

Theory & Practice of Domain Science & Engineering

Dines Bjørner

Technical University of Denmark
Fredsvej 11, DK-2840 Holte
bjorner@gmail.com – www.dtu.dk/~db

June 23, 2025: 09:02 am

A First “Final” Draft

Theory & Practice of Domain Science & Engineering

Dines Bjørner

June 23, 2025: 09:02 am

- **Warning:** Many formulas need being type checked, etc., etc. !

- I began writing this document in February, 2025.
- In my 87th year !
- I think about it and write “on” it, every day 7/7.
- Often twice a a day, 1-2 hours.
- More than that – and I get tired.

- **A first draft June 10, 2025.**
 - Now, as from June 10, 2025, I will
 - * Go through the entire document.
 - * Check that all index references to formulas are “correct” .
 - * Etcetera, et cetera !
 - I will release this and forthcoming versions to the Internet:

<https://www.imm.dtu.dk/~dibj/2025/transport/main.pdf>

- Section A.3 (pages 176–184) presents an index to all formulas !
- You may find, in the version of this report, that You are now perusing, that there are some “mysterious” vertical [i.e., line] spacing.
 They are there in order for the index entries to refer to pages (π) where both the (item ι) enumerated narrative and formal entries are on the same page !

- Pls. see Sect. 27.4 on page 164.
- Pls. refer to Appendix Chapter B on page 185 for a summary of main formal entities.

Prelude

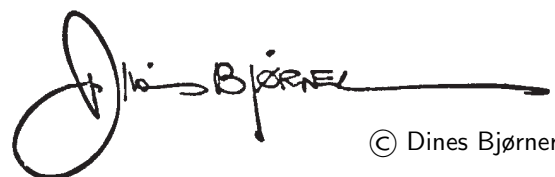
This is [a draft of a] science and engineering [tutorial] book.

- **Science:**

We summarize, in Chapter 1, the method – its principles, procedures, techniques and tools – for rigorously analyzing and describing [modeling] domains.

- **Engineering:**

In Chapters 2–24 we analyze and describe a conceptual domain of ‘transport’ in all its forms: passenger and goods, road, rail, water (navigable rivers and lakes as well as the open sea), and air. From the basis of an abstract notion of *graphs* with *labeled nodes* and *edges*, we define a notion of *routes of graphs*: sequences of node and edge labels. Nodes are then interpreted as street intersections, bus stops, railway stations, harbours and airports and edges as links between neighbouring nodes: street segments, bus routes, rail lines, sea lanes, and air routes. And from there it goes! We expand the treatment to cover customers, [sending and receiving] merchandises, conveyor companies and logistics companies.



© Dines Bjørner

June 23, 2025: 09:02 am

Contents

I	THE THEORY	1
1	The Theory	3
1.1	Domains	5
1.2	Six Languages	6
1.3	Endurants and Perdurants, I	8
1.4	A Domain Analysis & Description Ontology	8
1.5	The Name, Type and Value Concepts	10
1.6	Phenomena and Entities	10
1.7	Endurants and Perdurants, II	11
1.8	External and Internal Endurant Qualities	12
1.9	Perdurant Concepts	22
1.10	Facets	31
1.11	Conclusion	33
II	A PRACTICE	37
2	Introduction	39
2.1	On A Notion of 'Infrastructure'	39
2.2	Domain Models	39
2.3	A Dichotomy	40
2.4	A [Planned] Series of Infrastructure Domain Models	40
III	A SIMPLE BEGINNING	43
3	Kinds of Transports	45
3.1	Informal Outline	45
3.2	Narrative & Formalization	45
4	Overall "Single-Mode" Transport Endurants	47
4.1	Endurant Sorts & Observers	47
4.2	Unique Identification	48
5	Graphs: Transport Nets	49
5.1	The Endurant Sorts and Observers	49
5.2	Unique Identifiers	51
5.3	Mereology	52
5.4	Paths of a Graph	53
5.5	Attributes	56
6	Conveyors, I	59
6.1	Conveyor Endurant Sorts & Observers	59
6.2	Unique Identifiers	60
6.3	Mereology	60
6.4	Attributes	61
7	Intentional Pull, I	63
7.1	History Attributes	63
7.2	An Intentional Pull	64

8	Single-mode Transport Behaviours	65
8.1	Communication	65
8.2	Behaviours	66
8.3	Behaviour Signatures	66
8.4	Behaviour Definitions	66
8.5	Domain Instantiation	70
IV	A MULTI-MODE TRANSPORT: ENDURANTS	71
9	Multi-mode Transport	73
10	“Top” Transport Endurants	75
10.1	The Endurants – External Qualities	75
10.2	On Internal Qualities.	80
10.3	Conveyor Companies versus Logistics Companies.	80
10.4	Financial Matters	80
11	Merchandise	81
11.1	Merchandise Endurants	81
11.2	Representation of Merchandises	83
11.3	Humans	83
12	Customer	85
12.1	Customer Endurants	85
12.2	Customer Qualities	86
12.3	Customer Retrieval	87
12.4	Customer Commands	87
13	Conveyor Companies	89
13.1	Conveyor Authorities.	89
13.2	Conveyor Company Endurants.	89
13.3	Conveyor Company Internal Qualities	91
13.4	Conveyor Company Commands.	95
14	Conveyors, II	97
14.1	Conveyor Mereology	97
14.2	Conveyor Attributes	98
14.3	Conveyor Commands.	99
15	Logistics Companies	101
V	A MULTI-MODE TRANSPORT: INTENTIONAL PULL	103
16	Intentional Pull, II	105
VI	A MULTI-MODE TRANSPORT: COMMANDS	107
17	Multi-mode Transport Commands	109
17.1	Events and Commands	109
17.2	Command Traces	109
17.3	An Analysis	110
17.4	Material and “Immaterial” Commands	111
17.5	Abstracting an Essence of Transport	111
17.6	Commands – A First View	111
17.7	TR: Transport Routes	112
17.8	A Closer Analysis of Commands	115

VII IDENTITIES	121
18 Identities	123
VIII A MULTI-MODE TRANSPORT: BEHAVIOURS	125
19 Multi-mode Behaviours	127
19.1 Communication	127
19.2 Behaviour Signatures	128
19.3 Which Behaviours to Describe?	129
19.4 Multi-mode “Systems”	129
20 Customer Behaviours	131
20.1 Main Behaviour	131
20.2 Subsidiary Behaviours	132
21 Conveyor Company Behaviours	135
21.1 Main Behaviour	135
21.2 Main Reactive Behaviour	136
21.3 Subsidiary Behaviours	137
22 Conveyor Behaviour	141
22.1 Earlier Treatment	141
22.2 Main Behaviour	143
22.3 Subsidiary Behaviours	144
23 Logistics Company Behaviour	149
24 Edge Behaviour	151
24.1 Earlier Treatment	151
24.2 Main Behaviour	151
25 Node Behaviour	153
25.1 Earlier Treatment	153
25.2 Revised Node Attributes	153
25.3 [k10,k11,k14] Main Behaviour	154
IX CLOSING	155
26 Discussion	157
26.1 Wither Logistics Companies	157
26.2 Some Parts Modelled, Others Not!?	158
26.3 Formal Structuring	159
26.4 Mnemonics	159
26.5 Narratives	159
27 Conclusion	161
27.1 Logistics & Operations Research	161
27.2 Interpretations	161
27.3 Formality and Verification	163
27.4 On the Development of This Model	164
27.5 Acknowledgements	164
28 Bibliography	165
X APPENDIX	171
A Indexes	173
A.1 Domain Modeling Ontology	173
A.2 Transport Domain Concepts	174

A.3 Formal Entities	175
B Summaries	185
B.1 Commands	185
B.2 Mereologies and Attributes	185

Part I

THE THEORY

Chapter 1

The Theory

Contents

1.1 Domains	5
1.1.1 What are They ?	5
1.1.2 Some Introductory Remarks	6
1.1.2.1 A Discussion of Our Characterization of a Concept of Domain	6
1.1.2.2 Formal Methods and Description Language	6
1.1.2.3 Programming Languages versus Domain Semantics	6
1.1.2.4 A New Universe	6
1.2 Six Languages	6
1.2.1 The 6 Languages	7
1.2.2 Semiotics	7
1.2.3 Speech Acts	8
1.3 Endurants and Perdurants, I	8
1.4 A Domain Analysis & Description Ontology	8
1.4.1 The Chosen Ontology	8
1.4.2 Discussion of The Chosen Ontology	9
1.5 The Name, Type and Value Concepts	10
1.5.1 Names	10
1.5.2 Types	10
1.5.3 Values	10
1.6 Phenomena and Entities	10
1.7 Endurants and Perdurants, II	11
1.7.1 Endurants	11
1.7.2 Perdurants	11
1.7.3 Ontological Choice	11
1.8 External and Internal Endurant Qualities	12
1.8.1 External Qualities – Tangibles	12
1.8.1.1 The Universe of Discourse	12
1.8.1.2 Solid and Fluid Endurants	12
1.8.1.2.1 Solid cum Discrete Endurants.	13
1.8.1.2.2 Fluids.	13
1.8.1.3 Parts and Living Species Endurants	13
1.8.1.3.1 Parts	13
1.8.1.4 States	16
1.8.1.5 Validity of Endurant Observations	16
1.8.1.6 Summary of Endurant Analysis Predicates	16
1.8.1.7 “Trees are Not Recursive”	17
1.8.2 Internal Qualities – Intangibles	17
1.8.2.1 Unique Identity	17
1.8.2.1.1 Uniqueness of Parts	18
1.8.2.2 Mereology	18
1.8.2.3 Attributes	19

1.8.2.3.1	General	19
1.8.2.3.2	Michael A. Jackson's Attribute Categories	20
1.8.2.3.3	Analytic Attribute Extraction Functions:	21
1.8.3	Intentional Pull	21
1.8.4	Summary of Endurants	22
1.9	Perdurant Concepts	22
1.9.1	"Morphing" Parts into Behaviours	22
1.9.2	Transcendental Deduction	22
1.9.3	Actors – A Synopsis	23
1.9.3.1	Action	23
1.9.3.2	Event	23
1.9.3.3	Behaviour	24
1.9.4	Channel	24
1.9.5	Behaviours	24
1.9.5.1	Behaviour Signature	24
1.9.5.2	Inert Arguments: Some Examples	25
1.9.5.3	Behaviour Definitions	25
1.9.5.4	Action Definitions	27
1.9.5.5	Behaviour Invocation	28
1.9.5.6	Argument References	29
1.9.5.6.1	Evaluation of Monitorable Attributes.	29
1.9.5.6.2	Update of Biddable Attributes	29
1.9.5.7	Behaviour Description – Examples	30
1.9.6	Behaviour Initialization.	31
1.10	Facets	31
1.10.1	Intrinsics	32
1.10.2	Support Technology	32
1.10.3	Rules & Regulations	32
1.10.4	Scripts	32
1.10.5	License Languages	32
1.10.6	Management & Organization	32
1.10.7	Human Behaviour	32
1.11	Conclusion	33
1.11.1	Previous Literature	33
1.11.2	The Method	33
1.11.3	Specification Units	33
1.11.4	Object Orientation	33
1.11.5	Other Domain Modeling Approaches	34
1.11.6	How Much ? How Little ?	34
1.11.7	Correctness	34
1.11.8	Domain Facets	34
1.11.9	Perspectives	34
1.11.10	The Semantics of Domain Models	34
1.11.11	Further on Domain Modeling	35
1.11.12	Software Development	35
1.11.13	Modeling	35
1.11.14	Philosophy of Computing	35
1.11.15	A Manifesto	35

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*.

$\mathbb{D}, \mathbb{S} \models \mathbb{R}$:

In proofs of *Software* correctness,

with respect to *Requirements*,

assumptions are made with respect to the *Domain*.

We present a systematic *method*, its *principles*, *procedures*, *techniques* and *tools*, for efficiently *analyzing & describing* domains. This paper is based on [16, 19, 21]. It simplifies the methodology of these considerably – as well as introduces some novel presentation and description language concepts.

• • •

Alert: Before You start reading this paper, You are kindly informed of the following:

High Light 0.1 *What The Paper is All About:* The *Triptych Dogma*, above, says it all: this paper is about a new area of computing science – that of *domains*. It is about what domains are. How to model them. And their role in software development. There are many “domain things” it is not about: it is not about ‘derived’ properties of domains – beyond, for example, *intentional pull* [Sect. 1.8.3 on page 21]. Such are left for studies of domains based on the kind of formal domain descriptions such as those advocated by this paper •

High Light 0.2 *A Radically New Approach to Software Development:* The *Triptych Approach to Software Development*, calls for *software* to be developed on the basis of *requirements prescriptions*, themselves developed on the basis of *domain descriptions*. We furthermore advocate these specifications and their development be formal. That is: there are formal methods for the development of either of these three kinds of specifications:

- Development of domain descriptions is outlined in this paper.
- Development of requirements, from domain descriptions, is outlined in [21, *Chapter 9*].
- Development of software, from requirements prescriptions, is treated, extensively, in [11].

The reader should understand that the current paper, with its insistence of strictly following a method, formally, is at odds with current ‘software engineering’ practices. •

High Light 0.3 *Characterizations rather than Definitions:* The object of domain study, analysis and description, i.e., the domains, are, necessarily, informal. A resulting domain description is formal. So the domain items being studied and analyzed cannot be given a formal definition. Conventionally [so-called theoretical] computer scientists expect and can seemingly only operate in a world of clearly defined concepts. Not so here. It is not possible. Hence we use the term ‘characterization’ in lieu of ‘definition’ •

High Light 0.4 *Seemingly Fragmented Texts:* The text of this paper is a sequence of enumerated sections, sub-sections, sub-subsections and paragraphs, with short HIGHLIGHTS, CHARACTERIZATIONS, EXAMPLES, ONTOLOGICAL CHOICES, PROMPTS, SCHEMAS and ordinary short texts. The brevity is intentional. Each and all of these units outline important concepts. Each contain a meaning and can be read “in isolation” •

1.1 Domains

We start by delineating the informal concept of domain,¹

1.1.1 What are They ?

What do we mean by ‘domain’ ?

Definition 0.1 *Domain:* By a *domain* we shall understand a *rationaly 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 0.1 *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.) [27, *Chapter B*].
- **Road nets** – with street segments and intersections, traffic lights and automobiles – and the flow of these [27, *Chapter E*].
- **Pipelines** – with their liquids (oil, or gas, or water), wells, pipes, valves, pumps, forks, joins and wells and the flow of fluids [27, *Chapter I*].

¹Our use of the term ‘domain’ should not be confused with that of Dana Scott’s Domain Theory: https://en.wikipedia.org/wiki/Scott_domain.

- **Container terminals** – with their container vessels, containers, cranes, trucks, etc. – and the movement of all of these[27, *Chapter K*]•

Characterization 0.1 on the previous page 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**.² 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 modeling – have, as an important property, that they possess man-made entities.

1.1.2 Some Introductory Remarks

1.1.2.1 A Discussion of Our Characterization of a Concept of Domain

Characterization 0.1 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 5, to be reminded of the definition.

1.1.2.2 Formal Methods and Description Language

The reader is assumed to have a reasonable grasp of formal methods – such as espoused in [37, 38, 11, 70].

The descriptions evolving from the modeling approach of this paper are in the abstract, model-oriented specification language RSL [46] of the **Raise**³ Specification Language. But other abstract specification languages could be used: VDM [37, 38], Z [70], Alloy [53], CafeOBJ [45], etc. We have chosen RSL since it embodies a variant of CSP [51] – being used to express domain behaviours.

1.1.2.3 Programming Languages versus Domain Semantics

From around the late 1960s, spurred on by the works of John McCarthy, Peter Landin, Christopher Strachey, Dana Scott and others, it was not unusual to see publications of entire formal definitions of programming language semantics. Widespread technical reports were [5, 4, 1969, 1974] Notably so was [59, 1976]. There was the 1978 publication [37, *Chapter 5, Algol 60*, 1978]. Others were [38, *Chapters 6–7, Algol 60 and Pascal*, 1982] As late as into the 1980s there were such publications [6, 1980].

Formal descriptions of domains, such as we shall unravel a method for their study, analysis and description, likewise amount to semantics for the terms of the professional languages spoken by stakeholders of domains. So perhaps it is time to take the topic serious.

1.1.2.4 A New Universe

The concept of domain – such as we shall delineate and treat it – is novel. That is: new and not treated in this way before. Its presentation, therefore, necessarily involves the introduction of a new universe of concepts. Not the neat, well-defined concepts of neither “classical” computer science nor software engineering. It may take some concentration on the part of the reader to get used to this !

You will therefore be introduced to quite a universe of new concepts. You will find these concepts named in most display lines⁴ and in Figs. 1.1 on page 9 and 1.2 on page 21.

1.2 Six Languages

This section is an artifice, an expedient.

It summarizes, from an unusual angle, an aspect of the presentation style of this paper. *The road ahead of us introduces rather many new and novel concepts. It is easy to get lost. The presentation alternates, almost sentence-by-sentence, between 5 languages. The below explication might help You to keep track of where the paper eventually shall lead us !* This section, in a sense, tells the story backwards!⁵

²Another, “upside-down” – after the fact – [perhaps ‘cheating’] way of defining ‘describable’ is: is it describable in terms of the method of this paper !

³**RAISE** stands for **R**igorous **A**pproach to **I**ndustrial **S**oftware **E**ngineering [47].

⁴– that is, section, subsection, sub-subsection, paragraph and sub-paragraph lines

⁵Søren Kierkegaard: *Life is lived forwards but is understood backwards* [1843].

1.2.1 The 6 Languages

There are 6 languages at play in this paper:

- (i) technical English, as in most papers;
- (ii) RSL, the RAISE Specification Language [46];
- (iii) an augmented RSL language;
- (iv) the domain modeling language – which we can view as the composition of clauses from two [sub-ordinate] languages:
 - (v) a domain analysis language; and
 - (vi) a domain specification

language.

(i) Technical English is the main medium, as in most papers, of what is conveyed. (ii) Domain descriptions are (to be) expressed in RSL. (iii) The [few places where we resort to the] augmented RSL language is needed for expressing names of RSL types as values. (iv) The domain modeling language consists of finite sequences domain analysis and domain description clauses. (v) The domain analysis language just consists of prompts, i.e., predicate functions used informally by the domain analyzer in inquiring the domain. They yield either truth values or possibly augmented RSL texts. (vi) The domain description language consists of a few RSL text yielding prompts.

We presume that the reader is familiar with such languages as RSL. That is: VDM [37, 38], Z [70], Alloy [53], etc. They could all be use instead of, as here, RSL.

We summarize some of the language issues.

The Domain Analysis Language: We list a few, cf. Fig. 1.1 on page 9, of the predicate prompts, i.e., language prompts: `is_entity` [pg 10], `is_endurant` [pg 11], `is_perdurant` [pg 11], `is_solid` [pg 13], `is_fluid` [pg 13], `is_part` [pg 13], `is_atomic` [pg 14], `is_compound` [pg 14], `is_Cartesian` [pg 14], or `is_part-set` [pg 15]; and the extended RSL text yielding analysis prompts: `record_Cartesian_type_names` [pg 15], `record_part_set_type_names` [pg 15] and `record_attribute_type_names` [pg 19].

The Domain Description Language: RSL. We shall use a subset of RSL. That subset is a simple, discrete mathematics, primarily functional specification language in the style of VDM [37, 38]. Emphasis is on sets, Cartesians, lists, and maps (i.e., finite definition set, enumerable functions).

Domain Description: A domain description consists of one or more domain specification units. A specification unit is of either of 10 kinds, all expressed in RSL. (1) a universe-of-discourse **type** clause [pg 12]; (2) a part **type** and **obs_server value** clause [pg 15]; (3) a **value** clause; (4) a unique identifier **type** and (**uid_**) observer value (function) clause [pg 18]; (5) a mereology **type** and (**mereo_**) observer value (function) clause [pg 19]; (6) an attribute **type** and (**attr_**) observer value (function) definition clause [pg 20]; (7) an **axiom** clause; (8) a **channel** declaration clause [pg 24]; (9) a behaviour **value** (signature and definition) clause [pg 24 & pg 28]; and (10) a domain initialization clause [Sect. 1.9.6 on page 31]. These clauses are often combined in 2-3 such clauses, and may, and usually do, include further RSL clauses.

The use of RSL “outside” the domain specification units should not be confused with the RSL of the specification unit schemas and examples.

1.2.2 Semiotics

In *Foundations of the theory of signs* [60] defines semiotics as “consisting” of syntax, semantics and pragmatics.

- **Syntax:** The syntax of domain analysis and domain description clauses are simple atomic clauses consisting of a prompt (predicate or function) identifier, see above, and an identifier denoting a domain entity. The syntax of the domain modeling language prescribes a sequence of one or more domain analysis and domain description clauses.
- **Semantics:** The meaning of a domain analysis clause is that of a function from a domain entity to either a truth value or some augmented RSL text. The meaning of a domain description clause is that of a function from a domain entity to a domain specification unit.
- **Pragmatics:** The pragmatics of a domain analysis predicate clause, as applied to a domain entity e , is that of prompting the domain analyzer to a next domain analysis step: either that of applying a [subsequent, cf. Fig. 1.1] domain analysis predicate prompt to e ; or applying a [subsequent, cf. Fig. 1.1] domain analysis function to e , and noting – as writing down on a “to remember board” – the result of the [latter] query; or applying a [subsequent, cf. Fig. 1.1] domain description function to e . The pragmatics of a domain description function is that of including the resulting RSL domain description text in the emerging domain description. There is no hint as to what to do next!

1.2.3 Speech Acts

The above explication of a pragmatics for the domain modeling language relates to the concepts of *speech acts*. We refer to [3, How to do things with words], [63, Speech Acts: An Essay in the Philosophy of Language] and [62, Brain mechanisms linking language and action]. A further study of the *illocutionary* and *locutionary* aspects of the domain analysis language seems in place.

1.3 Endurants and Perdurants, I

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

Definition 0.2 *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 book solid endurants may be further unanalysed *living species*: *plants* and *animals* – including *humans*.

Definition 0.3 *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* •

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

• • •

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

1.4 A Domain Analysis & Description Ontology

1.4.1 The Chosen Ontology

Figure 1.1 expresses an ontology⁶ for our analysis of domains. Not a taxonomy⁷ for any one specific domain. The idea of Fig. 1.1 on the next page is the following:

- It presents a recipe for how to **analyze** a domain.
- You, the *domain analyzer cum describer*, are ‘confronted’⁸ with, or by a domain.
- You have Fig. 1.1 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 paper, 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. 1.1 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. 1.1 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.

⁶An 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].

⁷A taxonomy (or taxonomic classification) is a scheme of classification, especially a hierarchical classification, in which things are organized into groups or types [Wikipedia].

⁸By ‘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 !

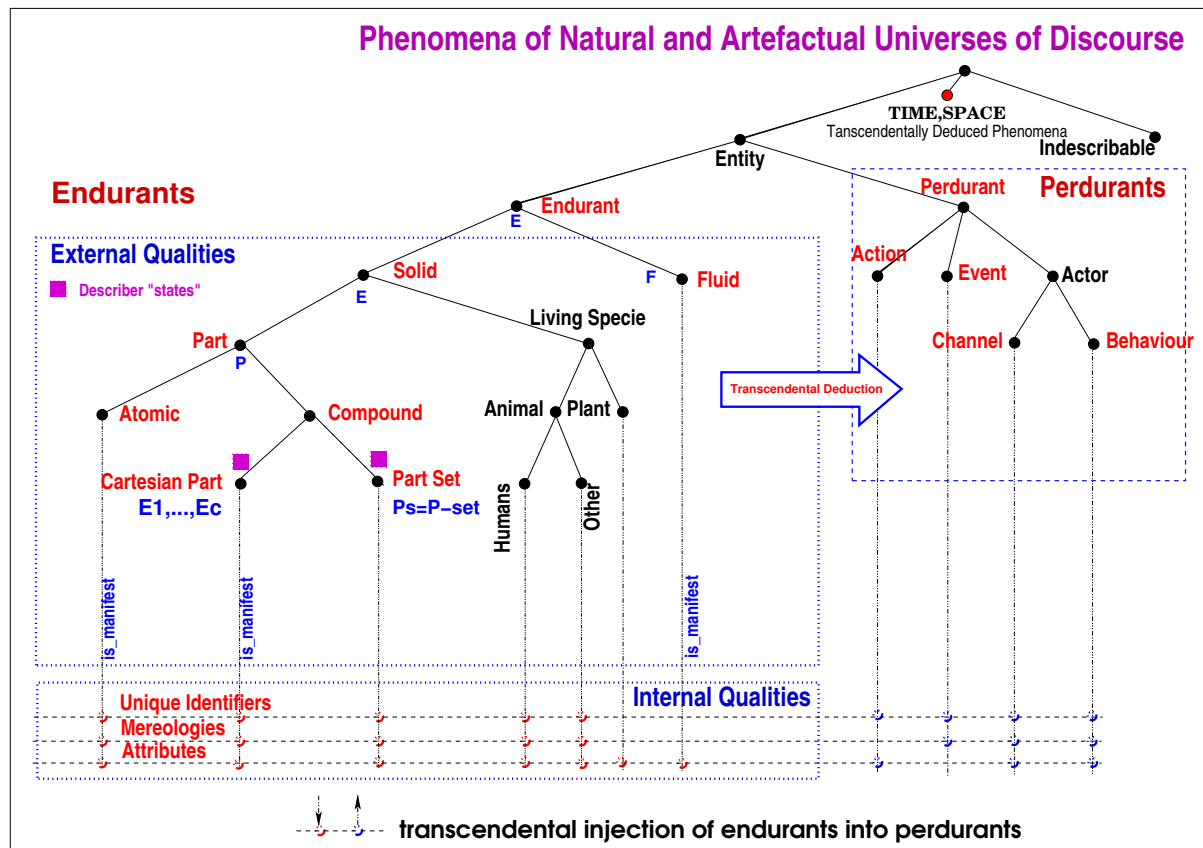


Figure 1.1: A Domain Analysis & Description Ontology

- 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 book, 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 descriptor **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”.

• • •

In Fig. 1.1 You will have noticed the positioning of the concepts of **TIME** and **SPACE** “right under” the *Phenomena* bullet •. These two concepts are neither endurants not perdurants. And they are not attributes of either. They can, however, as shown by Sørlander [67], be transcendently deduced by rational reasoning.

1.4.2 Discussion of The Chosen Ontology

We shall in the following motivate the choice of the *ontological classification* reflected in Fig 1.1. We shall argue that this classification is not “an accidental choice”. In fact, we shall try to justify the classification with reference to the philosophy of Kai Sørlander [64, 65, 66, 67]⁹. Kai Sørlander’s aim in these books is to examine *that which is absolutely necessary, inevitable, in any description of the world*. In [21, Chapter 2] we present a summary of Sørlander’s philosophy. In paragraphs, in the rest of this paper, marked **ONTOLOGICAL CHOICE**, we shall relate Sørlander’s philosophy’s “inevitability” to the ontology for studying domains.

⁹The 2022 book, [66], is presently a latest in Kai Sørlander’s work. It refines and further develops the theme of the earlier, 1994–2016 books. [67] is an English translation of [66]

1.5 The Name, Type and Value Concepts

Domain *modeling*, 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.

1.5.1 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.¹⁰ 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.

1.5.2 Types

The type concept is crucial to programming and modeling.

Definition 0.4 *Type*: A *type* is a class, i.e., a further undefined set, of values (“of the same kind”) •

We name types.

Example 0.2 *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*.

1.5.3 Values

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

1.6 Phenomena and Entities

Definition 0.5 *Phenomena*: By a *phenomenon* we shall understand a fact that is observed to exist or happen •

Some phenomena are rationally describable – to some degree¹¹ – others are not.

Definition 0.6 *Entities*: By an entity By an *entity* we shall understand a more-or-less rationally describable phenomenon •

Prompt 0.5 *is_entity*: We introduce the informal presentation language predicate *is_entity*. It holds for phenomena ϕ if ϕ is describable •

A *prompt*¹² is an informal “advice” to the domain analyzer to “perform” a mental inquiry wrt. the real-life domain being studied.

¹⁰The study of names is called *onomastics* or *onomatology*. *Onomastics* covers the naming of all things, including place names (toponyms) and personal names (*anthroponyms*).

¹¹That 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.

¹²French: *mot-clé*, German: *stichwort*, Spanish: *palabra clave*

Example 0.3 *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 paper: their external and internal qualities, unique identifiers, mereologies and attributes, channels and behaviours !*

Ontological Choice 0.6 *Phenomena*: 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.

1.7 Endurants and Perdurants, II

We repeat our characterizations of endurants and perdurants.

1.7.1 Endurants

We repeat characterization 0.2 on page 8.

Definition 0.7 *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 •

Example 0.4 *Endurants*: Examples of endurants are: a street segment [link], a street intersection [hub], an automobile •

Prompt 0.7 *is_endurant*: We introduce the informal presentation language predicate *is_endurant* to hold for entity *e* if *is_endurant(e)* holds •

1.7.2 Perdurants

We repeat characterization 0.3 on page 8.

Definition 0.8 *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* •

Example 0.5 *Perdurant*: A moving automobile is an example of a perdurant •

Prompt 0.8 *is_perdurant*: We introduce the informal presentation language predicate *is_perdurant* to hold for entity *e* if *is_perdurant(e)* holds •

1.7.3 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 •

1.8 External and Internal Endurant Qualities

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.

Definition 0.9 *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.

Definition 0.10 *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 0.9 *Rationality*: 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? The choice here is also that we separate our inquiry into examining *both external and internal qualities* of endurants [not ‘either or’] •

1.8.1 External Qualities – Tangibles

Example 0.6 *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 •

1.8.1.1 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.

Schema 0.10 *The Universe of Discourse*:

Narration:

The name, and hence the type, of the domain is UoD

The UoD domain can be briefly characterized by ...

Formalization:

type UoD •

1.8.1.2 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 0.11 *Solids vs. Fluids*: 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] •

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

1.8.1.2.1 Solid cum Discrete Endurants.

Definition 0.11 *Discrete cum Solid Endurants*: By a *solid cum 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 (or height), breadth and depth [58, *OED*, Vol. II, pg. 2046] •

Example 0.7 *Solid Endurants*: Pipeline system 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.) •

Prompt 0.12 *is_solid*: We introduce the informal presentation language predicate *is_solid* to hold for endurant *e* if *is_solid(e)* holds •

1.8.1.2.2 Fluids.

Definition 0.12 *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 [58, *OED*, Vol. I, pg. 774] •

Example 0.8 *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”.

Prompt 0.13 *is_fluid*: We introduce the informal presentation language predicate *is_fluid* to hold for endurant *e* if *is_fluid(e)* holds •

1.8.1.3 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 0.14 *Parts and Living Species*: With Sørlander, [67, Sect. 5.7.1, pages 71–72] we reason that one can distinguish between parts and living species •

1.8.1.3.1 Parts

Definition 0.13 *Parts*: The non-living solid species 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 0.9 *Parts*: Example 0.7, of solids, is an example of parts •

Prompt 0.15 *is_part*: 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 0.16 *Atomic and Compound Parts*: It is an empirical fact that parts can be composed from parts. That possibility exists. Hence we can [philosophy-wise] reason likewise •

— Atomic Parts.

Definition 0.14 *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 •

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

¹⁵ See footnote 14.

Example 0.10 *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 •

Prompt 0.17 *is_atomic*: We introduce the informal presentation language predicate `is_atomic` to hold for parts `p` if `is_atomic(p)` holds •

— **Compound Parts.**

Definition 0.15 *Compound Part*: Compound parts are those which are observed to [potentially] consist of several parts •

Example 0.11 *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 •

Prompt 0.18 *is_compound*: 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 0.19 *Cartesians*: 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! •

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

Definition 0.16 *Cartesians*: Cartesian parts are those compound parts which are observed to consist of two or more distinctly sort-named endurants (solids or fluids) •

Example 0.12 *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 •`

Prompt 0.20 *is_Cartesian*: 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.

Prompt 0.21 *record_Cartesian_part_type_names*:

value

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

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

Here \mathbb{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” it to an observed enduring and notes down, in her mind or jots it on a scratch of paper, her decision as to appropriate [new] type names.

Example 0.13 *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 •

— Part Sets.

Definition 0.17 *Part Sets*: Part sets are those compound parts which are observed to consist of an indefinite number of zero, one or more parts •

Prompt 0.22 *is_part_set* : 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.

Prompt 0.23 *record-part-set-part-type-names*:

value

```
record_part_set_part_type_names: E → TPs × TP
record_part_set_part_type_names(e:E) as ( $\eta Ps, \eta P$ ) •
```

Here the name of the value, e , and the type names ηPs 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 enduring and notes down, in her mind or jots it on a scratch of paper, her decision as to appropriate [new] type names.

Example 0.14 *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) = ($\eta As, \eta A$)` •

— 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 0.24 *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 ( $\eta$  Ps,  $\eta$  P) = record_part_set_part_type_names(e:E) in
  ‘‘type
    P, Ps = P-set,
  value
    obs_Ps: E  $\rightarrow$  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 domain describers’ “notice board”, and, as such, elements of the development of domain models. We have introduced a core domain modeling tool the **obs_...** observer function, one to be “applied” mentally by the domain describer, and one that appears in (RSL) 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, 0.24, 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 0.24 on the previous 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*.

1.8.1.4 States

Definition 0.18 *States*: By a *state* we shall mean any subset of the parts of a domain •

Example 0.15 *Road Transport State*:

variable

```

hs:AH := obs_AH(obs_RN(rt)),
ls:AL := obs_AL(obs_RN(rt)),
as:SA := obs_SA(rt),
 $\sigma$ : (H|L|A)-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 0.16 *Road Transport Development*: adding or removing hubs, links and automobiles •

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

1.8.1.5 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 [61]. 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.

1.8.1.6 Summary of Endurant Analysis Predicates

Characterizations 0.6–0.17 imply the following analysis predicates (Char.: δ , Page π):

¹⁷The concepts of monitorable, including biddable part attributes is treated in Sect. 1.8.2.3.2.

¹⁸testing, model checking and theorem proving

- `is_entity`, $\delta 0.6 \pi 10$
- `is_endurant`, $\delta 0.7 \pi 11$
- `is_perdurant`, $\delta 0.8 \pi 11$
- `is_solid`, $\delta 0.11 \pi 13$
- `is_fluid`, $\delta 0.12 \pi 13$
- `is_part`, $\delta 0.13 \pi 13$
- `is_atomic`, $\delta 0.14 \pi 13$
- `is_compound`, $\delta 0.15 \pi 14$
- `is_Cartesian`, $\delta 0.16 \pi 14$
- `is_part_set`, $\delta 0.17 \pi 15$

We remind the reader that the above predicates represent “formulas” in the presentation, **not** the description, language. They are not RSL 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.

1.8.1.7 “Trees are Not Recursive”

A ‘fact’, that seems to surprise many, is that parts are not “recursive”. Yes, in all our domain modeling experiments, [27], we have not come across the need for recursively observing compound parts. Trees, for example, are not recursive in this sense. Trees have roots. Sub-trees not. Banyan trees²⁰ have several “intertwined trees”. But it would be ‘twisting’ the modeling to try fit a description of such trees to a ‘recursion wim’! Instead, trees are defined as nets, such as are road nets, where these nets then satisfy certain constraints [27, *Chapter B*] – usually modeled by a mereology, see Sect. 1.8.2.2 on the following page.

1.8.2 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 0.6–0.17, and analysis functions, schemas 0.21–0.23, and a pair of description functions, schema 0.24 on page 15.

The application of description functions result in RSL 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 RSL 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*.

Definition 0.19 *Internal Qualities*: Internal qualities are those properties [of endurants] that do not occupy *space* but can be measured or spoken about •

Example 0.17 *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.

1.8.2.1 Unique Identity

Ontological Choice 0.25 *Unique Identity*: We postulate that separately discernible parts have unique identify. The issue, really, is a philosophical one. We refer to [21, *Sects. 2.2.2.3–2.2.2.4, pages 14–15*] for a discussion of the existence and uniqueness of entities •

Definition 0.20 *Unique Identity*: A unique identity is an immaterial property that distinguishes any two *spatially* distinct solids²¹ •

The unique identity of a part *p* of type *P* is obtained by the postulated observer **uid_P**:

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

²⁰<https://www.britannica.com/plant/banyan>

²¹For 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 modeling experiments have not found a need for such ascription !

Schema 0.26 *Describe-Unique-Identity-Part-Observer*

```

‘‘type
  P,PI
value
  uid_P: P → PI’’ •

```

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

Example 0.18 *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 modeling tool the **uid_...** observer function, one to be “applied” mentally by the domain describer, and one that appears in (RSL) domain descriptions The **uid_...** observer function is “applied” by the domain describer, it is not a computable function.

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

Example 0.19 *Road Transport Uniqueness*:

variable

$$\begin{aligned}
 hS_{uids}:HI\text{-set} &:= \{ \text{uid}_H(h) \mid h:H \cdot u \in \sigma \} \\
 lS_{uids}:LI\text{-set} &:= \{ \text{uid}_L(l) \mid l:L \cdot u \in \sigma \} \\
 aS_{uids}:AI\text{-set} &:= \{ \text{uid}_A(a) \mid a:A \cdot u \in \sigma \} \\
 \sigma_{uids}: (HI|LI|AI)\text{-set} &:= \{ \text{uid}_{(H|L|A)}(u) \mid u: (H|L|A) \cdot u \in \sigma \}
 \end{aligned}$$

axiom

□ **card** $\sigma = \text{card } \sigma_{uids}$ • For σ see Sect. 1.8.1.4 on page 16.

We have chosen, for the same reason as given in Sect. 1.8.1.4, 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).

1.8.2.2 Mereology

The concept of mereology is due to the Polish mathematician, logician and philosopher Stanisław Leśniewski (1886–1939) [69, 14].

Definition 0.21 *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 0.27 *Mereology*: 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 0.20 *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 0.21 *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 analyses and describe their mereologies. The inquiry: **mereo_P**(p) yields a mereology type (name), say PMer, and its description²²:

Schema 0.28 *Describe-Mereology*

```

‘ ‘type
  PMer =  $\mathcal{M}$ (PI1,PI2,...,PIm)
value
  mereo_P: P → PMer
axiom
   $\mathcal{A}$ (pm:PMer) ’ ’ •

```

where \mathcal{M} (PI1,PI2,...,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.

1.8.2.3 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.

1.8.2.3.1 General

Definition 0.22 *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 0.29 *Attributes*: 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 •

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 0.30 *record-attribute-type-names*

```

value
  record_attribute_type_names: P →  $\eta\mathbb{T}$ -set
  record_attribute_type_names(p:P) as  $\eta\mathbb{T}$ -set •

```

Example 0.22 *Road Net Attributes, I*: Examples of attributes are: hubs have states, $h\sigma:H\Sigma$: the set of pairs of link identifiers, (fli,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.

²²Cf. Sect. 1.8.1.3.1

```

type
  HΣ = (LI×LI)-set
  HΩ = HΣ-set
  LEN = Nat m
  APos = atH | onL
  atH :: HI
  onL :: LI × (fhi:HI × f:Real × thi:HI)
  HHis, LHis = (TIME×AI)*
  AHis = (TIME×(HI|LI))*

  attr_HΣ: H → HΣ
  attr_HΩ: H → HΩ
  attr_LEN: L → LEN
  attr_APos: A → APos
  attr_HHis: H → HHis
  attr_LHis: L → LHis
  attr_AHis: A → AHis

axiom
  ∀ (li, (fhi, f, thi)): onL • 0 < f < 1
    ∧ li ∈ ls_uids ∧ {fhi, thi} ⊆ hs_uids ∧ ... •

value

```

Schema 0.31 *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:  $\mathcal{A}(a1, a2, \dots, an)$  ’’
end •

```

1.8.2.3.2 Michael A. Jackson’s Attribute Categories Michael A. Jackson [54] 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 [54] and [21] [*Chapter 5, Sect. 5.4.2.3*] for details. We suggest a minor revision of Michael A. Jackson’s attribute categorization, see left side of Fig. 1.2 on the next page. We single out the inert from the ontology of Fig. 1.2 on the facing page, left side. Inert attributes seem to be “set externally” to the endurant. So we now distinguish between `is_external` and `is_internal` dynamic attributes. We summarize Jackson’s attribute and our revised categorization in Fig. 1.2.

This distinction has [pragmatic] consequences for how we treat arguments of the behaviours of parts, cf. Sect. 1.9.5.1 (page 25).

Example 0.23 *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 unto themselves and their surroundings”.

