The 𝔻𝕆𝕄𝔸𝕀ℕ* Method
A Study†

Dines Bjørner
Technical University of Denmark, DTU Compute
Fredsvej 11, DK-2840 Holte, Danmark
E–Mail: bjorner@gmail.com, URL: www.imm.dtu.dk/˜db

March 8, 2026: 02:43 am

Abstract

We outline the 𝔻𝕆𝕄𝔸𝕀ℕ *method.* The presentation is one of *methodology.* The background for this is our involvement in VDM, the Vienna Development Method, RAISE, a Rigorous Approach to Industrial Software Engineering, and 𝔻𝕆𝕄𝔸𝕀ℕ, a method for *analysing and describing* domains. By a *domain* we shall understand a *rationally describable* segment of a *discrete dynamics* fragment of a *human assisted* reality: the world that we daily observe – in which we work and act, a reality made significant by human-created entities. The domain embody *endurants* and *perdurants*. *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]. *Perdurants* are those quantities of domains for which only a fragment exists, in *space*, if we look at or touch them at any given snapshot in *time* [Merriam Webster]. A 𝔻𝕆𝕄𝔸𝕀ℕ description consists of descriptions of a selection of the observable and describable *phenomena,* the *entities* of a domain, its *endurants* and *perdurants.* A domain description thus consists of a number of *domain description units.* These describe the *external* and *internal qualities* of domains: the quantities we observe, their *unique identities, mereologies, attributes* and *intentional pulls;* and their *perdurants: behaviours, actions* and *events.* We outline a *domain analysis & description procedure* which, in 21 stages (!) help You develop *domain descriptions !*

*I decided, mid-February 2026, to give a name, 𝔻𝕆𝕄𝔸𝕀ℕ, to my approach to domain science & engineering. "Everybody else" has names attached to their work: B, Event B, Alloy, CafeOBJ, CASL, Petri nets, RAISE, RSL, VDM, etc.

†Work on this "exercise", i.e., document, was begun, late January 2026. On 28 February I reached a certain "state" of my understanding of 𝔻𝕆𝕄𝔸𝕀ℕs and decided, temporarily or ..., to halt further work on this document. I might consider "extracting" a possibly publishable paper from this document.

Contents

1 Introduction **4**

2 The DOMAIN Method **5**
2.1 A Summary . . . 5
2.2 A DOMAIN Ontology . . . 5
2.3 Some DOMAIN Principles . . . 7
2.4 The DOMAIN Procedure . . . 8
2.4.1 **Stage 0:** Initialization . . . 9
2.4.2 Sort and Types: Names and Meaning . . . 10
2.4.3 **Stage 1:** The *discover_domain* Procedure . . . 10
2.4.4 **Stage 2:** The *initialize_domain* Procedure . . . 10
2.4.5 **Stage 3:** The *discover_external_qualities* Procedures. . . . 11
2.4.5.1 **Stage 4:** The discover_endurant_descriptions . . . 11
2.4.5.2 Termination ? . . . 14
2.4.5.3 **Stage 5:** Describe Endurant Taxonomy. . . . 15
2.4.5.4 **Stage 6:** Discover Endurant States. . . . 15
2.4.6 **Stage 7:** The *discover_internal_qualities* Procedures. . . . 16
2.4.6.1 **Stage 8:** The *unique_identification* Procedure. . . . 16
2.4.6.1.1 **Stage 9:** The *unique_identifiers* Procedure. . . . 17
2.4.6.1.2 **Stage 10:** Unique Identifier States. . . . 17
2.4.6.1.3 **Stage 11:** Uniqueness. . . . 18
2.4.6.2 **Stage 12:** The *mereology* Procedure. . . . 18
2.4.6.3 **Stage 13:** The *discover_attributes* Procedure. . . . 19
2.4.6.4 **Stage 14:** The *discover_intential_pull* Procedure. . . . 20
2.4.7 **Stage 15:** The *discover_perdurant* Procedures . . . 21
2.4.7.1 **Stage 16:** The *describe* **channel** Procedure. . . . 22
2.4.7.2 On Behaviour Signatures. . . . 22
2.4.7.3 **Stage 17:** Characterisation of Actions. . . . 23
2.4.7.4 **Stage 18:** Behavioural Taxonomy . . . 23
2.4.7.5 **Stage 19:** Behaviour Signatures and Specs. . . . 23
2.4.7.6 **Stage 20:** Behaviour Definitions. . . . 24
2.4.7.7 Patterns of Behaviour Definition Bodies. . . . 24
2.4.7.8 **Stage 21:** Domain Initialisation. . . . 25
2.4.8 Procedure Iteration. . . . 25
2.5 Some DOMAIN Techniques . . . 26
2.6 Some DOMAIN Tools . . . 26

3 A DOMAIN Model of The Domain Modeling Procedure **26**

4 Review of the DOMAIN Method **26**
4.1 Skills Assessment: The DOMAIN Method . . . 26
4.2 My Characterisation of the Term 'Method' . . . 27
4.3 A Critical Review of the DOMAIN Method . . . 28

5 Conclusion **30**

Acknowledgments **31**

6 Bibliography **31**

A SI Units **36**
A.1 Base SI Units . . . 36
A.2 Derived SI Units . . . 36
A.3 Further SI Units . . . 37
A.4 Standard Prefixes for SI Units of Measure . . . 37
A.5 SI Units of Measure and Fractions . . . 37

B Example **38**
B.1 Universe of Discouse . . . 38
B.2 Compound Endurants . . . 38
B.3 Endurant Values . . . 39
B.4 Taxonomy . . . 39
B.5 Unique Identification . . . 40
B.6 Unique Identifier Values . . . 40
B.7 Uniqueness of Endurants . . . 41
B.8 Mereology . . . 41
B.9 Attributes . . . 42
B.10 Intentional Pull . . . 43
B.11 Auxiliary Types . . . 44
B.12 Simple Function Values . . . 45
B.13 Channel . . . 47
B.14 Variables . . . 47
B.15 Behaviours . . . 47
B.16 Initialization . . . 50

C Example Behavioural Taxonomies **51**
C.1 A Multi-mode Transport Behavioural Taxonomy . . . 51
C.2 – more to come – . . . 51
C.3 – more to come – . . . 51

D Behaviour Patterns **52**
D.1 Behaviour Definitions . . . 52
D.2 Action Definitions . . . 54

The Triptych Dogma

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

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

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

1 Introduction

By a **method** we shall understand a *set of principles* and *procedures* for *selecting* and *applying* a *set of techniques using* a *set of tools* in order to *construct* an *artefact* [DB]. [1] **Methodology**, to us, is the *comparative study* and *knowledge* of methods [DB].

In [3, *Towards a Meaning of 'M' in VDM*] I, perhaps, "vaingloriously"[2] made a first attempt at formulating aspects of VDM [31]). Since then I have tried to encircle the two concepts: 'method' and 'methodology' – finally settling on the two characterizations given above.

Since 2009 my work [15, 10, 16, 17, 11, 18, 19, 20, 21, 2, 29, 23, 24, 22, 27, 26, 28] has been focused on a method, 𝔻𝕆𝕄𝔸𝕀ℕ, for analysing and describing domains: This document is one of a quadruplet, [25, 26, 27, 28], of documents that I worked on during the winter and spring of 2026..

The work of [25, 26, 27, 28] has, as one of its core contributions, I claim, the enunciation, the clarification of 𝔻𝕆𝕄𝔸𝕀ℕ modeling, i.e., the analysis & formalization method. The [25, 26, 27, 28] documents are not aimed at publication. Their developments are done purely for elucidation, characterisation and my own pleasure.

A contribution of this document is the elucidation of the concept 'method'; another contribution is the clarification of the special, rigorous procedure that is suggested to the basic approach for the domain analyser cum describers to follow in their "discovery": analysis & description of domains.

• • •

An aside: Domain descriptions can be done for either of at least three reasons:

- (i) to understand a domain [2];
- (ii) as a basis for developing (middle school) text-book material for courses in banking [worldwide] [22], multi-mode transport [likewise worldwide: land, air and sea] [24], etc. [3]; or
- (iii) as a basis for developing software [29, *Chapter 9: Requirements*] – as expressed in the Triptych Dogma above ! [[4, 5, 6] shows how to specify software.

[1] [DB] indicates that this is my interpretation of the so characterized term ! See Sect. 4.2 on page 27.

[2] Vainglorious: in a way that shows someone is too proud of their own abilities or achievements [Cambridge]. Vaingloriously is an adverb describing actions done with excessive, unwarranted pride, boastfulness, or empty conceit regarding one's own abilities or achievements. Originating in the mid-1500s, it implies acting with "worthless glory" or vain conceit. Examples include bragging excessively, making unfounded, self-important predictions, or acting with arrogant superiority.

[3] We teach the sciences of the God-made phenomena of our world: physics, chemistry, geology, zoology, botany, etc. Why not the man-made infrastructures ?

The contributions of this document "is independent" of either of (i–iii) !

• • •

The reader is expected to be familiar with formal specification languages like VDM [31, 32, 35], Alloy [38], or RSL [36] – this document "favours" RSL; and to be familiar with the domain modeling approach of this document [19, 20, 21, 23, 29]. [23] is a sufficient prerequisite.

2 The DOMAIN Method

In this section I shall summarize the DOMAIN method elements: principles, procedures, techniques and tools.

2.1 A Summary

Recall the terms 'principles', 'procedures', 'techniques' and 'tools' in the characerisation of the term 'method': By a **method** we shall understand a *set of principles* and *procedures* for *selecting* and *applying* a *set of techniques using* a *set of tools* in order to *construct* an *artefact* [DB].

- Major principles[4] of DOMAIN are (i) abstraction, (ii) the use of mathematics, and (iii) a combination of rigorous narration and formalisation. More in Sect. 2.3 on page 7.
- The DOMAIN analysis and description has, as its major tool[5], two languages: DAL and DDL; DAL is a domain analysis language, DDL a domain description language. More in Sect. 2.6 on page 26.
- Their "deployment", i.e., use, is based on the procedure[6] which is implied by the domain analysis and description ontology shown graphically in Fig. 1 on page 7. More in Sect. 2.4 on page 8.
- Major techniques[7] of DOMAIN are ... More in Sect. 2.5 on page 26.

2.2 A DOMAIN Ontology

Ontology, according to the *Stanford Encyclopedia of Philosophy's section:* Ontology and Information Systems: https://plato.stanford.edu/entries/ontology-is/, is:

> *philosophical ontology identify the fundamental constituents of reality, the categories they fall into, and the relations they stand in to one another. Ontology in computer and information science shares with philosophy that it includes a study of categories (see the entry on categories) but with a practical focus on recording an inventory of categories in a formal syntax and often with some formal semantics. Some individual*

[4] **Principle**: a fundamental truth or proposition that serves as the foundation for a system of belief or behaviour or for a chain of reasoning [Wikipedia].

[5] **Tool**: A tool is an object that can extend an individual's ability to modify features of the surrounding environment or help them accomplish a particular task [Wikipedia].

[6] **Procedure**: an established or official way of doing something [Oxford Languages and Google: https://languages.oup.com/google-dictionary-en/].

[7] **Technique**: a way of carrying out a particular task, especially the execution or performance of an artistic work or a scientific procedure [Oxford Languages and Google].

> *information science ontologies are also concerned with a formal specification of the definitions of those categories. Ontology as a field in computer and information science is also reasonably characterized as the study of what there is, but not necessarily of what ultimately exists, rather of what is assumed to exist in some relevant domain for some computational purpose. A computer scientist is less concerned about the metaphysical status of categories –such as whether they actually exist or are just human constructs –than with their practical use in computation. In computer and information science, an ontology may also catalogue individuals and relationships, in addition to categories.*[8]

We shall interpret the concept of ontology, more specifically DOMAIN ONTOLOGY, is as follows, cf. Fig. 1 on the facing page:

- identify fundamental constituents of domains, such as we have defined the term domains[9];
- In the DOMAIN ONTOLOGY these constituents include TIME and SPACE as concepts that can be rationally deduced, that is, are not properties of specific or all entities of domains such as we have defined that term.
- Further fundamental constituent constituents are
  - *entity* (**is_entity**)
  - *endurant* (**is_endurant**)
  - *perdurant* (**is_perdurant**)
  - *solid* (**is_solid**)
  - *fluid* (**is_fluid**)
  - *part* (**is_part**)
  - *living_species* (**is_living_species**)
  - *atomic* (**is_atomic**)
  - *compound* (**is_compound**)
  - *Cartesian* (**is_Cartesian**)
  - *part_set* (**is_part_set**)
  - *animal* (**is_animal**)
  - *plant* (**is_plant**)
  - *human* (**is_human**)
  - *other* (**is_other**)
  - *action* (**is_action**)
  - *event* (**is_**`event`)
  - *actor* (**is_actor**)
  - *channel* (**is_channel**), and
  - *behaviour* (**is_behaviour**)
- The relationship between these fundamental constituents are as shown in Fig. 1 on the next page: if, for example, *an endurant* is *solid*, then it is either *a part* or *a living species*, etc.

