i),**mk\_XL**(xl))  $\rightarrow xl(i),$

55c (**mk\_L**(l),**mk\_XL**(xl))  $\rightarrow \mathbf{let} \text{ } i:Nat1, xt':XT \bullet i \in \mathbf{inds} \text{ xl} \wedge xl(i) = (l,xt') \mathbf{ in} \text{ } xt' \mathbf{ end},$

55d (**mk\_WldCrd**,**mk\_XS**(xs))  $\rightarrow \mathbf{let} \text{ } p:P \bullet p \in \mathbf{dom} \text{ xs} \mathbf{ in} \text{ } p \mathbf{ end},$

55d (**mk\_WldCrd**,**mk\_XL**(xl))  $\rightarrow \mathbf{hd} \text{ xl}$

56 **end**

56 **pre** **is\_Index**(i,xt)

### T.4.4 Path Indexing

57. Given an XTree,  $xt$ , a path,  $pth$ , may or may not identify an XTree,  $xtr'$ , of  $xt$ . The selection function, `Select` is defined recursively:

- (a) If the path is empty then the argument XTree,  $xt$ , is yielded.
- (b) If the head of the path is an index of the XTree,  $xt$ , then the so indexed XTree,  $xt_x$ , is selected.
- (c) Otherwise the path,  $pth$ , is ill-defined.

#### value

```

57 Select: XT × Path $\tilde{\rightarrow}$ XT | P
57a Select(xtop,⟨⟩) \equiv xtop
57b Select(xt,⟨i⟩ \wedge pth) \equiv
57b if is_Index(i,xt)
57b then
57b let e = Identify(i,xt) in
57b if e:P \wedge pth \neq ⟨⟩ then chaos end
57b Select(e,pth) end
57c else chaos end

```

## T.5 Queries

58. An XVSM query is a [piped] sequence of simple XVSM queries.

#### type

```
58 Q = SQ*
```

### T.5.1 Generally on Semantics

59. The idea is the following:

- (a) The meaning of a simple XVSM query,  $sq:SQ$ , as applied to an XTree,  $xt:XT$ , is expressed as  $MSQ(sq)(xt)$ , and is to be an XTree multiset or an XTree list. **Not an XTree value ?**
- (b) The meaning of an XVSM query,  $q:Q$ , as applied to an XTree,  $xt:XT$ , is expressed as  $MQ(q)(xt)$ , and is to be an XTree multiset or an XTree list.
- (c) The meaning function,  $MQ$ , when applied to an empty query,  $\langle \rangle$ , is  $MQ(\langle \rangle)(xt)$ , that is,  $xt$ .
- (d) The meaning function,  $MQ$ , when applied to a non-empty query,  $\langle sq \rangle \wedge q$ , is  $MQ(q)(MSQ(sq)(xt))$ .
- (e) Both  $MSQ$  and  $MQ$  may be undefined for some combinations of queries and Xtrees.

#### value

```

59a MSQ: SQ \rightarrow XT $\tilde{\rightarrow}$ XT
59b MQ: Q \rightarrow XT $\tilde{\rightarrow}$ XT
59b MSQ(sq) as xt
59c MQ(⟨⟩)(xt) \equiv xt
59d MQ(⟨sq⟩ \wedge q)(xt) \equiv MQ(q)(MSQ(sq)(xt))

59e MQ(⟨sq⟩ \wedge q)(xt) \equiv
59e if IS_UNDEFINED(MSQ(sq)(xt))

```

```

59e then IS_UNDEFINED(MQ((sq)^q)(xt))
59e else ... to be defined ...
59e end

```

### T.5.2 Syntax: Simple XVSM Queries

60. A simple XVSM query is either a selector query or a matchmaker query.

61. A [simple] selector [XVSM] query is either a predefined selector query or ...

#### type

60 SQ = SelQ | MatchQ

61 SelQ = PreSelQ | ...

#### T.5.2.1 Syntax: Predefined Selector Queries

62. A predefined selector query is either a count, a sort\_up, a sort\_down, a reverse, an identity, or a unique (selector) query.

- (a) A count query states a non-zero natural number.
- (b) A sort up query states a path.
- (c) A sort down query states a path.
- (d) A reverse query does not present an argument.
- (e) An identity query does not present an argument.
- (f) A unique (selector) query states a path.

62 PreSelQ = Cnt | SrtUp | SrtDo | Rev | Id | Uniq | ...

62a Cnt == mk\_Cnt(sel\_n:Nat)

62b SrtUp == mk\_SrtUp(sel\_p:Path)

62c SrtDo == mk\_SrtDo(sel\_p:Path)

62d Rev == mk\_Rev

62e Id == mk\_Id

62f Uniq == mk\_Uniq(sel\_p:Path)

#### T.5.2.2 Semantics: Predefined Selector Queries

##### T.5.2.2.1 Count

63. The **mk\_Cnt**(n) selector query applies to an **XTree**, **xt**, and,

- (a) if it is an **XTree** list and if the list is of length **n** or more, yields the **XTree** list **mk\_XL**(xl') of the first **n** properties of **xt** = **mk\_XL**(xl), else it yields **chaos**; or
- (b) if it is an **XTree** multiset and if the multiset has at least **n** properties, yields an **XTree** multiset, **mk\_XS**(xs'), of **n** arbitrarily chosen properties of **xt** = **mk\_XS**(xs), else it yields **chaos**.

63 MPreSelQ: PreSelQ → XT  $\overset{\sim}{\rightarrow}$  XT

63 MPreSelQ(**mk\_Cnt**(n))(xt) ≡

63 case xt of

63a **mk\_XL**(xl) →

63a if len xl ≥ n then **mk\_XL**((xl(i)|i:Nat•i:[1..n])) else **chaos** end,

63b **mk\_XS**(xs) →

```

63b if card dom xs ≥ n
63b then let ps:P-set•card ps=n ∧ ps⊆dom xs in
63b mk_XS([p→xs(p)|p:P•p ∈ ps]) end
63b else chaos
63b end,
63b _ → chaos
63b end

```

### T.5.2.2.2 Sort Up

64. The **mk\_SrtUp** selector query applies to a (relative) path,  $\text{pth}^\ell$ , and an **XTree**,  $\text{xt}$ .

- (a) First we **Select** from  $\text{xt}$  the **XTree**,  $\text{xt}''$ , identified by the path  $\text{pth}$ .
- (b) The selected **XTree**,  $\text{xt}''$ , is either a list or a multiset.
- (c) The result of  $\text{MPreSelQ}(\text{mk\_SrtUp}(\text{pth}^\ell))(\text{xt})$  is the **XTree** list  $\text{xt}'$  which has all the entries that  $\text{xt}$  has except that these are now ordered with respect to the ordering of the  $\ell$  values of  $\text{xt}''$ .

**value**

```

64 MPreSelQ: SrtUp → XT → XL
64 MPreSelQ(mk_SrtUp(pth^ℓ))(xt) ≡
64a let xt'' = Select(xt)(pth) in
64b let vl =

64c end end

```

MUCH MORE TO COME

## T.6 Conclusion

TO BE WRITTEN