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

³⁰`programmable`: values can be prescribed

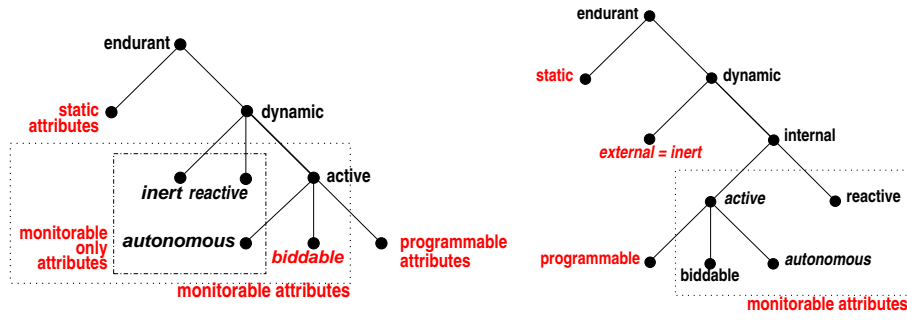


Figure 1.2: Michael Jackson's [Revised] Attribute Categories

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

value

`p:P`

`tns = record_attribute_type_names(p)`

`static_attributes: $\eta T\text{-set} \rightarrow \eta T\text{-set}$`

`static_attributes(tns) $\equiv \{ \eta tn \mid \eta tn:\eta T \cdot \eta tn \in tns \wedge \text{is_static}(tn) \}$`

`inert_attributes: $\eta T\text{-set} \rightarrow \eta T\text{-set}$`

`inert_attributes(tns) $\equiv \{ \eta tn \mid \eta tn:\eta T \cdot \eta tn \in tns \wedge \text{is_inert}(tn) \}$`

`monitorable_attributes $\eta T\text{-set} \rightarrow \eta T\text{-set}$`

`monitorable_attributes(tns) $\equiv \{ \eta tn \mid \eta tn:\eta T \cdot \eta tn \in tns \wedge \text{is_monitorable}(tn) \}$`

`programmable_attributes $\eta T\text{-set} \rightarrow \eta T\text{-set}$`

`programmable_attributes(tns) $\equiv \{ \eta tn \mid \eta tn:\eta T \cdot \eta tn \in tns \wedge \text{is_programmable}(tn) \}$`

`is_monitorable: $T \rightarrow \mathbf{Bool}$`

`is_monitorable(t) $\equiv \sim \text{is_static}(t) \wedge \sim \text{is_inert}(t) \wedge \sim \text{is_programmable}(t)$`

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

1.8.3 Intentional Pull

Ontological Choice 0.32 *Intentional Pull*: In [65, 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 endurants then it does have an intentionality and the components’ individual intentionalities does, i.e., shall relate to that.

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

Example 0.24 *Road Transport Intentionality:* Automobiles include the *intent*: 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 0.25 *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 0.26 *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}

1.8.4 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. We have carefully justified this ontology in ‘**Ontological Choice**’ paragraphs

1.9 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**.

1.9.1 “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 [51] constructs of RSL (derived from RSL [46]) to model concurrent behaviours.

1.9.2 Transcendental Deduction

Transcendse is the basic ground concept from the word’s literal meaning of climbing or going beyond, albeit with varying connotations in its different historical and cultural stages.

Definition 0.23 *Transcendence:* By **transcendence** we shall understand the notion: **the a priori or intuitive basis of knowledge, independent of experience** •

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.

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

Definition 0.24 *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 0.27 *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 [52], 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** [43] 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 0.33 *Transcendental Deduction of Behaviours from Parts*: So this, then, is, in a sense, our “final” ontological choice: that of transcendentially deducing behaviours from parts •

1.9.3 Actors – A Synopsis

This section provides a summary overview.

Definition 0.25 *Actors*: An actor is anything that can initiate an **action, event** or **behaviour** •

1.9.3.1 Action

Definition 0.26 *Actions*: An action is a function that can purposefully change a state •

Example 0.28 *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 •

1.9.3.2 Event

Definition 0.27 *Events*: An event is a function that surreptitiously changes a state •

Example 0.29 *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.

1.9.3.3 Behaviour

Definition 0.28 *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*: $\text{ch}[\{i,j\}] ! \text{value} / \text{ch}[\{i,j\}] ?$.

Example 0.30 *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*.

1.9.4 Channel

Definition 0.29 *Channel*: A channel is anything that allows synchronization and communication of values between behaviours •

Schema 0.34 Channel

We suggest the following schema for describing channels:

channel { $\text{ch}[\{ui,uj\}] \mid ui,uj:UI \cdot \dots$ } : 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 0.31 *Road Transport Interaction Channel*:

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

1.9.5 Behaviours

We single out the perdurants of behaviours – as they relate directly to the parts of Sect. 1.8. The treatment is “divided” into three sections.

1.9.5.1 Behaviour Signature

Schema 0.35 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

$B_p: \text{Uid}_p \rightarrow^{34} \text{Mereo}_p \rightarrow \text{Sta_Vals}_p \rightarrow \text{Inert_Vals}_p \rightarrow \text{Mon_Refs}_p \rightarrow \text{Prgr_Vals}_p \rightarrow \{ \text{ch}[\{i,j\}] \mid \dots \}$ **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 , $\text{mereo}_P(p) = \text{Mereo}_p$;

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

- 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 0.32 Road Transport Behaviour Signatures:

value

```

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

```

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

1.9.5.2 Inert Arguments: Some Examples

Let us give some examples of inert attributes of automobiles. (i) Driving uphill, one 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.

1.9.5.3 Behaviour Definitions

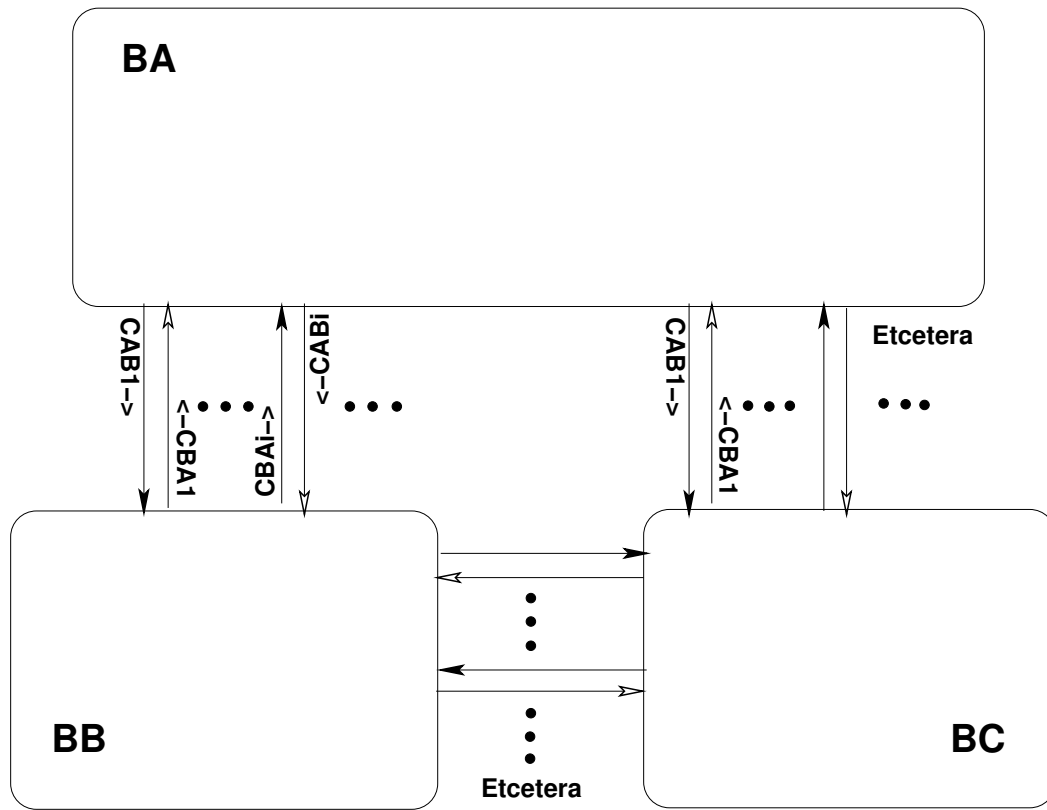
A typical, informal rendition of abstracted behaviours, BA, BC, BD, ... is shown in Fig. 1.9.5.3 on the next page.

Figure 1.9.5.3 on the following page should be understood as follows:³⁶ The **bold faced** labels **BA**, **BB**, **BC**, ... are meant to designate behaviours. The **black arrows**, from behaviour **Bx** to behaviour **By** are meant to designate CSP-like *communications* from **Bx** to **By**. The **open arrows** (“white”), from behaviour **Bx** to behaviour **By** are meant to designate possible CSP-like *communications* from **Bx** to **By**. These latter communications, the “possible” ones, are then thought of as *in response* to the “earlier”, in the figure: “immediately prior, next to” communication from **Bx** to **By**.

Figure 1.9.5.3 on the next page is now given a more precise “meaning” – with this “meaning” suggesting a general “pattern” for behaviour definitions:

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

³⁶The explanation of Fig. 1.9.5.3 is in now way an attempt to explain the semantics of behaviours. That is left to the RSL⁺ formalization's.

**Legend:**

BA, BB, BC, .. behaviours

CXY-> communication from behaviour X to Y

<-CYX communication from behaviour Y to X

→ initial comm.

← possible reply comm.

Figure 1.3: Communicating Behaviours

1. There are behaviours B, ... with identities b_1, \dots
 - (a) These behaviours, typically, have the form of internal, \square , non-deterministically “choosing” between
 - (b) *pro-actively* initiating communications with other behaviors
 - (c) and *re-actively* responding to such initiatives.

value

- 1a. $B(b_i)(\text{mereo})(\text{stat})(\text{mon})(\text{prg}) \equiv$
- 1b. $\text{pro_active_B}(b_i)(\text{mereo})(\text{stat})(\text{mon})(\text{prg})$
- 1c. $\square \text{re_active_B}(b_i)(\text{mereo})(\text{stat})(\text{mon})(\text{prg})$

2. $\text{pro_active_B}(b_i)(\text{mereo})(\text{stat})(\text{mon})(\text{prg})$ The pro-active behaviour (B) internal deterministically (\square) choosing between a number of initiating actions:

- (a) action 1,
- (b) action 2,
- (c) ...,
- (d) action n.

value

```

2., 11b  $\pi$ 26.  pro_active_B(bi)(mereo)(stat)(mon)(prg)  $\equiv$ 
2a.              B_action_1(bi)(mereo)(stat)(mon)(prg)
2b.               $\sqcap$  B_action_2(bi)(mereo)(stat)(mon)(prg)
2c.               $\sqcap$  ...
2d.               $\sqcap$  B_action_m(bi)(mereo)(stat)(mon)(prg)

```

3. 11b π 26. The responding behaviour (B) reacts to a number of such initiating actions by

- (a) external non-deterministically (\sqcap) offering to accept messages from responding behaviours,
- (b) and then performing corresponding actions.

value

```

3a., 11b  $\pi$ 26.  respond_B(bi)(mereo)(stat)(mon)(prg)  $\equiv$ 
3a.              let msg =  $\sqcap$  { comm[{bj,bi}]? | bj:BI • bj  $\in$  bis } in
3b.              react_behaviour_B(bi)(mereo)(stat)(mon)(prg)(msg) end

```

4. The react_behaviour_B inquires as to the type of the message, say, a command, received (?): if it is:

- (a) of type Cmd_i then it performs action act_cmd_i,
- (b) of type Cmd_j then it performs action act_cmd_j,
- (c) ..., or
- (d) of type Cmd_k then it performs action act_cmd_k.
- (e) If it is of neither of these types then it “skips” treatment of that response by resuming to be the behaviour B.

value

```

4.  react_behaviour_B(bi)(mereo)(stat)(mon)(prg)(msg)  $\equiv$ 
4a.      is_action_i(msg)  $\rightarrow$  B_action_i(bi)(mereo)(stat)(mon)(prg)(msg),
4b.      is_action_j(msg)  $\rightarrow$  B_action_j(bi)(mereo)(stat)(mon)(prg)(msg),
4c.      ...,
4d.      is_action_k(msg)  $\rightarrow$  B_action_k(bi)(mereo)(stat)(mon)(prg)(msg),
4e.       $\_ \rightarrow$  B(bi)(mereo)(stat)(mon)(prg)

```

1.9.5.4 Action Definitions

“Actions are what makes behaviours meaningful” We remind the reader that our function (incl. behaviour) definitions are expressed in a functional, “applicative”, style. [that is, there are no assignable variables] The actions elaborate to **values**. These values may be Booleans, numbers, sets, Cartesians, lists, maps and functions (over these), or the values may be (), of type **Unit**, as are the values (also of never-ending) behaviours.

Action signatures usually “follow that”, i.e., are the same as “their” initiating behaviour signatures.

5. Actions, as semantic quantities,

- (a) evaluate some values,
- (b) typically change some programmable attributes,

- (c) and may communicate, “issue” or inform, to some other behaviours, some requests, respectively information –
- (d) whereupon the “revert”, “tail-recursively” to the activating Behaviour.

```

5.  action_i(bi)(mereo)(stat)(mon)(prg) ≡
5a.    let v = evaluate_i(bi)(mereo)(stat)(mon)(prg) in
5b.    let (bj,prg') = elaborate_i(v)(bi)(mereo)(stat)(mon)(prg) in
5c.    comm[{bi,bj}] !  $\mathcal{E}$ (prg') ;
5d.    behaviour(bi)(mereo)(stat)(mon)(prg')
5.    end end

```

Variants of Item $\iota 5c \pi 28$ are also used:

$$\{ \text{comm}[\{bi,bj\}] ! \mathcal{E}(\text{prg}') \mid bj \in bis \} ;$$

where bj ranges over bis , a set of behaviour identities.

1.9.5.5 Behaviour Invocation

Schema 0.36 Behaviour Invocation

Behaviours are invoked as follows:

$$\begin{aligned}
 & \text{'B}_p(\text{uid}_p(p))^{37} \\
 & \quad (\text{mereo}_P(p)) \\
 & \quad (\text{attr_staA}_1(p), \dots, \text{attr_staA}_s(p)) \\
 & \quad (\text{attr_inertA}_1(p), \dots, \text{attr_inertA}_i(p)) \\
 & \quad (\text{attr_monA}_1, \dots, \text{attr_monA}_m) \\
 & \quad (\text{attr_prgA}_1(p), \dots, \text{attr_prgA}_p(p)) \text{' }
 \end{aligned}$$

- 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 •

³⁷We show the arguments of the invocation on separate lines only for readability. That is: normally we show the invocation arguments as $B(\dots)(\dots)(\dots)(\dots)(\dots)$.

1.9.5.6 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:

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

value

```
[*]   retr_part:  PI → P  read
[*]   retr_part(pi) ≡ let p:P · p ∈ c σ ∧ uid_P(p)=pi in p end
[*]   pre:  ∃ p:P · p ∈ c σ ∧ uid_P(p)=pi
```

You may think of the functions being illustrated in this section, Sect. 1.9.5.6, `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.

1.9.5.6.1 Evaluation of Monitorable Attributes.

6. 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.
7. Evaluation of the [current] attribute A value of p is defined by function `read_A_from_P`.

value

```
6.   pi:PI, a:A, ηA:ηT
7.   read_A_from_P: PI × T → read σ A
7.   read_A(pi, ηA) ≡ attr_A(retr_part(pi))
```

1.9.5.6.2 Update of Biddable Attributes

8. 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 .
9. Part p is retrieved from the global state.
10. 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.
11. That new p' replaces p in σ .

value

```

8.    $\sigma, a:A, \text{pi}:PI, \eta A:\eta\mathbb{T}$ 

8.   update_P_with_A:  $PI \times A \times \eta\mathbb{T} \rightarrow \mathbf{write} \ \sigma$ 
8.   update_P_with_A(pi,a, $\eta A$ )  $\equiv$ 
9.       let p = retr_part(pi) in
10.      let p':P .
10a.      uid_P(p')=pi
10b.       $\wedge$  mereo_P(p)=mereo_P(p')
10c.       $\wedge \forall \eta A' \in \text{record\_attribute\_type\_names}(p) \setminus \{\eta A\}$ 
10c.       $\Rightarrow \text{attr}_{A'}(p)=\text{attr}_{A'}(p')$ 
10d.       $\wedge \text{attr}_A(p')=a$  in
11.       $\sigma := \mathbf{c} \sigma \setminus \{p\} \cup \{p'\}$ 
8.      end end
9.      pre:  $\exists p:P \cdot p \in \mathbf{c} \sigma \wedge \text{uid}_P(p)=\text{pi}$ 

```

1.9.5.7 Behaviour Description – Examples

Behaviour descriptions rely strongly on CSPs' [51] expressivity. Leaving out some details (., '...'), and *without "further ado"*, we exemplify.

Example 0.33 *Automobile Behaviour at Hub:*

12. 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.

```

12  automobile(ai)(ris)(...)(atH(hi),ahis,_)  $\equiv$ 
12a  automobile_remains_in_hub(ai)(ris)(...)(atH(hi),ahis,_)
12b   $\sqcap$ 
12c  automobile_leaving_hub(ai)(ris)(...)(atH(hi),ahis,_)
12d   $\sqcap$ 
12e  automobile_stop(ai)(ris)(...)(atH(hi),ahis,_)

```

13. [12a] The automobile remains in the hub:

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

```

13  automobile_remains_in_hub(ai)(ris)(...)(atH(hi),ahis,_)  $\equiv$ 
13a  let  $\tau = \text{record\_TIME}()$  in
13c   $\text{ch}[\{ai,hi\}] ! \tau$  ;
13b  automobile(ai)(ris)(...)(atH(hi), $\langle(\tau,hi)\rangle^{\wedge} \text{ahis,}_\_) \mathbf{end}$ 

```


14. [12c] 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.

```

14  automobile_leaving_hub(ai)({li}Uris)(...)(atH(hi),ahis,_) ≡
14a      let  $\tau$  = record.TIME() in
14b      (ch[{ai,hi}] !  $\tau$  || ch[{ai,li}] !  $\tau$ ) ;
14c      automobile(ai)(ris)(...)(onL(li,(hi,0,_)),⟨( $\tau$ ,li)⟩^ahis,_) end
14      pre: [hub is not isolated]

```

The choice of link entered is here expressed (14) as a non-deterministic choice³⁸. One can model the leave hub/enter link otherwise.

15. [12e] Or the automobile “disappears — off the radar” !

```

15  automobile_stop(ai)(ris),(...)(atH(hi),ahis,_) ≡ stop •

```

rm

1.9.6 Behaviour Initialization.

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

Example 0.34 *Road Transport Initialization*: We “wrap up” the main example of this paper: We omit treatment of monitorable attributes.

- 16. Let us refer to the system initialization as an action.
- 17. All hubs are initialized,
- 18. all links are initialized, and
- 19. all automobiles are initialized.

value

```

16.  rts_initialisation: Unit → Unit
16.  rts_initialisation() ≡
17.      || { hub(uid_H(l))(mereo_H(l))(attr_HΩ(l),...)(attr_HΣ(l),...)| h:H • h ∈ hs }
18.      || || { link(uid_L(l))(mereo_L(l))(attr_LEN(l),...)(attr_LΣ(l),...)| l:L • l ∈ ls }
19.      || || { automobile(uid_A(a))(mereo_A(a))(attr_APos(a)attr_AHis(a),...) | a:A • a ∈ as }

```

We have here omitted possible monitorable attributes. For *hs,ls,as* we refer to Sect. 1.8.1.4 •

1.10 Facets

In this section we shall briefly overview the concept of facets. By a *domain facet* we shall understand one amongst a finite set of generic ways of analyzing a domain: a view of the domain, such that the different facets cover conceptually different views, and such that these views together cover the domain.

We leave it to [21, Chapter 8, pages 205–240] to detail the principles, procedures, techniques and tool for describing facets.

These are the facets that we have so far identified:

³⁸— as indicated by the **pre**- condition: the hub mereology must specify that it is not isolated. Automobiles can never leave isolated hubs.

- intrinsics
- support technology
- rules & regulations
- scripts
- license languages
- management & organization
- human behaviour

1.10.1 Intrinsics

By domain intrinsics we shall understand those *phenomena and concepts* of a domain *which are basic* to any of the other facets, with such domain intrinsics initially covering at least one specific, hence named, stakeholder view.

1.10.2 Support Technology

By a domain support technology we shall understand ways and means of *implementing* certain observed phenomena or certain conceived concepts.

1.10.3 Rules & Regulations

- By a *domain rule* we shall understand some text (in the domain) which prescribes how people or equipment are expected to behave when dispatching their duties, respectively when performing their functions.
- By a *domain regulation* we shall understand some text (in the domain) which prescribes what remedial actions are to be taken when it is decided that a rule has not been followed according to its intention.

1.10.4 Scripts

By a domain script we shall understand the structured, almost, if not outright, formally expressed, wording of a procedure on how to proceed, one that has legally binding power, that is, which may be contested in a court of law.

A special “subclass” of scripts are those of commands.

Commands are syntactic entities. Semantically they denote state changes. The state referred to is the state of the domain. Domain facets, as a wider concept than just commands, were first treated in [22, Chapter 8] which places facets in the wider context of domain modeling. Commands are but just one of the many kinds of script facets.

Commands are defined syntactically, and given semantics in the definition of perdurant behaviours, one set of simple actions per command.

1.10.5 License Languages

A *license* is a right or permission granted in accordance with law by a competent authority to engage in some business or occupation, to do some act, or to engage in some transaction which but for such license would be unlawful.

A *license language* is a [“small”] language (with syntax, semantics and pragmatics) in which to describe licenses.

1.10.6 Management & Organization

- By domain management we shall understand such people (such decisions) (i) who (which) determine, formulate and thus set standards (cf. rules and regulations, Sect. 8.4) concerning strategic, tactical and operational decisions; (ii) who ensure that these decisions are passed on to (lower) levels of management and to floor staff; (iii) who make sure that such orders, as they were, are indeed carried out; (iv) who handle undesirable deviations in the carrying out of these orders cum decisions; and (v) who “backstops” complaints from lower management levels and from “floor” staff.
- By domain organization we shall understand (vi) the structuring of management and non-management staff “oversee-able” into clusters with “tight” and “meaningful” relations; (vii) the allocation of strategic, tactical and operational concerns to within management and non-management staff clusters; and hence (viii) the “lines of command”: who does what, and who reports to whom, administratively and functionally.

1.10.7 Human Behaviour

By domain human behaviour we shall understand any of a quality spectrum of carrying out assigned work: from (i) careful, diligent and accurate, via (ii) sloppy dispatch, and (iii) delinquent work, to (iv) outright criminal pursuit.

1.11 Conclusion

We have summarized a method to be used by [human] domain analyzers cum describers in studying and modeling domains. Our previous publications [16, 19, 21] have, with this paper, found its most recent, we risk to say, for us, final form.

Of course, domain models can be developed without the calculi presented in this paper. And was for many years. From the early 1990s a number of formal models of railways were worked out [48, 8, 10, 35, 9]. The problem, though, was still, between 1992 and 2016, “*where to begin, how to proceed and when to end*”. The domain analysis & description ontology and, hence calculus, of this paper shows how. The systematic approach to domain modeling of this ontology and calculus has stood its test of time. The Internet ‘publication’ <https://www.imm.dtu.dk/~dibj/2021/dd/dd.pdf> include the following domain models³⁹ from the 2007–2024 period. Their development has helped hone the method of the present paper.

1.11.1 Previous Literature

To the best of my knowledge there is no prior, comparable publications in the field of domain science and engineering. Closest would be Michael A. Jackson’s [56]. Well, most computer scientists working in the field of correctness of programs, from somewhat “early on”, stressed the importance of making proper assumptions about the domain. They would then express these “in-line”, as appropriate predicates, with their proofs. Michael A. Jackson, lifted this, to a systematic treatment of the domain in his triplet ‘Problem Frame Approach’: *program, machine, domain* [55]. But Jackson did not lift his problem frame concern into a proper study of domains.

1.11.2 The Method

So the method procedure is this: (1) First analyze and describe the *external qualities* of the chosen domain. (2) For each of the so-described endurants You then analyze and describe their *internal qualities*. (2.1) First their *unique identification*. (2.2) Then their *mereology*. (2.3) Then their *attributes*. (2.4) And finally possible *intentional pulls*. (3) First then are You ready to tackle the issue of perdurants. (3.1) Decide upon the *state*. (That may already have been done in connection with (1).) (3.2) Then describe the *channels*. (3.3) Then analyze and describe [part] *behaviour signatures*. (3.4) Then describe *behaviour invocation*. (3.5) Then *behaviour (body) definitions*. (4) Finally describe *domain initialization*.

1.11.3 Specification Units

The method thus focuses, step-by-step, on the development of the following *specification units*: **type** specification units, **value** specification units, **axiom** specification units, **variable** declaration units, and **channel** declaration units.

There are two forms of *type* specifications: (α) introduction of sorts, i.e., type names, and (β) specification of types: pairs of new type names and type expressions – as atomic, alternate or composite types: set, Cartesian, list, map or function types.

There are basically three forms of value specification units: (i) (“simple”) naming of values, (ii) signature of functions: function name and function type, and (iii) signature of (endurant **obs**_, unique identifier **uid**_, mereology, **mereo**_, and attribute **attr**_.) observer functions.

1.11.4 Object Orientation

So far we have not used the term ‘object’ !

We shall now venture the following:

The combined description of enduring parts and their perdurant behaviour form an object definition.

You can then, for yourself, develop a way of graphically presenting these object definitions such that each part type is represented by a box that contains the specification units for [all] external and internal enduring qualities as well as for the perdurant [part] behaviour signatures and definitions; and such that the mereologies of these parts is represented by [possibly directed] lines connecting relevant boxes.

³⁹

- | | | | |
|--------------------------|-------------------------------|-------------------------------|---------------------------------|
| • <i>Graphs,</i> | • <i>The “7 Seas”,</i> | • <i>Urban Planning,</i> | • <i>Bookkeeping,</i> |
| • <i>Rivers,</i> | • <i>The “Blue Skies”,</i> | • <i>Swarms of Drones,</i> | • <i>Shipping,</i> |
| • <i>Canals,</i> | • <i>Credit Cards,</i> | • <i>Container Terminals,</i> | • <i>Stock Exchanges,</i> |
| • <i>Railways,</i> | • <i>Weather Information,</i> | • <i>A Retailer Market,</i> | • <i>Web Transactions, etc.</i> |
| • <i>Road Transport,</i> | • <i>Documents,</i> | • <i>Assembly Lines,</i> | |

That is, an object concept solely based on essentially inescapable world description facts – as justified by Sørlander’s Philosophy [64, 65, 66, 67] ! No “finicky” programming language “tricks” !

We leave it to the reader to compare this definition to those of so-called object-oriented programming languages.

1.11.5 Other Domain Modeling Approaches

[68] shows fragments of a number of expertly expressed domain models. They are all expressed in RAISE.⁴⁰ But they are not following the method of this paper. In other words, it is possible to develop domain models not using the method ! This author has found, however, that following the method – developed after the projects reported in [68] – leads to far less problematic situations – in contrast to my **not** adhering strictly to the method. In other words, based on this subjective observation, we advice using the method.

There is thus no proof that following the method does result in simpler, straightforward developments.

But we do take the fact that we can justify the method, cf. Fig. 1.1 on page 9, on the basis on the inevitability of describing the world as per philosophy of Kai Sørlander [64, 65, 66, 67], and that that may have a bearing on the experienced shorter domain description development efforts.

1.11.6 How Much ? How Little ?

How wide must we *cast the net* when studying a domain ? The answer to that question depends, we suggest, on whether our quest is for studying a domain in general, to see what might come out, or whether it is a study aiming at a specific model for a specific software development. In the former case *we cast the net* as we please – we suggest: as wide as possible, wider that for specific quests. In the latter case *we should cast the net* as “narrowly” as is reasonable: to fit those parts of a domain that we expect the requirements and software to deal with ! In this latter case we should assume that someone, perhaps the same developers, has first “tried their hand” on a wider domain.

1.11.7 Correctness

Today, 2024, software correctness appears focused on the correctness of algorithms, possibly involving concurrency. Correctness, of software, in the context of a specific domain, means that the software requirements are “correctly” derived from a domain description, and that the software design is correctly derived from the domain requirements, that is: $\mathbb{D}, \mathbb{S} \models \mathbb{R}$. Advances in program proofs helps little if not including proper domain and requirements specifications.

1.11.8 Domain Facets

There is more to *domain modeling* than covered in this paper. In [13] and in [21, Chapter 8] we cover the concept of *domain facets*. General examples of domain facets are *support technologies*, *rules & regulations*, *scripts*, *license languages*, *management & organization*, and *human behaviour*.

1.11.9 Perspectives

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 [49, 50] and [11, Vol. 3, Chapter 19, pages 404–412] for the concept of *business process engineering*. [(iii)] We refer to [21, Chapter 9] for the concept of *requirements engineering*.

1.11.10 The Semantics of Domain Models

The meaning of domain models, such as we describe them in this paper, is, “of course”, the actual, real domain “out there” ! One could, and, perhaps one should, formulate a mathematical semantics of the models, that is, of the **is_...**, **obs_...**, **uid_...**, **mereo_...** and **attr_...** analysis and description functions and what they entail (e.g., the type name labels: $\eta \mathbb{T}$ ’s; etc.). An early such semantics description is given in [15].

⁴⁰Other approaches could also be used: VDM [37, 38], Z [70], Alloy [53], CafeOBJ [45], etc.

1.11.11 Further on Domain Modeling

Additional facets of domain modeling are covered in [12] and [21, *Chapter 8: Domain Facets.*]

1.11.12 Software Development

[12] and [21, *Chapter 9 Requirements*] show how to develop *R*equirements prescriptions from *D*omain descriptions. [11] shows how to develop *S*oftware designs from *R*equirements prescriptions.

1.11.13 Modeling

Domain descriptions, such as outlined in this paper, are models of domains, that is, of some reality. They need not necessarily lead to or be motivated by possible development of software for such domains. They can be experimentally researched and developed just for the sake of understanding domains in which man has had an significantly influence. They are models. We refer to [44] for complementary modeling based on Petri nets. The current author is fascinated by the interplay between graphical and textual descriptions of HERAKLIT, well, in general Petri Nets.

1.11.14 Philosophy of Computing

The Danish philosopher Kai Sørlander [64, 65, 66, 67] has shown that there is a foundation in philosophy for domain analysis and description. We refer to [23, *Chapter 2*] for a summary of his findings.

1.11.15 A Manifesto

So there is no excuse, anymore! Of course we have developed interpreters and compilers for programming languages by first developing formal semantics for those languages [40, 42]. Likewise we must now do for the languages of domain stakeholders, at least for the domains covered by this paper. There really is no excuse!

Part II

A PRACTICE

Chapter 2

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.

This is one of a series, [28, 34, 33, 32, 25], of domain studies of such infrastructure components as government, public utilities, banking, transport, insurance, health care, etc. The current, this ‘Introduction’ chapter is common to these study reports.

2.1 On A Notion of ‘Infrastructure’

Central to our effort of studying “man-made” domains is the notion of *infrastructure*⁴¹. The *infrastructure* can be characterized as follows: *the basic physical and organizational structures and facilities (e.g. buildings, roads, power supplies) needed for the operation of a society or enterprise*, “the social and economic infrastructure of a country”. We interpret the “for example, e.g.,” to include, some already mentioned above: government structure: legislative, executive & judicial units, transport: roads, navigable rivers and lakes, the open sea, banking, educational system, health care, utilities: water, electricity, telecommunications (e.g. the Internet) gas, , etc.,⁴² Also: Winston Churchill is quoted to have said in the House of Commons: “*The young Labour speaker we have just listened to wants clearly impressing his constituency with the fact that he went to Eton and Oxford since he now uses such modern terms as ‘infrastructure’*”.

2.2 Domain Models

We rely on [30, 26, 22, 20, 17]. They provide a scientific foundation for modelling domains in the style of this report.

2.2.1 Some Characterizations

- **Domain:** By a *domain* we shall understand a *rationaly describable* segment of a *manifest*⁴³, *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*.
- **Endurants:** By *endurants* we mean 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].

⁴¹<https://en.wikipedia.org/wiki/Infrastructure>

⁴²According to the World Bank, ‘infrastructure’ is an umbrella term for many activities referred to as ‘social overhead capital’ by some development economists, and encompasses activities that share technical and economic features (such as economies of scale and spill-overs from users to non-users). We take a more technical view, and see infrastructures as concerned with supporting other systems or activities. Software for infrastructures is likely to be distributed and concerned in particular with supporting communication of data, people and/or materials. Hence issues of openness, timeliness, security, lack of corruption and resilience are often important.

⁴³The term ‘manifest’ is used in order to distinguish between these kinds of domains and those of computing and data communication: compilers, operating systems, database systems, the Internet, etc.

- **Perdurants:** By *perdurants* we mean 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].
- **Domain Description:** By a *domain description* we shall here mean a syntactic entity, both narrative and formal, describing the domain. That is, a domain description is a structured text, such as shown in Sects. 3–19 (pages 45–154).
- **Domain Model:** By a *domain model* we shall here mean the mathematical meaning, the semantics as denoted the domain description.

2.2.2 Purpose of Domain Models

The *Triptych* dogma (above) expresses a relation of domain models to software. But domain models serve a wider role. Mathematical models of, say, physics, are primarily constructed to record our understanding of some aspects of the world – only secondarily to serve as a basis for engineering work. So it is with manifest models of infra structure components such banking, insurance, health care, transport, etc. In this, and a series of papers, [33, 32], we shall therefore present the result of infra structure studies. We have, over the years, developed many domain models: [7].

2.2.3 Domain Science & Engineering

A series of publications [17, 20, 22, 26, 31] reflects scientific insight into and an engineering methodology for analyzing and describing manifest domains.

2.3 A Dichotomy

2.3.1 An Outline

As citizens we navigate, daily, in a *God-given* and a *Man-made world*. The God-given world can be characterized, i.e., “domain described”, as having natural science properties. The laws that these natural science properties obey are the same – all over the universe! The Man-made world can be characterized, i.e., “domain described”, as having infrastructure components⁴⁴. The “laws” that these properties obey are not necessarily quite the same around our planet!

2.3.2 The Dichotomy

For our society to work, we are being educated (in primary, secondary, tertiary schools, colleges and at universities). We are taught to to read, write and [verbally] express ourselves, recon and do mathematics, languages, history and the sciences: physics (mechanics, electricity, chemistry, biology, botany’s, zoology, geology, geography, ...), but we are not taught about most of the infrastructure structures⁴⁵. That is the dichotomy.

2.3.3 The Dichotomy Resolved

So there it is:

- first *study* a or several domains;
- then *analyze, describe* and *publish* infrastructure domains;
- subsequently *prepare educational texts* “over” these;
- finally introduce ‘*an infrastructures*’ school course.

2.4 A [Planned] Series of Infrastructure Domain Models

So this *domain science & engineering* paper – on banking – is one such infrastructure domain description. In all we are and would like to work on these infrastructure domains:

⁴⁴state, regional and local government: executive, legislative and judicial, banking, insurance, health care (hospitals, clinics, rehabilitation, family physicians, pharmacies, ...), passenger and goods transport (road, rail, sea and air), manufacturing and sales, publishing (newspapers, radio, TV, books, journals, ...), shops (stores, ...),

⁴⁵See footnote 44.

- **Transport**⁴⁶ [34]
- **Health Care**⁴⁹ [32]
- **Banking**⁴⁷ [28]
- **Insurance**⁴⁸ [33]
- etc.

A report on *double-entry bookkeeping* [25] relates strongly to most of these infra-structure component domains⁵⁰.

⁴⁶<https://www.imm.dtu.dk/dibj/2025/infra/main.pdf>

⁴⁷<https://www.imm.dtu.dk/dibj/2025/infra/banking.pdf>

⁴⁸<https://www.imm.dtu.dk/dibj/2025/infra/insurance.pdf>

⁴⁹<https://www.imm.dtu.dk/dibj/2025/infra/healthcare.pdf>

⁵⁰<http://www.imm.dtu.dk/dibj/2023/doubleentry/dblentrybook.pdf>

Part III

A SIMPLE BEGINNING

Chapter 3

Kinds of Transports

Contents

3.1 Informal Outline	45
3.2 Narrative & Formalization	45

3.1 Informal Outline

The transport we have in mind consists of a common transport net, in the following modelled as a graph of uniquely labeled, bi-directed edges and likewise labeled nodes. The transport net is [“intentional pull”] complemented, cf. Sect. 7 on page 63, by a set of conveyors.

Edges, nodes and conveyors are “of kind”: **“road”**, **“rail”**, **“sea”**, and **“air”**; these are literal values⁵¹. A conveyor is of one kind. Conveyors of kind **“road”** include taxis, buses, trucks and the like. Conveyors of kind **“rail”** include passenger trains, freight trains, etc. Conveyors of kind **“sea”** include sail boats, river and canal barges, fishing vessels, line and ramp freighters, passenger liners, etc. Conveyors of kind **“air”** include helicopters, freight and passenger planes. An edge is of one kind. Edges of kind **“road”** are called automobile roads. Edges of kind **“rail”**, **“sea”** and **“air”** are called rail tracks, sea lanes and air lanes. A node may be of one or more kinds. Nodes of kind **“road”** are called street point (street crossings, street ends, bus stops). Nodes of kind **“rail”** are called train stations. Nodes of kind **“sea”** are called harbours. Nodes of kind **“air”** are called airports.

3.2 Narrative & Formalization

20. There are four kinds of transportation: **“road, rail, sea”** and **“air”**.

type

20. Kind = **“road”|“rail”|“sea”|“air”**

People are not conveyors, so they are no “of a kind”! People may be merchandises.

• • •

That is: transport, in this report, is all about moving goods – here referred to as merchandises – around. By what/whichever means: on roads, rails, sea and/or by air – possibly combining two or more of these: moving from (road) trucks to (air) freight and/or by (sea) freighter– whether line or tramps⁵², or in some other order! We omit considering people as conveyors.

We divide the first formal presentation into five [further] segments: *Overall Transport Endurants*, *Graph Endurants*, *Conveyor Endurants*, *Intentional Pull* and *Perdurants*.

By an overall traffic domain we mean that of a graph⁵³ and a conveyor⁵⁴ sub-domain.

⁵¹– as are **true** and **false**

⁵²a boat or ship engaged in the tramp trade is one which does not have a fixed schedule, itinerary nor published ports of call, and trades on the so-called spot market [https://en.wikipedia.org/wiki/Tramp_trade].

⁵³[https://en.wikipedia.org/wiki/Graph_\(discrete_mathematics\)](https://en.wikipedia.org/wiki/Graph_(discrete_mathematics))

⁵⁴Conveyor: anything that conveys, transports or delivers. [Words are a conveyor of meaning] [<https://en.wiktionary.org/wiki/conveyor>]

A relation between graphs and conveyors is expressed in the intentional pull section.

The “co-operation” of graphs and conveyors is expressed in the perdurant section.

By a graph we mean a set of nodes and edges: nodes are then interpreted as road intersections (hubs); train stations; river, canal and sea harbours; and airports. A node may be one or more of these. Edges are accordingly interpreted as either street (or road) links, irail tracks, sailing or air routes. An edges can be only one of these. Hence there may be many edges between any two [neighbouring] nodes.

By conveyors we mean *buses, trains, boats, ships, and aircraft*.

The presentation follows the domain analysis & description ontology of Fig. 3.1.

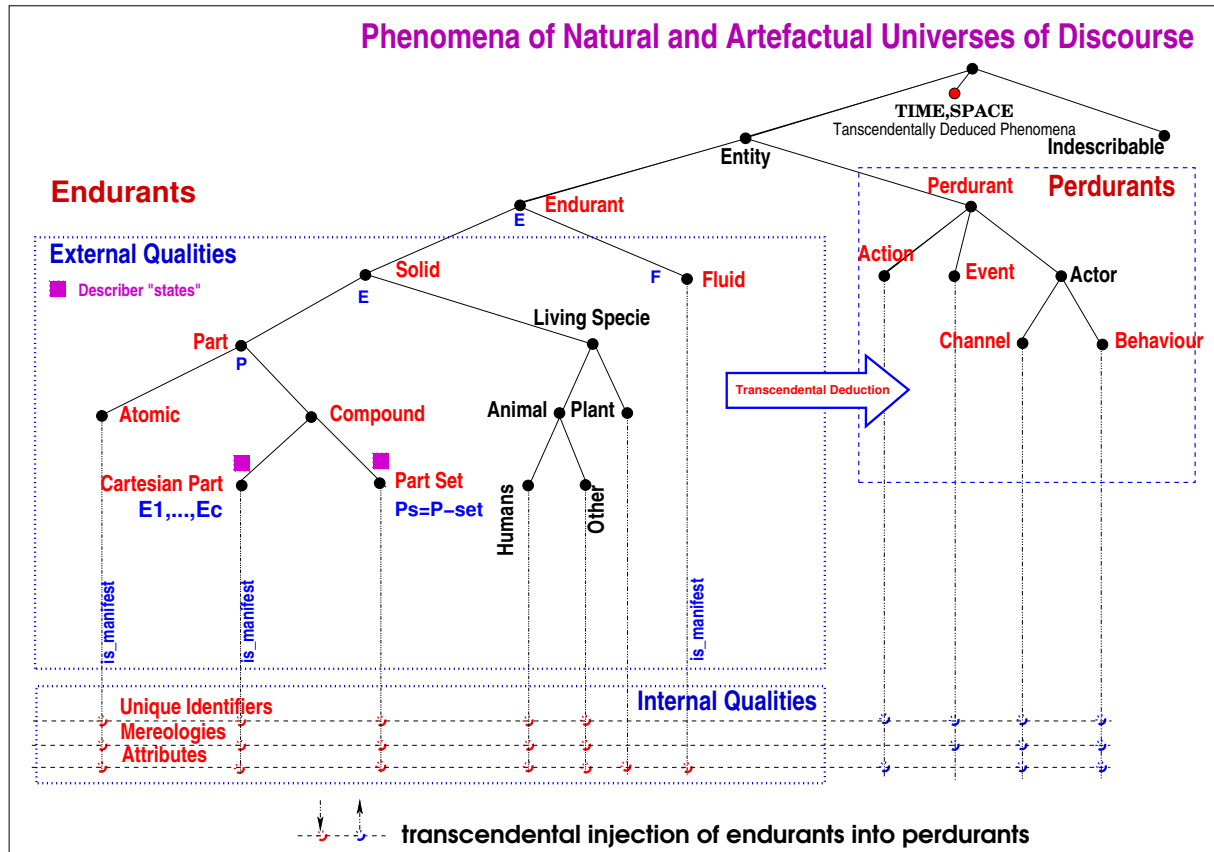


Figure 3.1: Domain Analysis Ontology

Chapter 4

Overall “Single-Mode” Transport Endurants

This early section introduces the, perhaps two most central classes of endurants: `transport nets`, in the abstracted form of `graphs`, and `conveyor aggregates`. Conveyor aggregates embody conveyors. Conveyors “move along” nets, and nets serve to [*intentional pull*⁵⁵] “carry” conveyor traffic.