The *Stanford Encyclopedia of Philosophy* `https://plato.stanford.edu/entries/ontology--is/` also mentions the term upper ontology[10] – since that term emerged within the computer cum information sciences. Our concept of a domain ontology is that it is an upper ontology.

[8] One popular description of an ontology in computer science is "a specification of a conceptualization" (Gruber 1993). But this requires an explanation of of what a specification is and what a conceptualization is. A conceptualization is a way to understand the structure of a particular concept by considering both its relationship to other concepts and the principles that may govern it. So a conceptualization includes a way to understand how a concept differentiates the objects falling under it (whether abstract or concrete), from other things.

[9] – hence we shall refer to our use of the term ontology as DOMAIN ONTOLOGY

[10] An upper ontology is a collection of concepts that are not specific to any application domain. The determination of whether a concept belongs in an upper ontology or not is not an objective decision, except relative to other terms in the same ontology. Some upper ontologies aim to be minimal, containing only a small number of the most general concepts,

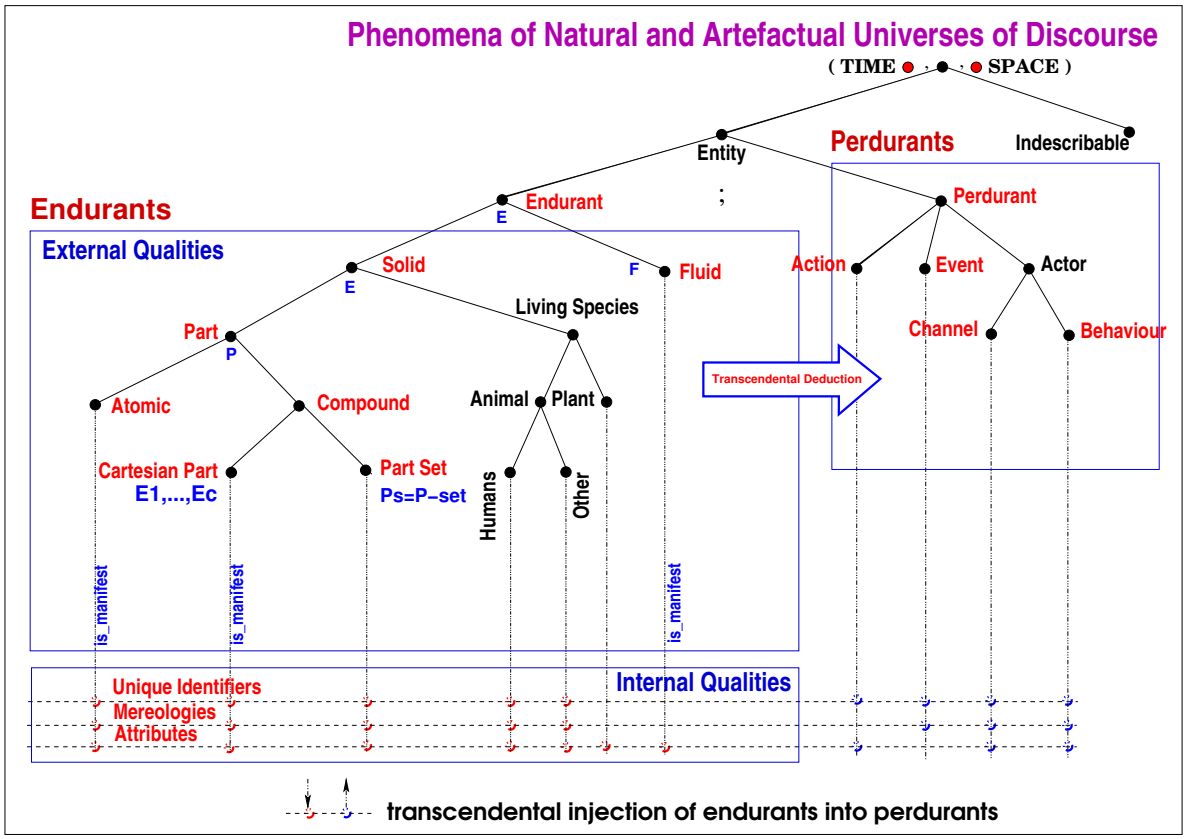

Figure 1: A Domain Analysis & Description Ontology

2.3 Some DOMAIN Principles

We list some DOMAIN PRINCIPLES:

- **Abstraction:**

```
Conception, my boy, fundamental brain-work,
    is what makes the difference in all art
```

D.G. Rossetti[11]*: letter to H. Caine*[12]

Quoting **C.A.R. Hoare:**

> *Abstraction is a tool,*
> *used by the human mind,*

and others include larger collections from the most general to the more specific. Many domain ontologies are created as extensions of upper ontologies. An upper ontology starts with a category of all things and then elaborates, adding categories (and sometimes, definitions) for physical and abstract things, relations and functions etc. But the boundary of where upper ontology becomes domain ontology is arbitrary. A domain ontology would not have a decomposition of categories starting from the class of all things, but many domain ontologies that do not extend an upper ontology still will require a few very general categories, such as the notion of a physical object or a process.

[11] Dante Gabrielli Rosetti, 1828–1882, English poet, illustrator, painter and translator

[12] T. Hall Caine, 1853–1931, British novelist, dramatist, short story writer, poet and critic.

and to be applied in the process of describing
(understanding) complex phenomena.
Abstraction is the most powerful such tool
available to the human intellect.
Science proceeds by simplifying reality.
The first step in simplification is abstraction.
Abstraction (in the context of science)
means leaving out of account all those empirical
data which do not fit the particular, conceptual framework
within which science at the moment happens to be working.
Abstraction (in the process of specification)
arises from a conscious decision
to advocate certain desired objects, situations and processes
as being fundamental;
by exposing, in a first, or higher, level of description, their similarities
and — at that level — ignoring possible differences.

- **Mathematics:** Thus it is implied, from the principle of *abstraction*, that we make use of mathematics, as a tool ! Its concepts of type, sets, Cartesians, lists, maps and functions.
- **Formal Specification:** Implied in the above is also that we focus on specifications: descriptions and prescriptions, [also] being formal, that is, in a [specification] language which [besides being readable and elegant] has a mathematical foundation. Several such *formal specification languages* exist: VDM [31, 32, 35], Z [40], Alloy [38], B [1], and RSL [36]. We shall be using RSL and variants, RSL$^+$text, thereof – but VDM and Alloy could easily be used.
- **Narration & Formalisation:** We insist on prefixing, or at least textually close, formulas with *narrative* text in plain natural language [e.g., English] text which expresses what the formulas [also tries to] express.[13]
- **Enumeration:** To ease cross-referencing between narration texts and formulas, we strongly advice to enumerate both, with all such enumerated items having distinct numbers !
- **Indexing:** Although not illustrated in this document[14], we also strongly advice to index all definitions, characterisations, formulas, etc.

2.4 The DOMAIN Procedure

The DOMAIN notion of *procedure* — implied by Fig. 1 on the preceding page — is that of the domain analyser cum describers proceed in steps, "traversing", as it were, (i) the domain, and ii) the graph as well as the taxonomy of the specific domain. Those steps and the taxonomy are covered in [26, *A Syntax for DDL*].

While the domain analyser cum describers are analysing & describing the domain they commit the state of their work to a "dashboard", here referred to as θ:Θ, an informal, conceptual (if not real) state.

[13] That is: the narrative is not meant to explain the formulas – although in this document they tend to do so – they are meant, in descriptions of domain facets, to explain what those facets are.

[14] – in a possible sequel, i.e., future version of this document, we hope to live up to this mandate.

This dashboard shall be understood as follows: It represents, in "abbreviated" form, various summaries of the evolving domain description. These summaries can either be written down, say on a chalkboard (slate, board, green board, white board, black board, writing board) or be deduced, all along, from the evolving domain description – which is [similarly to the dashboard] available to all domain analyser cum describers.

- **Please Note:** The formulas of this entire section are either those of the procedure steps, and they are in a variant of RSL [which we omit to name], or they are the RSL$^+$`text` being generated. Although they "look alike" they are different, informal, respectively [only] rigorously, i.e., not formally, formal languages – easy to, but should not be confused !

Steps of the above can be given a rigorous, although not quite formal specification. In the rendition below we assume just one domain analyser cum describer.

2.4.1 Stage 0: Initialization

1. There is, thus, a stage, say, discover_domain, which takes no argument, but delivers a domain description of a domain which that procedure elicits !
2. To "perform" discover_domain properly the dashboard state θ:Θ must be initialized.
   (a) $\theta_{esn} : \Theta_{ESN}$ is to hold a list of parts and their type under examination. This list will grow during "ontology traversal" examination of the of the domain.
   (b) $\theta_{dsu} : \Theta_{DSU}$ is to hold the list of all domain description units as they are issued during the analysis & description procedure. The list grows.
   (c) $\theta_{vde} : \Theta_{VDE}$ is to hold the list of RSL$^+$`texts` of value definitions.
   (d) $\theta_{beh} : \Theta_{BEH}$ is to hold the set of sort (names) of those [parts] which are to be "morphed" into behaviours. The set is set during procedure State 4, pages 12–14.

type
2a. Θ_{ESN} = (P×S)* [Pairs of values and endurant names]
2b. Θ_{DSU} = DSU* [Domain Description Units]
2c. Θ_{VDE} = RSL$^+$`text`* [Value definitions]
2d. Θ_{BEH} = S-**set** [Behaviour Sorts]
variable
2a. $\theta_{esn} : \Theta_{ESN} := \langle\rangle$
2b. $\theta_{dsu} : \Theta_{DSU} := \langle\rangle$
2c. $\theta_{vde} : \Theta_{VDE} := \langle\rangle$
2d. $\theta_{beh} : \Theta_{BEH} := \{\}$

Inquiry and update of the θ state is expressed below as if there is just one domain analyser cum describer. If indeed thee are more than one, a careful transaction processing protocol[15] – e.g., a la Jim Gray [37].

[15] Transaction processing is information processing that is divided into individual, indivisible operations called transactions. Each transaction must succeed or fail as a complete unit; it can never be only partially complete.

2.4.2 Sort and Types: Names and Meaning

We take an aside:

- Sorts are here sets of domain entities.
- Types are [more generally] sets of domain entities and mathematical quantities.
- Types have names T:$\mathbb{T}$.
- $\mathbb{T}$ is the type of all type names.
- Sorts have names: the name of all sorts is $\mathbb{S}$.
- The name of the sort of all names of sorts is Ξ [pronounced 'sigh'].
- If sort has name S:$\mathbb{S}$, then the name of that name is ηS:Ξ.

2.4.3 Stage 1: The *discover_domain* Procedure

3. The discover_domain procedure takes no explicit arguments and "delivers" its result deposited in $\theta_{dsu} : \Theta_{DSU}$. That is: the domain analyser cum describer, of course, work in the environment of the domain.
   (a) First the domain must be selected, that is, the universe of discourse must be chosen.
   (b) Then the external qualities must be analysed and described.
   (c) Then the internal qualities must be analysed and described.
   (d) Finally the perdurants must be analysed and described.

```
3.  discover_domain: Unit → Unit
3.  discover_domain() ≡
3a.     initialize_domain();
3b.     discover_external_qualities() ;
3c.     discover_internal_qualities() ;
3d.     discover_perdurants()
```

What "connects" these procedural phases is the dashboard state: θ:Θ.

2.4.4 Stage 2: The *initialize_domain* Procedure

[We refer to Example B.1 on page 38]

4. We define the analysis & description procedure.
   (a) The domain analyser cum describer choose a domain, and names it, say UoD.
   (b) A display line **The Universe-of-Discourse:** prefixes the domain description unit, and
   (c) the dashboard state θesn is initialised.
   (d) And that is it, for now !

```
value
4. discover_initialization_of_domain: Unit → Unit
4. discover_initialization_of_domain() ≡
4a.    let (text,uod:S) [ domain analyser cum describer choices ] in
4b.        { θdsu := ⟨ (" The Universe-of-Discourse:, text, value uod:𝕊 ") ⟩
4c.        ‖ θesn := ⟨ (text, value uod:𝕊) ⟩ } "
4d.    end
```

2.4.5 Stage 3: The *discover_external_qualities* Procedures.

There are three [sub-]stages here:

5. The discover_external_qualities
   (a) starts by discovering endurants;
   (b) then goes on to describe a domain endurant-taxonomy;
   (c) and to discover endurant states.

```
value
5. discover_external_qualities: Unit → Unit
5. discover_external_qualities() ≡
5a.    discover_endurant_descriptions() ;
5b.    describe_taxonomy() ;
5c.    describe_endurant_states()
```

2.4.5.1 Stage 4: The discover_endurant_descriptions [We refer to Example B.2 on page 38]

6. The examination (i.e., analysis & description) of the domain wrt. external qualities takes place in the/a current state $\theta : \Theta$.
   (a) θ_{esn} "holds" the list of names of the sorts yet to be examined. For every sort examination that list either grows or shrinks. It grows when examining compound sorts. It shrinks when examining atomic sorts. So as long as there are names in the list there is examination to do !
   (b) The endurant part, p, to be examined is selected.
   (c) It is then described – into RSL$^+$`text`.
   (d) And p is removed from θ_{esn}.

```
6.  discover_endurant_descriptions: Unit → Unit
6.  discover_endurant_descriptions() ≡
6a.     for i=1 to len c θesn do
6b.        let (p,E) = c θesn[i] in
6c.        traverse(p,E) end
6a.     end
```

The $\underline{\mathbf{c}}\,\theta_{esn}$ in the **for** i=1 **to len** $\underline{\mathbf{c}}\,\theta_{esn}$ **do** loop is to be evaluated "every time around the loop" as θ_{esn} changes within the loop !

Now we get to the "real" work.

7. The traverse step, as the name suggests, figuratively traverses the larger, blue lined box of Fig. 1 on page 7 – while observing a part p of type E.

   We read that traversal:

   (a) If p is an entity„
   (b) then
   (c) if p is an endurant
   (d) then
   (e) if p is a solid
   (f) then
   (g) if p is a part,
   (h) then
   (i) if p is atomic
   (j) then handle that atomic;
   (k) else [it is a compound]
   (l) if it is a Cartesian
   (m) then handle the Cartesian
   (n) else [it is a Part set] handle the Part set;
   (o) or else it is a living species[16],
   (p) or else it is a fluid[17],
   (q) or else it is a perdurant ![18]

```
value
7.  traverse: (P×E) → Unit
7.  traverse(p,E) ≡
7a.     if is_entity(p)
7b.       then
7c.         if is_endurant(p)
7d.           then [axiom: is_entity(p)]
7e.             if is_solid(p)
7f.               then [axiom: is_endurant(p)]
7g.                 if is_part(p)
7h.                   then
7i.                     if is_atomic(p)
7j.                       then handle_atomic(p,E)
7k.                       else [axiom is_compound(p)]
7l.                         if is_Cartesian(p)
7m.                           then handle_Cartesian(p,E)()
7n.                           else [axiom is_Part_set(p)] handle_Part_set(p,E)()
7l.                         end
7i.                     end
7o.                   else [axiom is_living_species(p)] skip
7g.                 end
7p.               else [axiom is_fluid(p)] then skip
7c.             end
```

[16] which we do not handle in this document
[17] which we do not handle in this document
[18] – which, or course, it is not since we initially invoked traverse with an endurant part !

```
7q.              else [ axiom: is_perdurant(p) ] skip
7c.           end
7a.       end
```

We mark in red the functions, i.e., the steps

7j. The handle_atom(p,E) function joins its sort to those of the behaviourals.

```
value
7j.   handle_atom(p,E) ≡ θ_beh := c θ_beh ∪ {E},
```

8. ,7k. The handle_Cartesian function behaves as follows.

   (a) The domain analyser cum describer observes/decides that one can observed n distinct parts, each of it own sort and

   (b) generates/supplies narrative text "explaining" these parts, and

   (c) updates the dashboard state accordingly – prefixing the domain description unit with a display line: **Endurant Sorts:**.

   (d) The domain analyser cum describer finally decides on the possibly empty, possibly full, subset of the sorts of the n Cartesian components are behavioral.

```
value
7m.     handle_Cartesian: (P × E) × Unit → Unit
7m.     handle_Cartesian(p,E)() ≡
8a.       let ((p1,E1),(p2,E2),...,(pn,En)) = observe(p) in
8b.       let (txt1,txt2,...,txtn) = analyser_cum_describer_choice(p) in
8c.       θ_esn := c θ_esn ⌢ ⟨ (p1,E1),(p2,E2),...,(pn,En) ⟩ ;
8c.       θ_dsu := c θ_dsu ⌢ ⟨" Endurant Sorts:
8c.                            Narrative:
8c.                              1. txt1,
8c.                              2. txt2,
8c.                              ...
8c.                              n. txtn.
8c.                            Formalisation:
8c.                              type
8c.                                1. E1, 2. E2, ..., En
8c.                              value
8c.                                1. obs_E1: E→E1
8c.                                2. obs_E2: E→E2,
8c.                                ...
8c.                                n. obs_En: E→En " ⟩ ;
8d.       θ_beh := c θ_beh ∪ select_Cartesian_endurants({E1,E2,...,En}) ,
8.          end end

8d.   select_Cartesian_endurants: E-set → E-set
8d.   select_Cartesian_endurants({E1,E2,...,En}) ≡
8d.     {Ei,Ej,...,Ek}[19] where {Ei,Ej,...,Ek} ⊆ {E1,E2,...,En}
```

9. ,7l If the part is considered a part set

   (a) then actions very similar to those for Cartesian parts are "performed" by the domain analyser cum describer.

```
value
7n.   handle_Part-set: (P × E) × Unit → Unit
7n.   handle_Part-set(p,E)() ≡
9a.    let (ps:PS) = observe(p) axiom ps ≠ {}  in
9a.    let p:P • p ∈ ps  in
9a.    let (text_ps,text_p) = analyser_cum_describer_Choice(ps,p)  in
9a.    θesn := c θesn ⌢ ⟨(p,P)⟩ ;
9a.    θdsu := c θdsu ⌢ ⟨ " Endurant Sorts:
9a.                              Narrative:
9a.                                  1. text_ps
9a.                                  2. text_p
9a.                              Formalisation:
9a.                                  type
9a.                                    1. PS = P-set[20],
9a.                                    2. P
9a.                                  value
9a.                                     1. obs_PS: E → PS " ⟩ ;
9a.    θbeh := c θbeh ∪ select_Part_set_endurant(P) ;
9.     end end end

9a.  select_Part_set_endurant: P → P-set
9a.  select_Part_set_endurant(P) ≡ if is_behavioral(P) then {P} else {} end
```

2.4.5.2 **Termination ?** Does the discover_endurant_descriptions procedure terminate ?
Yes.
The reasons are as follows:

- Endurants are not recursive ! That is, no endurant must be describable/definable in terms of itself. The reader may object ! How about trees, such as we see them in nature ? Well, these trees have roots below ground and trunks, branches and leaves above ground. The "above ground" latter are not trees. So we model trees as specially restricted graphs.
- For two distinct endurants, say two distinct Cartesians, E1 and E2, the same sort of part, p:E, does not occur in both. If the domain analyser cum describers think so, they are wrong. These two endurants sorts should be distinctly named, for example E1_E and E2_E or E_E1 and E_E2. They may be "alike", but somehow their, perhaps some or their "not enunciated" internal qualities differ.

So the discover_endurant_descriptions procedure terminate: eventually all examined endurants are atomic parts.

[19] The choice/selection of {Ei,Ej,...,Ek} is one that the domain analyser cum describer must decide upon.

2.4.5.3 Stage 5: Describe Endurant Taxonomy. [We refer to Example B.4 on page 39]

[We refer to Carl von Linneaeus, the originator of the term Taxonomy[21].] A visual domain model endurant taxonomy of the structure of parts do not add to the meaning of the domain model. It is merely of practical help in comprehending the possible complexity of the domain.

10. An endurant taxonomy domain description unit starts with a **Taxonomy** display line, usually prefixed by the name, E, of the "overall", i.e., a "main" part that is "at the root" of the taxonomy, i.e., the usually compound part whose siblings – and again their siblings, etc., till we "reach", as leaves of the taxonomy tree, the atomic parts.
    (a) The TAX_DSU tree-like diagram has as its root a box labeled with E.
    (b) In general now, if the taxonomy box stands for an atom then we leave it there: no taxonomy sub-tree emanating from that box.
    (c) If the taxonomy box, E, stands for a Cartesian endurant with components E1, E2, ..., En then we draw, on the same horizontal line below box E, n boxes, connected to box E by straight, usually slanted lines, n boxes, left-to-right, labeled E1, E2, ..., En.
    (d) If the taxonomy box, E, stands for a part set, then we draw, below it, a box labeled with the part set name, f.ex., PS, observed from E – connected to box E by a vertical line.
    (e) This is followed by drawing two or three boxes, "immediately" below box PS and on the same horizontal line and all identically labeled by the same endurant sort name, say P, where P is the type occurring in PS = P-**set** observed from E.
    (f) The above diagram construction instructions are followed till all domain endurants "derivable" from E have been followed.

```
value
10. describe_endurant_taxonomy: Unit → Unit
10. describe_endurant_taxonomy() ≡
10.     let endurant_taxonomy = design_endurant_taxonomy() in
10.     θdsu := c θdsu ⌢
10.         ⟨ " Endurant Taxonomy: endurant_taxonomy " ⟩
10.     end
```

Here design_endurant_taxonomy is a "function" which is performed by the domain analyser cum describer.

2.4.5.4 Stage 6: Discover Endurant States. [We refer to Example B.3 on page 39]

11. The discover_endurant_state stage
    (a) initially appends an **Endurant State**: prefix and
    (b) the RSL$^+$`text` **value** to θ_{dsu};
    (c) then proceeds, for all (part,sort)s that have been "discovered" and therefor are in θ_{ves},

[21] `https://en.wikipedia.org/wiki/Carl_Linnaeus` and to his main work: `https://ia600508.us.archive.org/9/items/mobot31753000798865/mobot31753000798865_bw.pdf`

(d) to select these elements of θ_{ves},

(e) and append them, as domain description units to θ_{dsu}.

12. Then a notion of "global" σtates is introduced:

(a) σ_{parts} – the union of all parts, and

(b) $\sigma_{behavioural_parts}$ the union of behavioural all parts.

```
11.  discover_endurant_state: Unit → Unit
11.  discover_endurant_state() ≡
11a.     θdsu := c θdsu ⌢ " ⟨ Endurant State: value ⟩ ";
11c.     for i=1 to len θves do
11d.       let (p,E) = θves[i] in
11e.       θdsu := c θdsu ⌢ " ⟨ ev:E = p ⟩ " ;
12.        θves := c θves ⌢ " ⟨ ev:E = p ⟩ " ;
12a.       σall_parts = [ ∪ of ∀ parts ] ,
12b.       σall_behavioural_parts = [ ∪ of ∀ hehaviourable parts ] "
11c.     end end
```

Here ev is a suitable, distinct value name chosen by the domain analyser cum describer.

2.4.6 Stage 7: The *discover_internal_qualities* Procedures.

13. The discover_internal_qualities procedure generates domain description units for the

(a) unique identification,

(b) mereologies,

(c) attributes and

(d) for possible intentional pull descriptions/

– and in that order !

```
13.  discover_internal_qualities: Unit → Unit
13.  discover_internal_qualities() ≡
13a.     describe_identification() ;
13b.     discover_mereologies() ;
13c.     discover_attributes() ;
13d.     discover_intentional_pulls()
```

2.4.6.1 Stage 8: The *unique_identification* Procedure. [We refer to Example B.5 on page 40]

14. The describe_identification procedure generates domain description units for the

(a) unique identifiers,

(b) unique identifier states,

(c) uniqueness of parts,

– and in that order !

14. describe_identification() ≡
14a. describe_unique_identifiers() ;
14b. describe_unique_identifier_states() ;
14c. describe_part_uniqueness() ;

2.4.6.1.1 Stage 9: The *unique_identifiers* Procedure. [We refer to Example B.5 on page 40]

Please Note: Let UI stand for the sort of all unique identifiers.

15. The *describe_unique_identifiers* procedure is to generate
    (a) a domain description unit **Unique Identification**: display line and
    (b) records the **type** and **uid_**E observers for all "behavioral" parts, that is: those parts which the domain analyser cum describer considers candidate to be "morphed" into behaviours – those which are dashboard recorded in θ_{beh}:Θ_{BEH}.
        i. For all E noted in the dashboard as behavioral
        ii. the**type** of the unique identifier and
        iii. the **value** and its observer
        iv. is added to the emerging domain description θ_{dsu}.

15. describe_unique_identifiers() ≡
15a. θ_{dsu} := **c** θ_{dsu} ⌢ ⟨ **Unique Identification:** ⟩ ;
15a. **for** ∀ E **in** **c** θ_{beh} **do**
15(b)iv. θ_{dsu} :=
15(b)iv. **c** θ_{dsu} ⌢
15(b)ii. ⟨ { " **type**
15(b)ii. EI
15(b)iii. **value**
15(b)iii. **uid_**E: E → EI "
15(b)i. | (_,E) ∈ **elems** θ_{esn} } ⟩
15. **end**

Since there will usually be many such unique identifier definitions, each with their **type** and **value** literals, these can be summarised into one such set of definitions.