4.1 Endurant Sorts & Observers

- 21. There is the domain of transport.
- 22. From transport endurants we can observe transport nets, i.e., graphs.
- 23. And from transport endurants we can observe a conveyor aggregate – embodying conveyors.

type

- 21. `T`
- 22. `G`
- 23. `CA`

value

- 22. `obs_G`: $T \rightarrow G$
- 23. `obs_CA`: $T \rightarrow CA$

4.1.1 An Endurant State Notion

We can speak of a transport state.

- 24. There is given a “global”⁵⁶ transport value, t . It contributes to a transport state.
- 25. From this transport value one can derive another transport state element: a global graph value, g .
- 26. And from this transport value one can derive another transport state element: a global conveyor aggregate value, ga .
- 27. We can postulate a transport state to consist of the three endurants: t, g, ca .

value

- 24. $t : T$
- 25. $g : G = \text{obs_G}(t)$
- 26. $ca : CA = \text{obs_CA}(t)$
- 24. $\sigma_t = \{t\} \cup \{g\} \cup \{ca\}$

⁵⁵cf. Sect. 7 on page 63

⁵⁶We shall be using this term: ‘global’ extensively. By double quoting it we intend to express that “global” values are values that can be referred to anywhere in the domain description. We emphasize their “globality” by use this kind of [mathematical] *font* !

4.2 Unique Identification

4.2.1 Unique Identifier Sorts & Observers

28. The transport enduring has a unique identifier.
29. So has the graph, and
30. the conveyor components.

type

28. TI
29. GI
30. CAI

value

28. $\mathbf{uid_T}: T \rightarrow TI$
29. $\mathbf{uid_G}: T \rightarrow GI$
30. $\mathbf{uid_CA}: T \rightarrow CAI$

4.2.2 A Unique Identifier State Notion

We can postulate a “global” transport state value, t .

31. From t we observe its unique identity.
32. Given t we can derive a “global” graph value g , hence its unique identity.
33. And a “global” conveyor aggregate value ca , hence its unique identity..
34. We can therefore postulate an “uppermost” enduring transport state to consist of the three endurants: ti, gi, cai .

value

31. $ti:TI = \mathbf{uid_T}(t)$
32. $gi:GI = \mathbf{uid_G}(g)$
33. $cai:CAI = \mathbf{uid_CA}(ca)$
34. $\sigma_{tis} = \{ti\} \cup \{gi\} \cup \{cai\}$

4.2.3 Uniqueness

35. The three [“uppermost”] transport endurants are distinct: have distinct unique identifiers.

axiom [Uniqueness of Transport Identifiers]

35. $\mathbf{card} \sigma_t = \mathbf{card} \sigma_{tis} = 3$

• • •

It seems that at least the overall transport enduring need not be a manifest one. Hence we leave out treatment of mereology and attributes of the transport enduring.

Chapter 5

Graphs: Transport Nets

In addition to describing the external and internal qualities of transport nets we introduce the concepts or *paths*, i.e., *routes*, through/across a transport net.

5.1 The Endurant Sorts and Observers

External qualities are the enduring sorts of graphs, node and edges aggregates and nodes and edges, their observers and enduring states.

- 36. From graphs one can observe an aggregate, i.e., a set, $ea:EA$, of edges –
- 37. From graphs one can observe an aggregate, i.e., a set, $na:NA$, of nodes –
- 38. From an aggregate of edges one can observe a set of edges.
- 39. From an aggregate of nodes one can observe a set of nodes.
- 40. Edges are considered atomic.
- 41. Nodes are considered atomic.
- 42. We can “lump” all endurants into a sort *parts*.

type

- 36. EA
- 37. NA
- 38. ES = E-set
- 39. NS = N-set
- 40. E
- 41. N

$$42. \quad P = G|EA|NA|ES|NA|N|E$$

value

- 36. **obs_EA**: $G \rightarrow ES$
- 37. **obs_NA**: $G \rightarrow NS$
- 38. **obs_ES**: $EA \rightarrow ES$
- 39. **obs_NS**: $NA \rightarrow NS$

A transport domain taxonomy is hinted at in Fig. 5.1 on the next page.

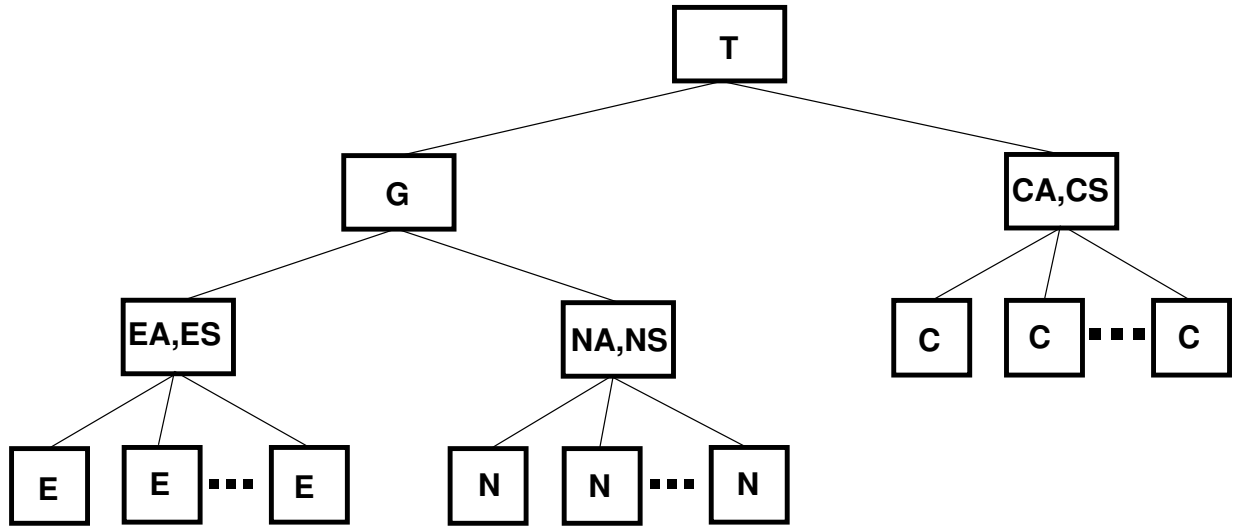


Figure 5.1: A Simplified Transport Domain Taxonomy: Transport Nets, **G**, and Conveyors, **C**

5.1.1 An Endurant State

43. Given the global graph value, there is therefore a “global” value of an edge aggregate.
44. Given the global graph value, there is therefore a “global” value of a node aggregate.
45. Given the global edge aggregate value, there is therefore a “global” node value of the set of all edges.
46. Given the global graph value, there is therefore a “global” value of the set of all nodes.
47. The state of all graph endurants is therefore the set of all graph parts.

value

43. $ea = \mathbf{obs_EA}(g)$
44. $na = \mathbf{obs_NA}(g)$
45. $es = \mathbf{obs_ES}(g)$
46. $ns = \mathbf{obs_NS}(g)$
47. $\sigma_{ps}:\mathbf{P-set} = \{g\} \cup \{ea\} \cup \{na\} \cup es \cup ns$

• • •

Internal qualities are fourfold: unique identification, mereology, attributes and intentional pull.

5.2 Unique Identifiers

Unique Identification has three facets: sort, observers and an axiom.

5.2.1 Unique Identifier Sorts and Observers

48. All parts have identification:
49. the graph,
50. the edge and node aggregates,
51. the sets of edges and nodes, and
52. each edge and node.
53. No two of these are the same, i.e., part identifiers are unique.

type

48. $PI = GI|EAI|NAI|ESI|NSI|EI|NI$
48. $GI, EAI, NAI, ESI, NSI, EI, NI$

value

49. $uid_G: G \rightarrow GI$
50. $uid_EA: EA \rightarrow EAI$, $uid_NA: NA \rightarrow NAI$
51. $uid_ES: ES \rightarrow ESI$, $uid_NS: NS \rightarrow NSI$
52. $uid_E: E \rightarrow EI$, $uid_N: N \rightarrow NI$

5.2.2 A Unique Identifier State

54. There is a “global” unique graph identifier.
55. There are, correspondingly, “global” edge and node aggregate identifiers.
56. There are, correspondingly, “global” edge set and node set identifiers; and
57. set of edge identifiers and
58. set of node identifiers.
59. The unique identifier state is the union of all the unique identifiers.

value

54. $gi = uid_G(g)$
55. $ea_{uis} = uid_EA(ea)$, $na_{uis} = uid_NA(na)$
56. $es_{uis} = uid_ES(ea)$, $ns_{uis} = uid_NS(na)$
57. $e_{uis} = \{uid_E(e) | e: E \bullet e \in es\}$
58. $n_{uis} = \{uid_N(n) | n: N \bullet n \in ns\}$
59. $\sigma_{uis}: PI\text{-}set = \{uid_P(p) | p: P \bullet p \in \sigma\}$
59. $\sigma_{uis} = \{gi\} \cup \{ea_{uis}\} \cup \{na_{uis}\} \cup \{es_{uis}\} \cup \{ns_{uis}\} \cup e_{uis} \cup n_{uis}$

5.2.3 Uniqueness

60. No two of these are the same, i.e., part identifiers are unique.

axiom [Uniqueness of Part Identification]

60. $card\sigma = card\sigma_{uis}$

5.3 Mereology

Mereology has three facets: types, observers and wellformedness.

5.3.1 Mereology Types and Wellformedness, I

- 61. The mereology of a node is a non-empty set of edge identifiers.
- 62. The mereology of an edge is a set of two node identifiers.

type

- 61. $NM = EI\text{-set}$ **axiom** $\forall nm:NM \cdot \text{card } nm > 0$
- 62. $EM = NI\text{-set}$ **axiom** $\forall em:EM \cdot \text{card } em = 2$

5.3.2 Mereology Observers

value

- 61. **mereo_N**: $N \rightarrow NM$
- 62. **mereo_E**: $E \rightarrow EM$

5.3.3 Mereology Wellformedness, II

- 63. The unique identifiers of a node must be those of the edges of the graph.
- 64. The unique identifiers of an edge must be those of the nodes of the graph.

axiom [Graph Mereology Wellformedness]

- 63. $\forall n:N \cdot \text{mereo_N}(n) \subseteq es_{uis}$
- 64. $\forall e:E \cdot \text{mereo_E}(e) \subseteq ns_{uis}$

5.4 Paths of a Graph

65. A path (of a graph) is a finite⁵⁷ sequence of one or more alternating node and edge identifiers such that
- (a) neighbouring edge identifiers are those of the mereology of the “in-between” node, and such that neighbouring node identifiers are/is those of the mereology of the “in-between” edge;
 - (b) and node identifiers of a path are node identifiers of the graph,
 - (c) and its neighbouring edge identifier(s) are in the mereology of the identified node;
 - (d) and edge identifiers of a path are edge identifiers of the graph,
 - (e) and its neighbouring node identifier(s) are/is in the mereology of the identified edge;
 - (f) the kinds of the adjacent nodes and edges “fit”.
66. Given a node [an edge] identifier we can retrieve the identified node [edge].

type

65. Path = (EI|NI)*

axiom [Wellformed Paths]

65. $\forall \text{ path:Path} \cdot$

65a. $\forall \{i, i+1\} \subseteq \text{inds path} \Rightarrow$

65a. $(\text{is_NI}(\text{path}[i]) \wedge \text{is_EI}(\text{path}[i+1]))$

65a. $\vee \text{is_EI}(\text{path}[i]) \wedge \text{is_NI}(\text{path}[i+1]))$

65b. $\wedge (\text{path}[i] \in ns_{uis} \Rightarrow \text{path}[i+1] \in es_{uis})$

65c. $\wedge \text{uid_N}(\text{retr_node}(\text{path}[i])) \in \text{mereo_E}(\text{retr_node}(\text{path}[i]))$

65d. $\wedge (\text{path}[i] \in es_{uis} \Rightarrow \text{path}[i+1] \in ns_{uis})$

65e. $\wedge \text{uid_E}(\text{retr_edge}(\text{path}[i])) \in \text{mereo_N}(\text{retr_edge}(\text{path}[i]))$

65f. $\wedge \text{kind}(\text{retr_unit}(\text{path}[i])) \cap \text{kind}(\text{retr_unit}(\text{path}[i+1])) \neq \{\}$

value

66. retr_node: NI \rightarrow N, retr_edge: EI \rightarrow E, retr_unit: UI \rightarrow U

66. retr_node(ni) as n • n $\in ns \wedge \text{uid_}(n)=ni$

66. retr_edge(ei) as e • e $\in es \wedge \text{uid_}(e)=ei$

66. retr_unit(i) as u • u $\in ns \cup es \wedge \text{uid_U}(u)=i$

66. uid_U(u) $\equiv \text{is_E}(u) \rightarrow \text{uid_U}(u), \text{is_N}(u) \rightarrow \text{uid_N}(u)$

The above **pre/post** condition allows for circular paths, i.e., possibly infinite paths that may contain the same node or edge identifier more than once.

We can define a function that given a graph calculates all its non-circular paths.

⁵⁷We shall only consider finite paths. The paths function, Item 67 below, can easily be modified to yield also infinite length paths !

67. The paths^{58} function takes a graph and yields a possibly infinite set of paths – satisfying the above well-formedness criterion.

We define the paths function in two ways.

68. Either axiomatically

69. in terms of an **as** predicate, with the result being the “largest” such set all of whose paths satisfy the well-formedness criterion;

70. or inductively⁵⁹:

- (a) **basis clause**: every singleton path of either node or edge identifiers of the graph form a path.
- (b) **inductive clause**: If p_i and p_j are finite, respectively possibly infinite paths of the “result”, ps , such that
 - (c) paths $p_i \hat{\langle u_i \rangle}$ and $\langle u_j \rangle \hat{p_j}$ are in ps , and
 - (d) the resulting concatenated path is not circular, and
 - (e) the mereology of the last element of p_i identifies the first element of p_j ,
 - (f) then their concatenation is a path in ps .
- (g) **extremal clause**: No path is an element of the desired set of paths unless it is obtained from the basis and the inductive clause by a finite number of uses.

value

```

67. paths: G → Path-infset
68. paths(g) as ps
69.   such that: ∀ p:ps satisfy the above wellformedness
70. paths(g) ≡
70a.   let ps = {⟨ni⟩ | ni:NI ∈ nsuis} ∪ {⟨ei⟩ | ei:EI ∈ esuis}
70f.   ∪ { pi∘⟨ui⟩∘⟨uj⟩∘pj | pi∘⟨ui⟩:Path-set, ⟨uj⟩∘pj:Path-infset •
70b.   ∧ ({pi∘⟨ui⟩, ⟨uj⟩∘pj} ⊆ ps
70c.   ∧ (ui~ ∈ elems pj ∧ uj~ ∈ elems pi)
70e.   ∧ (ui ∈ mereo_U(retr_unit(uj))
70e.   ∧ uj ∈ mereo_U(retr_unit(ui))) } in
70g.   ps end
type
67. U = E|N

```

Solution to the equation, lines 70a–70c, is “obtained” by a smallest set fix-point reasoning.

71. Given a “global” graph, g , we can calculate a “similarly global” paths value:

value

```

71. paths:Path-set = paths(g)

```

With the notion of paths of a graph one can now examine whether

- a graph is strongly connected, that is, whether any node or edge can be “reached” from any other node or edge; or
- a graph consists of two or more sub-graphs, i.e., there are no edges between nodes in two such sub-graphs;
- etc.

In the next section, i.e., Sect. 5.5.1, we shall now endow nodes and edges to reflect whether they are road intersections, railway stations, harbours, and road links, railway lines, or canal/river/sea- or air-routes, etc.

⁵⁸ **Alarm!** Check that this function indeed generates only finite length paths !

⁵⁹ https://www.cs.odu.edu/~toida/nerzic/content/recursive_def/more_ex_rec_def.html

72. We can formulate a *theorem*: for every graph we have that every path, p , in g , also contains its reverse path, $\text{rev}(p)$ in g .

theorem: [All finite paths have finite reverse paths]

72. $\forall g:G, p:\text{Path} \cdot p \in \text{paths}(g) \Rightarrow \text{rev_path}(p) \in \text{paths}(g)$

value

72. $\text{rev_path}: P \rightarrow P$

72. $\text{rev_path}(p) \equiv$

72. **case** p **of**

72. $\langle \rangle \rightarrow \langle \rangle,$

72. $\langle ui \rangle \rightarrow \langle ui \rangle,$

72. $\langle ui \rangle^{\sim p'} \langle uj \rangle \rightarrow \langle uj \rangle^{\sim \text{rev_path}(p')} \langle ui \rangle$

72. **end**

We can define auxiliary functions, for example:

73. Given a kind we can select all the graph paths of that kind.

value

73. $\text{path_kind}: \text{Path} \rightarrow \text{Kind} \rightarrow \text{Path-set}$

73. $\text{path_kind}(p)(k) \text{ as } pks$

73. $\bullet pks \subseteq \text{paths} \wedge$

73. $\forall pk:\text{Path} \cdot pk \in pks \wedge \forall \text{elems } pk \cdot \text{kind}(\text{retr_unit}(pk)) \cap \{k\} \neq \{\}$

5.5 Attributes

With endurants now being endowed with, i.e., having attributes, graphs come to “look”, more-and-more, as transport nets !

Attributes has three facets: types, observers and wellformedness.

5.5.1 Attribute Types & Observers

We introduce but just a few Graph Attributes.

74. From a node we can thus observe the “kind” of node: whether “**road crossing**”, train “**station**”, canal/river/sea boat/ship “**harbour**”, and/or “**airport**” – one or more ! [A static attribute]

Edge:

75. From an edge we can thus observe the “kind” of edge: whether it represents a street (segment between two neighbouring road crossings), or a rail track (between two neighbouring stations), or a sea route between two neighbouring (canal/river/sea) harbours or an aircraft route between two neighbouring airports.
76. From an edge we can we can observe its length⁶⁰. [Static Attribute]
77. and the cost⁶¹ of using the edge⁶². [Static Attribute]

type

74. `NodeKind = Kind-set` **axiom** $\forall nk:NodeKind \cdot nk \neq \{\}$
75. `EdgeKind = Kind-set` **axiom** $\forall ek:EdgeKind \cdot \mathbf{card} \, ek = 1$
76. `LEN = Nat`
77. `COST = Nat`

value

74. `attr_NodeKind: N \rightarrow NodeKind`
75. `attr_Edgekind: E \rightarrow EdgeKind`
76. `attr_LEN: E \rightarrow LEN`
77. `attr_COST: E \rightarrow COST`

⁶⁰LEN is here “formalized” in terms of **Natural** numbers. Whether such lengths stand for mm, cm, m, km, inches, feet, yard, mile or other we presently leave unspecified.

⁶¹COST is here “formalized” in terms of **Natural** numbers. Whether such costs stand for \$, €, £, or other we presently leave unspecified.

⁶²See [18]. The usual arithmetic operators apply: scaling between ... Check also [57].

78. Given a node or an edge we can observe its kinds.
79. Given a graph, and a “kind”, we can calculate all its paths of the same kind.
80. Given a finite route we can we can calculate its lengths
81. and costs.
82. We can also calculate the shortest route, possibly a set, of a graph,
83. and the least costly,⁶³
84. etc.

value

```

78. kind: (E|N) → EdgeKind|NodeKind
78. kind{en} ≡ is_E(en)→attr_Edgekind(en), is_N→attr_Edgekind(en)

79. route_kind: G → Kind → Path-set
79. route_kind(g)(k) ≡
79.   { ⟨p[i]|i: Nat, p:P·p∈paths(p) ∧ 1≤i≤len(p) ∧ k∈kind(p[i])⟩ }

80. path_length: P → LEN
80. path_length(p) ≡
80.   case p of
80.     ⟨⟩ → 0
80.     ⟨ui⟩ → retr_path_length(ui),
80.     ⟨ui⟩^p' → retr_length(ui)+path_path_length(p')
80.   end
80. retr_path_length: UI → LEN
80. retr_path_length(ui) ≡ (is_EI(ui)→attr_LEN(retr_edge(ui)), is_NI(ui)→0)

81. path_cost: P → LEN
81. path_cost(p) ≡
81.   case p of
81.     ⟨⟩ → 0
81.     ⟨ui⟩ → retr_cost(ui),
81.     ⟨ui⟩^p' → retr_path_cost(ui)+path_cost(p')
81.   end

81. retr_path_cost: UI → COST
81. retr_path_cost(ui) ≡ (is_EI(ui)→attr_COST(retr_edge(ui)), is_NI(ui)→0)

82. shortest_route: G → P-set
82. shortest_route(g) ≡
82.   let ps = paths(g) in
82.   { p | p:P · retr_len(p) ∧ ∀ p':P·p'∈ps ∧ retr_path_len(p)≤retr_path_len(p') }
82.   end

84. etc.

```

The “etc.” covers such auxiliary functions as shortest route of a given kind , least costly route of a given kind , etc. !

More Graph Attributes will be added [“later”].

⁶³See William Cook’s Web page: https://www.math.uwaterloo.ca/tsp/index.html?mc_cid=a51d99f2aa&mc_eid=783b63461a and *Quanta Magazine*’s Fundamentals Computer Science Web page <https://mail.google.com/mail/u/0/?ui=2#inbox/FMfcgz-QZTzdWzqtRWmVWkQrcNzzDrSnJ>

5.5.2 Attribute Wellformedness

- 85. If a node is of some kind, then there must be at least one edge leading to/from it of the same kind.
- 86. If an edge is of some kind, then the nodes connected to it must also be of that [same] kind.
- 87. If a node is of kind other than "car", then there must be an edge "of" that node of kind "car". [One must be able to drive to stations, harbours and airports by car, taxi, lorry (truck) or bus !]

axiom

85.

85.

86.

86.

87.

87.

Chapter 6

Conveyors, I

We remind the reader that conveyors are either for the **road**: cars, taxis, trucks, buses, etc.; or for the **rail**: trains, or for the **sea**: sailboats, barges, freighters, passenger liners, etc.; or for the **air**: helicopters and airplanes.

6.1 Conveyor Endurant Sorts & Observers

88. From a conveyor aggregate one can observe a finite set of conveyors.

89. A conveyor is either a

- a road conveyor
 - car,
 - taxi,
 - bus,
 - truck, etc.,
- or a rail conveyor
 - passenger train,
 - freight train, etc.,
- or a water conveyor
 - sailboat,
 - barge,
 - fishing vessel,
- or an airborne conveyor
 - freighter,
 - passenger liner, etc.,
 - civil aircraft,
 - freight plane, or
 - passenger aircraft, etc.

90. Conveyors are atomic parts.

91. Conveyors or “of kind”.

92. Conveyor aggregates are uniquely identified.

93. Conveyors are uniquely identified.

type

88. **CS** = **C-set**

89. **C** = **Road**|**Rail**|**Water**|**Air**

89. **Road** = ...

89. **Rail** = ...

89. **Sea** = ...

89. **Air** = ...

92. **CAI**

93. **CI**

value

92. **uid_CA**: **CA** → **CAI**

93. **uid_C**: **C** → **CI**

6.2 Unique Identifiers

6.2.1 Unique Identifier State

94. The unique identifier of a conveyor aggregate contributes to the unique identifier state for the [entire] transport domain.
95. The unique identifiers of all conveyors contribute to the unique identifier state for the [entire] transport domain.
96. The overall unique identifier state, σ_{uis} , is therefore the union of all the unique identifiers of all parts of a transport domain.

value

94. $cai:CAI = \mathbf{uid_CA}(ca)$
95. $cis:CI\text{-}\mathbf{set} = \{ \mathbf{uid_C}(c) \mid c:C \cdot c \in \mathbf{obs_CS}(ca) \}$
96. $\sigma_{uis} = \sigma_p \cup \{cai\} \cup cis$

6.2.2 Uniqueness

97. All parts are uniquely identified.

axiom [All parts are uniquely identified]

97. $\mathbf{card} \sigma = \mathbf{card} \sigma_{uis}$

6.2.3 Conveyor Retrieval

98. From a conveyor identifier one can obtain, via cs , the conveyor of that identification.

value

98. $\mathbf{retr_conveyor}: CI \rightarrow C$
98. $\mathbf{retr_conveyor}(ci) \equiv \iota c:C \cdot c \in cs \wedge \mathbf{uid_C}(c)=vi$

6.3 Mereology

6.3.1 Mereology Types & Observers

99. The mereology of a conveyor is a finite set of edge and node identifiers that it may “visit”.⁶⁴

type

99. $CM = UI\text{-}\mathbf{set}$

value

99. $\mathbf{mereo_C}: C \rightarrow CM$

⁶⁴We shall extend this mereology in Sect. 14.1 on page 97.

6.3.2 Mereology Wellformedness

100. The identifiers of a conveyor mereology must be those of the edges and nodes of the transport graph, g .

101. The kind of conveyor must “fit” the kind of edges and nodes⁶⁵.

axiom [Conveyor Mereology of Right Kind]

100. $\forall c:C \cdot c \in cs \Rightarrow \forall ui:UI \cdot ui \in \mathbf{mereo_C}(c)$

100. $\Rightarrow ui \in e_{uis} \cup n_{uis}$

101. $\wedge c_kind(c) \cap kind(retr_unit(ui)) \neq \{\}$ 1101

6.4 Attributes

6.4.1 Conveyor Attribute Types & Observers

In this section we deal with some attributes. Further conveyor attributes are brought forward in Sect. 13.3.3 page 94.

102. Conveyors are of kind. [Static Attribute]

103. These routes must be of the kind of the conveyors traveling them !

104. Conveyors either stand still or move. That is, they have position in the graph, an index on the service route. Either the position is at a node, or somewhere, a fraction, f , of a distance along an edge, from one node to an adjacent. [Programmable Attribute]

105. The service route index must be commensurate with the conveyor position.

106. We omit further possible attributes: Speed, Acceleration, Weight,

type

102. Kind

104. CPos = AtNode | OnEdge

104. AtNode :: NI

104. OnEdge :: NI \times (F \times EI) \times NI

104. F = **Real** **axiom** $\forall f:F \cdot 0 < f < 1$

value

102. **attr_Kind**: C \rightarrow Kind

104. **attr_CPos**: C \rightarrow CPos

106. ...

axiom [Routes of commensurate kind]

103. $\forall c:C \cdot \mathbf{let} \ ps = \mathbf{attr_Routes}(c) \mathbf{in} \forall p:\mathbf{Path} \cdot p \in ps \wedge ps \subseteq \mathbf{path_kind}(p)(kind(c)) \mathbf{end}$

⁶⁵Cars, Taxis, Buses, Trucks move along edges and nodes of kind **road** [a literal value, like **true** and **false** are literal values], Passenger and Freight Trains move along edges and nodes of kind **rail** [a literal value], Sail Boats, Barges, Fishing Vessels, Ferries, Freighters, Ferries and Passenger Liners move along edges and nodes of kind **sea** [a literal value] and Private Aircraft, Helicopters, Freight Planes and Passenger Aircraft move along edges and nodes of kind **air** [a literal value].

6.4.2 Routes

107. The following properties hold of any route:

- (a) the current route of a conveyor must always be in the routes of that conveyor.
- (b) The static attribute `Routes` must all start and end with a node identifier.
- (c) When initialized, a conveyor “starts” with a `CurrentRoute` chosen from the `Routes`.
- (d) At any moment a conveyor moves along a [programmable attribute] *current* route.
- (e) When moving from an edge to a node the current route is shortened by one.
- (f) When a route is thereby exhausted, i.e., $\langle \rangle$, the conveyor may decide to select a new route.
- (g) It does so from the static attribute `Routes`.
 - i. The previous, exhausted route ended with a node identifier.
 - ii. The next, to be current, route must start with that node identifier.

axiom [Commensurable Routes]

107. $\forall c:C, r:Routes, cr:CurrRoute \bullet r = attr_Routes(c) \wedge cr = attr_CurrRoute(c)$

107a. $cr \in r$

107b. $\wedge is_NI(hd\ r) \wedge is_NI(r[len\ r])$

For *cars* the `Routes` attribute may exclude certain paths, for example such toll-roads for which they have no license. When, for example, *buses*, *trains*, *ferries* and *passenger aircraft*, the routes are such that for every pat there is at least one path that “connects” to the former: ends, respectively starts with identical node identifiers. Usually the set of routes contains just two paths: one from node n_i to node n_j and the other from node n_j to node n_i . And so forth !

6.4.3 Conveyor Attribute Wellformedness

TO BE WRITTEN

Chapter 7

Intentional Pull, I

7.1 History Attributes

History attributes record when conveyors (cars, trains, boats and aircraft) were where and at which times. They are chronologically ordered, time-stamped sequences of event notices. History attributes are programmable.

History attributes “record” events. Conveyors, as controlled by, say humans, may not note down these **events**, and edges and nodes, which we in some sense consider innate⁶⁶, “most likely” do not notice them.

But we, “us”, humans, can speak about and recall [these, and “other”⁶⁷] events – and they are therefore an essential aspect of modelling any manifest domain.

108. We “lump” nodes and edges into single element ways [i.e., endurants].

109. The ordered, TIME ⁶⁸-stamped, history attribute event notices record the vehicles, by their unique identifiers.

110. The ordered, TIME -stamped, conveyor history attribute event notices record the ways, by their unique identifiers.

type

108. $W = N|E$

108. $WI = NI|EI$

109. $WHist = (s_t:\text{TIME} \times VI)^*$

110. $ConvHist = (s_t:\text{TIME} \times CI)^*$

value

108. $\text{retr}_W: WI \rightarrow N|E$

108. $\text{retr}_W(wi) \equiv ! \ w:W \cdot w \in nsUes \wedge \text{uid}_W(w)=wi$

109. $\text{attr}_WHist: W \rightarrow WHist$

109. $\text{attr}_ConvHist: C \rightarrow ConvHist \quad \text{!109}$

axiom [Ordered Way and Conveyor Histories]

109. $\forall \text{wh}:WHist \cdot \{i,i+1\} \subseteq \text{inds wh} \Rightarrow s_t(rh[i]) < s_t(wh[i+1])$

110. $\forall \text{ch}:ConvHist \cdot \{i,i+1\} \subseteq \text{inds ch} \Rightarrow s_t(ch[i]) < s_t(ch[i+1])$

⁶⁶An innate quality or ability is one that you were born with, not one you have learned. That is: we consider edges and nodes to be innate wrt. observing and recording the where-about events of conveyors – other than indirectly through the space they “occupy”, the possible wear & tear of the road surface or rail track, or possible pollution of the sea and air, etc.

⁶⁷By the seemingly cryptic “other” events, we may, in the context of transport, think of such events as *conveyor breakdown*, *edge collapse*, etc.

⁶⁸ TIME is a “global” phenomenon.

We say *15:23 June 23, 2025 CET*, and mean that it is now *23 minutes past 3pm, 25th of February 2025, Central European Time*.

TI stands for time-interval.

We say *3 hours and 23 minutes*.

7.2 An Intentional Pull

Nodes and edges are intended to “carry” traffic [only] in the form of vehicles, and vehicles are intended to move along [only] ways, i.e., nodes and edges.

111. for all conveyors (of a transport) if

- (a) a conveyor is said to be on a way, i.e, at a node [resp. on an edge], at time τ ,
- (b) then that way must “carry” that conveyor
- (c) at exactly that same time;

112. and vice-versa, if-and-only-if, for all ways

- (a) a way is said to “carry” a conveyor at time τ ,
- (b) then that conveyor must be on that way
- (c) at exactly that same time.

Intentional Pull:

```

111.   $\forall c:C \cdot c \in cs \cdot$ 
111a.  let  $ch:CH = attr\_CH(c)$  in
111a.   $\exists ! i:Nat \cdot i \in inds\ ch \cdot$ 
111a.  let  $(\tau, wi) = ch[i]$  in
111b.  let  $wh:WH = attr\_WH(retr\_way(wi))$  in
111c.   $\exists ! j:Nat \cdot j \in inds\ WH \cdot s\_t(wh[j]) = \tau$ 
111.  end end end
112.   $\equiv$ 
112.   $\forall w:W \cdot w \in es \cup ns \cdot$ 
112a.  let  $wh = attr\_WH(w)$  in
112a.   $\exists ! k:Nat \cdot k \in inds\ wh \cdot$ 
112a.  let  $(\tau, ci) = wh[k]$  in
112b.  let  $ch:CH = attr\_WH(retr\_conveyor(ci))$  in
112c.   $\exists \ell:Nat \cdot \ell \in inds\ ch \cdot s\_t(ch[\ell]) = \tau$ 
112.  end end end

```

Chapter 8

Single-mode Transport Behaviours

The previous sections, Sects. 4–7, studied, analyzed & described a transport domain syntactically, that is: its manifest forms and properties, but not its meaning, i.e., semantics. This sections is about that: the “meaning”, so-to-speak, of endurants. This will be done by **transcendentally deducing** *behaviours* and *actions* from the description of endurants. Endurants are **transcendentally deduced** into behaviours, and described as *s* with arguments. Their internal properties are **transcendentally deduced** into arguments of these behaviours. We choose to only endow edges, nodes and conveyors with behaviours. Behaviours synchronize and communicate via “the ether” – here RSL/CSP-modeled as a **channel** array that allows conveyor, node and edge behaviours (u_i, u_j, u_k) to cooperate !

8.1 Communication

8.1.1 Communication Medium

113. There is a “global” communication, i.e., behaviour interaction medium, **comm**. It allows transport Behaviours to synchronize and exchange information of type *M*.

channel

113. **comm**[$\{i, j\} \mid i, j : \text{UI} \bullet \{i, j\} \in \text{uis} \]$] MSG

8.1.2 Communication Causes

114. A conveyor, $ci : \text{CI}$, at a node decides to remain at that node.
115. A conveyor, $ci : \text{CI}$, at a node decides to change route.
116. A conveyor, $ci : \text{CI}$, at a node decides to leave the node, and
117. to enter an edge.
118. A conveyor, $ci : \text{CI}$, on an edge decides to move on.
119. A conveyor, $ci : \text{CI}$, on an edge decides to leave that edge, and
120. to enter the node.
121. And a conveyor, $ci : \text{CI}$, at a node or on an edge may decide, “surreptitiously” or otherwise, to just **stop**.

8.1.3 Communication Messages

122. The message is simple: a time stamp and the identity of a node, an edge or a conveyor.

type

122. $\text{MSG} = \text{TIME} \times (\text{NI} \mid \text{EI} \mid \text{CI})$

8.2 Behaviours

So we model conveyor, node and edge behaviours. Each of these behaviour functions has arguments of the following kind:

- a **unique identifier**, never changes, distinguishes between multiple instances of edges, or nodes, or conveyors;
- a *mereology*; and
- **attributes**:
 - **static attributes**, i.e., attributes whose value never changes;
 - **monitorable attributes**, i.e., attributes whose value changes “at their own volition”: itself nor cooperating behaviours cannot influence their value – we shall not consider monitorable attributes in this study; and
 - **programmable values**, i.e., attributes whose value may be changed by the behaviour – i.e., acts like variables that can be read and updated !

Each of these behaviours are modelled as processes that may “go-on-and-on-forever” – modelled in terms of *tail-recursion* – modelled also in the specifying **Unit** as part, “the last”, of the behaviour signature.

8.3 Behaviour Signatures

123. We present the conveyor, edge and node behaviour signatures.

value

```

123. conveyor: CI→CM→(Kind×Routes)→(CurrRoute×CPos×CH)→Unit
123. edge: EI→EM→(Kind×LEN×...)→NH→Unit
123. node: NI→NM→(Kind-set×...)→NH→Unit

```

8.4 Behaviour Definitions

8.4.1 Conveyor Behaviours

- A conveyor alternates between being at a node or on edge, so its behaviour is defined in terms of “either” and their “progress” onto “the other” !

• CONVEYOR Behaviour AT A NODE:

124. A conveyor at a node either

- (a) changes its current route, and choose another, the next current route, or
- (b) remains at that node, idling, or circling around, or
- (c) is entering an edge, or
- (d) **stops** at that node, i.e., leaves the transport altogether.

value

```

124. conveyor(ci)(cm)(k,routes)(cr,AtNode(ni),ch) ≡
124a. conveyor_change_route(ci)(cm)(k,routes)(cr,AtNode(ni),ch)
124b. || conveyor_remains_at_node(ci)(cm)(k,routes)(cr,AtNode(ni),ch)
124c. || conveyor_enters_edge(ci)(cm)(k,routes)(cr,AtNode(ni),ch)
124d. || conveyor_stops_at_node(ci)(cm)(k,routes)(cr,AtNode(ni),ch)

```

• CONVEYOR **Actions** AT A NODE:

125. A conveyor may non-deterministically decide to **change** its current route at a node

- (a) at time τ ,
- (b) selects of next, to be, current route from routes such that that the chosen route begins with the node being otherwise left,
- (c) so informing the node, and
- (d) updates its history,
- (e) whereupon it resumes being a conveyor with both updated current route and history.

```

125. conveyor_change_route(ci)(cm)(k,routes)(cr,AtNode(ni),ch) ≡
125a.   let  $\tau = \text{record\_TIME}()$ ,
125b.   ncr = select_next_route(ni,routes),
125d.    $ch' = \langle(\tau,ni)\rangle^{ch}$  in
125c.   comm[{ci,ni}] ! ( $\tau,ci$ ) ;
125e.   conveyor_at_node(ci)(cm)(k,routes)(ncr,AtNode(ni),ch') end

125b. selects_next_route:NI × Routes → CurrRoute
125b. selects_next_route(ni,routes) as ncr • ncr ∈ routes ∧ hd ncr = ni

```

126. A conveyor **remains** at a node

- (a) at some time, τ ,
- (b) which is to be noted by the node behaviour ni
- (c) whereupon the conveyor resumes being a conveyor except now with an updated conveyor history, ch .

value

```

126. conveyor_remains_at_node(ci)(cm)(k,routes)(cr,AtNode(ni),ch) ≡
126a.   let  $\tau = \text{record\_TIME}()$  in
126b.   comm[{ci,ni}] ! ( $\tau,ci$ );
126c.   conveyor(ci)(cm)(k,routes)(cr,AtNode(ni), $\langle(\tau,ni)\rangle^{ch}$ ) end

```

127. A conveyor at a node may non-deterministically choose to leave a node and **enter** an edge

- (a) at some time, τ , and as determined by the current route's next element, enters that route, i.e., edge,
- (b) which is to be noted by the node and designated edge behaviours ni ,
- (c) updates its position
- (d) and its history accordingly, and
- (e) resumes being a conveyor on an edge.

value

```

127. conveyor_enters_edge(ci)(cm)(k,routes)(cr,AtNode(ni),ch) ≡
127a.   let  $\tau = \text{record\_TIME}()$  in
127b.   ( comm[{ci,ni}] ! ( $\tau,ni$ ) || comm[{ci,ni}] ! ( $\tau,hd\ cr$ ) ) ;
127c.   let ei = hd cr in let {ni,ni'} = mereo.E(retr_edge(ei)(es)) in
127c.   let cpos = onEdge(hd cr,(ei,(ni,f,ni),ni')) in
127e.   conveyor(ci)(cm)(k,routes)(cr,cpos, $\langle(\tau,ni)\rangle^{ch}$ ) end end end end

```

128. And a conveyor may non-deterministically choose to abandon being a conveyor, i.e., leaving transport altogether – **stopping** !
129. But first it notifies the node at which it stops.

value

```

128. conveyor_stops_at_node(ci)(cm)(k,routes)(cr,AtNode(ni),ch) ≡
129.   let  $\tau = \text{record\_TIME}()$  in
129.   comm[{ci,ni}] ! ( $\tau$ ,ci) ;
128.   stop end

```

- A conveyor behaviour on an edge alternates.