2.4.6.1.2 Stage 10: Unique Identifier States. [We refer to Example B.6 on page 40]

16. The describe_unique_identifier_states stage
    (a) augments θ_{dsu} with a **Unique Identifier States** display line; followed by a
    (b) for all (p,E) known such in θ_{esn}
    (c) with "their" **value**
    (d) contribution to the unique identifier states; followed by

(e) a $\sigma_{uid_{all_parts}}$ definition, and
(f) a $\sigma_{uid_{all_behav_parts}}$ definition.

describe

16. describe_unique_identifier_states() ≡
16a. θ_{dsu} := $\underline{\mathbf{c}}\,\theta_{dsu}$ ⌢ ⟨ **Unique Identifier States:** ⟩ ;
16b. **for** ∀ (p,E) **in** $\underline{\mathbf{c}}\,\theta_{esn}$ **do**
16a. θ_{dsu} :=
16c. $\underline{\mathbf{c}}\,\theta_{dsu}$ ⌢ ⟨ **value** ⟩
16d. ⌢ ⟨ { " p_{uid}:EI = **uid_**E(p), " | (p,E) ∈ **elems** θ_{esn} } ⟩
16e. ⌢ ⟨ " $\sigma_{uid_{all_parts}}$:UI-set = [...] " ⟩
16f. ⌢ ⟨ " $\sigma_{uid_{all_behav_parts}}$:UI-set = [...] " ⟩
16b. **end**

2.4.6.1.3 Stage 11: Uniqueness. [We refer to Example B.7 on page 41]

17. The all parts are unique **axiom** is simple: the number of all parts equals the number of all part [unique] identifiers.

axiom
17. **card** σ_{all_parts} = **card** $\sigma_{uid_{all_behav_parts}}$

2.4.6.2 Stage 12: The *mereology* Procedure. [We refer to Example B.8 on page 41]

18. The *mereology* procedure is to generate a domain description unit which records the type and **obs_**E observers for all "behavioral" parts, that is: those parts which the domain analyser cum describer considers candidate to be "morphed"into behaviours – those which are dashboard recorded in θ_{beh}:Θ_{BEH}.

(a)
(b) For all E noted in the dashboard as behavioral the
(c) **type** and
(d) **value**
(e) of the mereology definition and its observer is added to the emerging domain description.

18. discover_mereologies: **Unit** → **Unit**
18. discover_mereologies ≡
18a. θ_{dsu} := $\underline{\mathbf{c}}\,\theta_{dsu}$ ⌢ ⟨ **Mereologies:** ⟩ ;
18b. **for** ∀ E **in** $\underline{\mathbf{c}}\,\theta_{beh}$ **do**
18e. θ_{dsu} :=
18e. $\underline{\mathbf{c}}\,\theta_{dsu}$ ⌢
18c. ⟨ " **type**
18c. MereoE = $\mathcal{M}_E$(UI1,UI2,...,UIn)
18d. **value**
18d. **mereo_**E: E → MereoE " ⟩
18. **end**

Since there will usually be many such mereology definitions, each with their **type** and **value** literals, these can be summarised into one such set of definitions.

The red text shall convey to the reader that this part of the description procedure requires the "creative" thinking on the part of the domain analyser cum describer.

The $\mathcal{M}$(UI1,UI2,...,UIn) expression is over unique identifiers. It may, for example, be expressed in terms of a triplet:

- ($\mathcal{M}_i$(iUI1,iUI2,...,iUIn),$\mathcal{M}_{io}$(ioUI1,ioUI2,...,ioUIn),$\mathcal{M}_o$(oUI1,oUI2,...,oUIn))

where the

- $\mathcal{M}_i$ part $\mathcal{M}_i$(iUI1,iUI2,...,iUIn) designate behaviours from which the "e:E" behaviour offers to accept inputs,
- $\mathcal{M}_{io}$ part, $\mathcal{M}$(ioUI1,ioUI2,...,ioUIn), designate behaviours from which the "e:E" behaviour accepts input and to which it offers communication, and
- $\mathcal{M}_o$ part $\mathcal{M}$(oUI1,oUI2,...,oUIn) designate behaviours to which the "e:E" behaviour offers communication.

The $\mathcal{M}$(UI1,UI2,...,UIn) are typically of the form UI-**set**.

2.4.6.3 Stage 13: The *discover_attributes* Procedure. [We refer to Example B.9 on page 42]

19. The **attr***ibutes* procedure is to generate
20. a domain description unit which records the type and **attr_**A observers for all "behavioral" parts, that is: those parts which the domain analyser cum describer considers candidate to be "morphed" into behaviours – those which are dashboard recorded in θ_{beh}:Θ_{BEH}.
    (a) **Attributes**: prefixes the individual attribute type and observer definitions.
    (b) For all E noted in the dashboard as behavioral the domain analyser cum describer
    (c) analyses that type to have a number of attributes (A1,A2,...,An);
    (d) with these possessing respective properties (AttrType_1,AttrType_2,...,AttrType_n).

    AttrType_i are of the form:

    - AttrType_i = AttrType_Expr_i × Attr_Cat [× SI_Unit]
    - Attr_Cat = **sta**|**mon** | **prg**
    - SI_Unit = ... We refer to Appendix A on page 36 ...

    The [...] in [× SI_Unit] mean that this term is optional. Not all attributes can be given an SI Unit[22]

    sta refers to static attributes, **mon** refers to static attributes, and **prg** refers to programmable attributes.

[22] The International System of Units for physical, observably measurable phenomena:

- International System of Units - Wikipedia: `https://share.google/kckTszukBHJWt4Nwm`
- SI Units, the US NIST: `https://share.google/4lkLUKbdHheQce1nT`
- Physical Quantities, Units, and Measurements –STEM for Educators: `https://share.google/uUs65ejqDO1RUqJ3n`.

(e) θ_{dsu} is therefore extended with a domain description unit with the list element containing

(f) attribute **type** definitions and

(g) **value** definitions of the **attr**ibute observers.

```
19.  discover_attributes: Unit → Unit
19.  discover_attributes() ≡
20a.     θdsu := c θdsu ⌢ ⟨ Attributes: ⟩ ;
20b.     for ∀ E ∈ c θbeh do
20c.        let (A1,A2,...,An) = observe_attribute_names(p:E) in
20d.        let (AttrType_1,AttrType_2,...,AttrType_n) = observe_attribute_types(p:E) in
20e.        θdsu := c θdsu ⌢
20f.            ⟨ " type
20f.                    A1 = AttrType_1, A2 = AttrType_2, ..., An = AttrType_n
20g.                value
20g.                    attr_A1: E → AttrType_1,
20g.                    attr_A2: E → AttrType_2,
20g.                    ...
20g.                    attr_An: E → AttrType_n " ⟩
19.     end end end
```

Attribute types AttrType_i are type expressions over "standard" RSL definable types, unique identifier sorts and mereo types.

Red "formulas" indicate work to be done by the domain analyser cum describer.

• • •

Important attributes, ones that can be ascribed to [almost all] parts, E, is the **E_history** attributes. They may not be observable, not physically measurable, but they can be *"remembered"*, can be *"talked about"* – say by humans. They therefore objectively, indisputably records the occurrence of **events**. In our domain models we record E_history attributes as ***sequences*** *of* TIME*-stamped events*. Examples are the events of automobiles deciding to remain at hubs or on links, or stopping, or leaving or entering these, etc.; cf. Item 76 on page 42.

2.4.6.4 Stage 14: The *discover_intential_pull* Procedure.

[We refer to Example B.10 on page 43]

Whereas the expression [i.e., definition of] unique identification, mereologies and attributes are specific to individual part types, intential pulls usually range over several part types: *"roads are meant for automobiles and automobiles are meant for roads, etc."*.

Intentional pulls are expressed in the form of $\mathcal{R}_i$ implies $\mathcal{A}_j$ if-and-only-if $\mathcal{A}_p$ implies $\mathcal{R}_q$: *"if an automobile, a, is on road r, at time τ, then road r observes automobile a on that road at that time* ***and*** *if road, r, observes automobile a, at time τ, then automobile a is on road r on that road at that time"*.

21. The *discover_intential_pulls* procedure [possibly] generates an "Intentional Pull" domain description unit with zero, one or more, n, intentional pulls.

    (a) θ_{dsu} is extended

(b) by an **axiom** and

(c) n, $n \geq 0$, intentional pull predicates.

The $\mathcal{B}(...)$ terms are $\mathcal{B}$oolean predicates.

```
21.  discover_intentional_pulls: Unit → Unit
21.  discover_intentional_pulls() ≡
21a.    θ_dsu := c θ_dsu ⌢ ⟨ " Intentional Pulls:
21b.          axiom
21c.             B_p1(...) ⇒ B_q1(...)
21c.             ≡
21c.             B_x1(...) ⇒ B_y1(...) ,
21c.
21c.             B_p2(...) ⇒ B_q2(...)
21c.             ≡
21c.             B_x2(...) ⇒ B_y2(...) ,
21c.
21c.             ...
21c.
21c.             B_pn(...) ⇒ B_qn(...)
21c.             ≡
21c.             B_xn(...) ⇒ B_yn(...)   "
21b.       ⟩
```

The red formulae designates work to be done by the domain analyser cum describer. [Hard work !]

2.4.7 Stage 15: The *discover_perdurant* Procedures

22. The *discover_perdurant* stage has four sub-stages:

    (a) the description of channels,

    (b) the characterisation of behaviour actions,

    (c) the discovery of behaviours, and

    (d) the description of domain initialization;

```
22.  discover_perdurants: Unit → Unit
22.  discover_perdurants() ≡
22a.     describe_channel() ;
22b.     characterisation_of_actions() ;
22c.     discover_behaviours() ;
22d.     describe_initialization()
```

2.4.7.1 Stage 16: The *describe* channel Procedure. [We refer to Example B.13 on page 47]

23. The *describe* **channel** procedure is very straightforward:
    (a) θ_{dsu} is extended with the declaration of a **channel** array
    (b) named ch, where the distinct array indexes, ui, uj [ui ≠ uj] range over the unique identifiers of all behavioral parts.

```
23.  discover_channel: Unit → Unit
23.  describe_channel() ≡
23a.      θdsu := c θdsu ⌢ " Channels:
23b.          channel { ch[{ui,uj}] | ui,uj:UI • {ui,uj}⊆σuid_parts }: M "
```

M is a type expression referring to values other than sort names, variables, ... MORE TO COME ?
The channels are referenced in CSP input/output clauses:

- ch[{ui,uj}] ?: input to behaviour ui from behaviour uj – with ch[{ui,uj}] ? denoting a[ny] value.
  ch[{ui,uj}] ? is an expression.
- ch[{ui,uj}] ! expr: output from behaviour ui to behaviour uj of he value denoted by the expression.
  ch[{ui,uj}] ! expr is a clause, i.e., like a statement.
- [] { ch[{ui,uj}] ? | uj:UI • ui ∈ set_of_uis }: expresses the **external** non-deterministic choice of input to behaviour ui from either one of the behaviours designated by uj.
- ⨅ { ch[{ui,uj}] ! expr | uj:UI • uj ∈ set_of_uis } expresses the **internal** non-deterministic output of the value of an expression from behaviour ui to one of the behaviours in set_of_uis.
- || { ch[{ui,uj}] ! expr | uj:UI • set_of_uis }: expresses the simultaneous deterministic output to all behaviours designated by uj in the from behaviour uj.

2.4.7.2 On Behaviour Signatures.

Behaviours transpire, as I say, by transcendental deduction, that is, are "morphed" from behaviourable parts. There is one behaviour for each such part. Their signatures follows "straight" from the composition of the internal qualities of the endurant parts from which they are "morphed":

- Their name is usually chosen to relate to the name of the part sort from which they emerge.
- A first argument, by convention, is the unique identifier of "the part".
- A second argument is the mereology of the part.
- A third argument is the zero, one or more static attribute values.
- A possible fourth argument the names zero, one or more monitorable attribute.
- A "final" argument is one or more programmable attribute values.
- The result "value" of a behaviour is the **Unit** value, ().

2.4.7.3 **Stage 17: Characterisation of Actions.** Behaviours, when observed, generally consists of sets of sequences of actions, events and behaviours. We shall "restrict" this characterization to not allow "embedded" behaviours. That is: In our understanding of the kind of domains that we have "limited' ourselves – i.e., the DOMAIN method – to deal with:

- (i) any behaviour is exclusively the result of a transcendental deduction of a specific part sort,
- (ii) and such behaviours do not invoke other part behaviours.

In the definition of part behaviours we let the behaviour tail-recursively invoke "itself" [in order to go on, and on, ...].

I have skipped, till now, an important procedural stage: it is that of analysing each "emerging" part behaviour for its actions. It is the composition of these actions that determine the specification of a behaviour, i.e., its definition.

So, as a procedural stage, perhaps right after the specification of channels and thus before the specification of behaviour signatures we suggest that the domain analyser cum describers, as a group, sketch a list of behaviour actions: a list for each behaviour, noting "all" of its actions, and noting whether such actions only involve themselves, are specific to that behaviour, or communicate with other behaviours – and then which !

2.4.7.4 **Stage 18: Behavioural Taxonomy** [We refer to Example C on page 51]

A behavioural taxonomy is a pictorial rendition, a two-dimensional graph, which shows "all" the behaviours, say as circles or [rounded] boxes with arrows between these. The arrows designate communications between behaviours.

```
value
10. describe_behavioural_taxonomy: Unit → Unit
10. describe_behavioural_taxonomy() ≡
10.     let behavioural_taxonomy = design_behavioural_taxonomy() in
10.     θ_dsu := c θ_dsu ⌢
10.         ⟨ " Behavioural Taxonomy: behavioural_taxonomy "⟩
10.     end
```

2.4.7.5 **Stage 19: Behaviour Signatures and Specs.** [We refer to Example B.15 on page 47]

Behaviours transpire, as I say, by transcendental deduction, that is, are "morphed" from behaviourable parts. There is one behaviour for each such part. Their signatures follows "straight" from the composition of the internal qualities of the endurant parts from which they are "morphed":

- Their name is usually chose to relate to the name of the part sort from which they emerge.
- A first argument, by convention, is the unique identifier of "the part".
- A second argument is the mereology of the part.
- A third argument is the zero, one or more static attribute values.

- A possible fourth argument the names zero, one or more monitorable attribute.
- A "final" argument is one or more programmable attribute values.
- The result "value" of a behaviour is the **Unit** value, ().

2.4.7.6 Stage 20: Behaviour Definitions.

24. The discover_behaviours stage "takes" the state of the dashboard and updates that dashboard.
25. First a **Behaviour Definitions** display line; then
26. for all behavioural parts
    (a) the domain analyser cum describer analyses and describes the [thus transcendentally "morphed"] part into a behaviour definition.
    (b) The general "pattern" of behaviour definitions start with a signature – [where we often find that we can omit the monitorable attributes] – and and a "token invocation", followed by ≡ – i.e., the definition.
    (c) The "body" of behavioural definitions is either of a simple form:
        i. B is a function, e.g., a behaviour which terminates delivering as its result a value, called state.
        ii. From state one can "extract" updated values and the mereology of behaviour ui – for the case that behaviour introduces new, or removes existing behavioural parts – and one or more of the programmable attributes.
        iii. whereupon behaviour resumes being the behaviour, but with an updated "state" !
    (d) Or B follows the patterns illustrated above and explained in Appendix D on page 52.

```
24.   discover_behaviours: Unit → Unit
24.   discover_behaviours() ≡
25.     θ_dsu := c θ_dsu ⌢ ⟨ " Behaviour Definitions: " ⟩ ;
26.     for ∀ E ∈ c θ_beh do
26.       θ_dsu := c θ_dsu ⌢
26.         ⟨ let p:E • [ where p ≡ recorded in θ as being of type E ] in
26a.            " behaviour: UI → Mereo → StatAttrs → [ MoniAttrs → ] ProgrAttrs → Unit
26b.                 behaviour(ui)(mereo)(sta_atts)(moni_attrs)(prgr_attrs) ≡[23]
26(c)i.                let state = B(ui)(mereo)(sta)[ (mon) ](prgr) in
26(c)ii.               let (mereo′,prgr′) = analyse(state) in
26(c)iii.                behaviour(ui)(mereo′)(sta_atts)(moni_attrs)(prgr′) end end "  end ⟩
26.     end
26d.    or: [See Appendix D on page 52]
```

2.4.7.7 Patterns of Behaviour Definition Bodies.

We refer to Appendix D on page 52

[23] The domain analyser cum describer must, based on the internal qualities defined for E "fill in" the details of Mereo, StaticAttrs, MoniAttrs and ProgrammableAttrs. It should be obvious (!) how to do that. It is too cumbersome to lay that out here.

2.4.7.8 Stage 21: Domain Initialisation. [We refer to Example B.16 on page 50]

27. The describe_initialisation procedure stage first generates
    (a) a display line: **Domain Initialisation**:
    (b) then, for each behavioural part, p:E, that has been encountered, i.e., each part in $\sigma_{uid_{parts}}$
    (c) the parallel composition of behaviour invocations, be(uid)(mereo)(sta)(moni)(prg), where the name, be, of the behaviour is a suitable name that has some connotation to the part type E,
    (d) where the unique identification, uid, is obtained from p:E,
    (e) the mereology, mereo, likewise,

    and so
    (f) the static attribute values, sta,
    (g) the monitorable attribute names, moni, and
    (h) the programmable attribute values, prgr.

```
value
27.  describe_initialisation: Unit → Unit
27.  describe_initialisation() ≡
27a.     θdsu := c θdsu ⌢ ⟨ " Domain Initialisation: " ⟩
27c.     ⌢ ⟨ "  ‖ { be
27d.                  (uid_E(p))
27e.                     (mereo_E(p))
27f.                        ( { attr_A(p) | A ∈ static_attrs(p) } )
27g.                           ( { ηA[24]| A ∈ moni_attrs(p) } )
27h.                              ( { attr_A(p) | A ∈ prgr_attrs(p) } ) "
27b.              | (p,E) ∈ elems θves ∧ E ∈ c θbeh } ⟩
```

The functions static_attrs, moni_attrs and prgr_attrs follow from the category of the attribute definitions for p:E.

2.4.8 Procedure Iteration.

The enunciation of the 21 stages – as a sequential, deterministic set, i.e., list, of procedural stages – shall not be understood as the only way to analyse & describe domains. "Mistakes", that is, unsuitable choices of description, do occur. The role of the preparatory domain modeling effort, the one that should precede any seriously full-scale modeling such as the one for which these 21 procedure stages are prescribed, is to "minimize" the likelihood of "mistakes". Anyway, "mistakes" do occur. In such situations, where domain analyser cum describer[s] decide that the model they are developing must be re-oriented/re-done [in a different direction], an iteration, a re-analysis/description, as from some stage, must take place – replacing domain description units.

A "roll-back" to some earlier step, must take place.

We do not prescribe the stages involved in such a roll-back. But it should be obvious that the more the procedure stages are followed and documented, the better an analysis & description iteration can succeed.

[24] All but the monitorable arguments are *"called-by-value"*,the monitorable arguments are *"called-by-name"*.

2.5 Some 𝔻𝕆𝕄𝔸𝕀ℕ Techniques

We remind the reader (and ourselves !) of what a technique is: *a way of carrying out a particular task, especially the execution or performance of an artistic work or a scientific procedure [Oxford Languages and Google].*

MORE TO COME

2.6 Some 𝔻𝕆𝕄𝔸𝕀ℕ Tools

The foremost tools of 𝔻𝕆𝕄𝔸𝕀ℕ are the domain analysis and the domain description languages, DAL and DDL. Not much more to say about that, other than citing [27, 26]. Computerised tools in the form of support for the use of DDL do not exist – yet ! ? But would be a much needed tool in commercial developments based on he use of the 𝔻𝕆𝕄𝔸𝕀ℕ method.

3 A 𝔻𝕆𝕄𝔸𝕀ℕ Model of The Domain Modeling Procedure

The rigorous, but informal, description, Δ, that has been given in the previous section is an "ordinary", informal description given in an informal, RSL-like language. It is not a 𝔻𝕆𝕄𝔸𝕀ℕ description. But one could "lift" Δ into a domain model: One behaviour is that of the domain, uod:UoD; another behaviour is that of the dashboard state, θ:Θ; and there are then n behaviours, one for each domain analyser cum describer. The 𝔻𝕆𝕄𝔸𝕀ℕ "domain" model would then explicate how the domain analyser cum describers interact amongst themselves, with the domain, and with the dashboard state $\theta : \Theta$, ! [25]

We leave it to You to further contemplate, etc., such a model !

4 Review of the 𝔻𝕆𝕄𝔸𝕀ℕ Method

The enunciation of a method, as in this document the 𝔻𝕆𝕄𝔸𝕀ℕ method, is, we believe "a first" such computing science contribution !

4.1 Skills Assessment: The 𝔻𝕆𝕄𝔸𝕀ℕ Method

We have "laid bare" 21 stages of a domain analyser cum describer process. How difficult are, i.e., what does it take to perform these stages !

Here is my assessment:

- Stages 0-3 (2.4.1 on page 9– 2.4.5 on page 11; Initialization, Discover Domain, Initialize Domain, External Qualities) are straightforward.
- Stage 4 (2.4.5.1 on page 11; Endurants) is a crucial stage: it calls for the experienced analytical skills of the domain analyser cum describers. The crucial are the observe [Item 8a] and analyser_cum_describer_choice [Item 8b on page 13].
- Stages 5-6 (2.4.5.3 on page 15– 2.4.5.4 on page 15; Endurant Taxonomy, Endurant State) are straightforward, respectively crucial: The choices, Items 8d and 9a page 13, of the Endurant State are seemingly easy to make, but a first such may be regretted, but it is "easy" to fix !

[25] The model would have to allow concurrent updates to θ, also those which "mess-up: matters. A requirements prescription would/might then sort out such things through appropriate *transaction protocols* – i.e., a la James Gray [37].

- Stages 7-11 (2.4.6 on page 16– 2.4.6.1.3 on page 18; Internal Qualities, Unique Identifiers, Unique Identifier State, Uniqueness of Parts) are straightforward.
- Stage 12-14 (2.4.6.2 on page 18– 2.4.6.4 on page 20; Mereology, Attributes, Intentional Pull) are crucial stages: they call for the experienced analytical skills of the domain analyser cum describers. The crucial tasks are those of determining $\mathcal{M}_E$(UI1,UI2,...,UIn) Item 18c on page 18, observe_attribute_names Item 20c on page 19, observe_attribute_types Item 20d on page 19, and the discover_intentional_pulls Item 21 on page 20.
- Stages 15–16 (2.4.7 on page 21– 2.4.5.3 on page 15; Perdurants, Channel) are straightforward.
- Stage 17 (2.4.7.3 on page 23; Actions) is a crucial stage: it calls for the experienced analytical skills of the domain analyser cum describers. I am presently not satisfied with this section !
- Stages 18-19 (2.4.7.4 on page 23– 2.4.7.5 on page 23; Behaviourable Taxonomy, Signatures) are, on the background of the completion of the previous stages, relatively straightforward.
- Stage 20 (2.4.7.6 on page 24; Behaviour Definitions) is a crucial stage: it calls for the experienced specification, "CSP programming" skills of the domain analyser cum describers. I am presently – likewise – not satisfied with this section !
- Stage 21 (2.4.7.8 on page 25; Domain Initialization) is straightforward.

4.2 My Characterisation of the Term 'Method'

Dear me and possible readers: is my characterisation[26] of the term 'method' commensurate with "other" characterisations of that term ?

First we refer to Descartes: [34]:

- *"**Le Discours de la méthode** (1637) de René Descartes est une œuvre fondatrice de la philosophie moderne, prônant l'usage de la raison pour trouver la vérité scientifique. Il propose une approche rigoureuse en quatre règles: (i) **ne rien accepter pour vrai**, (ii) **diviser les difficultés**, (iii) **ordonner les pensées du simple au complexe**, et (iv) **faire des dénombrements complets**."*.

Descartes thus proposed a simple, four-step process (i–iv) to ensure that the mind never accepts a falsehood as truth:

- **The Four Fundamental Rules:**
  - (i) **The Rule of Evidence:** Never accept anything as true unless it is so "clear and distinct" that there is no reason to doubt it. This is a rejection of prejudice and jumping to conclusions.
  - (ii) **The Rule of Analysis:** Break down large, complex problems into as many smaller parts as possible. Solving many small problems is easier than tackling one giant one.
  - (iii) **The Rule of Synthesis:** Organize your thoughts by starting with the simplest objects and gradually "climbing" to the most complex, assuming an order even where none naturally exists.

[26] By a **method** we shall understand a *set of principles* and *procedures* for *selecting* and *applying* a *set of techniques using* a *set of tools* in order to *construct* an *artefact* [DB].

(iv) **The Rule of Enumeration:** Make thorough lists and reviews to ensure that nothing has been omitted. This is essentially a final "audit" of your logic.

Then from the `Internet`. A method is:

- *"a way of doing something, especially a systematic way; implies an orderly logical arrangement (usually in steps)"* [https://www.vocabulary.com/dictionary/method];
- (i) *"a procedure, technique, or way of doing something, especially in accordance with a definite plan"*, (ii) *"a manner or mode of procedure, especially an orderly, logical or systematic way of instruction, investigation, experiment, presentation, etc."*, (iii) *"order or system of doing anything"*, (iv) *"order or systematic arrangement, sequence or the like"* [https://www.dictionary.com/browse/method];
- *"a procedure or process for attaining an object: such as as"* (a) *"a systematic plan followed in presenting material for instruction"*, or (b) *"a way, technique, or process of or doing something"*, or (c) *"a body of skills"*, or (d) *"a discipline that deals with the principles and techniques of scientific inquiry"* [https://www.merriam-webster.com/dictionary/method];
- Et ctera.

In view of the above, I do think that my characterisation is commensurate with those "of others" !

4.3 A Critical Review of the 𝔻𝕆𝕄𝔸𝕀ℕ Method

We risk a critical assessment of the 𝔻𝕆𝕄𝔸𝕀ℕ method. The itemized list below is not prioritized.