• **CONVEYOR Behaviour ON EDGE**

130. An edge [behaviour] at an edge external non-deterministically either:

- (a) **moves** along the edge, a fraction “at a time”,
- (b) **stops** on the edge and thereby “leaves” transport; or
- (c) **enters** a node.

```

130. conveyor(ci)(cm)(k,routes)(cr,OnEdge(nuif, (f,e), nuit), ch) ≡
130a.   conveyor_moves_on_edge(ci)(cm)(k,routes)(cr,OnEdge(nuif, (f,e), nuit), ch)
130c.   || conveyor_stops_on_edge(ci)(cm)(k,routes)(cr,OnEdge(nuif, (f,e), nuit), ch)
130b.   || conveyor_enters_node(ci)(cm)(k,routes)(cr,OnEdge(nuif, (f,e), nuit), ch)

```

• **CONVEYOR Actions ON AN EDGE:**

131. A conveyor **moving** along an edge

- (a) at time τ is modelled by
- (b) incrementing the fraction of its position
- (c) (while updating its history)
- (d) notifying the edge [behaviour]
- (e) [technically speaking] adjusting its position, and, finally,
- (f) resuming being a thus updated conveyor [OnEdge]

value

```

131. conveyor_moves_on_edge(ci)(cm)(k,routes)(cr,OnEdge(nuif, (f,e), nuit), ch) ≡
131a.   let  $\tau = \text{record\_TIME}()$ ,
131b.    $\varepsilon : \text{Real} \cdot 0 < \varepsilon \ll 1$  in
131b.   let  $f' = f + \varepsilon$ ,
131d.   cpos = OnEdge(nuif, (f',e), nuit) in
131c.   let  $ch' = \langle (\tau, ci) \rangle^{\sim} ch$  in
131e.   comm[{ci,ej}] ! ( $\tau$ ,ci) ;
131f.   conveyor(ci)(cm)(k,routes)(cr,cpos,ch') end end end
131. pre hd cr = nuif

```

132. A conveyor **enters** a node

- (a) at time τ is modelled by altering its position
- (b) notifying both the edge and designated node behaviours
- (c) resumes being an updated conveyor behaviour.

value

```

132. conveyor_enters_node(ci)(cm)(k,routes)(cr,OnEdge(nuif, (f,ei),nuit),ch) ≡
132a.   let  $\tau = \text{record\_TIME}()$ , cpos = AtNode(hd cr) in
132b.   ( comm[{ci,ei}] ! ( $\tau$ ,ci) || comm[{ci,nuit}] ! ( $\tau$ ,ci) ) ;
132c.   conveyor(ci)(cm)(k,routes)(tl cr,cpos,⟨( $\tau$ ,ci)⟩ch) end
132.   pre hd cr = nuif

```

133. A conveyor may non-deterministically choose to abandon being a conveyor, i.e., leaving transport altogether – **stopping** !

134. But first it notifies the edge at which it stops.

value

```

133. conveyor_stops_on_edge(ci)(cm)(k,routes)(cr,OnEdge(nuif, (f,e),nuit),ch) ≡
134.   let  $\tau = \text{record\_TIME}()$  in
134.   comm[{ci,ej}] ! ( $\tau$ ,ci) ;
133.   stop end
133.   pre hd cr = nuif

```

8.4.2 Node Behaviour

135. **Node** [behaviours]

- (a) external non-deterministically accept conveyor, ci, actions
- (b) at times τ
- (c) augment their histories accordingly and
- (d) resumes being node behaviours.

value

```

135. node: NI → NM → (NodeKind×...) → NH Unit
135a. node(ni)(nm)(nk,...)(nh) ≡
135c.   let msg = [] { comm[{ni,ci}] ? | ci:CI • ci ∈ nm } in
135d.   node(ni)(nm)(...)(⟨msg⟩nh) end

```

8.4.3 Edge Behaviour

136. **Edge** [behaviours] – similarly,

- (a) external non-deterministically, accept conveyor, ci, actions
- (b) augment their histories accordingly and
- (c) resumes being edge behaviours.

value

```

136. edge: EI → EM → (EdgeKind×LEN×COST×...) → EH Unit
136a. edge(ei)(em)(len,cost,...)(eh) ≡
136b.   let msg = [] { comm[{ei,ci}] ? | ci:CI • ci ∈ em } in
136c.   edge(ni)(em)(len,cost,...)(⟨msg⟩eh) end

```

8.5 Domain Instantiation

By domain initialization we mean the *invocation*⁶⁹ of all behaviours.

137. The overall initialization expresses the parallel composition of the initialization of

138. all conveyors,

139. all nodes and

140. all edges.

```

137. initialization: Unit → Unit
137. initialization() ≡ t
138.   || { conveyor
138.       (uid_C(c))
138.       (mereo_C(c))
138.       (attr_KindC(c), attr_RoutesC(c)) [Static Attrs.]
138.   [Programmable Attrs.] (attr_CurrRouteC(c), attr_CPoC(c)s, attr_CHC(c))
138.       | c:C•c ∈ cs}
139.   || || { edge
139.       (uid_E(e))
139.       (mereo_E(e))
139.       (attr_EdgeKind(e), ...) [Static Attrs.]
139.   [Programmable Attrs.] (attr_(e), attr_EH(e))
139.       | e:E•e ∈ es }
140.   || || { node
140.       (uid_N(n))
140.       (mereo_N(n))
140.       (attr_NodeKinds(n)) [Static Attrs.]
140.   [Programmable Attrs.] (attr_NH(n))
140.       | n:N•n ∈ ns}

```

But: the initializaton of conveyors is too simplified: To capture an essence of transport it seems reasonable to distinguish between the various kinds of conveyors.

Thus the initialization of conveyors “really” amounts to the initialization of all

- | | | |
|-------------------------------|-------------------------------|-----------------------|
| • cars, trucks, taxis, | • sailboats, barges, vessels, | • freight planes and |
| • buses, | • passenger liners, ferries, | |
| • passenger & freight trains, | • civil aircraft, | • passenger aircraft. |

⁶⁹Invocation – in the colloquial – “call”

Part IV

A MULTI-MODE TRANSPORT: ENDURANTS

Chapter 9

Multi-mode Transport

The domain description of Chapters 5–8 was for single-mode transport: It focused on transport nets and conveyors. For a model of *multi-mode transport* we suggest to introduce:

- **Merchandise:** By merchandise we shall here understand a wider concept than usually thought of. To us *merchandise* is what customers wishes to and actually send and receive: *goods*, if You will, that have weight, volume and value. Could be a car, a book, 10.000 barrels of oil, etc. Merchandise is treated in Sect. 11.
- **Customers:** A [multi-mode transport] *customer* is either, if persons, wishes to travel from one place to another, or if otherwise wishes to send merchandise from one place, e.g., the customer's place, e.g. a node or an edge, to be received by a *recipient* at that another place. In the latter case customers are persons, businesses, organizations, or other, i.e., are *senders* or *receiver*, i.e., *recipients*. Customers are treated in Sect. 12.
- **Conveyor Companies:** A *conveyor company* is a business which manages a fleet of conveyors: trucks, freight trains freighters (i.e., vessels) and freight aircraft. Conveyor Companies are treated in Sect. 13.
- **Logistics Companies:** A transport logistics company handles requests from *senders* of *passengers* or *goods* (containers, oil, coal, gas, grain, salt, cars, machinery, etc.) to have these conveyed from one node to another, world-wide, by whatever means of combinations of conveyors and routes. A logistics company thus is a company which arranges for transport of merchandise. To do so logistics firms have access to the transport offerings of a number of, not necessarily all, conveyor companies: their routes, timetable and costs. Logistics Companies are treated in Sect. 15.
- **“Overall Top” Transport Endurants:** The *graph*, *conveyors*, *merchandise*, *customers*, *conveyor companies* and *logistics companies* form the transport domain. As a whole they are defined in Sect. 10.

After these sections we

- outline an **intentional pull** for multi-mode domains, Sect. 16,
- summarize the syntax of multi-mode transport **commands**, Sect. 17,
- and cover multi-mode transport **behaviours**, Sect. 19.

• • •

To obtain the services of merchandise transport comes at a **price**, the *cost*.

The notion of *cost* is related to the notion of **cash**. It costs to have merchandise transported. Customers shall pay costs. Say, in the form of cash⁷⁰. Costs shall be modelled as integers. They are attributes of merchandise, customers, conveyor companies and logistics companies.

You may very well think of cash as manifest, i.e., as endurant parts. But in the context of transport we can abstract from that. If we were to model cash as endurants, then were we to model it as atomic or composite? Now we avoid such questions!

⁷⁰— or through withdrawal from bank accounts, or other. See [28].

Chapter 10

“Top” Transport Endurants

10.1 The Endurants – External Qualities

10.1.1 A Transport Taxonomy

We refer to Fig. 10.1 for a taxonomy of the transport domain.

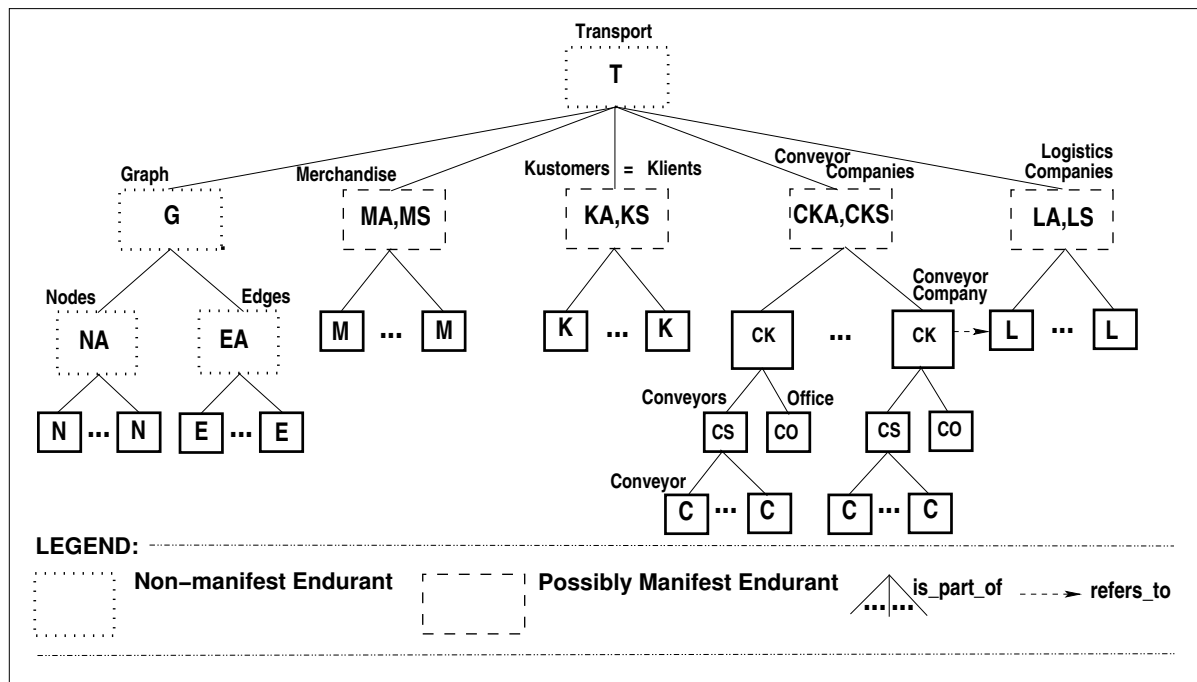


Figure 10.1: A Transport Domain Taxonomy

The “downwards” slanted lines express that the “lower” part is part of the “upper” part.

The “horizontal arrow” expresses that the source part embeds to “arrow” part. [Only one is illustrated; more could!]

10.1.2 An Overview of The Endurants

The Transport Domain

141. There is given the domain of interest, i.e., the universe of discourse, T .

type

141. T

value

141. $t:T$

Graphs

Graphs were treated in Sect. 5.

142. In a transport domain can observe the *transport net*, i.e., a *graph*, $g:G$.

143. From a graph we can observe a node aggregate,

144. and an edge aggregate.

145. From a node aggregate we can observe a set of nodes.

146. From an edge aggregate we can observe a set of edges.

type

142. G

143. NA

144. EA

145. $NS = \text{N-set}$

146. $ES = \text{E-set}$

145. N

146. E

value

142. $\text{obs}_G: T \rightarrow G$

143. $\text{obs}_{NA}: G \rightarrow NA$

144. $\text{obs}_{EA}: G \rightarrow EA$

145. $\text{obs}_{NS}: NA \rightarrow NA$

146. $\text{obs}_{ES}: EA \rightarrow ES$

And likewise for the unique identification of the manifest of these endurants.

type

142. GI

143. NAI

144. EAI

145. NO

146. EI

value

142. $\text{uid}_G: G \rightarrow GI$

143. $\text{uid}_{NA}: G \rightarrow NAI$

144. $\text{uid}_{EA}: G \rightarrow EAI$

145. $\text{uid}_N: N \rightarrow NI$

146. $\text{uid}_E: E \rightarrow EI$

Merchandise

Merchandise is treated in Sect. 11.

147. From a transport domain we can observe a *merchandise aggregate*, $\text{ma}:\text{MA}$;

148. and from a *merchandise aggregate* we can observe the set, $\text{ms}:\text{MS}$ of merchandise.

And likewise for the unique identification of the manifest of these endurants.

type	value
147. MA	147. obs _{MA} : $G \rightarrow \text{MA}$
148. $\text{MS} = \text{M-set}$	148. obs _{MS} : $\text{MA} \rightarrow \text{MS}$

type	value
147. MAI	147. uid _{MA} : $\text{MA} \rightarrow \text{MAI}$
148. MI	148. uid _M : $\text{M} \rightarrow \text{MI}$

Customers

Customers are treated in Sect. 12.

149. From a transport domain we can observe a “*k*” *customers aggregate*, $\text{ka}:\text{KA}$;

150. and from a *customer aggregate* we can observe the set, $\text{ks}:\text{KS}$ of customers.

151. We can speak of the set, ks , of all customers of a transport domain.

And likewise for the unique identification of the manifest of these endurants.

type	type
149. KA	149. KAI
150. $\text{KS} = \text{K-set}$	150. KI
value	value
149. obs _{KA} : $T \rightarrow \text{KA}$	149. uid _{KA} : $\text{KA} \rightarrow \text{KAI}$
150. obs _{KS} : $\text{KA} \rightarrow \text{KS}$	150. uid _K : $\text{K} \rightarrow \text{KI}$
	151. $\text{ks}:\text{K-set} = \text{obs}_{\text{KS}}(\text{obs}_{\text{KA}}(t))$

Conveyor Companies & Conveyors

Conveyors were treated in Sect. 6 and **Conveyor Companies** are treated in Sect. 13.

152. In a *transport domain*, $t:T$, we can observe the composite endurant of *conveyor companies aggregate*, $cca:CCA$.

153. From a *conveyor companies aggregate*, $cca:CCA$, we can observe a set, $cks:CKS$, of *conveyor companies*.

154. *Conveyor companies* are considered atomic.

From a *conveyor company*, $ck:CK$, we can observe

155. a *conveyor aggregate*, $ca:CA$,

156. and, from that, a *conveyor set*, $cs:CS$, which is a set of *conveyors*.

From a *conveyor company*, $ck:CK$, we can also observe

157. we can observe an atomic *conveyor company office*, $co:CO$,

158. and an atomic, optional *logistics subsidiary*, $ol:oL$, i.e., the conveyor company may operate its own *logistics company*.

type	value
152. CKA	152. obs_CKA : $T \rightarrow CKA$
153. $CKS = CK\text{-}set$	153. obs_CKS : $CKA \rightarrow CKS$
154. CK	155. obs_CA : $CK \rightarrow CA$
155. CA	156. obs_CS : $CA \rightarrow CS$
156. $CS = C\text{-}set$	157. obs_CO : $CK \rightarrow CO$
157. CO	158. obs_oL : $CK \rightarrow oL$
158. $oL = LI \mid nil$	

And likewise for the unique identification of the manifest of these endurants.

type	value
152. CKAI	152. uid_CKA : $CKA \rightarrow CKAI$
154. CKI	154. uid_CK : $CK \rightarrow CKI$
155. CAI	155. uid_CA : $CK \rightarrow CAI$
157. COI	157. uid_CO : $CK \rightarrow COI$
158. $oLI = LI \mid nil$	158. uid_oL : $CK \rightarrow oLI$

• • •

We shall, in the following, not treat the concepts of *conveyor [company] offices* and *the logistics company* parts of *conveyor companies*. We shall also not treat the concepts of *conveyor aggregates* and *conveyor sets*, but will treat the concept of *conveyors*.

Logistics Companies

Logistics Companies are treated in Sect. 15.

159. From a transport domain we can observe a *logistics companies aggregate*;

160. and from a *logistics companies aggregate* we can observe the set, $1s:LS$ of *logistics companies*.

And likewise for the unique identification of the manifest of these endurants.

type	
159.	LA
160.	LS = L-set
value	
159.	obs_{LA} : $T \rightarrow LA$
160.	obs_{LS} : $LA \rightarrow LS$

type	
159.	LAI
160.	LI
value	
159.	uid_{LA} : $LA \rightarrow LAI$
160.	uid_L : $LA \rightarrow LS$

Node and Edges were first treated in Sect. 5. To this we now add a widened understanding of their mereologies and attributes.

161. The mereology of nodes is a pair of the set identifiers of edges imminent upon the nodes and the set of identifiers of the customers and conveyors that can deposit merchandises “on hold” at the nodes.

162. The mereology of nodes is a pair of the set identifiers of [the pair of] nodes “at ether end of the edge” and the set of identifiers of conveyors that may travel along the edge.

Nodes and edges have the following attributes:

- (a) Nodes have merchandises “on hold” – by contract number,
- (b) and nodes have node histories: time-stamped events of which conveyors notified their presence at the node.
- (c) Edges have length,
- (d) cost of travel,
- (e) and event histories:: time-stamped events of which conveyors notified their presence at the edge.

type	
161.	$NM = EI\text{-set} \times (KI VI)\text{-set}$
162.	$EM = HI\text{-set} \times VI\text{-set}$
162a.	$OnHold = ContractNu \xrightarrow{m} M\text{-set}$
162b.	$NHist = (TIME \times CI)^*$
162c.	LEN
162d.	COST
162e.	$EHist = (TIME \times CI)^*$
value	
161.	mereo_N : $N \rightarrow NM$
162.	mereo_E : $E \rightarrow EM$
162a.	attr_{OnHold} : $N \rightarrow OnHold$
162b.	attr_{NHist} : $N \rightarrow NHist$
162c.	attr_{LEN} : $E \rightarrow LEN$
162d.	attr_{COST} : $E \rightarrow COST$
162e.	attr_{EHist} : $E \rightarrow EHist$

• • •

Atomic Parts:

163. Nodes, edges, merchandise, “k”ustomers, conveyors, conveyor company offices, and logistics firms are considered atomic.

type	
163.	N, E, M, K, C, CO, L

We shall not [really] consider conveyor offices and logistics firms in this report.

10.2 On Internal Qualities.

We discuss which endurants may be considered manifest. That is, to which of the parts – as, for example, shown by the boxes of Fig. 10.1 on page 75 – one might associate internal qualities, say in preparation for their part behaviours.

- With the *transport* part, $t:T$, we might – here rather loosely – associate a *ministry of transport*, or ...; We shall omit such associations.
- With the *graph* part, $g:G$, we might associate various other public (or private) institutions: *ministry of roads*, *ministry of railways*, *ministry of shipping*, and “*ministry of air*” ! We shall omit such associations.
- With the *merchandise* part one might associate some institution of *consumer protection* or other. We shall omit such associations.
- With the *customer* (*client*, *consumer*) part one might associate some kind of institutions. We shall omit such associations.
- With the *conveyor company* part one might associate some *conveyor association*. We shall omit such associations.
- With the *logistics companies* part one might similarly associate some associations. We shall omit such associations.
- With nodes, edges, merchandise, customers [clients], conveyor sets and conveyor offices we have and shall associate internal qualities – in respective sections 5, 6 and 11, 12, 13 and 15.

So we shall not elaborate on any internal qualities of the “top-level” endurants, that is those of T , G , NA , EA , MA , KA , CCA , and LA . But we shall, later, in indicated sections, elaborate on internal qualities of the “next-level” endurants, i.e., those of M , K , CK , CS , CO and L [Sects. 11, 12, 13 and 15] – as we already have for N , E and C [Sects. 5 and 6].

Figure 10.1 on page 75 hints at manifest, possibly manifest and non-manifest parts.

10.3 Conveyor Companies versus Logistics Companies.

Is it really necessary to distinguish between the two: conveyor and logistics companies? Examples of the two are:

- **conveyor companies:** Maersk⁷¹, DSV⁷² SAS, American Airlines, British Air, Deutsche Bahn, SNCF, Amtrack, Arriva, Greyhound, P&O, Dachser⁷³, etc.
- **logistics companies:** TUI, Expedia, etc.⁷⁴

As You may have deduced from the examples: some of the conveyor companies also operate “own” logistics departments, i.e., companies. But their functions must be separated: Conveyor companies fundamentally operate conveyors, and, only as a necessity, embody logistics departments – which basically only handle only their “mother”, i.e., the conveyor company’s own conveyors. Logistic companies, in general, make use of several conveyor companies.

10.4 Financial Matters

Transport implies expenses. *Cost* and *payment* of conveyance, is implied, but we have chosen to omit modelling these facets. Both conveyor and logistics companies rely on *creating*, *writing/editing*, *reading*, *copying* and *destroying documents*. The implied *double bookkeeping* will also not be modelled. These financial facets are not an essence, so we have decided, of the core aspects of transport. We refer to [28, 29] and [25], respectively, for treatments of these three domains.

⁷¹Maersk, Danish, is one of the world’s largest container shipping lines.

⁷²DSV, Danish, is one of the world’s largest trucking companies.

⁷³<https://www.dachser.dk/da/>

⁷⁴Yes, it has not gone unnoticed, that these “travel agencies” are, indeed, logistics companies – when seen from inside the daily operations of these. Also: I find it difficult to find conveyor companies that do not have a logistics [sub-office] !

Chapter 11

Merchandise

We shall use the term *merchandise* as a common denominator for “all that can be transported” ! living species: people⁷⁵, animals, plants, wheat, etc.; solid materials: iron ore, automobiles, timber, etc.; fluid materials: oil, gas, water, etc. Perhaps a better term would/should have been *goods*

11.1 Merchandise Endurants

11.1.1 External Qualities

164. There is the atomic endurant: merchandise.

type

164. M

value

164. $m:M$

⁷⁵Please do not be confused: No, we do not refer to people as slaves !

11.1.2 Internal Qualities

We lump the presentation of identification, mereology and attributes of merchandises into one, the present, section.

Unique Identifiers:

165. Merchandises have unique identification. [That is: no two items of merchandise have the same identification, and these are distinct from the identification of all other parts of the transport domain.]

Mereology:

166. The mereology of any [item or piece of] merchandise is the set of customers and conveyors that may possess or transport that merchandise.

Attributes:

167. Merchandises have practical identification: names, manufacture, place of origin, etc. Two or more merchandise may have the same such identification.
168. Merchandises have current position – a programmable attributes
169. Merchandises have size, approximate height, width and depth.
170. Merchandises have weight.
171. Merchandises have cost.
172. Merchandises have flammability.
173. Merchandises may be insured.
174. Merchandises have a history: an chronologically descending, ordered sequence of event notes:
175. Events are either ...
176. Et cetera ...

type

Unique Identifiers:

165. MI

Mereology:

166. $MM = KI\text{-}set \times CI\text{-}set$

Attributes:

167. $MId = Name \times Mfg \times Origin \times \dots$

168. $Position = (NI \times (F \times EI) \times NI) \mid NI \mid CI$

169. $Size = Nat \times Nat \times Nat$

170. $Weight = Real$

171. $Cost = Nat$

172. $Flammability = "flammable" \mid "inflammable" \mid "combustible" \mid \dots$

173. Insurance

174. $MHist = (TIME \times Event)^*$

175. $Event = \dots \mid \dots \mid \dots \mid \dots$

176. ...

value

165. $uid_M: M \rightarrow MI$

166. $mereo_M: M \rightarrow MM$

167. $attr_MId: M \rightarrow MId$

168. $attr_Position: M \rightarrow Position$

169. $attr_Size: M \rightarrow Size$

170. $attr_Weight: M \rightarrow Weight$

171. $attr_Cost: M \rightarrow Cost$

172. $attr_Flammability: M \rightarrow Flammability$

173. $attr_Insurance: M \rightarrow Insurance$

174. $attr_MHist: M \rightarrow MHist$

Merchandises must satisfy some axiom[s]:

177. No one merchandise must be at exactly one position at any one time.

axiom

177. ...

11.2 Representation of Merchandises

Merchandises are inert: does not move by their own volition ! But merchandises are being moved – by conveyors. So how do we present merchandise ? In Sect. 6.4 on page 61, when we first described conveyor attributes, we did not endow them with merchandise. That will be remedied in Sect. 13.3.5 Page 91.

We shall then, in Sect. 13.3.5 Page 91, see that we choose to model merchandises on a conveyor as a set of merchandise unique identifiers !

178. Here we shall model the existence of a set of merchandises as a state value.

value

178. $ms : \mathbf{M}\text{-set} = \mathbf{obs_MS}(\mathbf{obs_MA}(t))$

Given the unique identifier, mi , of a merchandise and given the “global” merchandises state we can “retrieve” the identified merchandise:

179. The retrieve merchandise function, `retr_merchandise`, takes a merchandise identifier and in the context of the “global” merchandises state ms ,

180. yields the unique $(t) m$ with that identifier in ms that has that identifier.

value

179. $\mathbf{retr_merchandise} : \mathbf{MI} \times \mathbf{MS} \rightarrow \mathbf{M}$

180. $\mathbf{retr_merchandise}(mi)(ms) \equiv \iota m : \mathbf{M} \cdot m \in ms \wedge \mathbf{uid_M}(m) = mi$

11.3 Humans

181. Humans can be merchandise.⁷⁶

type

181. `Human`

value

181. $\mathbf{is_Human} : \mathbf{M} \rightarrow \mathbf{Bool}$

⁷⁶Not in the sense of illegal immigrants, sadly, but in the sense of legally “ticketed” passengers of bus, train, ship and aircraft conveyors.

Chapter 12

Customer

We shall use the term ‘**customer**’ for any person or institution that requests transportation of or receives transported merchandise. Other terms could be ‘**client**’ or ‘**consumer**’. All have the advantage of beginning with a ‘c’. Which we [quickly] convert into a ‘k’ – for same pronunciation !

12.1 Customer Endurants

12.1.1 Endurant Sort

182. There is the atomic endurant: customer.

type

182. K

12.1.2 A State Notion

183. There is the “global” transport value, $t:T$.

184. From it we observe a likewise “global”, the set of all customers, $ks:KS$.

value

183. $t:T$

184. $ks:KS = \mathbf{obs_KS}(\mathbf{obs_KA}(t))$

12.2 Customer Qualities

We lump the presentation of identification, mereology and attributes of customers into one, the present, section.

Unique Identifiers:

- 185. Customers have unique identification.
- 186. We can speak of the identities of all customers, as a “globally” known value.

Mereology:

- 187. The mereology of any customer is the triple of the set of merchandises and the logistics firms that such firms may be requested to arrange transport.

Attributes:

- 188. Customers have practical identification: name and address.
- 189. Customers posses merchandise.
- 190. Customers have outstanding requests: a time-stamped set of shipping notices: to be or being sent, or to request to or expecting to receive.
- 191. Customers accumulate, for every event, a Customer History: A time-stamped, chronologically ordered sequence of event records: most recent event first.
- 192. Events are either ...
- 193.

type

Unique Identifiers:

185. KI

Mereology:

187. $KM = MI\text{-set} \times (CKI|LI)\text{-set} \times CI\text{-set}$

Attributes:

188. $CustId = CustNam \times CustAdd \times \dots$

189. $Possess = MI\text{-set}$

190. $OutReqs = \dots$

191. $CustHist = (TIME \times Event)^*$

192. $Event = \dots$

193. \dots

value

Unique Identifiers:

185. $uid_K: K \rightarrow KI$

value

186. $kis: KI\text{-set} = \{ uid_K(k) \mid k: K \cdot k \in ks^{77} \}$

Mereology:

187. $mereo_K: K \rightarrow KM$

Attributes:

188. $attr_CustId: K \rightarrow CustId$

189. $attr_Possess: K \rightarrow Possess$

190. $attr_OutReqs: K \rightarrow OutReqs$

191. $attr_CustHist: K \rightarrow CustHist$

⁷⁷ ks was defined in Item 184 on the previous page.

12.3 Customer Retrieval

194. The `retrieve_customer` function, `retr_customer`, takes a customer identifier and in the context of the “global” customers state, ks ,

195. yields the unique, ι, k with that identifier in ks that has that identifier.

value

178. `retr_customer`: $KI \times KS \rightarrow K$

180. `retr_customer(ki)(ks) $\equiv \iota k:K \cdot k \in ks \wedge \mathbf{uid_K}(k)=ki$`

12.4 Customer Commands

We refer to Sect. 17.6.1 on page 111.

Chapter 13

Conveyor Companies

We remind the reader of Sect. 10.3 on page 80.

The purpose of a conveyor company is to provide conveyors for the transport of merchandise. It does so in an interaction between customers and logistics companies.

Conveyor companies has basically two main functions wrt. transport provision: a conveyor office and an entity which manages the day-to-day movement of conveyors. A derivative, “in-house” function may be that of logistics: the more-or-less optimal allocation of conveyor resources, routes, etc.

13.1 Conveyor Authorities.

We shall not consider the various public government conveyor authorities that “oversee” specific kinds of conveyor traffic. In many countries there are, for example, several railway operators, but the underlying rail net is usually operated by a [semi-]public government authority.

13.2 Conveyor Company Endurants.

13.2.1 Conveyor Company External Qualities

13.2.1.1 Sorts and Observers

From page 78 we repeat:

type

```
152. CKA
153. CKS = CK-set
154. CK
155. CA
156. CS = C-set
157. CO
158. oL = LI | nil
```

value

```
152. obs_CKA: T → CKA
153. obs_CKS: CKA → CKS
155. obs_CA: CK → CA
156. obs_CS: CA → CS
157. obs_CO: CK → CO
158. obs_oL: CK → oL
```

13.2.1.2 A Conveyor Company Taxonomy

In preparation for our presentation of describing “the state” of the conveyor company segment we show a taxonomy for the full structure of conveyor company parts in Fig. 13.1. The rendition is just an edited segment of Fig. 10.1 on page 75.

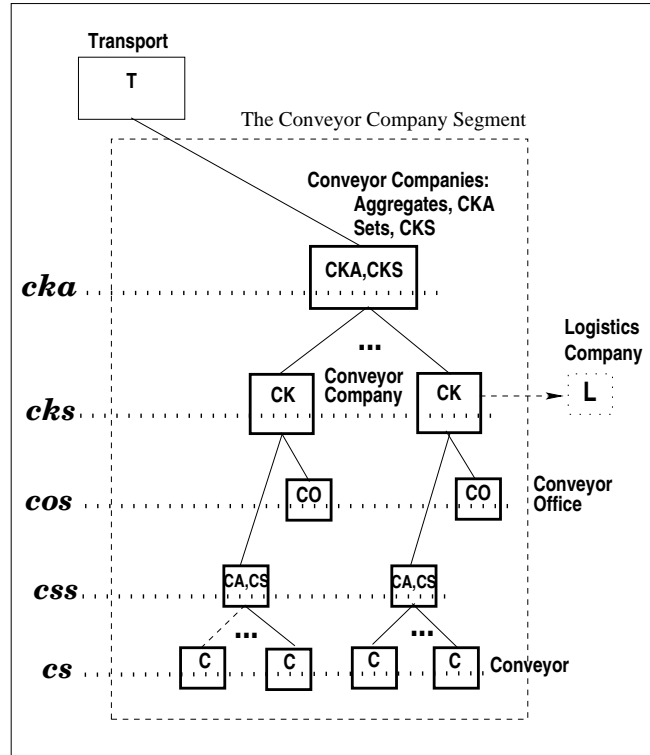


Figure 13.1: Conveyor Companies Taxonomy
We consider all parts to be manifest
Horizontal dotted lines indicate “state” components

13.2.2 A Conveyor Aggregate State Notion

There is the “global” transport domain value, $t:T$.

196. From t we can observe a likewise “global” *conveyor company aggregate* value, $cca:CCA$.

value

196. $cka:CKA = \mathbf{obs_CKA}(t)$

197. From cca we can observe a likewise “global” *set of conveyor companies* value, $cks:CKS$.

value

197. $cks:CKS = \mathbf{obs_CKS}(cca)$

198. From cks we can observe a likewise “global” *set of conveyors* value, $css:CS\text{-set}$.

value

198. $css:CS\text{-set} = \bigcup \{ \mathbf{obs_CS}(ck) \mid ck:CK \cdot ck \in css \}$

199. From *ccs* we can observe a likewise “global” of all set of *conveyors* value, *cs:C-set*.

value

$$199. \quad cs:C\text{-set} = \cup\{\mathbf{obs_CS}(cs) \mid cs:CS \bullet cs \in cks\}$$

200. From *cks* we can observe a likewise “global” set of *conveyor company offices* value, *cos:C0-set*.

value

$$200. \quad cos:C\text{-set} = \cup\{\mathbf{obs_C0}(cs) \mid cs:CS \bullet cs \in cks\}$$

201. From *cks* we can observe a likewise “global” set of optional *logistics companies* value, *ols:oL-set*. They do not contribute to the conveyor company segment state.

value

$$201. \quad ols:C\text{-set} = \cup\{\mathbf{obs_oL}(ck) \mid ck:CK \bullet ck \in cks\} \setminus \{\mathbf{nil}\}$$

202. We can postulate an overall conveyor company state, σ_{CK} .

value

$$202. \quad \sigma_{CK} = \{cca\} \cup \{cks\} \cup ccs \cup cs \cup cos$$

13.3 Conveyor Company Internal Qualities

13.3.1 Conveyor Company Identification

There are three issues here.

13.3.1.1 Conveyor Company Uniqueness of Identification.

The following conveyor companies parts have unique identifications:

203. the conveyor companies aggregate,

204. the conveyor companies set of conveyor companies

205. conveyor companies,

206. conveyors,

207. conveyor offices, and

208. optional logistics firms.

type

- 203. CCAI
- 204. CKSI
- 205. CAI
- 206. CI
- 207. C0I
- 208. oLI

value

- 203. **uid_CCA**: CCA \rightarrow CCAI
- 204. **uid_CKS**: CKS \rightarrow CKSI
- 205. **uid_CA**: CK \rightarrow CKI
- 206. **uid_C**: C \rightarrow CI
- 207. **uid_C0**: C0 \rightarrow C0I
- 208. **uid_oL**: oL \rightarrow oLI

13.3.1.2 Conveyor Company Unique Identifier State.

209. We can postulate, cf. Item 202 on the preceding page, an overall conveyor company unique identifiers state, $\sigma_{CK_{uid}}$.

value

209. $\sigma_{CK_{uid}} =$
 209. $\{uid_CCA(cca)\} [= cca_{ui}]$
 209. $\cup \{uid_CKS(cks)\} [= cks_{uid}]$
 209. $\cup \{uid_CK(ck) | ck:CK \cdot ck \in cks\} [= cks_{uid}]$
 209. $\cup \{uid_C(cs) | c:C \cdot cs \in cs\} [= cs_{uid}]$
 209. $\cup \{uid_CO(co) | co:CO \cdot co \in cos\} [= cos_{uid}]$

Where we use some non-RSL definitions of separate unique identifier sets – to be used in formulas 214–219 below.

13.3.1.3 Conveyor Company Uniqueness of Identification.

210. All conveyor company parts are uniquely identified.

axiom [Unique Conveyor Companies Parts]

210. $card\sigma_{CK} = card\sigma_{CK_{uid}}$

13.3.2 Conveyor Company Mereology

In the previous chapter Sect. 13.3.1, on unique identification, (pages 91-92), we treated all parts of the conveyor companies segment, as manifest. In the present chapter we shall only consider

- conveyor company set of conveyors, cks ,
- conveyor company conveyors, cs , and
- conveyor company offices, cos ,

as manifest.

211. The mereology of conveyor company sets of conveyors, are a pair of (i) the identities of the conveyors they “manage” and (ii) conveyor company, i.e., the conveyor company office they are “paired with”.
212. The mereology of a conveyor is the identity conveyor company set of conveyors they “belong to”.
213. The mereology of conveyor company office is a triplet: (i) the conveyor company sets of conveyors identity, (ii) a set of logistics company identities and (iii) a set of customers [who may handle their transport matters without the help of logistics firms].

type

211. $CAM = CI\text{-set} \times COI$
 212. $CM = CAI$
 213. $COM = CAI \times LI\text{-set} \times KI\text{-set}$

value

211. $mereo_CA: CA \rightarrow CAM$
 212. $mereo_C: C \rightarrow CAI$
 213. $mereo_CO: CO \rightarrow COM$

214. The Well-formed Conveyor Company Mereologies axiom has several clauses:

215. No two conveyor companies share [conveyor company sets of] conveyors.

216. The conveyor aggregate is correctly identified.

217. Conveyor, $c:C$, identities are those of actual conveyors,

218. **and** the identified logistics companies are actual

219. **and** the “k” ustomers are actual.

axiom [Well-formed Conveyor Company Mereologies]

215. $\text{share_conveyors}(cks)$

214. $\wedge \forall ck:CK \cdot ck \in cks \Rightarrow$
 let $(cai, lis, kis) = \text{mereo_CO}(ck),$
 $cs = \text{obs_CS}(\text{obs_CA}(ck))$ **in**
 216. $cai = \text{uid_CA}(\text{obs_CA}(ck))$
 217. $\wedge \{\text{uid_C}(c) | c:C \cdot c \in cs\} \in cs_{uid}$
 218. $\wedge lis \subseteq lis$
 219. $\wedge kis \subseteq kis$

end

215. $\text{share_conveyors}: CKS \rightarrow \mathbf{Bool}$

215. $\text{share_conveyors}(cks) \equiv$

215. $\forall ck, ck':CK \cdot ck \neq ck' \wedge \{ck, ck'\} \subseteq cks$

215. $\Rightarrow \text{obs_CS}(\text{obs_CA}(ck))ck \cap \text{obs_CS}(\text{obs_CA}(ck')) \neq \{\}$

13.3.3 Conveyor Company Attributes

Conveyor Companies have a number of attributes. We mention a few:

- 220. General conveyor company information, which conveyors it manages, their timed routes, capacity, maximum load, etc.⁷⁸
- 221. Resources: own and other conveyor companies' conveyors, their status, etc.
- 222. Contract history:
 - (a) for every contract, once “on the move”, which ways: from sending customer to node, from node to conveyor, from conveyor to node and from node to receiving customer⁷⁹.
- 223. Orders
 - (a) by contract number
 - (b) and an indexed set of offers,
 - (c) each index being a choice number.
- 224. Current business: set of command messages.⁸⁰
- 225. Past business: set of command messages.⁸¹
- 226. History: **TIME**-stamped, chronologically ordered, descending sequence of **Events**: the messages received from customers and conveyors.
- 227. From choice and contract numbers one can observe the identity of the issuing conveyor company.

type

```

220. ConvCompInfo = ...
221. Resources = ...
222. Contracts = ContractNu  $\xrightarrow{m}$  Move*
222a. Move = (KI  $\times$  NI) | (NI  $\times$  CI) | (CI  $\times$  NI) | (NI  $\times$  KI)
223. Orders = ContractNu  $\xrightarrow{m}$  Offers
223a. ContractNu
223b. Offers = ChoiceNu  $\xrightarrow{m}$  TR
223c. ChoiceNu
224. CurrBuss = MSG-set
225. PastBuss = MSG-set
226. CKHist = MSG*
```

value

```

220. attr_ConvCompInfo: C  $\rightarrow$  ConvCompInfo
222. attr_Contracts: CK  $\rightarrow$  Contracts
223. attr_Orders: CK  $\rightarrow$  Orders
224. attr_CurrBuss: CK  $\rightarrow$  CurrBuss
225. attr_PastBuss: CK  $\rightarrow$  PastBuss
226. attr_CKHist: CK  $\rightarrow$  CKHist
```

value

```

227. xtr_CKI: (ChoiceNu | ContractNu)  $\rightarrow$  CKI
```

⁷⁸**Note:** The conveyor company information attribute contains “all” the information that is needed for the calculation of offers etc.

⁷⁹**Note:** This conveyor company attribute is updated every time a conveyor [k12] and a customer [k15] acknowledges the transfer of merchandises

⁸⁰**Note:** Received messages are “stashed” here for future handling – and removed once handled.

⁸¹**Note:** Handled [current business] messages here “stashed” here, transferred from the **current business** attributes.

13.3.3.1 Progress Updates

Conveyor companies are involved in many actions. Most of the actions [referred to by these commands] entail an update of conveyor companies' *Progress* attribute. Some directly by the conveyor companies. Others specifically initiated by [the] so-called *Acknowledgment* actions originating with customers and conveyors.

These explicit acknowledgments are of the form:

- $\mathbf{mk_Acknowledgment}(\mathbf{TIME}, \mathbf{contract_number}, (ui, uj))$

where:

- $(ui, uj): (KI \times CKI) \mid (CKI \times KI) \mid (KI \times NI) \mid (NI \times CI) \mid (CI \times NI) \mid (NI \times KI)$

The explicit acknowledgments entail updates to conveyor companies' *Progress* attribute:

228. The `upd_contracts` function takes a `contracts` attribute and an `acknowledgment` and yields an updated `contracts` attribute.

value

228. `upd_contracts: Contracts \rightarrow Acknowledgment \rightarrow Contracts`

228. `upd_contracts(con)($\mathbf{mk_Acknowledgment}(\tau, \mathbf{cnu}, (ui, uj))$) \equiv`

228. `con \dagger [$\mathbf{cnu} \mapsto \mathbf{con}(\mathbf{cnu})^{\langle \mathbf{mk_Acknowledgment}(\tau, \mathbf{cnu}, (ui, uj)) \rangle}$]`

13.4 Conveyor Company Commands.

We refer to Sect. 17.8.2 on page 117.

Chapter 14

Conveyors, II

We have already dealt with conveyors: their external qualities, Sect. 6.1 on page 59, and two of their internal qualities, *unique identification*, Sect. 6.2 on page 60, and *mereology*, Sect. 6.3 on page 60. We shall, however, extend the mereology first sketched in Sect. 6.3 on page 60.

14.1 Conveyor Mereology

229. The mereology of a conveyor is a quadruple:

- the set of all identifiers of nodes and edges that the conveyor may travel;
- the set of all identifiers of conveyor companies that it may receive directives from and to which it shall have to acknowledge transfers of merchandises;
- the set of all identifiers of customers that it shall inform of pending collections and deliveries, and to which it shall deliver merchandises;

type

229. $CM = (NI|EI)_{\text{set}} \times CKI\text{-set} \times KI\text{-set}$

value

229. $\text{mereo_C}: C \rightarrow CM$

14.2 Conveyor Attributes

In Sect. 6.4 on page 61 we already touched upon some conveyor attributes.

We now extend these⁸².

- 230. Conveyors are of kind [unchanged] [Static Attribute].
- 231. Conveyors convey, i.e., stores (holds), merchandises by contract number.
- 232. They follow a *service route*⁸³, $sr:SR$ [programmable attribute] which is a path, of three or more node and edge identifiers – beginning with a node and ending with a node.
- 233. Conveyors “carry” and index attribute – $SRIndex$ – – indicating as to where in the service route they, at present, are.
- 234. Conveyors also operate according to two “tables”: for each node that it visits there are contracts to be unloaded, respectively loaded. This information is given to conveyors, at any time, by conveyor company directives.
- 235. Conveyors, having unloaded a contract at a final node informs the receiving customer of arrival. Note the difference between that attribute type name *Finals* (with a plural ‘s’) and the function argument identifier type *Final* (with no such plural).
- 236. Conveyors have, dynamically, a position – $CPos$ – either they are at a node or are en route, i.e., on an edge between two adjacent nodes.
- 237. The SR , $SRIndex$ and $CPos$ must be commensurate: if index $i:SRIndex$ designates a node ni , then $cpos:CPos$ must be a $AtNode(ni)$, else, it designates an edge, ej , and $cpos:CPos$ must be some $OnEdge(.,(.,ej),.)$.⁸⁴
- 238. And conveyors have a history.
- 239. We omit further possible attributes: *Speed*, *Acceleration*, *Weight*,
- 240. These routes must be of the kind of the conveyors traveling them !

type

- 230. $Kind$
- 231. $Stowage = ContractNu \multimap M\text{-}set$
- 234. $TBU, TBL = NI \multimap ContractNu\text{-}set$
- 232. $SR = Path$
- 233. $SRIndex = Nat$
- 235. $Finals = NI \multimap (KI \multimap ContractNu)$
- 235. $Final = NI \times ContractNu \times KI$
- 236. $CPos = [\text{Item 104 on page 61}]$
- 238. $CHist = MSG^*$ ⁸⁵
- 239. ...

value

- 230. $attr_Kind: Conveyor \rightarrow Kind$
- 230. $attr_Stowage: Conveyor \rightarrow Stowage$
- 234. $attr_TBU: Conveyor \rightarrow TBU$
- 234. $attr_TBL: Conveyor \rightarrow TBL$
- 232. $attr_SR: Conveyor \rightarrow SR$
- 233. $attr_SRIndex: Conveyor \rightarrow SRIndex$
- 235. $attr_Finals: Conveyor \rightarrow Finals$
- 236. $attr_CPos: Conveyor \rightarrow Position$
- 238. $attr_CHist: Conveyor \rightarrow CHist$

axiom [Routes of commensurate kind]

- 240. [left to the reader !]
- 237. $\square \dots$ [left to the reader] ...

⁸²Here we see a benefit from observing attributes, rather than explicitly defining the attributes of a part as a Cartesian of attributes.

⁸³This service route concept reflects that the conveyor, at any time, may carry merchandise from many distinct contracts.

⁸⁴The joint $i:SRIndex$ and $cpos:CPos$ may be a bit too much, but they come in conveniently for our subsequent formalizations.

14.3 Conveyor Commands.

We refer to Sect. 17.8.2 on page 117.

⁸⁵The messages are those directed at or emanating from conveyors

Chapter 15

Logistics Companies

We remind the reader of Sect. 10.3 on page 80.

The purpose of a logistics company is to arrange of transportation. It does so in interaction between customers and conveyor companies.

The functions of logistics companies very much overlaps with some of the functions of conveyor companies.

An “extreme” example of a logistics company is that of a *travel agency*!

We shall, however, not pursue the logistics concept further – since its role is also played by conveyor companies.

Part V

A MULTI-MODE TRANSPORT: INTENTIONAL PULL

Chapter 16

Intentional Pull, II

TO BE WRITTEN

Part VI

A MULTI-MODE TRANSPORT: COMMANDS

Chapter 17

Multi-mode Transport Commands

17.1 Events and Commands

We distinguish events from commands:

Events are perdurants. The “occur instantaneously”. At “their own” volition. In a state⁸⁶ and possibly cause a state change. Some events, the internal events, have their “root” in the [part] behaviour, hence “affect” the attributes of the underlying part. Other events, the external events, have their “root” “outside” the [part] behaviour, but may “affect” the attributes of the underlying part.

Commands are syntactic entities. Commands are “issued”⁸⁷ by part behaviours. They “occur” as the result of actions taken by [receiving] part behaviours. They have a syntax. They constitute a script facet⁸⁸ related to the part [behaviour]. They have a semantics. The semantics of commands is expressed by behaviour actions. We distinguish between directive commands and response commands. Directive commands are issued by a part behaviour and is directed at another part behaviour. Response commands are acted upon by a part behaviour in response to a command issued by another part behaviour. For both kinds of commands there are thus at least two behaviours involved in expressing their semantics.

17.2 Command Traces

In order to describe the very many commands it has proven useful to sketch a possible diagram of command traces. Figure 17.1 on the following page⁸⁹ shows schematically a possible trace of commands. The ordering, “i” in **ki**, shall indicate some temporal ordering of the issue of these commands.

We shall elaborate on the transport behaviours – with reference to Fig. 17.1 on the following page.⁹⁰

- k1** After some preparatory work a sending customer inquires as to possible transport at a chosen conveyor or logistics company.
- k2** After some preparatory work the conveyor or logistics company replies to the inquiry.
- k3** After some preparatory work the customer places and order for transport.
- k4** After some preparatory work the chosen conveyor or logistics company confirms the order,
- k5** which the customer now [likewise firmly] accepts – with payments.
- k6** At some point logistics companies hand over customer orders to [respective] conveyor companies.
- k7** After some preparatory work these conveyor companies, one or more, select a the set of conveyors and inform them of the order, i.e., give them directives.
- k8** The conveyor company, at some time after [k7] informs the customer that a designated node is ready to accept its merchandises for transport – “on hold”, at a node.

⁸⁶By ‘state’ we shall, in the context of perdurants, mean the value of all dynamic attributes of all behaviours.

⁸⁷By “issued” we shall here mean that they are communicated, in the style of CSP communications by behaviours directed at other behaviours.

⁸⁸For *facets* and *scripts* see [22, Chap. 8].

⁸⁹In Fig. 17.1 on the next page we have “merged” the logistics company handling of commands with that of the conveyor company handling – as there is some “overlap” in their functionalities.

⁹⁰That is: figures like Fig. 17.1 on the next page are not given a semantics. The “semantics” of Fig. 17.1 on the following page “transpires from the entire formal model of this report.

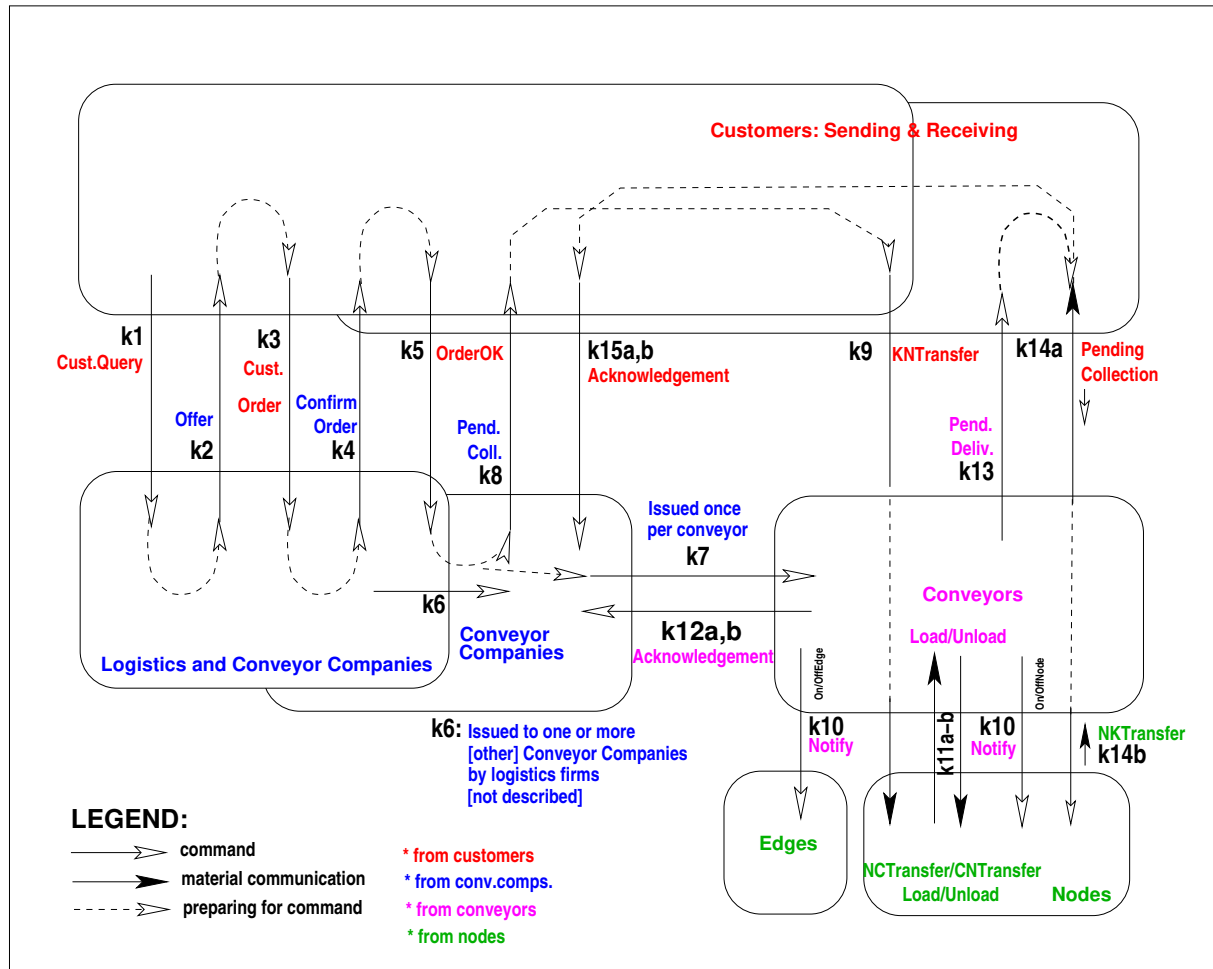


Figure 17.1: Command & Material Traces [→]

- k9** Having been so notified by a conveyor the customer delivers the merchandises, to be transported, at a node, to be “on hold” for the conveyor.
- k10** Conveyors, “on the move”, notify edges and nodes of their presence.
- k11** In synchronous communications conveyors exchange merchandises with nodes: either loading ([k11a]) or unloading ([k11b]).
- k12** Those conveyors inform their companies of transfers.
- k13** The “last” conveyor notifies the “end” customer receiver of pending arrival.
- k14** Having been notified, by the conveyor, the “end” customer receives the transported merchandises.
- k15** That customer informs the [final] conveyor company of the [final] transfer.

17.3 An Analysis

We now analyze Sect. 17.2 on the previous page.

It seems tat there are four kinds of “commands”: *ab initio*, *deferred*, *triggered* and *cascaded*.

- **Ab Initio:** There is only one command of this category: the customer query command, [k1]. Customers, at their own instigation, that is, internal non-deterministically, decides to have some merchandises transported.
- **Deferred:** Most commands are of this category: they are implied by issue of other, that is, “previous” [k2] thus follows from [k1], [k3] from [k2], etc.
There is no guarantee that [k2] will occur. The conveyor (or logistics) company may simply ignore that it has received [k1], respectively [k3] may not occur in response to [k2]. Etcetera.

- **Triggered:** “Commands” [k11a] and [k11b] are not “directly issued”, external non-deterministically, “at some time” in response to [k7].

[k7], such as we small model it, shall result in conveyors having an appropriate attribute, the *to be loaded* and *to be unloaded*, containing such information as when conveyors at nodes shall *load* and *unload* merchandises – and when conveyors are **At** such **Nodes**, this attribute information is said to **trigger** these merchandise transfers.

- **Cascaded:** [k8] is issued either at the same time as [k7], or shortly thereafter. [k9] is issued when [k8] has been received – after which a first [k15] is issued.

17.4 Material and “Immaterial” Commands

Commands **k1-k8**, **k10**, **k13**, **k15** and **k18** are “immaterial” in that they “just” communicate information. Commands **k9**, **k11** and **k14** are “material” in that they, besides information (data) also communicate, i.e., physically transfer material, i.e., merchandises.

17.5 Abstracting an Essence of Transport

By “*abstracting an essence of transport*” we mean that a number of transport “details” are omitted for “the benefit” of emphasizing “other details” ! For examples: (i) we omit details of the structure and contents of what is to be transported, (ii) keeping, somehow, details of who is sending, the address, by whom the merchandise is to be received, etc., (iii) omitting details of merchandise, identification, quantity, weight, value, etc., (iv) cost, payments, etc. In the description of commands, below, we therefore abstract “to the core” these commands – assuming that the various “actors”: the customers, the logistics and conveyor companies and the conveyors can otherwise, i.e., somehow “find out” !

17.6 Commands – A First View

As You see, there are many commands. In this section we shall “*take an abstract view of these*” before, in Sect. 17.8 we go into the detailing of these commands This “abstract view” should then enable us to “design”, as it were, a systematic form and set of less abstract commands.

17.6.1 Customer Commands, I

241. **k1** Customers inquire either logistics companies or conveyor companies about many things, for example time-tables, cost, etc., for the transport of merchandises from one customer to another, etc.
242. **k3** Customers place orders, with either logistics companies or conveyor companies for the transport – according to some offers, **k2**, made by these.
243. **k5** Customers “signs” the **k4** offer.
244. **k9** Customers deliver merchandise to nodes.
245. **k15** Customers acknowledge receipt of merchandises.

type

- | | | |
|------|-------|----------------|
| 241. | [k1] | CustQuery |
| 242. | [k3] | CustOrder |
| 243. | [k5] | OrderOK |
| 244. | [k9] | CustDel |
| 245. | [k15] | Acknowledgment |

17.6.2 Conveyor Company Commands, I

- 246. **k2** Conveyor companies place an offer for transport in response to an inquiry, **k1**.
- 247. **k4** Conveyor companies OKs an order in response to an customer order, **k3**.
- 248. **k7** Conveyor companies inform conveyors of orders, **k4**, to be carried out.
- 249. **k8** Conveyor companies inform customers of pending collection of merchandises.

type

- 246. [k2] ConvCompOffer
- 247. [k4] ConvCompOrdOK
- 248. [k7] ConvCompConvDir
- 249. [k8] PendColl

17.6.3 Conveyor Commands, I

- 250. **k8** Conveyor notify customers of pending collection.
- 251. **k10** Conveyor notify edges and nodes of its presence.
- 252. **k11a-k11b** Conveyor transfers merchandises to and from node.
- 253. **k12** Conveyor acknowledges conveyor company of merchandise transfer.
- 254. **k13** Conveyor informs customer of pending delivery.

- 251. [k10] Notify
- 252. [k11a] CNTransfer
- 253. [k12] Acknowledgement
- 254. [k13] PendDel

• • •

Conveyors collect and deliver merchandise not only from and to nodes, but also from and to other conveyors. Therefore the **k10–k15a-b**. sequence of commands also takes place between distinct conveyors.

17.6.4 Logistics Company Commands

We shall skip this section,

17.7 TR: Transport Routes

We may have “*abstracted too much*” in Sect. 17.6. For example, where in the conveyor company and logistics company to customer order OK commands is the information “hidden” that outlines the course of actions: which route to take, with which conveyors, at which approximate times? That information may be formalized:

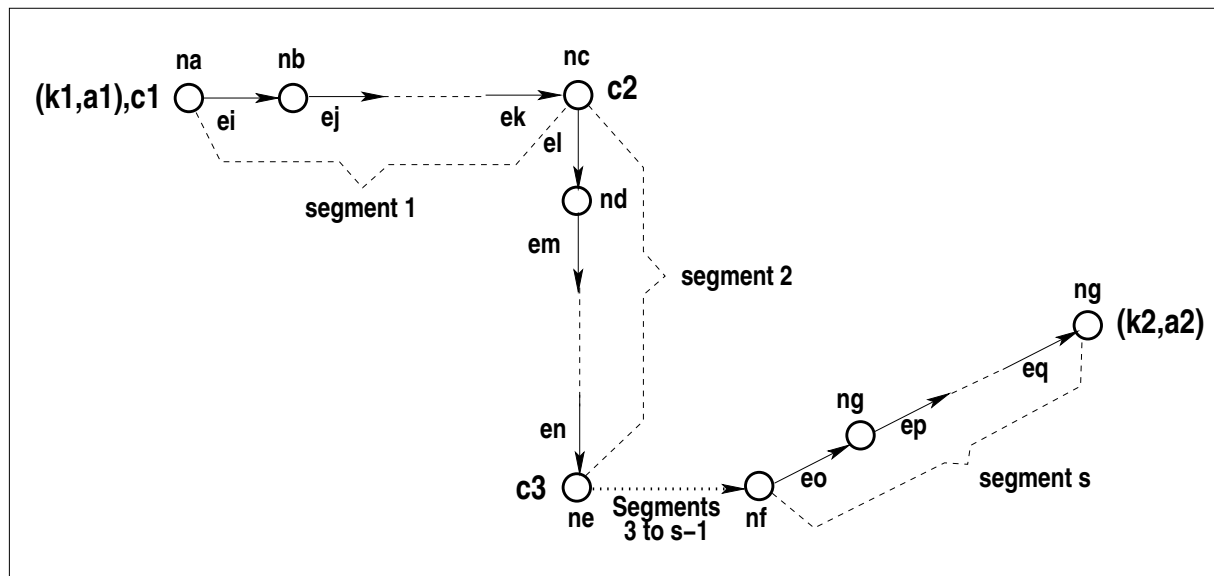


Figure 17.2: A Transport Route: k:kustomer, c:conveyor, a:address, n:node, e:edge

255. A *transport [route]* is a composite of
256. first a sending customer's identifier and place of pick-up ($(k1, a1)$);
257. then the storage: a non-empty set of unique identifiers of the merchandises transported – indexed by contract number;
258. followed by a sequence of one or more segments (**segment 1, segment 2, ..., segment n**)–
259. each segment beginning with a conveyor ($c1, c2, \dots, c3$) identifier, then a node identifier (na, nc, \dots, ne), and finally a non-empty edge-node-path –
260. an edge-node-path is sequence of alternating edge and node identifiers ($(ei, nb, ej, \dots, ek, nc)$);
261. finally ending with a receiving customer's identifier and place of delivery ($k2, a2$)).
262. The two addresses must be different $a1 \neq a2$.
263. The paths formed by edge-node-paths headed by a, i.e., the, node identifier must be paths of the transport net⁹¹,
264. and these paths must be of the same kind as the conveyor for those paths.
265. The time ordering is strictly ascending –
266. and the “end” node of one segment must match, i.e., be equal to the “beginning” node of the next segment.
267. The storage must be well-formed: no two contracts identify the same merchandises.
268. From a contract number one can observe, i.e., extract, the issuing conveyor company identifier.

type

255. $TR = s_sndr: (KI \times Addr)$
257. $\times s_cos: (ContractNu \xrightarrow{m} MI\text{-}set) \text{ axiom } \forall mis: MI\text{-}set \cdot mis \neq \{\}$
258. $\times s_sgl: Segment^* [\text{axiom } \forall sl: Segment^* \cdot sl \neq \langle \rangle]$
261. $\times s_rcvr: (KI \times Addr)$
259. $Segment = TIME \times CI \times NI \times Edge_Node_Path$
260. $Edge_Node_Path = (s_ei: EI \times s_ni: NI)^* \text{ axiom } \forall enp: Edge_Node_Path \cdot enp \neq \langle \rangle$
257. $ContractNu$

value

268. $xtr_CKI: ContractNu \rightarrow CKI$

axiom

257. $\forall tr: TR \cdot \text{let } cos = s_cos(tr) \text{ in } \forall cnu: \text{dom } cos \cdot xtr_MIs(cnu) = cos(cnu) \text{ end}$

Wellformed Transports⁹²**axiom** [Wellformed Transports]

262. $\forall ((_, a1), _, sl, (_, a2)): TR \cdot a1 \neq a2 \wedge$
261. $\forall seg: Segment \cdot seg \in \text{elems } sl \Rightarrow$
261. $\forall (_, ci, ni, enp): Segment \cdot (ci, enil, ei) \in \text{elems } enp$
263. $\wedge \langle ni \rangle^{enp} \in \text{paths} \wedge enil \in \text{paths}$
264. $\wedge \text{same_kind}(enp, ci)$
265. $\wedge \forall i, i+1 \cdot \{i, i+1\} \subseteq \text{inds } sl \Rightarrow$
265. $\text{let } (\tau_i, ci, ni, enp) = sl[i], (\tau_j, cj, nj, enpj) = sl[i+1] \text{ in } \tau_i < \tau_j$
266. $\wedge s_ni(enpi[\text{len } enp]) = nj \text{ end}$
266. $\forall storage: (ContractNu \xrightarrow{m} MI\text{-}set) \cdot$
266. $\forall cni, cnj: ContractNu \cdot \{cni, cnj\} \subseteq \text{dom } storage \wedge cni \sim cnj$
266. $\Rightarrow storage(cni) \cap storage(cnj) = \{\}$

⁹¹Cf. Item 71 on page 54

⁹²Axiom 262–266 must be carefully checked

Auxiliary Functions

value

264. `same_kind: Edge_Node_Path × CI → Bool`
 264. `same_kind(enpath,ci) ≡ ...` [Left to the reader]

An aspect of the transport routes, `tr:TR`, when a transport route has more than one segment, is that the node between two adjacent segments, serve as a repository for merchandises. A conveyor unloading merchandises destined for other, one or more, conveyors may not arrive when either or all of these conveyors have arrived⁹³, so they deposit, put “on hold”, those merchandises. For respective kinds of nodes these “deposit holds” are, for example, called *bus stops* for kind **road**, *train station waiting rooms* for kind **rail**, *airport passenger lounges* for kind **air**, and *container terminals* for kind **sea**.

Segments (Item 259 on the preceding page) are static descriptions of where conveyors are to move. Service Routes, SRs (Item 232 on page 98), are static descriptions of when conveyors are to move.

17.8 A Closer Analysis of Commands

We refer back to the overview of all commands given in Sect. 17.6.

17.8.1 Customer Commands, II

241. For a customer to formulate a proper query about possible transports such a query must contain the following information:

- (a) a unique, customer-chosen inquiry identification and
- (b) a query compound.

269. The query compound, it seems, should contain such information as:

- (a) name, address, and other such data that “pin-points”, “validates” the inquirer;
- (b) characterization of the merchandise to be transported: product information, quantity, total weight, total volume, total value [for insurance purposes], etc.;
- (c) time interval of transport;
- (d) from where to where;
- (e) expected cost frame; and, possibly, more!
- (f) Addresses are further unspecified.

type

241. `[k1] CustQuery ::`
 268a. `QueryId`
 268b. `× QueryComp`

 269. `QueryComp =`
 269a. `Addr`
 269b. `× MInfo [...]`
 269c. `× TI [= (TIME×TIME), axiom ∀ (ft,tt):TI•ft<tt]`
 269d. `× FT [= NI×(NI×KI×AddrInfo), axiom ∀ (nf,(nt,__,__)):FT•nf≠nt]`
 269e. `× ExpCost`
 269f. `Addr`

⁹³The conveyor or logistics company, when preparing the offers, are assumed to make sure that there is appropriate time intervals between unloading and loading conveyors for relevant merchandises.

242. For a customer to formulate a proper order for a specific transport such a query must be based on the conveyor or logistics company offer to a query like that outlined in Item 269, above, the order must contain the following information:

- (a) the customer inquiry identification, and
- (b) a reference to the logistics or conveyor company contract number given in query reply.
- (c) Then more-or-less the same information, formulated as a compound, as given in the original query – which is also expected to be contained in the reply offer;
- (d) name, address, etc.,
- (e) merchandise information,
- (f) precise times.
- (g) from-to transport details,
- (h) the offered cost,
- (i) etc.

270. From a query identifier one can extract the customer identity.

type

```
242. [k3]  CustOrd ::
269a.      QueryId
269b.      × ContractNu
269c.      × OrdrComp
```

```
269c.  OrdrComp =
269d.      Addr
269e.      × MerchInfo
269f.      × TI
269g.      × FT
269h.      × Cost
269i.      × ...
```

value

```
270. xtr_KI: QueryId → KI
```

243. For a customer to OK a proposed transport the the customer must provide

- (a) the contract number,
- (b) the choice number,
- (c) payment.

```
243. [k5]  OrderOK ::
270a.      ContractNu
270b.      × ChoiceNo
270c.      × Payment
```

243. For a customer to deliver the merchandises according to the contracted order the customer must provide

- (a) a reference to to the contract number and
- (b) the therein indicated number of actual merchandises !

type

```
243. [k9]  KNTransfer ::
270a.      ContractNu
270b.      × M-set
```

271. [k15] A customer having received merchandises (from another customer via conveyors) at a node acknowledges this receipt by so informing the conveyor company.

type

271. [k15] Acknowledgment :: $TIME \times ContractNu \times (NI \times KI)$

Observe that the first two commands and the last command were strictly “informational”, i.e., syntactic, whereas the Customer tDelivery command is “rather” physical; i.e., semantic: the command, so-to-speak, “embodies” an action, the manifest movement of volumes of possibly heavy material !

There may be other customer commands – such as inquiring as to the progress of an actual transport, etc. We leave that to the reader.

17.8.2 Conveyor Commands, II

The conveyor commands, first outlined in Sect. 17.6.3 on page 112, are now summarized and detailed. First we list their treatment in Sect. 17.6.3 on page 112.

250. **k8** Conveyor informs customer of pending collection.
251. **k10** Conveyor notifies edges and nodes of conveyor presence.
252. **k11** Conveyor transfers (loads [k11a], unloads [k11b]) merchandises.
253. **k12** Conveyor acknowledges conveyor company of merchandise transfer.
254. **k13** Conveyor informs customer of pending delivery.

250. [k8] PendColl
 251. [k10] Notify
 252. [k11a,b] Transfer = CNTransfer | NCTransfer
 253. [k12] Acknowledgment
 254. [k13] PendDel

272. [k8] Conveyors inform either a customer of pending collection of merchandises.
 They do so by simply mentioning the contract number and the set of unique identifiers of the merchandise to be collected.

273. [k10] Conveyors notify edges and nodes of their presence.

Conveyors transfer:

274. [k11a] load from a node.
275. [k11b] or unload merchandises to a node.

They do so by stating the contract number and presenting the set of merchandise to be transferred.

276. [k12] Conveyors, time-stamped, acknowledges its company of, and at the completion of a transfer, collection or delivery of merchandise. They do so by mentioning the contract number and the two “parties” to the transfer:

277. either a customer and a node, or a of conveyor and a node.
278. [k13] Conveyors inform customers of pending delivery (at a node).

type

272. [k8] PendColl :: $(NI \times (ContractNu > MI\text{-}set))$
 273. [k10] Notify :: $AtNode \mid OnEdge$
 274. [k11a] NCTransfer :: $(ContractNu \times M\text{-}set)$
 275. [k11b] CNTransfer :: $(ContractNu \times M\text{-}set)$
 276. [k12] Acknowledgment :: $TIME \times ContractNu \times FromTo$
 277. FromTo = $(NI \times CI) \mid (CI \times NI)$
 278. [k13] PendDel :: $(NI \times (ContractNu \times MI\text{-}set))$

17.8.3 Conveyor Company Commands, II

Review:

type

```

ι246 π112. [k2] ConvCompOffer
ι247 π112. [k4] ConvCompOrdOK
ι248 π112. [k7] ConvCompConvDir

```

We now detail these.

279. An offer for transport must state

- (a) the conveyor company identity;
- (b) a contract⁹⁴ number;
- (c) refer to an inquiry, for example by stating its number or by repeated it; and
- (d) a set of zero, one or more choice number indexed offer-choices.

An offer-choice

- (e) a timed route of transport, and
- (f) a cost.

280. An OK, binding acknowledgment of an order must state

- (a) the conveyor company identity,
- (b) a contract number,
- (c) refer to an offer and choice number,
- (d) “repeats” the contracted timed route of transport,
- (e) and the cost.

281. The conveyor company information to be given to conveyors of orders, **k4**, state

- (a) the conveyor company identity;
- (b) a contract number and
- (c) the contracted time route of transport.

282. From Offer numbers, contract numbers and choice numbers one can extract the offering and contracting company’s identity

283. as well as the identity of the customer being offered and contracted.

type

```

279. [k2] ConvCompOffer :: CKI × ContractNu × QueryNu × (ChoiceNu  $\overline{m}$  OfferChoice)
279b. ContractNu
279d. ChoiceNu
279e. OfferChoice = TR × ost

```

280. [k4] ConvCompOrdOK :: CKI × ContractNu × ChoiceNu × TR × Cost

281a. [k7] ConvCompConvDir :: CKI × ContractNu × Segment

value

```

282. xtr_CKI: (OfferNu|ChoiceNu|ContractNu) → CKI
283. xtr_KI: (OfferNu|ChoiceNu|ContractNu) → KI

```

⁹⁴ – even though this may not result in a contract

17.8.4 Node Commands

Nodes, as behaviours, have now become reactive. They store contracted merchandises – “on hold between” conveyors. So they must react to conveyor commands requesting merchandises, unloaded, to be put “on hold” or fetched, to be loaded. They react by accepting and delivering merchandises from, respectively to conveyors and customers. To these requests node behaviours must react [immediately (?)]. These are the only transport commands that must be so synchronized⁹⁵. All other transport commands are “buffered”⁹⁶

284. [k14] Nodes transfer merchandises (from another customer via conveyors) from the ‘on-hold’ of a node to a customer.

type

284. [k14] `NKTransfer :: NI × ContractNu`

17.8.5 Edge Commands

There are no edge commands. Edge behaviours receive notifications from conveyors as to their presence on edges.

⁹⁵ **Alert:** Check that I actually describe so !

⁹⁶ **Alert:** Perhaps one should reconsider the customer to conveyor and conveyor to customer transfers of merchandises to also be synchronized.

Part VII

IDENTITIES

Chapter 18

Identities

So far we have introduced a variety of identities:

- unique identities of endurants,
- query ‘numbers’,
- offer ‘numbers’,
- contract numbers,
- etc.

These are, of course, not identifiers nor numbers or numerals. They are abstract entities.

We can say a lot about these:

285. From the identity of a customer we can “extract” (i.e., “observe”) such things as the name of the customer, the address (road name & number, district name, city name, county name, country name, telephone ‘numbers’, e-mail addresses, etc., etc.).
286. From the identity of a conveyor we can ‘extract’ the identity of its owner: a conveyor company.
287. From a query ‘number’ we can extract the identity of the querying customer.
288. From offer, order and contract ‘numbers’ we can extract the identities of conveyor (logistics) company and customer identities.
289. From a contract number we can extract the set of merchandise identifiers “involved” in the identified contract.
290. From a contract number we can extract a waybill⁹⁷,
291. From a contract number we can extract a bill-of-lading⁹⁸.
292. From a contract number we can observe whether it (i.e.,the waybill/bill-of-lading) represents a ticket for human “merchandise” (cf. Sect. 11.3 on page 83).
293. Et cetera.

value

285. `xtr_Name: KI→Name`
285. `xtr_Addr: KI→((RoadNam×Nat)×DisNam×CounNam×LandNam×PhonNu×Email×...)`
286. `xtr_CKI: CI→CKI`
287. `xtr_CI: QueryNu→CI`
288. `xtr_CKI: (OfferNu|OrderNu|ContractNu)→CKI`
288. `xtr_CI: (OfferNu|OrderNu|ContractNu)→CI`
289. `xtr_MIs: ContractNu→MI-set`

type

285. `RoadNam, DisNam, CounNam, LandNam, PhonNu, Email`
290. `WayBill`
291. `BoL`

value

290. `xtr_WayBill: CKI→WayBill`
291. `xtr_BoL: CKI→BoL`
292. `is_Ticket: (WayBill|BoL)→Bool`

MORE TO COME

⁹⁷A waybill is a document issued by a carrier acknowledging the receipt of goods by the carrier and the contract for shipment of a consignment of that cargo. Typically it will show the names of the consignor and consignee, the point of origin of the consignment, its destination, and route [Wikipedia].

⁹⁸A 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. Although the term is historically related only to carriage by sea, a bill of lading may today be used for any type of carriage of goods. 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.[a] Whereas a bill of lading is negotiable, both a policy and an invoice are assignable [Wikipedia].

Part VIII

A MULTI-MODE TRANSPORT: BEHAVIOURS

Chapter 19

Multi-mode Behaviours

Contents

19.1 Communication	127
19.2 Behaviour Signatures	128
19.3 Which Behaviours to Describe ?	129
19.4 Multi-mode “Systems”	129
19.4.1 Multi-mode Domain Initialization	129
19.4.2 Multi-mode Domain Instantiation	130

19.1 Communication

294. There is a medium for synchronization of and communication between behaviours.
295. **comm** $\{\{ui, uj\}\} ! value$ expresses an event [an action]: the “output” of value, from the behaviour identified by ui towards the behaviour identified by uj.
296. **comm** $\{\{ui, uj\}\} ?$ expresses a value, i.e., the “input” of a value, from the behaviour identified by ui by the behaviour identified by uj.

channel

297. $\{ \text{comm}\{\{ui, uj\}\} \mid ui, uj : UI \cdot \{ui, uj\} \subseteq \sigma_{uis} \} : MSG$

297. The **comm** channel declaration above expresses that this medium is “two-dimensional” and communicates (“mediates”) messages of type M.
298. Messages are timed commands
299. and the commands are those of customers, conveyor companies, logistics companies and conveyors.

type

298. $MSG = (UI \times TIME \times UI)^{99} \times Command$

298. $UI = KI \mid CKI \mid CI$

298. [k1]	Command = CustQuery	[Customer → Company]
298. [k3]	CustOrd	...
298. [k5]	OrderOK	...
298. [k15]	Acknowledgment	...
298. [k9]	KNTransfer	[Customer → Node]
298. [k14a]	PendColl	...
298. [k2]	ConvCompOffer	[Company → Customer]
298. [k4]	ConvCompOrdOK	...
298. [k8]	PendColl	...
298. [k7]	ConvCompConvDir	[Company → Conveyor]
298. [k12]	Acknowledgment	[Conveyor → Company]

298.	[k13]		PendDeliv	[Conveyor→Customer]
298.	[k11a]		CNTransfer	[Conveyor→Node]
298.	[k10]		Notify	...
298.	[k11b]		NCTransfer	[Conveyor→Edge]
298.	[k10]		Notify	...
298.	[k11b]		NCTransfer	[Node→Conveyor]
298.	[k14]		NKTransfer	[Node→Customer]

• • •

A core property of CSP is that behaviours both **synchronize** their behaviours and **exchange** messages, from one, **!**, to another, **?**.

19.2 Behaviour Signatures

We omit consideration of aggregate and merchandise behaviours. There are:

- | | |
|--|-------------------------------|
| 300. the customer behaviours, | 303. the conveyor behaviours, |
| 301. the logistics company behaviours, | 304. the edge behaviours, and |
| 302. the conveyor company behaviours, | 305. the node behaviours. |

In some other order their signatures are:

value	
300. customer: KI	[identifier]
300. → KM ¹⁰¹	[mereology]
300. → (CustId × AddrInfo × ...)	[static attrs.]
300. → (Possess × OutReqs × CustHist) Unit	[progr. attrs.]
302. conv_comp: CKI →	[identifier]
302. → CKM	[mereology]
302. → (ConvCompInfo × ...)	[static attrs.]
302. → (Resources×Contracts×Orders×CurrBuss×PastBuss×CKHist) Unit	[progr. attrs.]
303. conveyor: CI	[identifier]
303. → CM	[mereology]
303. → (Kind × ...)	[static attrs.]
303. → (Stowage×TBU×TBL×SR×SRIndex×Final×CPos×CHist) Unit	[progr. attrs.]
301. logistics: LI →	[identifier]
301. → LM	[mereology]
301. → (LogisticsCompInfo × ...)	[static attrs.]
301. → (PastBusiness × CurrBusiness × LHist) Unit	[progr. attrs.]
304. edge: EI	[identifier]
304. → EM	[mereology]
304. → (EdgeKind × LEN × COST × ...)	[static attrs.]
304. → EHist Unit	[progr. attrs.]
305. node: NI	[identifier]
305. → NM	[mereology]
305. → (NodeKind × ...)	[static attrs.]
305. → (OnHold × NHist) Unit	[progr. attrs.]

⁹⁹The triplet: (fui,t,tui) is subject to the following constraint, which we leave to the reader to formalize: if tui:KI then tui:CKI or tui:CI; if tui:CKI then tui:KI or tui:CI; if tui:CI then tui:KI or tui:CKI.

19.3 Which Behaviours to Describe ?

We treat the transcendently deduced behaviours of some, but not all, the manifest parts: customers, conveyor companies, but **not** their conveyor company offices **nor** their conveyor aggregates, but their conveyors. We omit, also treatment of Logistics companies as their “function” is “very much like, i.e., “overlapping” with, that of conveyor companies.

• • •

The arrangement of the [narrative & formal] descriptions is by enduring, i.e., part, type; but the “reading” of these should be by pairs: each pair represents an arrow in Fig. 10.1 on page 75, one of the pair represents the source of the arrow, the “sending” behaviour, the second of the pair represents the target of the arrow, the “receiving” behaviour,

19.4 Multi-mode “Systems”

We can initialize a domain, and we can instantiate a domain.

19.4.1 Multi-mode Domain Initialization

- 306. An initialization of a transport domain means the parallel composition of the
- 307. parallel composition of the initialization of all customer behaviours with the
- 308. parallel composition of the initialization of all conveyor company behaviours with the
- 309. parallel composition of the initialization of all conveyor behaviours with the
- 310. parallel composition of the initialization of all logistics behaviours with the
- 311. parallel composition of the initialization of all edge behaviours with the
- 312. parallel composition of the initialization of all node behaviours.

```

306. instantiation: Unit → Unit
306. instantiation() ≡
307.   || { customer(uid_K(k))
307.         (mereo_K(k))
307.         (attr_CustId(k),...)
307.         ([], {}, ⟨⟩)
307.         | k:K • k∈ks } [ks, see Item 151 on page 77]
307.   ||
308.   || { conv_comp(uid_CK(ck))
308.         (mereo_CK(ck))
308.         (attr_ConvCompInfo(ck),...)
308.         (attr_Resources(c), [], [], {}, {}, ⟨⟩)
308.         | ck:CK • ck∈cks } [cks, see Item 197 on page 90]
308.   ||
309.   || { conveyor(uid_C(c))
309.         (mereo_C(c))
309.         (attr_Kind(c),...)
309.         ([], [], [], attr_SR(c), 1, [], attr_Position(c), ⟨⟩)
309.         | c:C • c∈cs } [cs, see Item 199 on page 91]
309.   ||
310.   || { logistics( ... ) | ... } [see remark on page 149]
310.   ||
311.   || { edge(uid_E(e))
311.         (mereo_E(e))
311.         (attr_EdgeKind(e), attr_LEN(e), attr_COST(e),...)
311.         (⟨⟩)
311.         | e:E • e∈es } [es, see Item 45 on page 50]
311.   ||
312.   || { node(uid_N(n))
312.         (mereo_N(n))
312.         (attr_NodeKind(n),...)
312.         ([], ⟨⟩)
312.         | n:N • n∈ns } [ns, see Item 46 on page 50]

```

19.4.2 Multi-mode Domain Instantiation

```

306. instantiation: Unit → Unit
306. instantiation() ≡
307.   || { customer(uid_K(k))
307.         (mereo_K(k))
307.         (attr_CustId(k),...)
307.         (attr_Possess(k),attr_OutReqs(k),attr_CustHist(k))
307.         | k:K • k∈ks } [ks, see Item 151 on page 77]
307.   ||
308.   || { conv_comp(uid_CK(ck))
308.         (mereo_CK(ck))
308.         (attr_ConvCompInfo(ck),...)
308.         (attr_Resources(c),attr_Contracts(ck),attr_Orders(ck),
308.         attr_CurrBuss(ck),attr_PastBuss(ck),attr_CKHist(ck))
308.         | ck:CK • ck∈cks } [cks, see Item 197 on page 90]
308.   ||
309.   || { conveyor(uid_C(c))
309.         (mereo_C(c))
309.         (attr_Kind(c),...)
309.         (attr_Stowage(c),attr_TBU(c),attr_TBL(c),attr_SR(c),
309.         attr_SRIndex(c),attr_Final(c),attr_Position(c),attr_CHist(c))
309.         | c:C • c∈cs } [cs, see Item 199 on page 91]
309.   ||
310.   || { logistics( ... ) | ... } [see remark on page 149]
310.   ||
311.   || { edge(uid_E(e))
311.         (mereo_E(e))
311.         (attr_EdgeKind(e),attr_LEN(e),attr_COST(e),...)
311.         (attr_EHist(e))
311.         | e:E • e∈es } [es, see Item 45 on page 50]
311.   ||
312.   || { node(uid_N(n))
312.         (mereo_N(n))
312.         (attr_NodeKind(n),...)
312.         (attr_OnHold(n),attr_NHist(n))
312.         | n:N • n∈ns } [ns, see Item 46 on page 50]

```

We refer to Sect. 8.5 on page 70 for a first example of domain initialization.