- **Part Sets:** Part sets are endurants of the form PS = P-**set**. We have, not arbitrarily, but from experimental evidence [2], decided to limit parts p:P] of part sets] to be atomic. Is that limitation well-founded: not really necessary ? We must admit that it is not ! Let us give an **example:**

  > Let us take a domain of companies, say like A. P. Møller Maersk[27] https://www.maersk.com/. It has many divisions, typically one could model these many divisions as a part set. That is: a division is a part p:P where PS = P-**set**.
  >
  > Now these individual divisions would in the present 𝔻𝕆𝕄𝔸𝕀ℕ method be modeled as atoms. No chance for divisions to have sets of subdivisions, that is: being [non-atomic] compounds !

  So, there really is no good reason to mandate that p:P in PS = P-**set** be atomic.

  So why have we, so far, restricted p:P in PS = P-**set** to be atomic. The simple, but perhaps embarrassing answer is: to keep some matters simple[28].

  An example of where the present limitation makes the presentation of the method simpler is in the Discover Endurant State stage of the procedure, Sect. 2.4.5.4 on page 15. See items 11d–11e on page 16. "Lifting" the restriction from atoms to parts in general would mean that, say a

[27] Maersk is a Danish company. From its homepage: *It is an integrated logistics company that offers supply chain solutions for managing shipments and cargo*

[28] Einstein: *"Everything Should Be Made as Simple as Possible, But Not Simpler"* https://quoteinvestigator.com/2011/05/13/einstein-simple.

recursively defined exploration of sub-parts in line 11e on page 16 would have to be specified – and all the domain examples [2] have "survived" with the restriction ! We leave it to the reader to lift the restriction !

- **Sequentiality of DOMAIN Modeling Procedure:** A "backbone" of the procedure of Sect. 2.4 (pages 8–25) is that it suggests a sequential domain modeling process.

  But is it always possible to model domains sequentially ?

  No: domain analyser cum describers may find that they have "come across" a "more correct", a more elegant, a shorter, ..., way for a domain model than the one they are currently pursuing. So what to do ? Stick to the less desirable, "past/current" approach – or shift to the more "correct" model ! We suggest, of course, that they shift.

  Such shifts imply that the modeling retrace: "go back to an earlier stage" – the one marking point where the currently less desirable model was "mis-directed" and redo the modeling stages from there.

  The question is then, what, if any of the current description can be salvaged, that is: edited into "a more desirable" model ?

  The current procedure does not prescribe such domain modeling iterations !

- **Weak Treatment of Actions:** The treatment of actions, Sect. 2.4.7.3 on page 23 is presently not satisfactory. It lacks a systematic, convincingly "line of thought". The aim of Sect. 2.4.7.3 should include specific guidelines so that domain analyser cum describers can rigorously formulate the "tasks" of corresponding behaviour specifications.

- **Weak Treatment of Behaviour Patterns:** Similar to the treatment of actions, the treatment of behaviour patterns, that is the way behaviour definitions are structured, in Sect. 2.4.7.5 on page 23, is presently not satisfactory. It lacks a systematic, convincingly "line of thought". The aim of Sect. 2.4.7.5 should include specific guidelines so that domain analyser cum describers can rigorously formulate behaviour specifications.

- **Possible Formal Relations Between Taxonomies and The Formal Descriptions:** Sections 2.4.5.3 on page 15 and 2.4.7.4 on page 23 presented issues of endurant taxonomy, respectively behavioural taxonomy. It was suggested that these graphical renditions of domain facets in no way could be claimed to augment or detract from the [mathematical] meaning of the [emerging] domain model.

  But suppose the could ! That is, can one give a semantics to these graphical renditions and formally relate them to the semantics of "the rest" of the [emerging] domain model. Or, put it differently, can these graphical renditions be automatically "derived" from that [emerging] model ?

- **Not Quite Satisfactory Treatment of Monitorables:** Monitorable attributes are such whose values are beyond "control" of "their" behaviours. Examples are *the velocity of an automobile, its acceleration, the temperature of an endurant, etc.*

  The way the current version of the DOMAIN method treats monitorable attributes is as follows. First, monitorables, A, "appear" as arguments in behaviour definitions "by name", that is as ηA. Then, whenever encountered, as a, during the elaboration of a behaviour invocation, where the unique identifier of the behaviour is ui, that elaboration proceeds as follows:

(i) The part, p, with unique identifier ui in state value σ_{parts} is "retrieved".

(ii) If the elaboration is for the purposes of obtaining the current value, v of attribute a:A, then **attr_**A(p) is obtained and is the value of a.

(iii) If the elaboration is for the purposes of changing the value of a, then its occurrence in behaviour ui is [somehow, say through a function ϕ] "paired" with a value, w with which to update attribute A of p. The new value, in p of σ_{parts}, is set to ϕ(v,w)[29] – leaving all other internal qualities unchanged !

We are not quite happy with this model of monitorable attributes – but have not spent much time of finding other models. The reason for this "omission" is that we have not been confronted with domains where we have had to model parts with monitorable attributes. Having and original, engineering background in control theory, i.e., in its application[30], it appears that we have "shied" away and focused the 𝔻𝕆𝕄𝔸𝕀ℕ method on discrete, rather than continuous, dynamics !

But it might be worthwhile to reformulate the role of monitorables !

- **Precise Syntax of DDL:** There is, as yet, no document which specifies the syntax of DDL and RSL$^+$text; but see [27, 26].

There will undoubtedly appear more critical items. [This list was first made up February 26, 2026.]

5 Conclusion

We have presented a method for analysing & describing domains – such as we have "defined" that term. The method entails a linear procedure, outlined here in 21 stages. It is, we suggest, an interesting "feature" that it is linear ! At each stage we need not refer to issues that are first introduced in later stages.

There is, however, more to domain modeling than the procedural model presented here !

There is, for example, the modeling of *domain facets* [29, *Chapter 8*]. We leave it to others to systematize their pursuit.

• • •

This document is one of a quadruple set of documents [25, 26, 27, 28].

As of March 8, 2026: 02:43 am Sects. 2.5 on page 26 and 2.6 on page 26, have yet to be written. So it is a bit too early to comment on whether a reason for this study has been achieved. But let me try.

The work of [25, 26, 27, 28] has, as one of its core contributions, I claimed, the enunciation, the clarification of 𝔻𝕆𝕄𝔸𝕀ℕ modeling, i.e., the analysis & formalisation method. The [25, 26, 27, 28] documents are not aimed at publication. Their developments are done purely for elucidation, characterisation and my own pleasure.

Now, in thinking about and writing the present document, have I clarified, i.e., improved my own understanding of the 𝔻𝕆𝕄𝔸𝕀ℕ method ?

Yes. Section 4.3 summarizes "sticky" points – to [possibly] be tackled next !

[29] We leave the details of ϕ to the reader's imagination !

[30] (i) Sandviken Steel: 18-9 Chrome-Nickel steel production automation [https://www.home.sandvik/en/]; (ii) Billerud: paper production automation [https://www.billerud.com/] – at the IBM Nordic Laboratories, Stockholm, Sweden in the early 1960s

Acknowledgments

Todays computer scientists appear, to me, to really not be [scholarly] interested in the work of their colleagues if it "falls" outside their own, usually highly specialised [theoretical] area. So, in the winter/spring of 2026, when the topic of this document is being researched: tought about, reflected upon and written down, I have to come to terms with the apparent fact that nobody, really, cares about my work on 𝔻𝕆𝕄𝔸𝕀𝕟s.

Here I am very happy to acknowledge the works of Tony Hoare, Michael A. Jackson, my IBM Vienna Laboratory 1973–1975 colleagues: the late Peter Lucas, the late Hans Bekič and Cliff B. Jones, and Hans Bruun, Chris W. George and the late Søren Prehn.

6 Bibliography

I generously cite my own work ! The recent `https://plato.stanford.edu/entries/ontology-is/` is most readable, and is recommended.

References

[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] Dines Bjørner. Domain Modeling Case Studies:

- 2025: *Transport: A Domain Description*, March 2025, `https://www.imm.dtu.dk/ dibj/2025/transport.pdf`
- 2025: *Banking: A Domain Description*, August 2025, `https://www.imm.dtu.dk/~dibj/2025/banking/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 – A Technical Note*, 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*, Inst. of Softw., Chinese Acad. of Sci., Peking, China `www.imm.-dtu.dk/~dibj/2017/swarms/swarm-paper.pdf`
- 2013: *Road Transport*, Techn. Univ. of Denmark `www.imm.dtu.dk/~dibj/2013/-road/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-1.pdf`, `www.imm.dtu.dk/~db/todai/tse-2.pdf`
- 2009: *Pipelines*, Techn. Univ. of Graz, Austria `www.imm.dtu.dk/~dibj/pipe-p.pdf`
- 2007: *A Container Line Industry Domain*, Techn. Univ. of Denmark `www.imm.dtu.-dk/~dibj/container-paper.pdf`
- 2002: *The Market*, Techn. Univ. of Denmark `www.imm.dtu.dk/~dibj/themarket.pdf`
- 1995–2004: *Railways*, Techn. Univ. of Denmark - a compendium `www.imm.dtu.-dk/~dibj/train-book.pdf`

.

[3] Dines Bjørner. Towards a Meaning of 'M' in VDM. In E.J. Neuhold and M. Paul, editors, *Formal Description of Programming Concepts*, IFIP State–of–the–Art Reports, pages 137–258. Springer-Verlag, Heidelberg, Germany, 1991. An IFIP TC2 Seminar, Persepolis, Brasil.

[4] Dines Bjørner. *Software Engineering, Vol. 1: Abstraction and Modeling.* Texts in Theoretical Computer Science, the EATCS Series. Springer, 2006. See [7, 12].

[5] Dines Bjørner. *Software Engineering, Vol. 2: Specification of Systems and Languages.* Texts in Theoretical Computer Science, the EATCS Series. Springer, 2006. Chapters 12–14 are primarily authored by Christian Krog Madsen. See [8, 13].

[6] Dines Bjørner. *Software Engineering, Vol. 3: Domains, Requirements and Software Design.* Texts in Theoretical Computer Science, the EATCS Series. Springer, 2006. See [9, 14].

[7] Dines Bjørner. ***English**: Software Engineering, Vol. 1: Abstraction and Modeling.* Qinghua University Press, 2008. Republication of [4].

[8] Dines Bjørner. ***English**: Software Engineering, Vol. 2: Specification of Systems and Languages.* Qinghua University Press, 2008. Republication of [5].

[9] Dines Bjørner. ***English**: Software Engineering, Vol. 3: Domains, Requirements and Software Design.* Qinghua University Press, 2008. Republication of [6].

[10] Dines Bjørner. From Domains to Requirements `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.

[11] Dines Bjørner. *Domain Engineering: Technology Management, Research and Engineering.* Research Monograph (# 4); JAIST Press, 1-1, Asahidai, Nomi, Ishikawa 923-1292 Japan, 2009. This Research Monograph contains the following main chapters:

1. *On Domains and On Domain Engineering – Prerequisites for Trustworthy Software – A Necessity for Believable Management*, pages 3–38.
2. *Possible Collaborative Domain Projects – A Management Brief*, pages 39–56.
3. *The Rôle of Domain Engineering in Software Development*, pages 57–72.
4. *Verified Software for Ubiquitous Computing – A VSTTE Ubiquitous Computing Project Proposal*, pages 73–106.
5. *The Triptych Process Model – Process Assessment and Improvement*, pages 107–138.
6. *Domains and Problem Frames – The Triptych Dogma and M.A.Jackson's PF Paradigm*, pages 139–175.
7. *Documents – A Rough Sketch Domain Analysis*, pages 179–200.
8. *Public Government – A Rough Sketch Domain Analysis*, pages 201–222.
9. *Towards a Model of IT Security — – The ISO Information Security Code of Practice – An Incomplete Rough Sketch Analysis*, pages 223–282.
10. *Towards a Family of Script Languages – – Licenses and Contracts – An Incomplete Sketch*, pages 283–328.

.

[12] Dines Bjørner. ***Chinese**: Software Engineering, Vol. 1: Abstraction and Modeling*. Qinghua University Press. Translation of [4] by Dr Liu BoChao et al., 2010.

[13] Dines Bjørner. ***Chinese**: Software Engineering, Vol. 2: Specification of Systems and Languages*. Qinghua University Press. Translation of [5] by Dr Liu BoChao et al., 2010.

[14] Dines Bjørner. ***Chinese**: Software Engineering, Vol. 3: Domains, Requirements and Software Design*. Qinghua University Press. Translation of [6] by Dr Liu BoChao et al., 2010.

[15] 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.

[16] Dines Bjørner. Domain Science & Engineering – *From Computer Science to The Sciences of Informatics, Part I of II: The Engineering Part*. *Kibernetika i sistemny analiz*, 2(4):100–116, May 2010.

[17] Dines Bjørner. Domain Science & Engineering – *From Computer Science to The Sciences of Informatics Part II of II: The Science Part*. *Kibernetika i sistemny analiz*, 2(3):100–120, June 2011.