Chapter 20

Customer Behaviours

Contents

20.1	Main Behaviour	131
20.1.1	Overall Behaviour	131
20.1.2	Overall Reactive Behaviour	132
20.2	Subsidiary Behaviours	132
20.2.1	Proactive Behaviours	132
20.2.1.1	[k1] Customer Issues Query	132
20.2.2	Reactive Behaviours	133
20.2.2.1	[k3] Customer Issues Order	133
20.2.2.2	[k5] Customer Accepts Offer	133
20.2.2.3	[k9] Customer Delivers Mercandises	134
20.2.2.4	[k14a-b,k15b] Customer Requests & Receives Mercandises	134

20.1 Main Behaviour

20.1.1 Overall Behaviour

313. The customer internal non-deterministically alternates between being

- (a) a private entity, doing whatever,
or possibly
- (b) [k1]¹⁰² querying conveyor or logistics companies about a possible transport;
- (c) [k3] examining a conveyor or logistics company offer;
- (d) [k5] accepting an offer from a conveyor or logistics company;
- (e) [k9] delivering merchandises to nodes;
- (f) [k14] requesting contracted onhold merchandises from nodes, and
- (g) external non-deterministically possibly receiving messages from conveyor companies or logistics companies, conveyors, or nodes ([k2,k4,k8,k13,k14]).

u The [k1] query is pivotal. It “sets everything else in motion”. Responses from the conveyor company are “temporarily stored”, cf. *customer receives messages*, i.e., *cust_receiv_messages*, Item 313g. “Storage” is in the form of an additional behaviour argument.

value

```
313. customer(ki)(cm)(kid,kaddr)(po,or,ch) ≡
313a. ...
313b. [k1]  ⌈ cust_issues_query(ki)(cid,...)(...)(po,or,r,ch)
313c. [k3]  ⌈ cust_issues_order(ki)(cid,...)(...)(po,or,r,ch)
313d. [k5]  ⌈ cust_order_OK(ki)(cid,...)(...)(po,or,r,ch)
313e. [k9]  ⌈ cust_delivers_merchandises(ki)(cid,...)(...)(po,or,r,ch)
313e. [k14] ⌈ cust_requests_merchandises(ki)(cid,...)(...)(po,or,r,ch)
313g.      ⌈ cust_receives_messages(ki)(cid,...)(...)(po,or,r,ch)
```

¹⁰²The bracketed numbers refer to those of Fig. 17.1 on page 110.

20.1.2 Overall Reactive Behaviour

314. The external non-deterministic reception of messages, **msg**:¹⁰³ MSG, proceed as follows:

- (a) Customer awaits messages¹⁰⁴ from either conveyor companies or conveyors.
- (b) Customers “remember” these messages as outstanding requests. They will be handled by [recursively] iterated invocations of the conveyor behaviour !

So we “handle” that “lastly” listed behaviour “first” !

value

```
314. cust_receives_messages(ki)(cid,...)(...)(po,or,r,ch) ≡
314a.   let msg = [] { comm[k1,ui]? | ui ∈ ckis∪cis } in
314b.   customer(ki)(cid,...)(...)(po,or∪{(k1,\tida,ui),msg},r,<msg>^ch) end
```

The “handling” of the orders, or “buffered” are defined in the ‘Reactive Behaviours’ subsections:

- Customer Issues Order [k3], Sect. 20.2.2.1, item 316 on the facing page;
- Customer Accepts Offer [k5] (order OK), Sect. 20.2.2.2, item 316 on the next page;
- Customer Delivers Mercandises [k9], Sect. 20.2.2.3, item 317 on page 134; and
- Customer Requests & Receives Merchandises [k14a-b,k15b], Sect. 20.2.2.4, item 318 on page 134.

20.2 Subsidiary Behaviours

20.2.1 Proactive Behaviours

20.2.1.1 [k1] Customer Issues Query

315. [k1] The customer decides

- (a) to inquire, with some conveyor or logistics company, with a selected query command¹⁰⁵,
- (b) which it then communicates to the conveyor company or logistics company, updates its outstanding requests and augments its history,
- (c) whereupon it resumes being a customer.

This query action [k1] is “matched” by the suggest offer action [k2] Sect. 21.3.1 on page 137; cf. formula lines 315b and 321d on page 137.

```
315. cust_issues_query(ki)(cid,...)(...)(po,or,ch) ≡
315a.   let (coli,mk_CustQuery(qi,qc)) = sel_q(ki,(cid,...),(...),(po,or,ch)) in
315a.   let msg = ((ki,record TIME(),coli),mk_CustQuery(qi,qc)) in
315b.   comm[{ki,coli}] ! msg; [k1]
315c.   customer(ki)(cid,...)(...)(po,or∪{msg},<msg>^ch) end end

315a.   sel_q: KI × (CustId × AddrInfo × ...) × ..
315a.   × (Posses × OutReqs × CustHist) → CustInq
315a.   sel_q(ki,(cid,ai,...),(...),(po,or,ch)) ≡ ... see footnote 105 pg 132
```

¹⁰³We have emphasized the **message** arguments as these play a pivotal role in the behavior interaction.

¹⁰⁴These messages are either [k4] ConvCompOffers, [k8] ConvCompOrderOK, [k9] PendColl, [k13] ConvCustPendDel, [k14] NKTransfer messages.

¹⁰⁵— we leave unspecified how that query is formed from the basis of the customer attributes

20.2.2 Reactive Behaviours

20.2.2.1 [k3] Customer Issues Order

316. [k3] If there is an ongoing (or outstanding) conveyor company offer

- (a) then the customer selects a suitable one. If there is not such the choice number is forced to 0.
- (b) Time is recorded.
- (c) If the customer does not finds a suitable offer
- (d) it so informs the conveyor company.
- (e) Else it likewise informs the conveyor company of order and choice number.
- (f) Whereupon it resumes being a customer.¹⁰⁶

This issues_order action [k3] is “matched” by the confirm_order action [k4] Sect. 21.3.2 on page 137.

value

```

316. cust_issues_order(ki)(cid,ai,...)(...)
316.      (po,{((cki,t,ki),mk_ConvCompOffer(on,t,choices))107}∪or,ch) ≡
315a.      let (cn,offer) = select_offer(choices) in
315c.      let msg = ((ki,recordTIME(),cki),if cn=0
315c.          then mk_OrderOK(on,no)
315c.          else mk_OrderOK(on,cn,offer) end) in
315d.      comm[{cid,cki}]! msg; [k3]
315f.      customer(ki)(cid,ai,...)(...)(po,or,<msg>^ch) end end

```

20.2.2.2 [k5] Customer Accepts Offer

316. Customers

- (a) examine transport company offers: the examine analysis function is left to Your imagination; the status value is either a **no**, or is **OrderOK**.
- (b) A time-stamped message to that effect is communicated to the conveyor company.
- (c) And the customer resumes being so.

This customer_order_OK action [k5] is “matched” by the conveyor_directives action [k7] Sect. 21.3.3 on page 138. And also the pending_collection action [k8] Sect. 21.3.4 on page 139.

```

316. cust_order_OK(ki)(cid,...)(...)
316.      (po,{(ki,τ,cki),m:mk_ConvCompOffer(cki109,cnu,qno,offers)}∪or,ch) ≡
316a.      let okonok = examine(ki)(cid,...)(...)(po,{(ki,τ,cki),m}∪or,ch) in
316b.      let msg = ((ki,TIME,cki),mk_OrderOK(oknok)) in
316b.      comm[{cid,cki}]! msg;
316c.      customer(ki)(cid,...)(...)(po,or,<msg>^ch) end end

```

¹⁰⁶We have used some informal notation, i.e., [orderOK=]

¹⁰⁷Note the formal argument “trick”: If the ongoing_requests argument contains an element, ConvCompOffer(on,t,choices), then the cust_accept_offer behaviour applies. If it does not, then **skip**!

¹⁰⁹The two argument ckis are/must be [!] identical.

20.2.2.3 [k9] Customer Delivers Merchandises

317. [k9] Customer delivers merchandises:

- (a) collecting the identified merchandises;
- (b) composing messages to node and contracting conveyor company;
- (c) then transferring the merchandises to the identified node;
- (d) informing the contracting conveyor company; and
- (e) finally resuming being a customer.

This delivery action [k9] is “in consequence” of the pending collection action [k8] Sect. 21.3.4 on page 139.

value

```

317.  cust_delivers_merchandises(ki)(cid,ai,...)(...)
317.      (po,{mk_PendColl(cki,on,mis,ni)}Uor,ch) ≡
317a.      let ms = {m|M:m∈po^uid_M(m)∈mis}, τ = record TIME() in
317b.      let msg1 = ((ki,τ,ni),mk_KNTransfer(on,ms)),
317b.      msg2 = ((ki,,τ,cki),mk_Acknowledgment(τ,cnu,(ki,ni))) in
317c.      [k9] (comm[{ki,ni}]! msg1
317d.      [k15a] || comm[{ki,cki}]! msg1);
317e.      customer(ki)(cid,ai,...)(...)(po\ms,or,<{msg1,msg2}>^ch) end end

```

20.2.2.4 [k14a-b,k15b] Customer Requests & Receives Merchandises

318. [k14a] Customers are ready to receive merchandises once a message of pending delivery has been received from a conveyor.

- (a) [k14a] They can therefore accept such a delivery notice;
- (b) concocts an acknowledgment to the conveyor company,
- (c) [k15b] communicates this to the conveyor company,
- (d) whereupon it resumes being a customer.

This `cust_requests_merchandises` action [k14] is “matched” by the node action Sect. 25.3 on page 154; cf. formula lines 318a and 342 on page 154.

value

```

318.  cust_requests_merchandises(ki)(cid,ai,...)(...)
318.      (po,{(ci,t,ki),mk_PendDeliv(ci,cnu,mis)}Uor,ch) ≡
318b.  [k14a] comm[{ki,ni}]! mk_((ki,record TIME(),ni),PendColl(ni,(cnu,mis)))110;
318a.  [k14b] let mk_NKTransfer(cms) = comm[{ki,ni}]?111 in
318c.  [k15b] comm[{ki,cki}]! mk_Acknowledgment(record TIME(),cnu,(ci,ki)) ;
318d.      customer(ki)(cid,ai,...)(...)(poUUrng cms,or,<ms,msg>^ch) end

```

¹¹⁰Observe that the received message `ki` [in `(cki,t,ki)`] must match the formal argument `ki`. This informative communication is symbolized by the “open, white arrowhead” of the [k14] “double arrow” in Fig. 17.1 on page 110.

¹¹¹This material communication is symbolized by the “black arrowhead” of the [k14] “double arrow” in Fig. 17.1 on page 110.

Chapter 21

Conveyor Company Behaviours

Contents

21.1 Main Behaviour	135
21.2 Main Reactive Behaviour	136
21.3 Subsidiary Behaviours	137
21.3.1 [k2] Suggest Offer	137
21.3.2 [k4] Confirm Order	137
21.3.3 [k7] Conveyor Directives	138
21.3.4 [k8] Pending Collection	139

21.1 Main Behaviour

319. Conveyor companies non-deterministically alternates between

- (a) being “themselves”, sorting out daily, “internal” operations,

internal non-deterministically issuing

- (b) [k2] (i.e., suggesting) offers,
- (c) [k4] order confirmations,
- (d) [k7] messages to conveyors about transports and
- (e) [k8] pending collection;

external non-deterministically awaiting

- (f) [k1] queries from customers, [k3] orders, [k5] sign-off on orders, or [k12,k15] acknowledgments of merchandise transfers.

```

319. conveyor_company(cki)(me)(info)(res,co,ors,cb,pb,ckh) ≡
319a. ...
319b. [k2]    [] suggests_offer(cki)(me)(info)(res,co,ors,cb,pb,ckh)
319c. [k4]    [] confirms_offer(cki)(me)(info)(res,co,ors,cb,pb,ckh)
319d. [k7]    [] informs_conveyors(cki)(me)(info)(res,co,ors,cb,pb,ckh)
319e. [k8]    [] pending_collection(cki)(me)(info)(res,co,ors,cb,pb,ckh)
319f. [k12,k15] [] awaits_msg(cki)(me)(info)(res,co,ors,cb,pb,ckh)

```

21.2 Main Reactive Behaviour

320. The conveyor company external non-deterministic reception of messages, i.e., responses, proceed as follows:

- (a) The conveyor company awaits responses from either customers or conveyors.¹¹²
- (b) If the message
- (c) is an acknowledgment, [k12,k15], of merchandise transfers,
- (d) then the `contracts` attribute is updated accordingly and
- (e) the conveyor company resumes being so,
- (f) else the conveyor company resumes being so, with updated `current business`,

```

320. awaits_msg(cki)(me)(info)(res,co,ors,cb,pb,ckh) ≡
320a. let msg : ((koci,τ,cki),cmd)
320a. = [] { comm[cci,koci]|koci:(KI|CI)•koci∈kisUcis113 } in
320b. case msg of
320c. (ui,τ,cki),mk_Acknowledgment(τ,cnu,(ui,uj))
320d. → let co' = upd_contracts(co,mk_Acknowledgment(τ,cnu,(ui,uj))) in
320e. conveyor_company(cki)(me)(info)(res,co',ors,cb,pb,⟨msg⟩ckh) end
320f. _ → conveyor_company(cki)(me)(info)(res,co,ors,cb∪msg,pb,⟨msg⟩ckh)
320. end end

```

320d. upd_contracts: Contracts×Acknowledgment → Contracts

320d. upd_contracts(co,(τ,cnu,ft)) ≡ ⟨(τ,cnu,ft)⟩^{con}

¹¹²These responses are either [k1] customer queries, [k3] customer orders, [k5] customer order confirmation (and payment), or [k12,k15] conveyor and customer acknowledgment of merchandise transfers. Any other messages will be ignored

¹¹³*kis* and *cis* were defined in Items 186 on page 86 and 95 on page 60, respectively.

21.3 Subsidiary Behaviours

21.3.1 [k2] Suggest Offer

321. The conveyor company, with a customer query in its “in-basket”: current business, decides

- (a) to calculate an offer, commensurate with the query –
- (b) while updating the Offers and Orders attributes –
- (c) to form this offer into a commands, and to
- (d) communicate this offer to the inquiring customer,
- (e) updates its “past business” and history, and
- (f) resumes being a conveyor company.

This suggest offer action [k2] is “matched” by the query action [k1] Sect. 20.2.1.1 on page 132; cf. formula lines 315b on page 132 and 321d.

```

321.  suggests_offer(cki)(me)(info)
321.      (res,co,ors,msg:{{{cki,τ,ki},mk_CustQuery(qi,qc)}}∪cb114,pb,ckh) ≡
321a.      let offer:ConvCompOffer
321a.          = calc_offer(cki,res,co,ors,cb,pb,ckh)(mk_CustQuery(qi,ic)) in
321b.      let (res',ors') = update_res_and_ors(res,ors)(offer),
321c.          msg = ((cki,record TIME(),ki),offer) in
321d. [k2] comm[{{cki,ki}}]! msg;
321e.      let pb' = pb∪{msg}, ckh' = ⟨msg⟩^ckh in
321f.      conveyor_company(cki)(me)(info)(res',co,ors',cb,pb',ckh') end end end
320.  post: commensurate_query_offers(mk_CustQuery(ki,iq,ic),offer)

```

```

321.  commensurate_query_offers: ...
321a.  calc_offer(...) ≡ ...
321b.  update_res[ources]_and_or[der]s ...

```

21.3.2 [k4] Confirm Order

(319c) The conveyor company with an OrderOK, decides to handle that:

- (a) If the order was not OK'ed then it does nothing,
- (b) else it cashes the payment¹¹⁶ –
- (c) updates its current business and history,
- (d) and resumes being a conveyor company.

This confirm order action k4 is “matched” by the customer accepts offer action k5 Sect. 20.2.2.2 on page 133.

```

319c.  confirms_offer(cki)(me)(info)
319c.      (res,co,ors,{msg:((ki,t,cki),nok)}∪cb,pb,ckh) ≡
321a.  conveyor_company(cki)(me)(info)(res,co,ors,cb,{msg}∪pb,ckh)

319c.  confirms_offer(cki)(me)(info)
319c.      (res,co,ors,{msg:((ki,t,cki),mk_OrderOK(con,cn,pay))}∪cb,pb,ckh) ≡
321b.  [payment is registered ;]
321c.  let ors' = update_orders(co,ors)(msg), ckh' = ⟨pay⟩^ckh in
321d.  conveyor_company(cki)(me)(info)(res,co,ors',cb,pb∪{msg},ckh') end

```

¹¹⁴See footnote 107 on page 133.

¹¹⁶The receipt and registration of payments, etc., etc., is a role for the conveyor company office.

322. The `update_orders` [auxiliary] function

- (a) examines the choice identified `offer`, and
the identified choice, `tr`, and
updates the contract to now only reflect that choice.

The “stashing” of `msg` in the “past business book” serves to remind the conveyor company to – sooner or later – issue [k7]. See next!

```
322.  update_orders:  Orders → MSG → Orders
322.  update_orders(ors)((ki,t,cki),mk_OrderOK(cnu,cn,pay)) ≡
322a.      ors \ {cn} ∪ [cn → (ors(cn))(cnu)]
```

21.3.3 [k7] Conveyor Directives

(319d) “Sooner or later” the conveyor company reacts on the **orderOK** and

323. informs the one or more conveyors to be involved in the contracted transport.

- (a) If the **orderOK** was a **no** it does nothing, i.e., resumes being a conveyor company.
- (b) Else it decomposes the possibly multiple element segment list into separate conveyor company to conveyor directives,
- (c) communicates these to each involved conveyor, and
- (d) updates its history, and resumes being a conveyor company.

This conveyor directives action [k7] is “matched” by the [k5] action customer accepts offer Sect. 20.2.2.2 on page 133; cf. formula lines 316b on page 133 and 323c.

```
319d.  informs_conveyors(cki)(me)(info)
323.      (res,co,ors,cb,pb ∪ {((ki,t,cki),mk_OrderOK(cnu,chn,status))},ckh) ≡
323a.  if status = no axiom status ≠ orderOK
323a.      then conveyor_company(cki)(me)(info)(res,co,ors,cb,pb,ckh117)
323b.      else let (status,tr) = (co(con))(chn) in
323b.          let dir1 = elems construct_dirs(ki,record TIME(),cki,cnu,tr) in
323c.          {comm[{cki,ci}] ! dir:ConvDir•dir ∈ elems dir1 ∧ dir = ((cki,t,ci),_)} end end
323d.  conveyor_company(cki)(me)(info)(res,co,ors,cb,pb,(dir|dir ∈ dirs)^ckh) end
```

324. The `construct_dirs` function

- (a) from each segment from the contracted, `con` and chosen, [choice no.] `chn`, transport offer, it constructs a *convoy directive*,
- (b) and assembles into a Conveyor Company to Conveyor Directive command.
- (c) A *convoy directive* is a pair of *unload* and *load* directives.
- (d) An unload [load] directive is a quadruple of `TIME`, a node identifier, a contract number and a set of merchandise identifiers.

type

```
324c.  ConvDir = Unload × Load × [Final]118
235.  Finals = NI  $\overrightarrow{m}$  (ContractNu  $\overrightarrow{m}$  KI)
235.  Final = (NI × (ContractNu × KI)) | not_final
324d.  Load, Unload = TIME × NI × ContractNu × MI-set
```

value

```
324.  construct_dirs: KI × TIME × CKI × ContrNo × TR → ConvDir*
324.  construct_dirs(ki,t,cki,cnu,((fki,faddr),mis,sgl,(tki,taddr))) ≡
324a.      let dir1 = ⟨ extract_dir(sgl[i],con,mis,i,lensgl,tki) | i: Nat.1 ≤ i ≤ lensgl ⟩ in
324b.      ⟨ ((cki,t,ci),ConvDir(dir1[i],not_final)) | i: Nat.1 ≤ i < lensgl ⟩
324b.      ^ ⟨ ((cki,t,ci),ConvDir(dir1[lensgl],(ni,(cnu,ki)))) ⟩ end
```

¹¹⁷ **Alert:** I am not sure with what, if anything, to prefix the history with is OK. I was not ready to think about it when I wrote it, March 31, 2025, 16:01

325. The `extract_directive` function applies to a segment, contract number, a set of merchandise identifiers, the index of the segment list being examines, the length of that list, and the “end” customer identifier.

- (a) If the current index is less than the segment list, the no “final” is issued, just a pair of unload/loads.
- (b) Otherwise a final: `nj, cnu, ki`, the identifier of the last node where the contracted merchandises will be held for the customer `ki`.

type

325. `Segment = TIME × CI × NI × (EI|NI)*`

value

325. `extract_dir: Segment × ContractNu × MI-set × Nat × Nat × KI`
`→ ((UnLoad×Load)×Final)`

325. `extract_dir(sg:(t,ci,ni,enl^⟨nj⟩),cnu,mis,i,li,ki) ≡`

325a. `if i<li then (((t,ni,con,mis),(t,nj,cnu,mis)),nil)`

325b. `else (((t,ni,con,mis),(t,nj,cnu,mis)),(nj,cnu,ki)) end`

325. **pre:** the edge-node identifier list is not empty, i.e., $\neq \langle \rangle$

We apologize for the somewhat “tricky” functions: `construct_dirs` and `extract_dir`¹¹⁹.

21.3.4 [k8] Pending Collection

326. At some time conveyor companies react to customers’ [k5] order OK (accepts offer) messages

- (a) by replying with a pending collection message –
- (b) whereupon the resume being conveyor companies,

This pending collection action [k8] is “in consequence” of the [k5] action order OK (accepts offer) Sect. 20.2.2.2 on page 133.

value

326. `pending_collection(cki)(me)(info)`

326. `(res,co,ors,{((ki,τ,cki),mk_OrderOK(ni,cnu,chn,orderOK))}∪cb,pb,ckh) ≡`

326a. **let** `msg = mk_((cki,record TIME(),ki),(PendColl(ni,(cnu,mis))))` **in**

326a. `comm[{cki,ki}]! msg;`

326b. `conveyor_company(cki)(me)(info)(res,co,ors,cb,pb,⟨msg⟩^ckh) end`

¹¹⁸The type expression [T] stands for T|nil

¹¹⁹Most other function definitions are, in our opinion, straightforward

Chapter 22

Conveyor Behaviour

Contents

22.1 Earlier Treatment	141
22.2 Main Behaviour	143
22.3 Subsidiary Behaviours	144
22.3.1 Proactive Behaviours	144
22.3.1.1 [k7] Directives	144
22.3.1.2 [k10] Conveyor to Node and Edge Notifications	144
22.3.1.3 Conveyor on Edge	145
22.3.1.4 Conveyor at Node	146

22.1 Earlier Treatment

In Sect. 8.4.1 on page 66 we first treated conveyor behaviours:

Signatures then:

value

$\iota 123 \pi 66.$ conveyor: $CI \rightarrow CM \rightarrow (Kind \times Routes) \rightarrow (CurrRoute \times CPos \times CH)$ **Unit**

Behaviour, then at node:

value

$\iota 124 \pi 66.$ conveyor(ci)(cm)($k, routes$)($cr, AtNode(ni), ch$) \equiv
 $\iota 124a \pi 66.$ conveyor_change_route(ci)(cm)($k, routes$)($cr, AtNode(ni), ch$)
 $\iota 124b \pi 66.$ \parallel conveyor_remains_at_node(ci)(cm)($k, routes$)($cr, AtNode(ni), ch$)
 $\iota 124c \pi 66.$ \parallel conveyor_enters_edge(ci)(cm)($k, routes$)($cr, AtNode(ni), ch$)
 $\iota 124d \pi 66.$ \parallel conveyor_stops_at_node(ci)(cm)($k, routes$)($cr, AtNode(ni), ch$)

$\iota 125 \pi 67.$ conveyor_change_route(ci)(cm)($k, routes$)($cr, AtNode(ni), ch$) \equiv
 $\iota 125a \pi 67.$ **let** $\tau = \text{record_TIME}()$,
 $\iota 125b \pi 67.$ $ncr = \text{select_next_route}(ni, routes)$,
 $\iota 125d \pi 67.$ $ch' = \langle (\tau, ni) \rangle^{ch}$ **in**
 $\iota 125c \pi 67.$ **comm**[$\{ci, ni\}$] ! (τ, ci) ;
 $\iota 125e \pi 67.$ conveyor_at_node(ci)(cm)($k, routes$)($ncr, AtNode(ni), ch'$) **end**

$\iota 125b \pi 67.$ selects_next_route: $NI \times Routes \rightarrow CurrRoute$
 $\iota 125b \pi 67.$ selects_next_route($ni, routes$) **as** $ncr \cdot ncr \in routes \wedge \text{hd } ncr = ni$

Behaviour, then on edge:

```

l130 π68. conveyor(ci)(cm)(k, routes)
l130 π68.      (cr, mk_OnEdge(nuif, (f, e), nuir), ch) ≡
l130a π68.      conveyor_moves_on_edge(ci)(cm)(k, routes)
l130a π68.      (cr, mk_OnEdge(nuif, (f, e), nuir), ch)
l130c π68.      || conveyor_stops_on_edge(ci)(cm)(k, routes)
l130c π68.      (cr, mk_OnEdge(nuif, (f, e), nuir), ch)
l130b π68.      || conveyor_enters_node(ci)(cm)(k, routes)
l130b π68.      (cr, mk_OnEdge(nuif, (f, e), nuir), ch)

l126 π67. conveyor_remains_at_node(ci)(cm)(k, routes)(cr, AtNode(ni), ch) ≡
l126a π67.      let τ = record.TIME() in
l126b π67.      comm[{ci, ni}] ! (τ, ci);
l126c π67.      conveyor(ci)(cm)(k, routes)(cr, AtNode(ni), ⟨(τ, ni)⟩ch) end

l127 π67. conveyor_enters_edge(ci)(cm)(k, routes)(cr, AtNode(ni), ch) ≡
l127a π67.      let τ = record.TIME() in
l127b π67.      ( comm[{ci, ni}] ! (τ, ni) || comm[{ci, ni}] ! (τ, hd cr) ) ;
l127c π67.      let ei = hd cr in let {ni, ni'} = mereo_E(retr_edge(ei)(es)) in
l127c π67.      let cpos = onEdge(hd cr, (ei, (ni, f, ni), ni')) in
l127e π67.      conveyor(ci)(cm)(k, routes)(cr, cpos, ⟨(τ, ni)⟩ch) end end end end

l128 π68. conveyor_stops_at_node(ci)(cm)(k, routes)(cr, AtNode(ni), ch) ≡
l129 π68.      let τ = record.TIME() in
l129 π68.      comm[{ci, ni}] ! (τ, ci) ;
l128 π68.      stop end

```


22.2 Main Behaviour

In the context of customers and logistics and conveyor companies, as illustrated by Fig. 17.1 on page 110, conveyors, i.e., their behaviour, are a bit more intricate !

327. Conveyors non-deterministically alternates between

(a) being themselves,

or external non-deterministically receiving

(b) [k7] directives from conveyor companies – their own or other,

(c) and then handling these messages,¹²⁰

and internal non-deterministically sending messages

(d) [k10] notifying edges and nodes of their presence,

(e) [k12] and acknowledgments of transfer of merchandises from and to customers and nodes.

When not responding to and handling messages from other behaviours ([k7] conveyor companies, or [k9] customers),

(f) a conveyor is either at a node, possibly unloading or loading merchandises, or

(g) along, i.e., on, an edge.

```

327.    conveyor(ci)(cm:(uis,ckis,kis,cis))(k,...)
327.    (stow,tbu,tbl,sr,idx,finals,pos,ch) ≡
327a.    ... conveyor(ci)(cm:(uis,ckis,kis,cis))(k,...)
327a.    (stow,tbu,tbl,sr,idx,finals,pos,ch)
327b.    [k7]  || let msg [] { comm[{ci,cki}]? | cki∈ckis } in
327c.    conv_dir_handling(ci)(uis,ckis,kis,cis)(k,...)
327c.    (stow,tbu,tbl,sr,idx,finals,pos,<msg>^ch)(msg) end
327d.    [k10] || conv_node_notification(ci)(uis,ckis,kis,cis)(k,...)
327d.    (stow,tbu,tbl,sr,idx,finals,pos,ch)
327d.    [k10] || conv_edge_notification(ci)(uis,ckis,kis,cis)(k,...)
327d.    (stow,tbu,tbl,sr,idx,finals,pos,ch)
327e.    [k12] || conv_comp_ack(ci)(uis,ckis,kis,cis)(k,...)
327e.    (stow,tbu,tbl,sr,idx,finals,pos,ch)
327f.    || conv_at_node(ci)(uis,ckis,kis,cis)(k,...)
327f.    (stow,tbu,tbl,sr,idx,finals,pos,ch)
327g.    || conv_on_edge(ci)(uis,ckis,kis,cis)(k,...)
327g.    (stow,tbu,tbl,sr,idx,finals,pos,ch)

```

¹²⁰**Note:** This is the only message received by conveyors from contracting conveyor companies in this, the present transport domain model. For more realistic transport domain models there will, of course, be other such messages – but they deal, not with the *intrinsic* facets of transport (logistics) but with technology support, management & organization, human, and other facets – cf. Chapter 8 of my book [22].

22.3 Subsidiary Behaviours

22.3.1 Proactive Behaviours

22.3.1.1 [k7] Directives

328. The `conv_directive_handling` behaviour for handling conveyor company to conveyor directives

- (a) updates the to-be-unloaded, the to-be-loaded and the finals attributes, and
- (b) resumes being a conveyor.

This conveyor directives handling action k7 is “matched” by the informs conveyors action k7 Sect. 21.3.3 on page 138; cf. formula lines 328 and 323c on page 138.

```

328. [k7] conv_dir_handling(ci)(me)(k,r)
328.      (stow,tbu,tbl,sr,idx,finals,pos,ch)
328.      ((cki,t,ci),ConvDir((t',ni,cnu,mis),(t'',nj,cnu,mis)),final) ≡
328a.      let tbu' = tbu ∪ [nj→tbu∪{cnu}],      [we disregard t,t',t'']
328a.      tbl' = tbl ∪ [ni→tbl∪{cnu}],      [we disregard t,t',t'']
328a.      finals' = upd_finals(finals,final) in
328b.      conveyor(ci)(me)(k,r)(stow,tbu',tbl',sr,idx,finals',pos,⟨dirs⟩^ch) end

328a.      upd_finals(finals,(ni,cnu,ki)) ≡ finals ∪ [ni→[ki→cnu]]

```

22.3.1.2 [k10] Conveyor to Node and Edge Notifications

329. Conveyor notify the edges and nodes along which it is moving:

- (a) either at a node,
- (b) or on an edge.

This `conv_node_notification` action k10 is “matched” by the node action k10 Sect. 25.3 on page 154; cf. formula lines 329a and 341b on page 154.

value

```

329. conv_node_notification(ci)(uis,ckis,kis,cis)(k,...)
329.      (stow,tbu,tbl,sr,idx,finals,mk_AtNode(ni),ch) ≡
329a.      let msg = ((ci,record TIME(),ni),mk_AtNode(ni)) in
329a. [k10] comm[{ci,ni}]! msg ;
329.      conveyor(ci)(uis,ckis,kis,cis)(k,...)
329.      (stow,tbu,tbl,sr,idx,finals,mk_AtNode(ni),⟨msg⟩^ch) end

```

This `conv_edge_notification` action k10 is “matched” by the edge action k10 Sect. 25.3 on page 154; cf. formula lines 329b and 341d on page 154.

```

329. conv_edge_notification(ci)(uis,ckis,kis,cis)(k,...)
329.      (stow,tbu,tbl,sr,idx,finals,pos:mk_OnEdge(⟨_,(⟨_,ei⟩),_⟩),ch) ≡
329b.      let msg = ((ci,record TIME(),ei),mk_OnEdge(ei)) in
329b. [k10] comm[{ci,ei}]! msg ;
329.      conveyor(ci)(uis,ckis,kis,cis)(k,...)
329.      (stow,tbu,tbl,sr,idx,finals,pos,⟨msg⟩^ch) end

```

¹²⁰The `ci` is that of the conveyor

¹¹⁹The two formal argument occurrences of `ci`, respectively `cki`, must be pairwise identical! See also the next `conv_msg_handling` definitions.

22.3.1.3 Conveyor on Edge

Conveyor on Edge – Then:

```

1130  $\pi$ 68.    conveyor(ci)(cm)(k,routes)(cr,mk_OnEdge(nuif, (f,e),nuit),ch)  $\equiv$ 
1130a  $\pi$ 68.    conveyor_moves_on_edge(ci)(cm)(k,routes)(cr,mk_OnEdge(nuif, (f,e),nuit),ch)
1130c  $\pi$ 68.     $\sqcap$  conveyor_stops_on_edge(ci)(cm)(k,routes)(cr,mk_OnEdge(nuif, (f,e),nuit),ch)
1130b  $\pi$ 68.     $\sqcap$  conveyor_enters_node(ci)(cm)(k,routes)(cr,mk_OnEdge(nuif, (f,e),nuit),ch)

```

We leave it to the reader, this time, to review the functions: `conveyor_moves_on_edge` Sect.8.4.1 items 131 on page 68 etc., `conveyor_stops_on_edge` Sect.8.4.1 items 133 on page 69 etc. and `conveyor_enters_node` Sect.8.4.1 items 132 on page 69 etc.

• • •

Conveyor on Edge – Now:

330. An edge [behaviour] at an edge external non-deterministically either:

- (a) **moves** along the edge, a fraction “at a time”, or
- (b) **stops** on the edge and thereby “leaves” transport; or
- (c) **enters** a node.

```

330.    conveyor_on_edge(ci)(me:(uis,ckis,kis,cis))(k,len,cost)
330.    (stow,tbu,tbl,sr,idx,finals,mk_OnEdge((fni,(ej,f),tni)),ch)  $\equiv$ 
330a.     $\sqcap$  conveyor_moves_on_edge(ci)(me:(uis,ckis,kis,cis))(k,len,cost)
330a.    (stow,tbu,tbl,sr,idx,finals,mk_OnEdge((fni,(ej,f),tni)),ch)
330b.     $\sqcap$  conveyor_stops_on_edge(ci)(me:(uis,ckis,kis,cis))(k,len,cost)
330b.    (stow,tbu,tbl,sr,idx,finals,mk_OnEdge((fni,(ej,f),tni)),ch)
330c.     $\sqcap$  conveyor_enters_node(ci)(me:(uis,ckis,kis,cis))(k,len,cost)
330c.    (stow,tbu,tbl,sr,idx,finals,mk_OnEdge((fni,(ej,f),tni)),ch)

```

The next behaviour is “patterned” over Items 131a– 131e on page 68.

331. A conveyor which is moving along an edge, some fraction down the edge/road/track/route, but not “yet” near “the end”:

- (a) at time τ ,
- (b) increments the fraction of its position
- (c) (while updating its history)
- (d) notifying the edge [behaviour]
- (e) [technically speaking] adjusting its position], and, finally,
- (f) resuming being a thus updated conveyor [OnEdge.

```

331.    conveyor_moves_along_edge(ci)(me)(__,__,__)
331.    (stow,tbu,tbl,sr,idx,finals,mk_OnEdge((fni,(ej,f),tni)),ch)  $\equiv$ 
331a.    let  $\tau = \text{record\_TIME}()$ ,  $\varepsilon: \text{Real} \cdot 0 < \varepsilon \ll 1$  in
331b.    let  $f' = f + \varepsilon$ ,  $\text{cpos} = \text{mk\_OnEdge}(n_{ui_f}, (f',e), n_{ui_t})$  in
331c.    let  $\text{ch}' = \langle (\tau, ci) \rangle^{\wedge} \text{ch}$  in
331d.    comm[{ci,ej}]!  $(\tau, ci)$  ;
331e.    conveyor(ci)(me)(__,__,__)
331f.    (stow,tbu,tbl,sr,idx,finals,mk_AtNode(tni),ch) end end end
331.    pre:  $f \simeq 1 \wedge \text{sr}(\text{idx}) = \text{tni}$ 

```

332. A conveyor may, “surreptitiously” as it were, “decide” to stop being a conveyor altogether!

```
332. conveyor_stops_on_edge(ci)(me:(uis,ckis,kis,cis))(k,len,cost)
332.      (stow,tbu,tbl,sr,idx,finals,mk_OnEdge((fni,(ej,f),tni)),ch) ≡ stop
```

333. A conveyor enters a node

- (a) at time τ , by altering its position,
- (b) notifying both edge and node behaviours,
- (c) and resumes being a conveyor.

```
333. conveyor_enters_node(ci)(me)(__,__,__)
333.      (stow,tbu,tbl,sr,idx,finals,mk_OnEdge(fni,(ej,1),tni),ch) ≡
333.      let  $\tau = \text{record TIME}()$  in
333a.      (comm[{ci,ej}]! ( $\tau$ ,ci)||comm[{‘tau’,tni}]! ( $\tau$ ,ci)) ;
333b.      conveyor(ci)(me)(__,__,__)
333b.      (stow,tbu,tbl,sr,idx,finals,mk_atNode(tni),(( $\tau$ ,mk_atNode(tni)))ch) end
```

22.3.1.4 Conveyor at Node

Conveyor at Node – Then:

value

```
i124  $\pi$ 66. conveyor(ci)(cm)(k,routes)(cr,mk_AtNode(ni),ch) ≡
i124a  $\pi$ 66.      conveyor_change_route(ci)(cm)(k,routes)(cr,mk_AtNode(ni),ch)
i124b  $\pi$ 66.      || conveyor_remains_at_node(ci)(cm)(k,routes)(cr,mk_AtNode(ni),ch)
i124c  $\pi$ 66.      || conveyor_enters_edge(ci)(cm)(k,routes)(cr,mk_AtNode(ni),ch)
i124d  $\pi$ 66.      || conveyor_stops_at_node(ci)(cm)(k,routes)(cr,mk_AtNode(ni),ch)
```

• • •

Conveyor at Node – Now:

A primary “business” of a conveyor at a node is to unload and load merchandises.

334. In general, a conveyor at a node internal non-deterministically “alternates” between

- (a) **unloading** merchandises,
- (b) **loading** merchandises,
- (c) **stopping** altogether, and
- (d) **entering** a next edge – if not the end of the conveyor route –
– an in these cases resuming being a conveyor.

```
334. conveyor_at_node(ci)(uis,ckis,kis,cis)(k,...)
334.      (stow,tbu,tbl,sr,idx,finals,mk_AtNode(ni),ch) ≡
334a.      conveyor_unloads_merch(ci)(uis,ckis,kis,cis)(k,...)
334a.      (stow,tbu,tbl,sr,idx,finals,mk_AtNode(ni),ch)
334b.      || conveyor_loads_merch(ci)(uis,ckis,kis,cis)(k,...)
334b.      (stow,tbu,tbl,sr,idx,finals,mk_AtNode(ni),ch)
334c.      || conveyor_stops_at_node(ci)(uis,ckis,kis,cis)(k,...)
334c.      (stow,tbu,tbl,sr,idx,finals,mk_AtNode(ni),ch)
334d.      || conveyor_enters_edge(ci)(uis,ckis,kis,cis)(k,...)
334d.      (stow,tbu,tbl,sr,idx,finals,mk_AtNode(ni),ch)
```

335. Conveyors unload (deliver), onto the node they are at,

- (a) from their stowage, the one-or-more contracted merchandises, for that node,
- (b) [k11a] and communicates these to that node,
- (c) [k12a] and acknowledges that to the contracting conveyor companies.
- (d) For final ‘unloads’, if any, receiving customers
- (e) are informed of pending delivery.
- (f) Whereupon the conveyor resumes being a conveyor at that node.

value

```

335. conveyor_unloads_merch(ci)(uis,ckis,kis,cis)(k,...)
335.      (stow,tbu,tbl,sr,idx,finals,mk_AtNode(ni),ch) ≡
335a.      let unls = tbu(ni), stow' = stow\{ni} in
335b.      [k11a] comm[{ci,ni}]! mk_CNTransfer(stow/unls)120
335c.      [k12a] || {comm[{ci,xtr_CKI(ci)]! mk_Acknowledgment(record TIME(),cnu,(ci,ni))
335c.      | cnu:ContractNu•cnu∈unls } ;
335d.      if ni∉dom finals
335d.      then skip
335e.      else { let cnu=(finals(ni))(ki), mis=(tbu(nu))(cnu) in
335e.      [k13]      comm[{ci,ki}]! mk_PendDeliv(ni,(cnu,mis)) ;
335e.      | ki:KI•ki∈dom finals(ni) end }
335d.      end
335f.      conveyor_unloads_merch(ci)(uis,ckis,kis,cis)(k,...)
335f.      (stow',tbu\{ni},tbl,sr,idx,finals\{ni},mk_AtNode(ni),⟨v⟩^ch) end

```

Alert: Fix **v**: CNTransfer(unls) ?

336. Conveyors load (fetch)

[from the node they are at, onto their stowage]

contracted merchandises:

- (a) if there are merchandises to
- (b) load these
- (c) communicate them to the node
- (d) and the contracting conveyor company notified.
- (e) otherwise nothing is done;
- (f) and the conveyor resumes being a conveyor at that node.

value

```

336. conveyor_loads_merch(ci)(uis,ckis,kis,cis)(k,...)
336.      (stow,tbu,tbl,sr,idx,finals,mk_AtNode(ni),ch) ≡
336a.      if ni∈dom tbl
336b.      then let lds = tbl(ni), cki = xtr_CKI(cnu) in
336c.      comm[{ci,ni}]! mk_NCTransfer(lds) ;
336d.      comm[{ci,cki}]! mk_Acknowledgment(cnu,(ci,ni)) end
336a.      else skip end
336f.      conveyor_loads_merch(ci)(uis,ckis,kis,cis)(k,...)
336f.      (stow,tbu,tbl\{ni},sr,idx,finals,mk_AtNode(ni),⟨load⟩^ch)

```

Alert: Check for proper **load** onto ch

¹²⁰The value of stow/unls is that of stow [domain-]restricted to unls.

The next behaviour:

value

```
conveyor_stops_at_node(ci)(uis,ckis,kis,cis)(k,...)
      (stow,tbu,tbl,sr,idx,finals,mk_AtNode(ni),ch)  $\equiv$  stop
```

is a “mere” transcription” of the similarly named behaviour of Sect. 8.4.1 on page 66, items 133 on page 69-...

337. Finally, the conveyor may [be ready to] leave the node for possibly continuing its journey.

- (a) If the conveyor is at the end of its current service route, sr,
- (b) then
- (c) it reverts sr, into rs,
- (d) which defines the next **mk_onEdge**(fni,(0,ei),tni) elements,
- (e) and the conveyor continues being a conveyor, on that edge.
- (f) Otherwise
- (g) the next **mk_onEdge**(fni,(0,ei),tni) elements, are defined by the current service route, sr,
- (h) and the conveyor continues being a conveyor, on that edge.

```
337. conveyor_enters_edge(ci)(me)(k,...)
337. (stow,tbu,tbl,sr,idx,finals,mk_AtNode(ni),ch)  $\equiv$ 
337a. if idx = len sr
337b. then
337c.   let rs = revert(sr) in
337d.   let fni = rs[1], ei = rs[2], tni = rs[3] in
337d.   let e = mk_onEdge(fni,(0,ei),tni) in
337e.   conveyor(ci)(me)(k,...)
337e.   (stow,tbu,tbl,rs,1,finals,e,(e)^ch) end end end
337f. else
337g.   let fni = sr[idx], ei = sr[idx+1], tni = sr[idx+3] in
337h.   let e = mk_onEdge(fni,(0,ei),tni) in
337h.   conveyor(ci)(me)(k,...)
337h.   (stow,tbu,tbl,sr,idx+1,finals,e,(e)^ch) end end
337f. end

337c. revert: Path  $\rightarrow$  Path
337c. revert(p)  $\equiv$ 
337c.   case p of
337c.      $\langle \rangle \rightarrow \langle \rangle$ ,
337c.      $r^{\langle u \rangle} \rightarrow \langle u \rangle^{\text{revert}(q)}$ 
337c.   end
```

The above reflects but one choice for continuing a conveyor once it has “exhausted” its current service route. Others can be thought of.

Chapter 23

Logistics Company Behaviour

We skip this chapter: the conveyor company behaviour “says it all !”.

Chapter 24

Edge Behaviour

Contents

24.1 Earlier Treatment	151
24.2 Main Behaviour	151

24.1 Earlier Treatment

value

123. edge: $EI \rightarrow EM \rightarrow (Kind \times LEN \times Cost) \rightarrow NH \rightarrow \mathbf{Unit}$

value

136. edge: $EI \rightarrow EM \rightarrow (EdgeKind \times LEN \times Cost) \dots \rightarrow EH$
136a. $edge(ei)(em)(ekind, len, cost)(eh) \equiv$
136b. **let** **msg** = $\square \{ \mathbf{comm}[\{ei, ci\}] ? \mid ci:CI \cdot ci \in em \}$ **in**
136c. $edge(ni)(em)(eki\dots)(\langle \mathbf{msg} \rangle^{eh})$ **end**

24.2 Main Behaviour

338. An edge behaviour revolves around:

- (a) conveyors moving along, being so notified by messages which it remembers by “adding” them to their histories,
- (b) before resuming being adge behaviours.

338. $edge(ei)(em)(ekind, len, cost)(eh) \equiv$
338a. **let** **msg** = $\square \{ \mathbf{comm}[\{ei, ci\}] ? \mid ci:CI \cdot ci \in em \}$ **in**
338b. $edge(ei)(em)(ekind, len, cost)(\langle \mathbf{msg} \rangle^{eh})$ **end**

That is, no change !

Chapter 25

Node Behaviour

Contents

25.1 Earlier Treatment	153
25.2 Revised Node Attributes	153
25.3 [k10,k11,k14] Main Behaviour	154

25.1 Earlier Treatment

value

```
l135  $\pi$ 69.   node:  NI  $\rightarrow$  NM  $\rightarrow$  NodeKind  $\rightarrow$  NH
l135a  $\pi$ 69.   node(ni)(nm)(nkind)(nh)  $\equiv$ 
l135c  $\pi$ 69.     let msg =  $\sqcap \{ \text{comm}[\{ni,ci\}] ? \mid ci:CI \bullet ci \in nm \}$  in
l135d  $\pi$ 69.     node(ni)(nm)(nkind)( $\langle \text{msg} \rangle^{nh}$ ) end
```

25.2 Revised Node Attributes

339. Each node may potentially provide [also] as a temporary “on-hold” storage for customer merchandises.

type

```
339.   OnHold = ContractNu  $\xrightarrow{m}$  M-set
```

value

```
339.   attr_OnHold:  N  $\rightarrow$  OnHold
```

25.3 [k10,k11,k14] Main Behaviour

340. Node behaviours revolves around:

341. nodes external non-deterministically accepting messages from conveyors where these messages are

- (a) [k10] either notifications of the presence of (moving) conveyors – duly recorded in the node history attribute;
- (b) [k11a] or from conveyors unloading at nodes duly updated in the node onhold and history attributes;
- (c) [k11b] or from conveyors loading at nodes
- (d) [k12] and informing the “originating” conveyor company,
– in which latter case
- (e) the merchandises identified in the load are communicated (“back”) to the conveyor.

or non-deterministically externally receiving requests from customers to

342. to deliver contracted onhold merchandises,

```

340. node(ni)(nm:(eis,kis,cis))(nkind)(onhold,nh) ≡
341.   let msg = [] {comm[{ni,ci}]?|ci:CI·ci∈cis} in
341.   case msg of

341a. [k10]   (_,mk_AtNode(ni))
341a.       → node(ni)(nm)(nkind)(onhold,<msg>^nh),

341b. [k11a] ((ci,τ,ni),mk_CNTransfer(cnu,lds)) [cf. 335b on page 147]
341b.       → node(ni)(nm)(nkind)(onhold∪lds121,<msg>^nh),

341c.       ((ci,τ,ni),mk_NCTransfer(cnu,mis)) [cf. 336a on page 147]
341d. [k12]   → let ms = {m:M|m∈onhold(cnu)∧uid_M(m)∈mis} in
341e. [k11b]   comm[{ni,ci}]!mk_NCTransfer([cnu→ms]) ;
341d.       node(ni)(nm)(nkind)(onhold\cnu,<msg>^nh) end

341.   end end

342.   [] let msg:mk_PendColl(ni,(cnu,mis)) = [] {comm[{ni,ki}]?|ki:KI·ki∈kis} in
342.   let ms = {m|M·m∈onhold(cnu)∧uid_M(m)∈mis} in
342.   let τ = record TIME() in
342.   msg = ((ni,τ,ki),mk_NKTransfer(ms)) in
342. [k14] comm[{ni,ki}]! msg ;
342.   node(ni)(nm)(nkind)(onhold\cnu,<((ni,τ,ki),ms_to_mis(ms))>^nh) end end
135.   end

```

¹²⁰ domlds ∩ domonhold = {}

¹²¹ **Alert:** Fic unls; one or more !?

Part IX

CLOSING

Chapter 26

Discussion

Contents

26.1 Wither Logistics Companies	157
26.2 Some Parts Modelled, Others Not!?	158
26.3 Formal Structuring	159
26.4 Mnemonics	159
26.5 Narratives	159

26.1 Wither Logistics Companies

It was a mistake, it seems, to distinguish between conveyor and logistics companies. A conveyor company with no conveyors is a logistics company. Examples are travel agencies. A revised taxonomy for conveyor companies is as shown in Figs. 26.1 and 26.2 on the next page. They are revisions of Figs. 13.1 on page 90 and 10.1 on page 75.

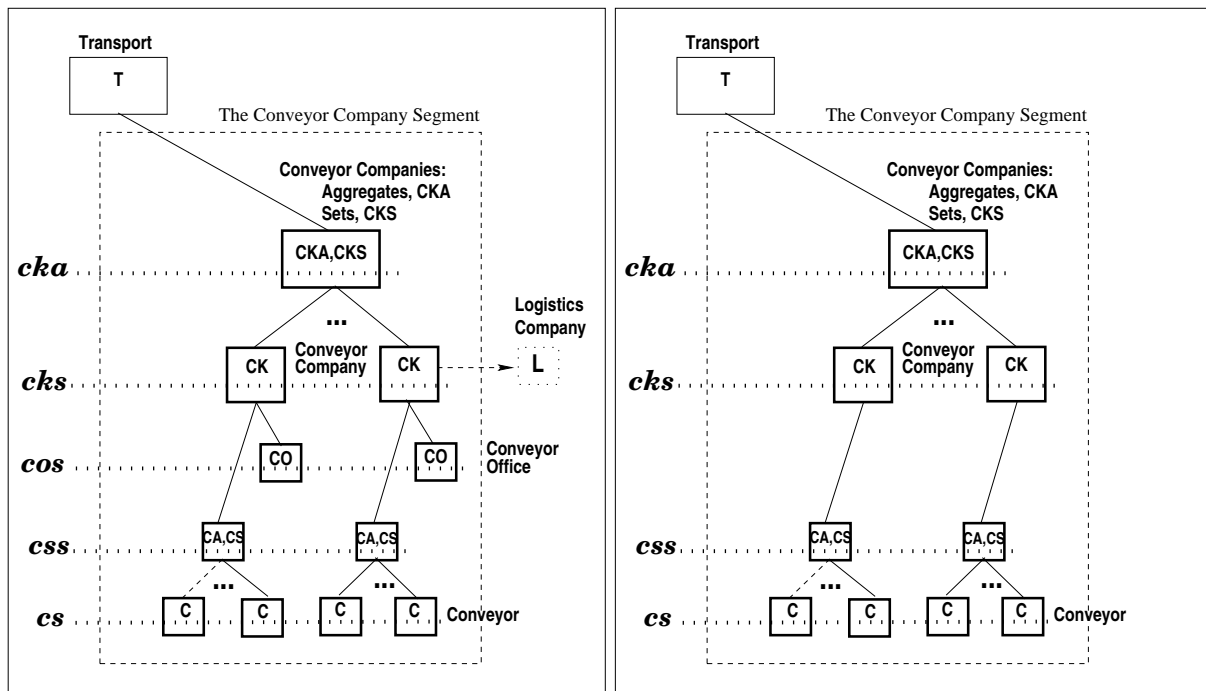


Figure 26.1: Old and Revised Conveyor Company Taxonomies

The corresponding Command & Material Traces figures is Fig. 26.3 on the following page:

MORE TO COME

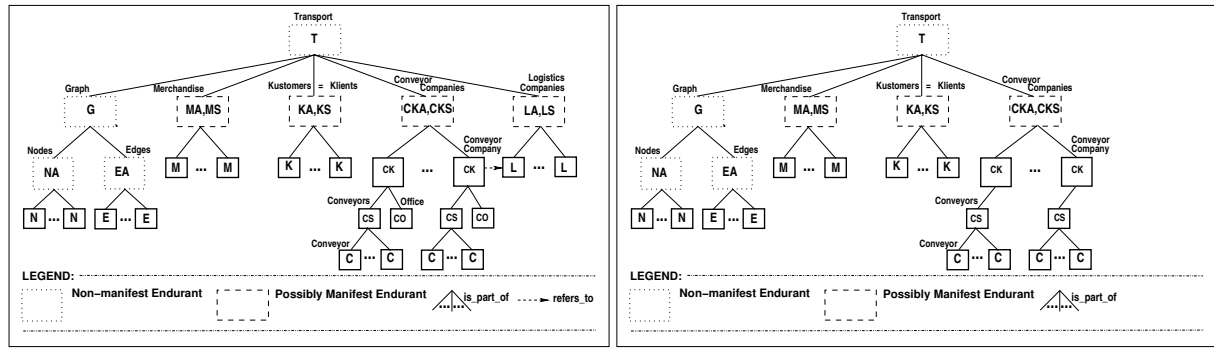


Figure 26.2: Old and Revised Transport Taxonomies

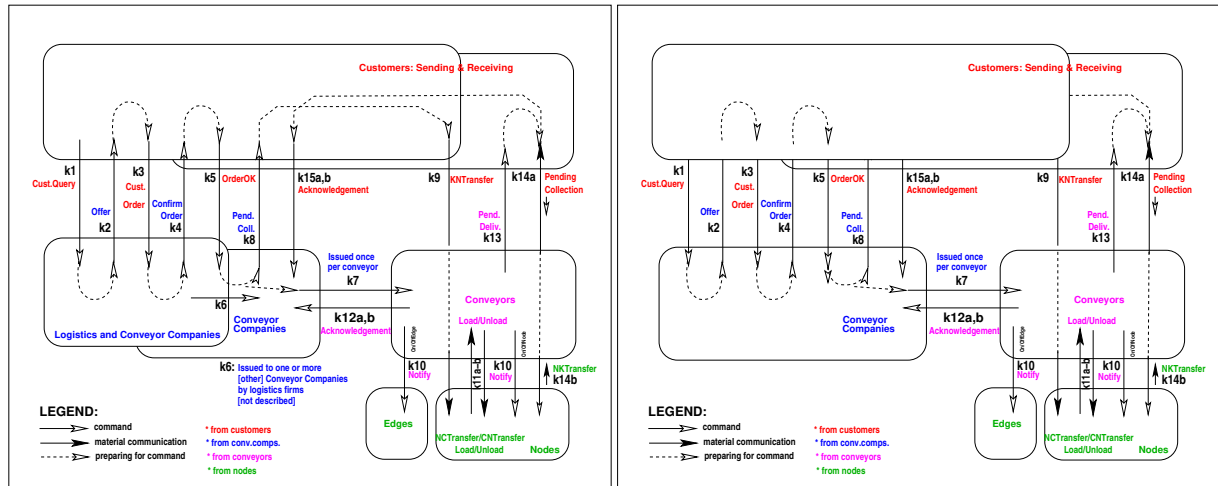


Figure 26.3: Old and Revised Command & Material Traces [→]

26.2 Some Parts Modelled, Others Not ! ?

The reader will have observed that we model only some of the internal qualities of composite parts ! Why ? Well the answer is this: We have chosen to emphasize the modelling of essential aspects of transport. The “omitted” full modelling of some, well most, composite parts [endurants], and hence their behaviours [perdurants], is therefor motivated as follows:

- **Graphs:** With G, EA and NA we do not associate any manifest “authority”. But we could ! ? With G we could associate such more-or-less public authorities as the road authorities of Your city or country, rail net authorities, coastal and sea authorities, air traffic command & control, incl. *ICAO*¹²², etc.
- **Merchandise Aggregate:** With MA we also do not associate any manifest “authority”. But we could ! ? There are an abundance of private/public association which monitor and control publically available merchandise categories: food, toy, automobile, etc., agencies.
- **Customer Aggregate:** With KA we do not associate any manifest “authority”. But we could ! ? We leave it to the reader to identify possibly relevant such candidates !
- **Conveyor Companies Aggregate:** With CKA we do not associate any manifest “authorities”. But we could ! ? There are public/private associations which handle concerns of the conveyor industry, one or more for each *kind*. We omit their modelling.
- **Logistics Companies Aggregate:** With LA we do not associate any manifest “authorities”. We could ! ? But we do not.

¹²²<https://www.icao.int/about-icao/Pages/default.aspx>

26.3 Formal Structuring

By *formal structuring* we mean the way we have chosen some endurant parts to be composite, i.e., Cartesians and sets of parts. This structuring is most clearly reflected in Fig. 10.1. We now regret the “messy” handling of logistics, both as separate parts, and as an element of conveyor companies. A better “decomposition” must be found in a continuation project. There are other, in our mind, minor, such restructurings to be made.

26.4 Mnemonics

Mnemonics is the study and development of systems for improving and assisting the memory¹²³. One such system is naming. We have strived some “logic” in choosing names. Endurant parts have been given very short one, two or three letter identifiers. Commands, functions and behaviours have been assigned longer identifiers, trying to compress their full names in the informal texts. A careful review, for any possible continuation project should carefully review these latter names.

26.5 Narratives

All (or almost all) **formulas** have been preceded by **narratives**. Pairwise their numbering “match” ! But these narratives are, in our mind, far from satisfactory. Much more care should be taken in formulating and “repetitively” express these narratives. Perhaps one should serve two narratives for each one presented here ? One, short, coupled with and receding the formulas; another, longer, perhaps appearing as footnotes, or as notes in a separate appendix ?

¹²³<https://languages.oup.com/google-dictionary-en/> and <https://dictionary.cambridge.org/dictionary/english/mnemonic>

Chapter 27

Conclusion

Contents

27.1	Logistics & Operations Research	161
27.1.1	Logistics	161
27.1.2	Operations Research	161
27.2	Interpretations	161
27.2.1	Socio-Economic Study	162
27.2.2	Business Process Re-Engineering	162
27.2.3	Primary and Secondary School Topic	162
27.2.4	Algorithms & Data Structures	162
27.2.5	Software System Development	162
27.3	Formality and Verification	163
27.4	On the Development of This Model	164
27.5	Acknowledgements	164

Chapters 3–25 (pages 45–154) sketched a “strict” narrative coupled to a formal description of an essence of transport domains. These were engineering descriptions. Your understanding of these rely on Your having understood [30, 26, 22, 20, 17].

27.1 Logistics & Operations Research

As for ‘logistics companies’: Yes, I have left them out.

27.1.1 Logistics

343. By *logistics* we shall mean *the detailed planning of the organization and implementation of a complex operation*..

In this report logistics, in this sense of *planning* has been concentrated in the function `cal_offer`, cf. Item 321a on page 137.

27.1.2 Operations Research

That is: the often exciting and beautiful properties of optimization algorithms are to be “buried” here. They do not belong to the ‘transport’ aspects – but to the *strategic, tactical an operational facets* of the transport domain¹²⁴.

27.2 Interpretations

The domain description of Sects. 3–19 (pages 45–154) can be viewed in three ways:

- (i) as a step in the general, say socio-economic study of a specific infra-structure [sub-]domain;
- (ii) as a prerequisite for *business process re-engineering*;

¹²⁴Cf. Sect.8.7, Example 107, pages 232–233 of my book [22].

- (iii) as an, albeit, in this case, and this stage of unfolding study, basis document for preparing teachers material for subsequent development, i.e., writing, of secondary school course element for teaching such specific infra-structure [sub-]domains; and
- (iv) as an initial feasibility study for possible subsequent development of software for multi-mode transport systems.

We shall now comment on each of these.

27.2.1 Socio-Economic Study

TO BE WRITTEN

27.2.2 Business Process Re-Engineering

TO BE WRITTEN

27.2.3 Primary and Secondary School Topic

We should like to see reports on the study, analysis and description of several societal infrastructure components:

- **the banking system**, from Your local, “brick and mortar” branch office via its head quarter, the national bank of Your country¹²⁵, the regional bank of your continent to The World Bank¹²⁶ and the IMF¹²⁷;
- **the insurance industry**;
- **the health care industry**, from Your family doctor, via local clinics, to hospitals – with pharmacies, home care and health insurance providers included;
- **the education system**, from primary and secondary schools, to high schools, colleges and universities;
- **et cetera !**

27.2.4 Algorithms & Data Structures

Many functions, like `get_offers`, imply, for their software realization, rather complex data structures and intricate algorithms. Since we are describing domains, and not designing software. we need, in a sense, not be concerned. But we have achieved, one might say, a clear identification, of where such clever software designs may be warranted.

27.2.5 Software System Development

This study and experimental report began with espousing **The Triptych Dogma**. But we have advocated that domain modelling be used for other purposes than “just” software development. Now we “*return to the fore*” ! We now assume that there is, indeed, to be professionally & commercially, at least in a seriously funded effort, to be developed actual software for essential aspects of transport as they have been laid out in this study and experimental report. How would we go about doing that ?

Based on more than 40 years of experience¹²⁸ we would do as follows:

- First we would, as we have already started doing, perform the three phases of so-called ‘ ‘SEA’ ’ preparatory work.
 - **Study**,
 - **Analyze**, and
 - **Experiment**.

We have just, more-or-less, completed these three phases.

- Now we are ready for a project committed to produce a “full-blown” domain model.

¹²⁵<https://www.nationalbanken.dk/en>

¹²⁶<https://www.worldbank.org/ext/en/home>

¹²⁷<https://www.imf.org/en/Home>

¹²⁸We refer to the Dansk Datamatik Center’s [36] CHILL and Ada projects [39]

- After that, the similar development of a requirements prescription.
- And after that, the development of a software design, is coding, validation, etc.

How would we organize the “full-blown” domain modelling

- First we would assemble, in this case, six people, well-familiar with the domain modelling approach pursued in this report.
- They would be organized with the following responsibilities – being responsible for the development of:
 - the transport net, i.e., graph, model – 1 person;
 - the conveyor model – 2 persons;
 - the merchandises model – 1 person; and
 - the logistics and conveyor companies model – 2 persons.

All under the leadership of an overall domain modelling “architect” !

They would each have “an own”, private and “inviolable” office. After a very few days of domain modelling they would

- each morning review the previous day’s work of a colleague, on a rotating shift basis, a “new colleague” on consecutive days;
- meet around a coffee/tea machine and a white board mid-morning for the possible discussion of common issues – across their modelling – while also handing back the possibly annotated work of their reviewed colleague;
- go back to correcting possible collegial remarks;
- and otherwise continue their main assigned work !

27.3 Formality and Verification

Jean-Raymond Abrial¹²⁹ passed away 26 May 2025. He was one of the greats of our science. His contributions, especially through *Z*, *B* and *The B Methods* [2, 1] to *construction by proof* are seminal.

So where, in our description, do we find “traces” of that ?

The answer is: nowhere !

Why ?

Well, usually proof of program correctness is usually [carried out] with respect to some property, some “prior” specification. For domains there is no prior “specification” ! There is the manifest reality of the subject domain.

Thus we must first specify, i.e., describe that domain.

A domain description, a domain model, cannot be said to be correct.

It is either a bad, or a not so bad, or not quite so “approximate” a description as to be accepted by domain stakeholders; or it is a reasonably good model.

Verification of a domain model is by its acceptance by domain stakeholders.

When, below, we refer to verification we mean that properties of the description can be expressed, in mathematical logic and then formally proved: verified, tested, checked !

• • •

But: But the above is not good enough ! Certainly J.R. Abrial’s work must or ought apply here !? A study should be made, by professionals well-familiar with, for example, Event B¹³⁰. Based on the description/modelling taxonomy, cf. Fig. 3.1, it might very well be possible to formulate the formal model along the principles set out by J.R. Abrial

• • •

The next remarks were written before the J.R. Abrial discourse above.

• • •

¹²⁹https://en.wikipedia.org/wiki/Jean-Raymond_Abrial

¹³⁰<https://www.event-b.org/>, <https://www.southampton.ac.uk/~tsh2n14/publications/chapters/eventb-dbook13.pdf>

The reader may well have observed two aspects of our “formal” model:

- (i) **“Formality” of the Specification:** I have been rather “lax”, some would say, in my use for RSL. An example is “trick”, referred to in footnote 107 on page 133, and used in several formal parameter of behaviours. Other examples is the use of discriminated union of `:-`-defined command types. These “lax” uses have been done, deliberately, in the interest of shortening the formulas. They can all be edited into “correct” RSL.
- (ii) **Lack of Verification:** Yes, indeed. I have not been as careful as I would wish, to highlight all the places where appropriate **theorems** should be enunciated, let alone proved. Similarly for **axioms**. I trust the reader can spot these places. And I trust that appropriate proofs be provided. Not necessarily formal proofs in the sense of there being a proof system for the RSL for all of these cases: there is not. But then I am “almost” sure that classical proofs, such as mathematicians “always” do, can suffice. And, for cases that that is not immediately possible? Well, great, then this domain description provides rich possibilities for the able computer scientist to excel!

27.4 On the Development of This Model

I started on this document on Saturday February 22, 2025. I finished, “more-or-less” all the formalization and this concluding section on Monday March 3, 2025. Nine days, Nine days of great fun.

I am not really ashamed to confess that other than the RSL formula text editing system I have not had access to proper RSL tools, such as they indeed do exist. Thus I have not been able to more-or-less automatically check my RSL formulas. Et cetera - et cetera!

During the development many model-formulations changed. Figure 17.1 on page 110, for example, underwent numerous versions.

27.5 Acknowledgements

Chapter 28

Bibliography

- [1] 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.
- [2] Jean-Raymond Abrial. From Z to B and then Event B: Assigning Proofs to Meaningful Programs. In *IFM 2013*, LNCS 7940, Åbo, Finland, June 2013. Springer.
- [3] J. L. Austin. *How to Do Things with Words*. Harvard University Press, Cambridge, Mass., 2 edition, 1975. (William James Lectures).
- [4] H. Bekič, D. Bjørner, W. Henhapl, C.B. Jones, and P. Lucas. A Formal Definition of a PL/I Subset. Technical Report 25.139, Vienna, Austria, December 1974.
- [5] Hans Bekič, Peter Lucas, Kurt Walk, and Many Others. Formal Definition of PL/I, ULD Version III. IBM Laboratory, Vienna, 1969.
- [6] D. Bjørner and O. Oest. *Towards a Formal Description of Ada*, volume 98 of *LNCS*. Springer-Verlag, 1980.
- [7] Dines Bjørner. Domain Case Studies:
 - 2025: *Documents – a Domain Description*, Winter/Spring 2025, www.imm.dtu.dk/dibj/2025/documents/main.pdf
 - 2023: *Nuclear Power Plants, A Domain Sketch*, 21 July, 2023 www.imm.dtu.dk/dibj/2023/nupopl/nupopl.pdf
 - 2021: *Shipping*, April 2021. www.imm.dtu.dk/dibj/2021/ral/ral.pdf
 - 2021: *Rivers and Canals – Endurants*, March 2021. www.imm.dtu.dk/dibj/2021/Graphs/Rivers-and-Canals.pdf
 - 2021: *A Retailer Market*, January 2021. www.imm.dtu.dk/dibj/2021/Retailer/BjornerHeraklit27January2021.pdf
 - 2019: *Container Terminals*, ECNU, Shanghai, China www.imm.dtu.dk/dibj/2018/yangshan/maersk-pa.pdf
 - 2018: *Documents*, TongJi Univ., Shanghai, China www.imm.dtu.dk/dibj/2017/docs/docs.pdf
 - 2017: *Urban Planning*, TongJi Univ., Shanghai, China www.imm.dtu.dk/dibj/2017/urban-planning.pdf
 - 2017: *Swarms of Drones*, IS/CAS¹³¹, Peking, China www.imm.dtu.dk/dibj/2017/swarms/swarm-paper.pdf
 - 2013: *Road Transport*, Techn. Univ. of Denmark www.imm.dtu.dk/dibj/road-p.pdf
 - 2012: *Credit Cards*, Uppsala, Sweden www.imm.dtu.dk/dibj/2016/credit/accs.pdf
 - 2012: *Weather Information*, Bergen, Norway www.imm.dtu.dk/dibj/2016/wis/wis-p.pdf
 - 2010: *Web-based Transaction Processing*, Techn. Univ. of Vienna, Austria, 186 pages www.imm.dtu.dk/dibj/wfdftp.pdf
 - 2010: *The Tokyo Stock Exchange*, Tokyo Univ., Japan www.imm.dtu.dk/db/todai/tse-2.pdf
 - 2009: *Pipelines*, Techn. Univ. of Graz, Austria www.imm.dtu.dk/dibj/pipe-p.pdf

¹³¹Inst. of Softw., Chinese Acad. of Sci.

- 2007: *A Container Line Industry Domain*, Techn. Univ. of Denmark [www.imm.dtu.dk/ dibj/container-paper.pdf](http://www.imm.dtu.dk/dibj/container-paper.pdf)
 - 2002: *The Market*, Techn. Univ. of Denmark [www.imm.dtu.dk/ dibj/themarket.pdf](http://www.imm.dtu.dk/dibj/themarket.pdf)
 - 1995–2004: *Railways*, Techn. Univ. of Denmark - a compendium [www.imm.dtu.dk/ dibj/train-book.pdf](http://www.imm.dtu.dk/dibj/train-book.pdf)
- Experimental research carried out to “discover”, try-out and refine method principles, techniques and tools, 1995–2025.
- [8] 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.
 - [9] 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).
 - [10] 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).
 - [11] Dines Bjørner. *Software Engineering, Vol. 1: Abstraction and Modelling; Vol. 2: Specification of Systems and Languages; Vol. 3: Domains, Requirements and Software Design*. Texts in Theoretical Computer Science, the EATCS Series. Springer, Heidelberg, Germany, 2006.
 - [12] Dines Bjørner. From Domains to Requirements [www.imm.dtu.dk/ dibj/2008/ugo/ugo65.pdf](http://www.imm.dtu.dk/dibj/2008/ugo/ugo65.pdf). In *Montanari Festschrift*, volume 5065 of *Lecture Notes in Computer Science* (eds. Pierpaolo Degano, Rocco De Nicola and José Meseguer), pages 1–30, Heidelberg, May 2008. Springer.
 - [13] 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.
 - [14] 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).
 - [15] Dines Bjørner. Domain Analysis: Endurants – An Analysis & Description Process Model [www.imm.dtu.dk/ dibj/2014/kanazawa/kanazawa-p.pdf](http://www.imm.dtu.dk/dibj/2014/kanazawa/kanazawa-p.pdf). In Shusaku Iida and José Meseguer and Kazuhiro Ogata, editor, *Specification, Algebra, and Software: A Festschrift Symposium in Honor of Kokichi Futatsugi*. Springer, Heidelberg, Germany, May 2014.
 - [16] 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.
 - [17] 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.
 - [18] 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.
 - [19] 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.
 - [20] Dines Bjørner. Domain Analysis & Description. [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.

- [21] 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 [24].
- [22] 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 [26].
- [23] Dines Bjørner. *Domain Modelling – A Primer*. A short and significantly revised version of [21]. xii+202 pages¹³², May 2023.
- [24] Dines Bjørner. *Domain Science & Engineering – A Foundation for Software Development*. Revised edition of [21]. xii+346 pages¹³³, January 2023.
- [25] Dines Bjørner. *Double-entry Bookkeeping*. Research, Institute of Mathematics and Computer Science. Technical University of Denmark, DK-2800 Kgs.Lyngby, Denmark, August 2023. <http://www.imm.dtu.dk/~dibj/2023/doubleentry/dblentrybook.pdf>. One in a series of planned studies: [28, 34, 33, 32].
- [26] Dines Bjørner. *Domain Modelling – A Primer*. A significantly revised version of [22]. xii+202 pages¹³⁴, Summer 2024.
- [27] Dines Bjørner. *Domain Models – A Compendium*. Internet: <http://www.imm.dtu.dk/~dibj/2024/-models/domain-models.pdf>, March 2024. This is a very early draft. 19 domain models are presented.
- [28] Dines Bjørner. *Banking – A Domain Description*. Sci. & techn. study, Technical University of Denmark, Fredsvej 11, DK 2840 Holte, Denmark, March 2025. One in a series of planned studies: [34, 33, 32, 25].
- [29] Dines Bjørner. *Documents – A Domain Description*. Sci. & techn. study, Technical University of Denmark, Fredsvej 11, DK 2840 Holte, Denmark, March 2025. One in a series of planned studies: [28, 34, 33, 32, 25].
- [30] Dines Bjørner. *Domain Analysis & Description*. *To be submitted*, page 33, March 2025. Institute of Mathematics and Computer Science. Technical University of Denmark.
- [31] Dines Bjørner. *Domain Modelling*. *Submitted to ACM FAC*, page 18, February 2025. Institute of Mathematics and Computer Science. Technical University of Denmark.
- [32] Dines Bjørner. *Health Care – A Domain Description*. Sci. & techn. study, Technical University of Denmark, Fredsvej 11, DK 2840 Holte, Denmark, March 2025. One in a series of planned studies: [28, 34, 33, 25].
- [33] Dines Bjørner. *Insurance – A Domain Description*. Sci. & techn. study, Technical University of Denmark, Fredsvej 11, DK 2840 Holte, Denmark, March 2025. One in a series of planned studies: [28, 34, 32, 25].
- [34] Dines Bjørner. *Transport – A Domain Description*. Sci. & techn. study, Technical University of Denmark, Fredsvej 11, DK 2840 Holte, Denmark, March 2025. One in a series of planned studies: [28, 33, 32, 25].
- [35] 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.
- [36] Dines Bjørner, Chr. Gram, Ole N. Oest, and Leif Rysrøm. *Dansk Datamatik Center*. In Benkt Wangler and Per Lundin, editors, *History of Nordic Computing*, Stockholm, Sweden, 18-20 October 2010. Springer.
- [37] Dines Bjørner and Cliff B. Jones, editors. *The Vienna Development Method: The Meta-Language*, volume 61 of *LNCS*. Springer, Heidelberg, Germany, 1978.
- [38] Dines Bjørner and Cliff B. Jones, editors. *Formal Specification and Software Development*. Prentice-Hall, London, England, 1982.

¹³²This book is currently being translated into Chinese by Dr. Yang ShaoFa, IoS/CAS (Institute of Software, Chinese Academy of Sciences), Beijing and into Russian by Dr. Mikhail Chupilko and his colleagues, ISP/RAS (Institute of Systems Programming, Russian Academy of Sciences), Moscow

¹³³Due to copyright reasons no URL is given to this document's possible Internet location. A primer version, omitting certain chapters, is [23]

¹³⁴This book is currently being translated into Chinese by Dr. Yang ShaoFa, IoS/CAS (Institute of Software, Chinese Academy of Sciences), Beijing and into Russian by Dr. Mikhail Chupilko and colleagues, ISP/RAS (Institute of Systems Programming, Russian Academy of Sciences), Moscow

- [39] Dines Bjørner and Ole N. Oest. The DDC Ada Compiler Development Project. In Dines Bjørner and Ole N. Oest, editors, *Towards a Formal Description of Ada*, [41], volume 98 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 1980.
- [40] Dines Bjørner and Ole N. Oest, editors. *Towards a Formal Description of Ada*, volume 98 of *LNCS*. Springer, Heidelberg, Germany, 1980.
- [41] Dines Bjørner and Ole N. Oest, editors. *Towards a Formal Description of Ada*, volume 98 of *Lecture Notes in Computer Science*. Springer, Heidelberg, Germany, 1980.
- [42] Geert Bagge Clemmensen and Ole N. Oest. Formal specification and development of an Ada compiler – a VDM case study. In *Proc. 7th International Conf. on Software Engineering*, 26.–29. March 1984, Orlando, Florida, pages 430–440, New York, USA, 1984. IEEE.
- [43] Patrick Cousot. *Principles of Abstract Interpretation*. The MIT Press, 2021.
- [44] Peter Fettke and Wolfgang Reisig. *Understanding the Digital World – Modeling with HERAKLIT*. Springer, 2024. To be published.
- [45] K. Futatsugi, A.T. Nakagawa, and T. Tamai, editors. *CAFE: An Industrial-Strength Algebraic Formal Method*, Sara Burgerhartstraat 25, P.O. Box 211, NL–1000 AE Amsterdam, The Netherlands, 2000. Elsevier. Proceedings from an April 1998 Symposium, Numazu, Japan.
- [46] 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.
- [47] Chris W. George, Anne Elisabeth Haxthausen, Steven Hughes, Robert Milne, Søren Prehn, and Jan Storbank Pedersen. *The RAISE Development Method*. The BCS Practitioner Series. Prentice-Hall, Hemel Hempstead, England, 1995.
- [48] Chris W. George, Hung Dang Van, Tomasz Janowski, and Richard Moore. *Case Studies using The RAISE Method*. FACTS (Formal Aspects of Computing: Theory and Software) and FME (Formal Methods Europe). Springer-Verlag, London, 2002. This book reports on a number of case studies using RAISE (Rigorous Approach to Software Engineering). The case studies were done in the period 1994–2001 at UNU/IIST, the UN University's International Institute for Software Technology, Macau (till 20 Dec., 1997, Chinese Territory under Portuguese administration, now a Special Administrative Region (SAR) of (the so-called People's Republic of) China).
- [49] 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.
- [50] Michael Hammer and Stephen A. Stanton. *The Reengineering Revolution: The Handbook*. HarperCollinsPublishers, 77–85 Fulham Palace Road, Hammersmith, London W6 8JB, UK, 1996. Paperback.
- [51] 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 (2004).
- [52] Gerard J. Holzmann. *The SPIN Model Checker, Primer and Reference Manual*. Addison-Wesley, Reading, Massachusetts, 2003.
- [53] Daniel Jackson. *Software Abstractions: Logic, Language, and Analysis*. The MIT Press, Cambridge, Mass., USA, April 2006. ISBN 0-262-10114-9.
- [54] Michael A. Jackson. *Software Requirements & Specifications: a lexicon of practice, principles and prejudices*. ACM Press. Addison-Wesley, Reading, England, 1995.
- [55] Michael A. Jackson. *Problem Frames — Analyzing and Structuring Software Development Problems*. ACM Press, Pearson Education. Addison-Wesley, England, 2001.
- [56] Michael A. Jackson. Program Verification and System Dependability. In Paul Boca, Jonathan Bowen, and Jawed Siddiqi, editors, *Formal Methods: State of the Art and New Directions*, pages 43–78, London, UK, December 2009. Springer.

- [57] Andrew Kennedy. *Programming languages and dimensions*. PhD thesis, University of Cambridge, Computer Laboratory, April 1996. 149 pages: cl.cam.ac.uk/techreports/UCAM-CL-TR-391.pdf. Technical report UCAM-CL-TR-391, ISSN 1476-298.
- [58] 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.
- [59] R. Milne and C. Strachey. *A Theory of Programming Language Semantics*. Chapman and Hall, London, Halsted Press/John Wiley, New York, 1976.
- [60] Charles W. Morris. *Foundations of the theory of signs*, volume I of *International encyclopedia of unified science*. The University of Chicago Press, 1938.
- [61] 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.
- [62] F. Pulvermüller. Brain mechanisms linking language and action. *Nature Reviews: Neuroscience*, 6:576582, 2005. <https://doi.org/10.1038/nrn1706>.
- [63] John R. Searle. *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press, 1969.
- [64] 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.
- [65] Kai Sørlander. *Indføring i Filosofien [Introduction to The Philosophy]*. Informations Forlag, Copenhagen, Denmark, 2016. 233 pages.
- [66] Kai Sørlander. *Den rene fornufts struktur [The Structure of Pure Reason]*. Ellekær, Slagelse, Denmark, 2022. See [67].
- [67] Kai Sørlander. *The Structure of Pure Reason*. Springer, February 2025. This is an English translation of [66] – done by Dines Bjørner in collaboration with the author.
- [68] Hung Dang Van, Chris George, Tomasz Janowski, and Richard Moore, editors. *Specification Case Studies in RAISE*. Springer, 2002.
- [69] Achille C. Varzi. *On the Boundary between Mereology and Topology*, pages 419–438. Hölder-Pichler-Tempsky, Vienna, 1994.
- [70] James Charles Paul Woodcock and James Davies. *Using Z: Specification, Proof and Refinement*. Prentice Hall International Series in Computer Science, London, England, 1996.