[18] Dines Bjørner. Domains: Their Simulation, Monitoring and Control – A Divertimento of Ideas and Suggestions. In *Rainbow of Computer Science, Festschrift for Hermann Maurer on the Occasion of His 70th Anniversary.*, Festschrift (eds. C. Calude, G. Rozenberg and A. Saloma), pages 167–183. Springer, Heidelberg, Germany, January 2011. `www.imm.dtu.dk/~dibj/maurer-bjorner.pdf`.

[19] Dines Bjørner. Manifest Domains: Analysis & Description. *Formal Aspects of Computing*, 29(2):175–225, March 2017. First Online: 26 July 2016. DOI 10.1007/s00165-016-0385-z.

[20] Dines Bjørner. Domain Analysis & Description – Principles, Techniques and Modeling Languages. *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, January 2020. xii+346 pages[31].

[22] Dines Bjørner. Banking – A Domain Description. Technical report, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, 18 August 2025. https://www.imm.dtu.dk/~dibj/2025/-banking/main.pdf.

[23] Dines Bjørner. Domain Analysis & Description. In *Theoretical Aspects of Computing, ICTAC 2025*, number 16237 in Lecture Notes in Computer Science, pages 39–66. Springer, November 24–28 2025. A Tutorial. DOI 10.1145/3796228.

[24] Dines Bjørner. Transport – A Domain Description. Technical report, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, 12 June 2025. https://www.imm.dtu.dk/~dibj/2025/transport/main.pdf.

[25] Dines Bjørner. A Linguistics Study of 𝔻𝕆𝕄𝔸𝕀ℕ. Technical report, DTU Compute, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, Denmark, January–March 2026. `https://www.imm.dtu.dk/~dibj/2026/linguistics/main.pdf`. See also [27, 26, 28].

[26] Dines Bjørner. A Syntax for DADL – First Attempt. Technical report, DTU Compute, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, Denmark, January–March 2026. `https://www.imm.dtu.dk/~dibj/2026/syntax/main.pdf`. See also [27] and [28].

[27] Dines Bjørner. DADL: A 𝔻𝕆𝕄𝔸𝕀ℕ Analysis & Description Language. Technical report, DTU Compute, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, Denmark, February 16, 2026 2026. `https://www.imm.dtu.dk/~dibj/2026/ddl/ddl.pdf`. See also [26] and [28].

[28] Dines Bjørner. Methodology – A Study. Technical report, DTU Compute, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, Denmark, January–March 2026. `https://www.imm.dtu.dk/~dibj/2026/method/main.pdf`. See also [27] and [26].

[29] Dines Bjørner. Domain Science & Engineering, a Primer[32] Technical University of Denmark. Significantly revised edition of [21]. xii+211 pages., February 2026 To be submitted [2026/2027].

[30] Dines Bjørner and Mikhail Chupilko. Моделирование предметной области: основы. Translation of [29]. [Book], To be submitted [2026/2027].

[31] Dines Bjørner and Cliff B. Jones, editors. *The Vienna Development Method: The Meta-Language*, volume 61 of *LNCS*. Springer, Heidelberg, Germany, 1978.

[32] Dines Bjørner and Cliff B. Jones, editors. *Formal Specification and Software Development*. Prentice-Hall, London, England, 1982.

[31] Due to copyright reasons no URL is given to this document's possible Internet location. A revised version of this book is [29]

[32] This book is currently being translated into Russian, [30], by Dr. Mikhail Chupilko and his colleagues, ISP/RAS (Institute of Systems Programming, Russian Academy of Sciences), Moscow and into Chinese, [33], by Dr. Yang ShaoFa, IoS/CAS (Institute of Software, Chinese Academy of Sciences), Beijing.

[33] Dines Bjørner and Yang ShaoFa. 领域科学与工程导论. Translation of [29]. [Book], To be submitted [2026/2207].

[34] René Descartes. *Discours de la méthode. Texte et commentaire par Étienne Gilson.* Paris: Vrin, 1987.

[35] John Fitzgerald and Peter Gorm Larsen. *Software System Design: Formal Methods into Practice.* Cambridge University Press, The Edinburgh Building, Cambridge CB2 2RU, UK, 1997.

[36] 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 Hampstead, England, 1992.

[37] James Gray and Andreas Reuter. *Transaction Processing : Concepts and Techniques.* Series in Data Management Systems. Morgan Kaufmann, 1993. ISBN.

[38] Daniel Jackson. *Software Abstractions: Logic, Language, and Analysis.* The MIT Press, Cambridge, Mass., USA, April 2006. ISBN 0-262-10114-9.

[39] Gawin Lowe. Analysing a Library of Concurrency Primitives using CSP. *Formal Aspects of Computing*, December 2025. 41 Pages: https://dl.acm.org/doi/10.1145/3779649.

[40] James Charles Paul Woodcock and James Davies. *Using Z: Specification, Proof and Refinement.* Prentice Hall International Series in Computer Science, London, England, 1996.

A SI Units

We refer to:

- International System of Units - Wikipedia: `https://share.google/kckTszukBHJWt4Nwm`
- SI Units, the US NIST: `https://share.google/4lkLUKbdHheQce1nT`
- Physical Quantities, Units, and Measurements –STEM for Educators: `https://share.google/uUs65ejqDO1RUqJ3n`

A.1 Base SI Units

Base quantity	Name	Type
length	meter	m
mass	kilogram	kg
time	second	s
electric current	ampere	A
thermodynamic temperature	kelvin	K
amount of substance	mole	mol
luminous intensity	candela	cd

Table 1: Base SI Units

A.2 Derived SI Units

Name	Type	Derived Quantity	Derived Type
radian	rad	angle	m/m
steradian	sr	solid angle	$m^2 \times m^{-2}$
Hertz	Hz	frequency	s^{-1}
newton	N	force, weight	$kg \times m \times s^{-2}$
pascal	Pa	pressure, stress	N/m^2
joule	J	energy, work, heat	N×m
watt	W	power, radiant flux	J/s
coulomb	C	electric charge	s×A
volt	V	electromotive force	W/A ($kg \times m^2 \times s^{-3} \times A^{-1}$)
farad	F	capacitance	C/V ($kg^{-1} \times \cdot\ m^{-2} \times s^4 \times A^2$)
ohm	Ω	electrical resistance	V/A ($kg \times m^2 \times s^{-3} \times A^{-2}$)
siemens	S	electrical conductance	A/V (kg–1×$m^{-2} \times s^3 \times A^2$)
weber	Wb	magnetic flux	V×s ($kg \times m^2 \times s^{-2} \times A^{-1}$)
tesla	T	magnetic flux density	Wb/m^2 ($kg \times s^{-2} \times A^{-1}$)
henry	H	inductance	Wb/A ($kg \times m^2 \times s^{-2} \times A^2$)
degree Celsius	^{o}C	temp. rel. to 273.15 K	K
lumen	lm	luminous flux	cd×sr (cd)
lux	lx	illuminance	lm/m^2 ($m^{-2} \times cd$)

Table 2: Derived SI Units

A.3 Further SI Units

Name	Explanation	Derived Type
area	square meter	m^2
volume	cubic meter	m^3
speed	meter per second	m/s
wave number	reciprocal meter	m-1
mass density	kilogram per cubic meter	kg/m^3
specific volume	cubic meter per kilogram	m3/kg
current density	ampere per square meter	A/m^2
magnetic field strength	ampere per meter	A/m
substance concentration	mole per cubic meter	mol/m3
luminance	candela per square meter	cd/m^2
mass fraction	kilogram per kilogram	kg/kg = 1

Table 3: Further SI Units

A.4 Standard Prefixes for SI Units of Measure

Prefix name		deca	hecto	kilo	mega	giga
Prefix symbol		da	h	k	M	G
Factor	10^0	10^1	10^2	10^3	10^6	10^9

Prefix name	tera	peta	exa	zetta	yotta
Prefix symbol	T	P	E	Z	Y
Factor	10^{12}	10^{15}	10^{18}	10^{21}	10^{24}

Table 4: Standard Prefixes for SI Units of Measure

A.5 SI Units of Measure and Fractions

Prefix name		deca	hecto	kilo	mega	giga
Prefix symbol		da	h	k	M	G
Factor	10^0	10^1	10^2	10^3	10^6	10^9

Prefix name	tera	peta	exa	zetta	yotta
Prefix symbol	T	P	E	Z	Y
Factor	10^{12}	10^{15}	10^{18}	10^{21}	10^{24}

Prefix name		deci	centi	milli	micro	nano
Prefix symbol		d	c	m	μ	n
Factor	10^0	10^{-1}	10^{-2}	10^{-3}	10^{-6}	10^{-9}

Prefix name	pico	femto	atto	zepto	yocto
Prefix symbol	p	f	a	z	y
Factor	10^{-12}	10^{-15}	10^{-18}	10^{-21}	10^{-24}

Table 5: SI Units of Measure and Fractions

B Example

B.1 Universe of Discouse

28. The universe of discourse is that of
29. road traffic, RT –
30. whose narrative is then presented.

28. **universe of discourse**:
29. **type** RT
30. **narrative**: The road traffic domain consists of a road net aggregate and an automobile aggregate.
    Road net aggregates consists of aggregates of hubs (street [link] intersections) and links.
    The aggregates of hubs and links consists of sets of hubs and sets of links.
    Hubs and links are here considered atomic.
    Automobile aggregates are″ a set **of** automobiles −− that are here considered atomic.
    Hubs, links and automobiles have unique identification, mereologies and attributes.
    Autombile ride along hubs and links; leave hubs [links] to enter links [hubs], etc.

B.2 Compound Endurants

31. In the road traffic domain we can observe a road net aggregate and an automobile aggregate.
32. In a road net aggregate we can observer an aggregate of hubs and an aggregate of links.
33. In an aggregate of automobiles we can observe a set of [atomic] automobiles.
34. Aggregates of hubs ad links are sets of [atomic] hubs and links.

type
31. RNA, AA
32. AH, AL
33. AS = A-**set**
34. HS = H-**set**, LS = L-**set**

value
31. **obs_**RNA: RT → RNA, **obs_**AA: RT → AA
32. **obs_**AH: RNA → AH, **obs_**AL: RNA → AL
33. **obs_**AS: AA → AS
34. **obs_**HS: AH → RS, **obs_**LS: AL → LS

B.3 Endurant Values

35. There is the road transport [universe of discourse].
36. There is the road net aggregate.
37. There is the automobile aggregate.
38. There is the hub aggregate.
39. There is the link aggregate.
40. There is the set of automobiles.
41. There is the set of hubs.
42. There is the set of links.
43. And there is the entire set of all of these.
44. And there is the entire set of just all automobiles, hubs and links.

value
35. rt:RT
36. rna:RNA = **obs_**RNA(rt)
37. aa:AA = **obs_**AA(rt)
38. ah:AH = **obs_**AA(rna)
39. al:AL = **obs_**AL(rna)
40. as:AS = **obs_**AS(ah)
41. hs:HS = **obs_**HS(ah)
42. ls:LS = **obs_**LS(al)
43. σ_{parts} = {rt,rna,aa,ah,al} ∪ aa ∪ as ∪ hs ∪ ls
44. σ_{atoms} = aa ∪ hs ∪ ls

B.4 Taxonomy

See Fig. 2 on the following page.

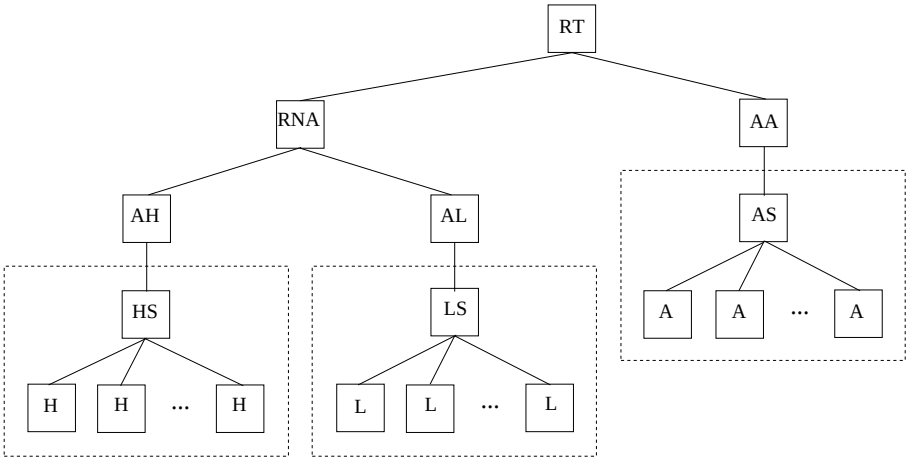

Figure 2: A Road Transport Taxonomy

B.5 Unique Identification

The unique identifierss of a road transport, rt:RT, is here limited to just hubs, links and automobiles:

45. The universe of discourse has a unique identifier.
46. The road net agregate has a unique identifier.
47. The automobile aggregate has a unique identifier.
48. The hub aggregate has a unique identifier.
49. The link aggregate has a unique identifier.
50. Each hub has a unique identifier.
51. Each link has a unique identifier.
52. Each automobile has a unique identifier.

type
45. RTI
46. RNAI
47. AAI
48. HAI
49. LAI
50. HI
51. LI
52. AI

value
45. **uid_**RT: RT → RTI
46. **uid_**RNA: RNA → RNAI
47. **uid_**AA: AA → AAI
48. **uid_**HA: HA → HAI
49. **uid_**LA: LA → LAI
50. **uid_**H: H → HI
51. **uid_**L: L → LI
52. **uid_**A: A → AI .