Part X

APPENDIX

Appendix A

Indexes

A.1 Domain Modeling Ontology

General

Domain Modeling Method: By a systematic domain analysis & description method we mean a set of *principles, procedures, techniques* and *tools*, for efficiently *analyzing & describing* domains., 5

Domain: By a *domain* we shall understand a *rationaly 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*., 5

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], 8

Entity: By an entity By an *entity* we shall understand a more-or-less rationally describable phenomenon., 10

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*, 8

Phenomena: By a *phenomenon* we shall understand a fact that is observed to exist or happen., 10

Prompt: By a prompt we shall understand an informal “advice” to the domain analyzer to “perform” a mental inquiry wrt. the real-life domain being studied., 10

Rationality: The rational, analytic philosophy issues of the inevitability of these external and internal 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? The choice here is also that we separate our inquiry into examining *both external and internal qualities* of endurants [not ‘either or’], 12

Transcendence: By transcendence we shall understand the philosophical notion: the a priori or intuitive basis of knowledge, independent of experience, 23

Transcendental Deduction: By a transcendental deduction we shall understand the notion: a “conversion” of one kind of knowledge into a seemingly different kind of knowledge, 23

T: the name of the type of all type names., 15

Endurants

External Qualities

Atomic Part: y 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., 13

Cartesian and Part Sets: A description prompt, 15

Cartesians: Cartesian parts are those compound parts which are observed to consist of two or more distinctly sort-named endurants (solids or fluids). , 14

Compound Part: Compound parts are those which are observed to [potentially] consist of several parts, 14

Domain Description Schema. *Cartesian and Part Sets:* ..., 15

Domain Description Schema. *Describe Attributes:* ..., 20

Domain Description Schema. *Describe Unique Identity:* ..., 17

Domain Description Schema: *Describe Mereology:* ..., 19

External Quality: 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. , 12

Fluid Endurant: 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; , 13

Part Sets: Part sets are those compound parts which are observed to consist of an indefinite number of zero, one or more parts, 15

Part: Non-living solid species are what we shall call parts., 13

Solid Endurant: By a *solid* cum *discrete* enduring we shall understand an enduring which is separate, individual or distinct in form or concept, or, rephrasing, have body (or magnitude) of three-dimensions: length (or height), breadth and depth, 13

State: By a *state* we shall mean any subset of the parts of a domain., 16

is_ Cartesian: An analysis prompt, 14

is_ atomic: An analysis prompt, 14

is_ compound: An analysis prompt, 14

is_ fluid: An analysis prompt, 13

is_ part: An analysis prompt, 13

is_ part_ set: An analysis prompt, 15

is_ solid: An analysis prompt, 13

record_ Cartesian_ part_ type_ names: An analysis function, 15

record_ part_ set_ part_ type_ names: An analysis function, 15

Internal Qualities

Internal Quality: Internal qualities are those properties [of endurants] that do not occupy *space* but can be measured or spoken about. , 12

Internal Qualities: Unique Identification

Domain Description Schema. Describe Unique Identity: ..., 17

Unique Identity: A unique identity is an immaterial property that distinguishes any two *spatially* distinct solids. , 17

uid_ : unique identifier observer., 17

Internal Qualities: Mereology

Domain Description Schema: Describe Mereology: ..., 19

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. , 18

mereo_ : mereology observer., 19

Internal Qualities: Attributes

Attribute: 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. , 19

Domain Description Schema. Describe Attributes: ..., 20

is_ active: An attribute category observer, 20

is_ autonomous: An attribute category observer, 20

is_ biddable: An attribute category observer, 20

is_ dynamic: An attribute category observer, 20

is_ inert: An attribute category observer, 20

is_ monitorable_ attribute: An attribute category observer, 21

is_ programmable: An attribute category observer, 20

is_ programmable_ attribute: An attribute category observer, 21

is_ static: An attribute category observer, 20

is_ static_ attribute: An attribute category observer, 21

record_ attribute_ type_ names: An analysis function, 19

attr_ : attribute observer, 20

Perdurants

Action: An action is a function that can purposefully change a state, 23

Behaviour: Behaviours are sets of sequences of actions, events and behaviours, 24

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

Description Schema: Behaviour Invocation: ..., 25

Description Schema: Behaviour Signatures: ..., 24

Description Schema: Channels: channel ... , 24

Events: An event is a function that surreptitiously changes a state, 23

A.2 Transport Domain Concepts

- action, 61
 - of behaviour, 105
- argument
 - of behaviour, 61
- behaviour, 61
 - action, 105
 - argument, 61
 - of part, 105
- bookkeeping, double, 76
- business process re-engineering, 157
- cash, 69
- client, 81
- command, 105
 - directive, 105
 - response, 105
- consumer, 81
- conveyor, 41
 - kind, 41
- conveyor company, 69
- cost, 69
 - of conveyance, 76
- current, 58
- customer, 69, 81
 - aggregate, 73
- directive
 - command, 105
- double bookkeeping, 76
- edge
 - kind, 41
 - label
 - unique, bi-directed, 41
- entity
 - syntactic, 105
- event, 59, 105
 - external, 105
 - internal, 105
- event notice, 59
- external
 - event, 105
- facet
 - script, 105
- function, 61
- goods = merchandises, 77
- graph [= net], 41
- history attribute, 59
- infrastructure
 - component, 36
- intentional pull, 41
- internal
 - event, 105
- invocation, 66
- kind, 41
 - conveyor, 41
 - edge, 41
 - node, 41
- kustomer
 - aggregate, 73
- logistics, 157
- merchandise, 69, 77
 - = goods, 77
 - aggregate, 73
- merchandises, 77
- multi-mode transport, 69
- node
 - kind, 41
 - label, 41
- overall, top transport endurants, 69
- part
 - behaviour, 105
- path, 45
- payment
 - of conveyance, 76
- people, 41
- receiver, 69
- recipient, 69
- response
 - command, 105
- route, 45
- routes, 58
- script
 - facet, 105
- semantics, 105
- sender, 69
- single-mode transport, 69
- state
 - change, 105
- syntactic entity, 105
- tail-recursion, 62
- theorem, 51
- time-stamp, 59
- transport, 41
 - multi-mode, 69
 - net, 41
 - route, 110
 - single-mode, 69

A.3 Formal Entities

The formal entries first lists formula entries by ontological category, then all:

Endurants

External Qualities

- * **Parts: Sorts and Observers**
- * **A Part State Concept**

Internal Qualities

Unique Identification

- * **Unique Identifiers: Sorts and Observers**
- * **A Unique Identifier State Concept**
- * **A Wellformedness Axiom**

Mereology

- * **Mereology: Sorts and Observers**
- * **A Wellformedness Axiom**

Attributes

- * **Attributes: Sorts and Observers**
- * **Wellformedness Axioms**

- * **Intentional Pull**

- * **Commands**

Perdurants

- * **Communication**
- * **Messages**
- * **Behaviour Signatures**
- * **Behaviour Definitions**
- * **Initialization**

- * **Values**

- * **Auxiliary Types**

- * **Auxiliary Functions**

- * **Theorems**

Only the *'ed entries are listed.

Endurant**sorts**

EA 131, 45
 ES 133, 45
 E 136, 45
 NA 132, 45
 NS 134, 45
 N 135, 45
 P 137, 45
 C 184, 55
 CA 1150, 74, 85
 CA 118, 43
 CK 1149, 74, 85
 CKA 1147, 74, 85
 CKS 1148, 74, 85
 CO 1152, 74, 85
 CS 1151, 74, 85
 CS 183, 55
 E 1141, 72
 E 135, 45
 EA 1139, 72
 EA 131, 45
 EAI 1139, 72
 EI 1141, 72
 ES 1141, 72
 ES 133, 45
 G 1137, 72

G 117, 43
 GI 1137, 72
 KA 1144, 73
 KS 1145, 73
 LA 1154, 75
 LS 1155, 75
 M 1159, 77
 MA 1142, 73
 MS 1143, 73
 N 1140, 72
 N 136, 45
 NA 1138, 72
 NA 132, 45
 NAI 1138, 72
 NI 1140, 72
 NS 1140, 72
 NS 134, 45
 oL 1153, 74, 85
 P 137, 45
 T, 72
 T 116, 43
 U, 50

auxiliary types

Air 184, 55
 Rail 184, 55
 Road 184, 55
 Sea 184, 55

observers

obs_CA *ι*150, 74, 85
obs_CKA *ι*147, 74, 85
obs_CKS *ι*148, 74, 85
obs_CO *ι*152, 74, 85
obs_CS *ι*151, 74, 85
obs_EA *ι*139, 72
obs_EA *ι*31, 45
obs_ES *ι*141, 72
obs_ES *ι*33, 45
obs_GT *ι*137, 72
obs_KA *ι*144, 73
obs_KAI *ι*144, 73
obs_KI *ι*145, 73
obs_KS *ι*145, 73
obs_LA *ι*154, 75
obs_LS *ι*155, 75
obs_MA *ι*142, 73
obs_MAI *ι*142, 73
obs_MI *ι*143, 73
obs_MS *ι*143, 73
obs_NA *ι*138, 72
obs_NA *ι*32, 45
obs_NS *ι*140, 72
obs_NS *ι*34, 45
obs_oL *ι*153, 74, 85
obs_obs_ CA *ι*18, 43
obs_obs_ G *ι*17, 43

Unique Identification**sorts**

CAI *ι*150, 74
 CAI *ι*200, 87
 CAI *ι*25, 44
 CAI *ι*87, 55
 CCAI *ι*198, 87
 CI *ι*201, 87
 CI *ι*88, 55
 CIK *ι*149, 74
 CKAI *ι*147, 74
 CKSI *ι*199, 87
 COI *ι*152, 74
 COI *ι*202, 87
 EAI, 47
 EI, 47
 ESI, 47
 GI, 47
 GI *ι*24, 44
 KAI *ι*144, 73
 KI *ι*145, 73
 KI *ι*180, 82
 LAI *ι*154, 75
 LI *ι*155, 75
 MAI *ι*142, 73
 MI *ι*143, 73
 MI *ι*160, 78
 NAI, 47
 NI *ι*43, 47
 NSI, 47
 oLI *ι*153, 74
 oLI *ι*203, 87

PI *ι*43, 47

TI *ι*23, 44

observers

uid_C *ι*201, 87
uid_CA *ι*150, 74
uid_CAI *ι*25, 44
uid_CAI *ι*87, 55
uid_CCA *ι*198, 87
uid_CI *ι*88, 55
uid_CK *ι*200, 87
uid_CKAI *ι*147, 74
uid_CKI *ι*149, 74
uid_CKS *ι*199, 87
uid_CO *ι*152, 74
uid_CO *ι*202, 87
uid_E *ι*47, 47
uid_EA *ι*45, 47
uid_EAI *ι*139, 72
uid_EI *ι*141, 72
uid_ES *ι*46, 47
uid_G *ι*44, 47
uid_GI *ι*137, 72
uid_GI *ι*24, 44
uid_K *ι*180, 82
uid_LAI *ι*154, 75
uid_LI *ι*155, 75
uid_M *ι*160, 78
uid_N *ι*47, 47
uid_NA *ι*45, 47
uid_NAI *ι*138, 72
uid_NI *ι*140, 72
uid_NS *ι*46, 47
uid_TI *ι*23, 44
uid_oL *ι*153, 74
uid_oL *ι*203, 87

Axioms

*ι*235, 94
 All parts are uniquely identified *ι*92, 56
 Commensurable Routes *ι*102, 58
 Conveyor Mereology of Right Kind *ι*95, 57
 Graph Mereology Wellformedness *ι*58 *ι*59, 48
 Ordered Way and Conveyor Histories *ι*104, 59
 Routes of commensurate kind *ι*98, 57
 Unique Conveyor Companies Parts *ι*205, 88
 Uniqueness of Part Identification *ι*55, 47
 Uniqueness of Transport Identifiers *ι*30, 44
 Wellformed Conveyor Company Mereologies
 *ι*210, 89
 Wellformed Transports *ι*257, 110

Mereology**types**

CAM *ι*206, 88, 182
 CM *ι*207, 88
 CM *ι*224, 93
 CM *ι*94, 56
 COM *ι*208, 88
 COST *ι*157d, 75
 Cost *ι*166, 78
 EHist *ι*157e, 75

EM *t*157, 75
 EM *t*57, 48, 182
 Flammability *t*167, 78
 Insurance *t*168, 78
 KM *t*182, 82, 181
 LEN *t*157c, 75
 MHist *t*169, 78
 MId *t*162, 78
 MM *t*161, 78
 NHist *t*157b, 75
 NM *t*156, 75
 NM *t*56, 48, 182
 OnHold *t*157a, 75
 Position *t*163, 78
 Size *t*164, 78
 Weight *t*165, 78

observers

mereo_ C *t*207, 88
mereo_ C *t*224, 93
mereo_ C *t*94, 56
mereo_ CA *t*206, 88
mereo_ CO *t*208, 88
mereo_ EM *t*157, 75
mereo_ EM *t*57, 48
mereo_ K *t*182, 82
mereo_ M *t*161, 78
mereo_ NM *t*156, 75
mereo_ NM *t*56, 48

auxiliary types

Event *t*170, 78

Attribute

types:

AtNode *t*99, 57
 CKHist *t*221, 90, 182
 Contracts *t*217, 90, 182
 Contracts *t*218, 90
 ConvCompInfo *t*215, 90, 182
 ConvHist *t*105, 59
 COST *t*72, 52
 CPos *t*99, 57
 CurrBuss *t*219, 90, 182
 CustHist *t*186, 82, 181
 CustId *t*183, 82, 181
 EdgeKind *t*70, 52
 F *t*99, 57
 Kind *t*225, 94
 Kind *t*97, 57
 LEN *t*71, 52
 NodeKind *t*69, 52
 OnEdge *t*99, 57
 OnHold *t*334, 149
 Orders *t*218, 182
 OutReqs *t*185, 82, 181
 PastBuss *t*220, 90, 182
 Position *t*231, 94
 Possess *t*184, 82, 181
 Resources *t*216, 90
 SR *t*227, 94, 182
 SRIndex *t*228, 94
 Stowage *t*226, 94

WHist *t*104, 59

observers:

attr_ CHist *t*233, 94
attr_ CKHist *t*221, 90
attr_ COST *t*157d, 75
attr_ COST *t*72, 52
attr_ CPos *t*231, 94
attr_ CPos *t*99, 57
attr_ Contracts *t*217, 90
attr_ ConvCompInfo *t*215, 90
attr_ ConvHist, 59
attr_ Cost *t*166, 78
attr_ CurrBuss *t*219, 90
attr_ CustHist *t*186, 82
attr_ CustId *t*183, 82
attr_ EHist *t*157e, 75
attr_ Edgekind *t*70, 52
attr_ Finals *t*230, 94
attr_ Flammability *t*167, 78
attr_ Insurance *t*168, 78
attr_ Kind *t*225, 94
attr_ Kind *t*97, 57
attr_ LEN *t*157c, 75
attr_ LEN *t*71, 52
attr_ MHist *t*169, 78
attr_ MId *t*162, 78
attr_ NHist *t*157b, 75
attr_ NodeKind *t*69, 52
attr_ OnHold *t*157a, 75
attr_ OnHold *t*334, 149
attr_ Orders *t*218, 90
attr_ OutReqs *t*185, 82
attr_ PastBuss *t*220, 90
attr_ Position *t*163, 78
attr_ Possess *t*184, 82
attr_ SR *t*227, 94
attr_ SRIndex *t*228, 94
attr_ Size *t*164, 78
attr_ Stowage *t*226, 94
attr_ TBL *t*229, 94
attr_ TBU *t*229, 94
attr_ WH *t*104, 59
attr_ Weight *t*165, 78

auxiliary types:

CHist *t*233, 94
 ChoiceNu *t*218c, 90, 182
 ContractNu *t*218a, 90, 182
 Event *t*187, 82, 181
 Final *t*230, 94
 Finals *t*230, 94
 Move *t*217a, 90, 182
 Offer *t*218b, 90
 Offers *t*218b, 182
 TBL *t*229, 94
 TBU *t*229, 94

Intentional Pull

Vehicles, Nodes and Edges *t*106, 60

Commands

syntax

Acknowledgement *t*248, 108
 Acknowledgment *t*240, 107
 Acknowledgment *t*248, 113
 Acknowledgment *t*266, 113
 Acknowledgment *t*271, 113
 CNTransfe *t*269, 113
 CNTransfer *t*247, 108
 ConvCompConvDir *t*243, 108, 114
 ConvCompConvDir *t*276, 114
 ConvCompOffer *t*241, 108, 114
 ConvCompOffer *t*274, 114
 ConvCompOrdOK *t*242, 108, 114
 ConvCompOrdOK *t*275, 114
 CustDel *t*239, 107
 CustOrd *t*237, 112
 CustOrder *t*237, 107
 CustQuery *t*236, 107, 111
 K *t*177, 81
 KNTransfer *t*238, 112
 NCTransfer *t*269, 113
 NKTransfer *t*279, 115
 Notify *t*246, 108, 113
 Notify *t*268, 113
 OrderOK *t*238, 107, 112
 PendColl *t*244, 108
 PendColl *t*245, 113
 PendColl *t*267, 113
 PendDel *t*249, 108, 113
 PendDel *t*273, 113
 Transfer *t*247, 113

auxiliary types

Addr *t*264a, 111
 Addr *t*264f, 111
 ChoiceNu *t*274d, 114
 ContractNu *t*252, 110
 ContractNu *t*265a, 112
 ContractNu *t*274b, 114
 ExpCost *t*264e, 111
 FromTo *t*272, 113
 FT *t*264d, 111
 MInfo *t*264b, 111
 M-set *t*265b, 112
 OfferChoice *t*274e, 114
 OrdComp *t*264c, 112
 QueryComp *t*263b, 111
 QueryId *t*263a, 111
 TI *t*264c, 111
 TR *t*250, 110

auxiliary functions

Addr *t*264d, 112
 ContractNu *t*264b, 112
 Cost *t*264h, 112
 FT *t*264g, 112
 MerchInfo *t*264e, 112
 OrdComp *t*264c, 112
 QueryId *t*264a, 112
 TI *t*264f, 112

Channel

comm, 61
comm *t*292, 123

M *t*293, 123

Message

Types

M *t*117, 61

Behaviour

Signatures

conv_comp *t*297, 124
 conveyor *t*118, 62, 137
 conveyor *t*298, 124
 customer *t*295, 124
 edge *t*118, 62, 147
 edge *t*131, 65, 147
 edge *t*299, 124
 initialization *t*132, 66
 logistics *t*296, 124
 node *t*118, 62
 node *t*130, 65, 149
 node *t*300, 124

Definitions

awaits_msg *t*315, 132
 confirms_offer *t*314c, 133
 conv_msg_handling *t*323, 140
 conveyor *t*119, 62, 142
 conveyor *t*125, 64, 141
 conveyor *t*322, 139
 conveyor_change_route *t*120, 63, 137
 conveyor_company *t*314, 132
 conveyor_enters_edge *t*122, 63, 138
 conveyor_enters_node *t*127, 65
 conveyor_moves_on_edge *t*126, 64
 conveyor_remains_at_node *t*121, 63, 138
 conveyor_stops_at_node *t*123, 64, 138
 conveyor_stops_on_edge *t*128, 65
 cust_delivers_merchandises *t*312, 130
 cust_issues_order *t*311, 129
 cust_order_OK *t*311, 129
 cust_requests_merchandises *t*313, 130
 customer *t*308, 127
 customer_issues_query *t*310, 128
 customer_receiv_messages *t*308g, 128
 edge *t*131, 65, 147
 inform_conveyors *t*314d, 134
 initialization *t*132, 66
 instantiation *t*301, 125, 126
 node *t*130, 65, 149
 pending_collection *t*321, 135
 suggests_offer *t*314b, 133

Values

TIME, 59
 TI, time-interval, 59
 $\sigma_{CK_{uid}}$ *t*204, 88
 σ_{CK} *t*197, 87
 σ_{ps} *t*42, 46
 σ_{uis} *t*29, 44
 σ_t *t*19, 43
 σ_{uis} *t*54, 47
 σ_{uis} *t*91, 56
 ca *t*21, 43

cai 128, 44
cai 189, 56
cca_{ui} 1204, 88
ccks_{uid} 1204, 88
cis 190, 56
cka 1191, 86
cks 1192, 86
cks_{uid} 1204, 88
cos 1195, 87
cos_{uid} 1204, 88
cs 1194, 87
cs_{uid} 1204, 88
css 1193, 86
e_{uis} 152, 47
ea 138, 46
ea_{uis} 150, 47
es 140, 46
es_{uis} 151, 47
g 120, 43
gi 127, 44
gi 149, 47
ks 1146, 73
ks 1179, 81
m 1159, 77
n_{uis} 153, 47
na 139, 46
na_{uis} 150, 47
ns 141, 46
ns_{uis} 151, 47
ols 1196, 87
paths 166, 50
t, 72
t 1178, 81
t 119, 43
ti 126, 44

Auxiliary

Types

ConvDir 1319c, 134
 Edge_Node_Path 1255, 110
 Kind, 41
 Load 1319d, 134
 Path 160, 49
 Segment 1254, 110
 Unload 1319d, 134
 W 1103, 59
 WI 1103, 59

Functions

calc_offer 1316a, 133
 commensurate_query_offer 1315, 133
 commensurate_query_offers 1316a, 133
 construct_dirs 1319, 134
 ContractNu 1263, 110
 extract_dir 1320, 135
 kind 173, 53
 least_costly_route_of_kind, 53
 path_cost 176, 53
 path_kind 168, 51
 path_length 175, 53
 paths 162, 50
 retr_conveyor 193, 56

retr_customer 1173, 83
 retr_edge 161, 49
 retr_merchandise 1174, 79
 retr_merchandise 1175, 79
 retr_node 161, 49
 retr_path_cost 176, 53
 retr_path_length 175, 53
 retr_unit 161, 49
 retr_W 1103, 59
 rev_path 167, 51
 route_kind 174, 53
 same_kind 1259, 111
 select_next_route 1120b, 63, 137
 share_conveyors 1210, 89
 shortest_route 177, 53
 shortest_route_of_kind, 53
 update_orders 1317, 134
 update_res_and_ors 1316b, 133
 update_resources_and_orders 1316b, 133
 xtr_Addr 1280, 120
 xtr_CI 1282, 120
 xtr_CI 1283, 120
 xtr_CKI 1277, 114
 xtr_CKI 1281, 120
 xtr_CKI 1283, 120
 xtr_KI 1265, 112
 xtr_KI 1278, 114
 xtr_MIs 1284, 120
 xtr_Name 1280, 120

Theorems

All finite paths have finite reverse paths 167, 51

All

attr_CHist 1233, 94
attr_CKHist 1221, 90
attr_COST 1157d, 75
attr_COST 172, 52
attr_CPos 1231, 94
attr_CPos 199, 57
attr_Contracts 1217, 90
attr_ConvCompInfo 1215, 90
attr_ConvHist, 59
attr_Cost 1166, 78
attr_CurrBuss 1219, 90
attr_CustHist 1186, 82
attr_CustId 1183, 82
attr_EHist 1157e, 75
attr_Edgekind 170, 52
attr_Finals 1230, 94
attr_Flammability 1167, 78
attr_Insurance 1168, 78
attr_Kind 1225, 94
attr_Kind 197, 57
attr_LEN 1157c, 75
attr_LEN 171, 52
attr_MHist 1169, 78
attr_MId 1162, 78
attr_NHist 1157b, 75
attr_NodeKind 169, 52
attr_OnHold 1157a, 75

attr_OnHold ι 334, 149
attr_Orders ι 218, 90
attr_OutReqs ι 185, 82
attr_PastBuss ι 220, 90
attr_Position ι 163, 78
attr_Possess ι 184, 82
attr_SR ι 227, 94
attr_SRIndex ι 228, 94
attr_Size ι 164, 78
attr_Stowage ι 226, 94
attr_TBL ι 229, 94
attr_TBU ι 229, 94
attr_WH ι 104, 59
attr_Weight ι 165, 78
 All finite paths have finite reverse paths ι 67, 51
 M ι 117, 61
 ι 235, 94
TIME, 59
TI, time-interval, 59
 $\sigma_{CK_{uid}}$ ι 204, 88
 σ_{CK} ι 197, 87
 σ_{ps} ι 42, 46
 $\sigma_{t_{uis}}$ ι 29, 44
 σ_t ι 19, 43
 σ_{uis} ι 54, 47
 σ_{uis} ι 91, 56
ca ι 21, 43
cai ι 28, 44
ccaui ι 204, 88
ckcsuid ι 204, 88
cis ι 90, 56
cka ι 191, 86
cks ι 192, 86
ckcsuid ι 204, 88
cos ι 195, 87
cosuid ι 204, 88
cs ι 194, 87
csuid ι 204, 88
css ι 193, 86
euis ι 52, 47
ea ι 38, 46
eauis ι 50, 47
es ι 40, 46
esuis ι 51, 47
g ι 20, 43
gi ι 27, 44
gi ι 49, 47
ks ι 146, 73
ks ι 179, 81
m ι 159, 77
nuis ι 53, 47
na ι 39, 46
nauis ι 50, 47
ns ι 41, 46
nsuis ι 51, 47
ols ι 196, 87
paths ι 66, 50
t ι 19, 43
t, 72
ti ι 26, 44
 Air ι 84, 55
 All parts are uniquely identified ι 92, 56
 AtNode ι 99, 57
 C ι 224, 93
 C ι 84, 55
 C ι 94, 56
 CA ι 150, 74, 85
 CA ι 18, 43
 CA ι 206, 88
 CAI ι 150, 74
 CAI ι 200, 87
 CAI ι 25, 44
 CAI ι 87, 55
 CAM ι 206, 88, 182
 CCAI ι 198, 87
 CHist ι 233, 94
 CI ι 88, 55
 CIK ι 149, 74
 CK ι 149, 74, 85
 CKA ι 147, 74, 85
 CKAI ι 147, 74
 CKHist ι 221, 90, 182
 CKS ι 148, 74, 85
 CKSI ι 199, 87
 CM ι 224, 93
 CM ι 94, 56
 CO ι 152, 74, 85
 CO ι 208, 88
 COI ι 152, 74
 COI ι 202, 87
 COM ι 208, 88
 COST ι 157d, 75
 COST ι 72, 52
 CPos ι 99, 57
 CS ι 151, 74, 85
 CS ι 83, 55
 ChoiceNu ι 218c, 90, 182
 Commensurable Routes ι 102, 58
 ContractNu ι 218a, 90, 182
 ContractNu ι 263, 110
 Contracts ι 217, 90, 182
 Contracts ι 218, 90
 ConvCompInfo ι 215, 90, 182
 ConvDir ι 319c, 134
 ConvHist ι 105, 59
 Conveyor Mereology of Right Kind ι 95 , 57
 Cost ι 166, 78
 CurrBuss ι 219, 90, 182
 CustHist ι 186, 82, 181
 CustId ι 183, 82, 181
 E ι 141, 72
 E ι 35, 45
 EA ι 139, 72
 EA ι 31, 45
 EAI ι 139, 72
 EAI, 47
 EHist ι 157e, 75
 EI ι 141, 72
 EI, 47
 EM ι 157, 75
 EM ι 57, 48, 182
 ES ι 141, 72

- ES *ι*33, 45
- ESI, 47
- EdgeKind *ι*70, 52
- Edge_ Node_ Path *ι*255, 110
- Event *ι*170, 78
- Event *ι*187, 82, 181
- F *ι*99, 57
- Final *ι*230, 94
- Finals *ι*230, 94
- Flammability *ι*167, 78
- G *ι*137, 72
- G *ι*17, 43
- GI *ι*137, 72
- GI *ι*24, 44
- GI, 47
- Graph Mereology Wellformedness *ι*58 *ι*59, 48
- Insurance *ι*168, 78
- K *ι*182, 82
- KA *ι*144, 73
- KAI *ι*144, 73
- KI *ι*145, 73
- KI *ι*180, 82
- KM *ι*182, 82, 181
- KS *ι*145, 73
- Kind *ι*225, 94
- Kind *ι*97, 57
- Kind, 41
- LA *ι*154, 75
- LAI *ι*154, 75
- LEN *ι*157c, 75
- LEN *ι*71, 52
- LI *ι*155, 75
- LS *ι*155, 75
- Load *ι*319d, 134
- M *ι*159, 77
- M *ι*161, 78
- M *ι*293, 123
- MA *ι*142, 73
- MAI *ι*142, 73
- MHist *ι*169, 78
- MI *ι*143, 73
- MI *ι*160, 78
- Mid *ι*162, 78
- MM *ι*161, 78
- MS *ι*143, 73
- Move *ι*217a, 90, 182
- N *ι*140, 72
- N *ι*36, 45
- NA *ι*138, 72
- NA *ι*32, 45
- NAI *ι*138, 72
- NAI, 47
- NHist *ι*157b, 75
- NI *ι*140, 72
- NI *ι*43, 47
- NM *ι*156, 75
- NM *ι*56, 48, 182
- NS *ι*140, 72
- NS *ι*34, 45
- NSI, 47
- NodeKind *ι*69, 52
- Offer *ι*218b, 90
- Offers *ι*218b, 182
- OnEdge *ι*99, 57
- OnHold *ι*157a, 75
- OnHold *ι*334, 149
- Ordered Way and Conveyor Histories *ι*104, 59
- Orders *ι*218, 182
- OutReqs *ι*185, 82, 181
- P *ι*37, 45
- PI *ι*43, 47
- PastBuss *ι*220, 90, 182
- Path *ι*60, 49
- Position *ι*163, 78
- Position *ι*231, 94
- Possess *ι*184, 82, 181
- Rail *ι*84, 55
- Resources *ι*216, 90
- Road *ι*84, 55
- Routes of commensurate kind *ι*98, 57
- SR *ι*227, 94, 182
- SRIndex *ι*228, 94
- Sea *ι*84, 55
- Segment *ι*254, 110
- Size *ι*164, 78
- Stowage *ι*226, 94
- T *ι*16, 43
- TBL *ι*229, 94
- TBU *ι*229, 94
- TI *ι*23, 44
- T, 72
- Unique Conveyor Companies Parts *ι*205, 88
- Uniqueness of Part Identification *ι*55, 47
- Uniqueness of Transport Identifiers *ι*30, 44
- Unload *ι*319d, 134
- U, 50
- Vehicles, Nodes and Edges *ι*106, 60
- W *ι*103, 59
- WHist *ι*104, 59
- WI *ι*103, 59
- Weight *ι*165, 78
- Wellformed Conveyor Company Mereologies *ι*210, 89
- Wellformed Transports *ι*257, 110
- comm** *ι*292, 123
- comm**, 61
- awaits_ msg *ι*315, 132
- calc_ offer *ι*316a, 133
- commensurate_ query_ offer *ι*315, 133
- commensurate_ query_ offers *ι*316a, 133
- confirms_ offer *ι*314c, 133
- construct_ dirs *ι*319, 134
- conv_ comp *ι*297, 124
- conv_ msg_ handling *ι*323, 140
- conveyor *ι*118, 62, 137
- conveyor *ι*119, 62, 142
- conveyor *ι*125, 64, 141
- conveyor *ι*298, 124
- conveyor *ι*322, 139
- conveyor_ change_ route *ι*120, 63, 137
- conveyor_ company *ι*314, 132
- conveyor_ enters_ edge *ι*122, 63, 138

conveyor_enters_node *t*127, 65
 conveyor_moves_on_edge *t*126, 64
 conveyor_remains_at_node *t*121, 63, 138
 conveyor_stops_at_node *t*123, 64, 138
 conveyor_stops_on_edge *t*128, 65
 cust_delivers_merchandises *t*312, 130
 cust_issues_order *t*311, 129
 cust_order_OK *t*311, 129
 cust_requests_merchandises *t*313, 130
 customer *t*295, 124
 customer *t*308, 127
 customer_issues_query *t*310, 128
 customer_receive_messages *t*308g, 128
 edge *t*118, 62, 147
 edge *t*131, 65, 147
 edge *t*299, 124
 extract_dir *t*320, 135
 inform_conveyors *t*314d, 134
 initialization *t*132, 66
 instantiation *t*301, 125, 126
 kind *t*73, 53
 least_costly_route_of_kind, 53
 logistics *t*296, 124
 node *t*118, 62
 node *t*130, 65, 149
 node *t*300, 124
 oL *t*153, 74, 85
 oLI *t*153, 74
 oLI *t*203, 87
 path_cost *t*76, 53
 path_kind *t*68, 51
 path_length *t*75, 53
 paths *t*62, 50
 pending_collection *t*321, 135
 retr_W *t*103, 59
 retr_conveyor *t*93, 56
 retr_customer *t*173, 83
 retr_edge *t*61, 49
 retr_merchandise *t*174, 79
 retr_merchandise *t*175, 79
 retr_node *t*61, 49
 retr_path_cost *t*76, 53
 retr_path_length *t*75, 53
 retr_unit *t*61, 49
 rev_path *t*67, 51
 route_kind *t*74, 53
 same_kind *t*259, 111
 select_next_route *t*120b, 63, 137
 share_conveyors *t*210, 89
 shortest_route *t*77, 53
 shortest_route_of_kind, 53
 suggests_offer *t*314b, 133
 update_orders *t*317, 134
 update_res_and_ors *t*316b, 133
 update_resources_and_orders *t*316b, 133
 xtr_Addr *t*280, 120
 xtr_CI *t*282, 120
 xtr_CI *t*283, 120
 xtr_CKI *t*277, 114
 xtr_CKI *t*281, 120
 xtr_CKI *t*283, 120
 xtr_KI *t*265, 112
 xtr_KI *t*278, 114
 xtr_MIs *t*284, 120
 xtr_Name *t*280, 120
 obs_CA *t*150, 74
 obs_CKA *t*147, 74
 obs_CKS *t*148, 74
 obs_CO *t*152, 74
 obs_CS *t*151, 74
 obs_EA *t*139, 72
 obs_EA *t*31, 45
 obs_ES *t*141, 72
 obs_ES *t*33, 45
 obs_GT *t*137, 72
 obs_KA *t*144, 73
 obs_KAI *t*144, 73
 obs_KI *t*145, 73
 obs_KS *t*145, 73
 obs_LA *t*154, 75
 obs_LS *t*155, 75
 obs_MA *t*142, 73
 obs_MAI *t*142, 73
 obs_MI *t*143, 73
 obs_MS *t*143, 73
 obs_NA *t*138, 72
 obs_NA *t*32, 45
 obs_NS *t*140, 72
 obs_NS *t*34, 45
 obs_oL *t*153, 74
 obs_obs_CA *t*18, 43
 obs_obs_G *t*17, 43
 uid_CA *t*150, 74
 uid_CAI *t*25, 44
 uid_CAI *t*87, 55
 uid_CCA *t*198, 87
 uid_CI *t*88, 55
 uid_CK *t*200, 87
 uid_CKAI *t*147, 74
 uid_CKI *t*149, 74
 uid_CKS *t*199, 87
 uid_CO *t*152, 74
 uid_CO *t*202, 87
 uid_E *t*47, 47
 uid_EA *t*45, 47
 uid_EAI *t*139, 72
 uid_EI *t*141, 72
 uid_ES *t*46, 47
 uid_G *t*44, 47
 uid_GI *t*137, 72
 uid_GI *t*24, 44
 uid_K *t*180, 82
 uid_LAI *t*154, 75
 uid_LI *t*155, 75
 uid_M *t*160, 78
 uid_N *t*47, 47
 uid_NA *t*45, 47
 uid_NAI *t*138, 72
 uid_NI *t*140, 72
 uid_NS *t*46, 47
 uid_TI *t*23, 44
 uid_oL *t*153, 74

uid_ oL *t203*, 87
 Acknowledgement *t248*, 108
 Acknowledgment *t240*, 107
 Acknowledgment *t248*, 113
 Acknowledgment *t266*, 113
 Acknowledgment *t271*, 113
 Addr *t264a*, 111
 Addr *t264d*, 112
 Addr *t264f*, 111
 ChoiceNu *t274d*, 114
 CNTransfe *t269*, 113
 CNTransfer *t247*, 108
 ContractNu *t252*, 110
 ContractNu *t264b*, 112
 ContractNu *t265a*, 112
 ContractNu *t274b*, 114
 ConvCompConvDir *t243*, 108, 114
 ConvCompOffer *t241*, 108, 114
 ConvCompOrdOK *t242*, 108, 114
 ConvCompOrdOK *t275*, 114
 Cost *t264h*, 112
 CustDel *t239*, 107
 CustOrder *t237*, 107
 CustQuery *t236*, 107
 ExpCost *t264e*, 111
 FromTo *t272*, 113
 FT *t264d*, 111
 FT *t264g*, 112
 K *t177*, 81
 KNTransfer *t238*, 112
 MerchInfo *t264e*, 112
 MInfo *t264b*, 111
 M-set *t265b*, 112
 NCTransfer *t269*, 113
 NKTransfer *t279*, 115
 Notify *t246*, 108, 113
 Notify *t268*, 113
 OfferChoice *t274e*, 114
 OrderOK *t238*, 107, 112
 OrdComp *t264c*, 112
 PendColl *t244*, 108
 PendColl *t245*, 113
 PendColl *t267*, 113
 PendDel *t249*, 108, 113
 PendDel *t273*, 113
 QueryComp *t263b*, 111
 QueryId *t263a*, 111
 QueryId *t264a*, 112
 TI *t264c*, 111
 TI *t264f*, 112
 TR *t250*, 110
 Transfer *t247*, 113

There are 483 formal RSL entities, and there are 504 RSL definitions – the former counted among the latter.

Appendix B

Summaries

B.1 Commands

```
ι241 π111. [k1] CustQuery :: <QueryId × QueryComp
ι279 π118. [k2] ConvCompOffer :: CKI × ContractNu × QueryNu × (ChoiceNu  $\xrightarrow{m}$  OfferChoice)
ι279e π118. OfferChoice = TR × Cost
ι242 π111. [k3] CustOrd :: QueryId × ContractNu × OrdComp
ι280 π118. [k4] ConvCompOrdOK :: CKI × ContractNu × ChoiceNu × TR × Cost
ι243 π111. [k5] OrderOK :: ContractNu × ChoiceNu × Payment
ι281a π118. [k7] ConvCompConvDir :: CKI × ContractNu × Segment
ι272 π117. [k8] PendColl :: (NI × ContractNu × MI-set) mayby not the MI-set
ι243 π111. [k9] KNTransfer :: ContractNu × M-set
ι273 π117. [k10] Notify :: AtNode | OnEdge
ι274 π117. [k11a] NCTransfer :: ContractNu  $\xrightarrow{m}$  M-set
ι275 π117. [k11b] CNTTransfer :: ContractNu  $\xrightarrow{m}$  M-set
ι276 π117. [k12] Acknowledgment :: TIME × ContractNu × ((NI × CI) | (CI × NI))
ι278 π117. [k13] PendDel :: NI × ContractNu × MI-set mayby not the MI-set
ι284 π119. [k14a] NKTransfer :: NI × ContractNu × MI-set mayby not the MI-set
ι271 π117. [k15a] Acknowledgment :: TIME × ContractNu × (NI × KI)
ι271 π117. [k15b] Acknowledgment :: TIME × ContractNu × (KI × NI)
```

B.2 Mereologies and Attributes

B.2.1 Customers

Mereology:

ι187 π86. $KM = MI\text{-set} \times (CKI|LI)\text{-set} \times CI\text{-set}$

Attributes:

ι188 π86. $CustId = CustNam \times CustAdd \times \dots$

ι189 π86. $Possess = MI\text{-set}$

ι190 π86. $OutReqs = \dots$

ι191 π86. $CustHist = (TIME \times Event)^*$

ι192 π86. $Event = \dots$

ι193 π86. \dots

B.2.2 Conveyor Companies

Mereology:

l211 π 92. CAM = CI-set \times COI

Attributes:

l220 π 94. ConvCompInfo = ...

l222 π 94. Contracts = ContractNu \xrightarrow{m} Move*

l222a π 94. Move = (KI \times NI)|(NI \times CI)|(CI \times NI)|(NI \times KI)

l223 π 94. Orders = ContractNu \xrightarrow{m} Offers

l223a π 94. ContractNu

l223b π 94. Offers = ChoiceNu \xrightarrow{m} TR

l223c π 94. ChoiceNu

l224 π 94. CurrBuss = MSG-set

l225 π 94. PastBuss = MSG-set

l226 π 94. CKHist = MSG*

B.2.3 Conveyors

Mereology:

l229 π 97. CM = (NI|EI)set \times CKI-set \times KI-set

Attributes:

l230 π 98. Kind

l231 π 98. Stowage = ContractNu \xrightarrow{m} M-set

l234 π 98. TBU,TBL = NI \xrightarrow{m} ContractNu-set

l232 π 98. SR = Path

l233 π 98. SRIndex = Na

l235 π 98. Finals = NI \xrightarrow{m} (KI \xrightarrow{m} ContractNu)

l235 π 98. Final = NI \times ContractNo \times KI

l236 π 98. CPos = OnEdge (= NI \times (F \langle >EI) \times NI)

l236 π 98. CPos = AtNode (= NI)

l238 π 98. CHist = MSG*

B.2.4 Nodes and Edges

Mereology:

l61 π 52. NM = EI-set **axiom** \forall nm:NM \cdot card nm $>$ 0

l62 π 52. EM = NI-set **axiom** \forall em:EM \cdot card em=2

Attributes:

l74 π 56. NodeKind = Kind-set **axiom** \forall nk:NodeKind \cdot nk \neq {}

l75 π 56. EdgeKind = Kind-set **axiom** \forall ek:EdgeKind \cdot card ek=1

l76 π 56. LEN = Nat

l77 π 56. COST = Nat

l339 π 153. OnHold = ContractNu \xrightarrow{m} M-set