B.6 Unique Identifier Values

53. There is the unique identifier of the road transport.

54. There is the unique identifier of the road net aggregate.
55. There is the unique identifier of the autombile aggregate.
56. There is the unique identifier of the hub aggregate.
57. There is the unique identifier of the link aggregate.
58. There are the unique identifiers of the automobiles of the set of autombiles.
59. There are the unique identifiers of the hubs of the set of hubs.
60. There are the unique identifiers of the links of the set of links.
61. And there is the entire set of all of these.
62. And there is the entire set of all identifiers of the automobiles, hubs and links.

value
53. rt$_{uid}$RT: RT = **uid_**RT(rt)
54. rna$_{uid}$RNA: RNA = **uid_**RNA(rt)
55. aa$_{uid}$AA: AA = **uid_**AA(rt)
56. ah$_{uid}$AH: AH = **uid_**AA(rna)
57. al$_{uid}$AL: AL = **uid_**AL(rna)
58. as$_{uid}$AS: AS = {**uid_**A(a)|a:A•a∈as}
59. hs$_{uid}$:HS = {**uid_**H(h)|h:H•h∈hs}
60. ls$_{uid}$:LS = {**uid_**L(l)|l:L•a∈ls}
61. $\sigma_{parts_{uid}}$ = {rt,rna,aa,ah,al} ∪ aa ∪ hs unon ls
62. $\sigma_{atoms_{uids}}$ = aa ∪ hs ∪ ls

B.7 Uniqueness of Endurants

63. Parts are uniquely identfied.

axiom
63. **card** σ_{atoms} = **card** $\sigma_{atoms_{uids}}$

B.8 Mereology

We shall be concerned only with the mereology of some manifest parts.

64. The mereology of links is a 2 element set of hub identifiers.
65. The mereology of a hub is a possibly empty set of hub identifiers.
66. The mereology of an automobile is a set of hub and link identifiers

type
64. ML = LI-**set** **axiom** ∀ ml:MK • **card** ml = 2 ∧ ml ⊆ ls_{uis}
65. MH = HI-**set** **axiom** ∀ mh:MH • mh ⊆ hs_{uis}
66. MA = (HI|LI)-**set** **axiom** ∀ ma:MA • ma ⊆ as_{uis}

value
64. **mereo_**L: L → ML
65. **mereo_**H: H → MH
66. **mereo_**A: A → MA .

B.9 Attributes

Example attributes are:

- **Hubs:**
  67. Hubs have states, hσ:HΣ: the set of pairs of link identifiers, ($f$li,$t$li), of the links $f$rom and to which automobiles may enter, respectively leave the hub.
  68. Hubs have state spaces, hω:HΩ: the set of hub states "signaling" which states are open/closed, i.e., **green**/red.
- **Links:**
  69. Links that have lengths, LEN;
  70. Links have states, lσ:LΣ: the set of pairs of link identifiers, ($f$li,$t$li), of the links $f$rom and to which automobiles may enter, respectively leave the hub.
  71. Links have state spaces, lω:LΩ: the set of link states "signaling" which states are open/closed, i.e., **green**/red.
- **Automobiles:**
  72. Automobiles have road net positions, APos,
      (a) either *at a hub*, atH,
      (b) 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.
  73. Automobiles have velocity;
  74. Automobiles have acceleration;
  75. Etc.[33]
- **Hubs, Links, Automobiles:**
  76. Hubs, links and automobiles 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.

[33]

77. Link positions have well-defined identifiers and fractions.

type
67. HΣ = (LI×LI)-**set** [**prg**]
68. HΩ = HΣ-**set** [**sta**]
69. LEN = **Nat** [**sta**] [m]
70. LΣ = (HI×HI)-**set** [**prg**]
71. LΩ = LΣ-**set** [**sta**]
72. APos = atH | onL [**prg**]
72a. atH :: HI
72b. onL :: LI × (fhi:HI × f:**Real** × thi:HI)
73. Veloc = **Real** [**mon**] [km/h]
74. Accel = **Real** [**mon**] [m/sec]
75. ...
76. HHis,LHis = ($\mathbb{TIME}$×AI)* [**prg**]
76. AHis = ($\mathbb{TIME}$×(HI|LI))* [**prg**]

value
67. attr_HΣ: H → HΣ
68. attr_HΩ: H → HΩ
69. attr_LEN: L → LEN
70. attr_LΣ: L → LΣ
71. attr_LΩ: L → LΩ
72. attr_APos: A → APos
73. attr_Veloc: A → Veloc
74. attr_Accel: A → Accel
76. attr_HHis: H → HHis
76. attr_LHis: L → LHis
76. attr_AHis: A → AHis

axiom
77. ∀ mk_onL(li,(fhi,f,thi)):onL • 0<f<1 ∧ li∈ls_{uids} ∧ {fhi,thi}⊆hs_{uids} ∧ ....

B.10 Intentional Pull

78. An *intentional pull* of any road transport system, rts, is then:
    (a) if for any automobile, a, of rts,
        on a link, ℓ (hub, h),
        at time τ,
    (b) then that link, ℓ, (hub h)
        "records" automobile a
        at that time.
79. and:
    (c) if for any link, ℓ (hub, h)
        being visited by an automobile, a,
        at time τ,
    (d) then that automobile, a,
        is visiting that link, ℓ (hub, h),
        at that time.

axiom
78a. ∀ a:A • a ∈ as ⇒
78a. **let** ahist = attr_AHist(a) **in**
78a. ∀ ui:(LI|HI) • ui ∈ **dom** ahist ⇒
78b. ∀ τ:$\mathbb{TIME}$ • τ ∈ **elems** ahist(ui) ⇒
78b. **let** hist = is_LI(ui) → attr_LHist(retr_L(ui))(σ),
78b. _ → attr_HHist(retr_H(ui))(σ) **in**

78b. $\tau \in$ **elems** hist(uid_A(a)) **end end**
79. $\wedge$
79c. $\forall$ u:(L|H) • u $\in ls \cup hs \Rightarrow$
79c. **let** uhist = attr(L|H)Hist(u) **in**
79d. $\forall$ ai:AI • ai $\in$ **dom** uhist $\Rightarrow$
79d. $\forall \tau$:𝕋𝕀𝕄𝔼 • $\tau \in$ **elems** uhist(ai) $\Rightarrow$
79d. **let** ahist = attr_AHist(retr_A(ai))(σ) **in**
79d. $\tau \in$ **elems** uhist(ai) **end end**

B.11 Auxiliary Types

We introduce the concepts or *paths*, i.e., *routes*, through/across a road net.

80. A path element identifier is either a link identifier or a hub identifier.
81. A path (of a road net) is a finite[34] sequence of one or more alternating hub and link identifiers
82. such that
    (a) neighbouring link identifiers are those of the mereology of the "in-between" hubs, and such that neighbouring hub identifiers are/is those of the mereology of the "in-between" link;
    (b) and hub identifiers of a path are hub identifiers of the road net,
    (c) and its neighbouring link identifier(s) are in the mereology of the identified hub;
    (d) and link identifiers of a path are link identifiers of the road net,
    (e) and its neighbouring hub identifier(s) are/is in the mereology of the identified link.
83. Given a hub [a link] identifier we can retrieve the identified hub [link].

type
80. PEI = LI | HI
81. Path = PEI*
82. **axiom** [Well-formed Paths]
81. $\forall$ path:Path •
82a. $\forall$ {i,i+1}$\subseteq$**inds** path $\Rightarrow$
82a. ((is_HI(path[i])$\wedge$is_LI(path[i+1]) $\vee$ is_LI(path[i])$\wedge$is_HI(path[i+1]))
82b. $\wedge$ (path[i]$\in hs_{uis}\Rightarrow$path[i+1]$\in ls_{uis}$
82c. $\wedge$ **uid_**H(retr_hub(path[i]))$\in$**mereo_**L(retr_hub(path[i])))
82d. $\wedge$ (path[i]$\in ls_{uis}\Rightarrow$path[i+1]$\in hs_{uis}$
82e. $\wedge$ **uid_**L(retr_link(path[i])$\in$**mereo_**H(retr_link(path[i]))))

value
83. retr_hub: HI $\to$ H, retr_link: LI $\to$ L, retr_unit: UI $\to$ U
83. retr_hub(hi) **as** h • h $\in hs \wedge$ **uid_**H(h)=hi
83. retr_link(li) **as** l • l $\in ls \wedge$ **uid_**L(l)=li
83. retr_unit(ui) **as** u • u $\in hs \cup ls \wedge$ **uid_**U(u)=ui
83. **uid_**U(u) $\equiv$ is_L(u)$\to$**uid_**L(u),is_H(u)$\to$**uid_**H(u)

[34] We shall only consider finite paths. The `paths` function, Item 84 below, can easily be modified to yield also infinite length paths !

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

B.12 Simple Function Values

We define a function that given a road net calculates all its non-circular paths.

84. The paths[35] function takes a road net – represented here by its "global" sets of hubs and links – and yields a possibly infinite set of paths – satisfying the wellformedness criterion of Sect. B.11 on the preceding page.

    We define the paths function in two ways.

85. Either axiomatically

86. in terms of an **as** predicate, with the result being the "largest" such set all of whose paths satisfy the wellformedness criterion;

87. or inductively[36]:

    (a) **basis clause:** every singleton path of either hub or link identifiers of the road net form a path.

    (b) **inductive clause:** If `pi` and `pj` are finite, respectively possibly infinite paths of the "result", `ps`, such that

        i. paths `pi`⌢⟨`ui`⟩ and ⟨uj⟩⌢pj are in `ps`, and
        ii. the resulting concatenated path is not circular, and
        iii. the mereology of the last element of `pi` identifies the first element of `pj`,
        iv. then their concatenation is a path in `ps`.

    (c) **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
84. paths: Unit → Path-infset
85. paths() as ps
86.     such that: ∀ p:ps satisfy the wellformedness of Sect. B.11 on the facing page
```

We can also express the `paths` functions explicitly:

```
value
84. paths: Unit → Path-infset
87. paths() ≡
87a.    let ps = {⟨ni⟩ | ni:NI ∈ hs_uis}∪{⟨ei⟩ | ei:EI ∈ ls_uis}
87(b)iv.        ∪ { pi⌢⟨ui⟩⌢⟨uj⟩⌢pj | pi⌢⟨ui⟩:Path-set, ⟨uj⟩⌢pj:Path-infset •
87b.              ∧ ({pi⌢⟨ui⟩,⟨uj⟩⌢pj}⊆ps)
87(b)i.             ∧ (ui~ ∈ elems pj ∧ uj~ ∈ elems pi)
87(b)iii.             ∧ (ui ∈ mereo_U(retr_unit(uj)))
```

[35] **Alarm !** Check that this function indeed generates only finite length paths !

[36] https://www.cs.odu.edu/ toida/nerzic/content/recursive_def/more_ex_rec_def.html

87(b)ii. ∧ (uj ∈ **mereo_**U(retr_unit(ui)))} **in**
87c. ps **end**
type
84. U = H|L

Solution to the equation, lines 87a–87(b)i, is "'obtained' by a smallest set fix-point reasoning.

88. Given a "global" road net, g, we can calculate a "similarly global" $paths$ value:

value
88. $paths$:Path-**set** = paths(g)

With the notion of paths of a road net one can now examine whether

- a road net is strongly connected, that is, whether any hub or link can be "reached" from any other hub or link; or
- a road net consists of two or more sub-graphs, i.e., there are no links between hubs in two such sub-graphs;
- etc.

89. We can formulate a *theorem:* for every road net we have that every path, p, in g, also contains its reverse path, `rev(p)` in g.

theorem: [All finite paths have finite reverse paths]
89. ∀ g:G,p:Path•p ∈ paths(g) ⇒ rev_path(p) ∈ paths(g)
value
89. rev_path: P → P
89. rev_path(p) ≡
89. **case** p **of**
89. ⟨⟩ → ⟨⟩,
89. ⟨ui⟩ → ⟨ui⟩,
89. ⟨ui⟩⌢p′⌢⟨uj⟩ → ⟨uj⟩⌢rev_path(p′)⌢⟨ui⟩
89. **end**

We can define further functions. For example:

90. `path_length`,
91. `shortest_path`, etc.

value
90. path_length: P → LEN
90. path_length(p) ≡
90. **case** p **of**
90. ⟨⟩ → 0
90. ⟨ui⟩ → retr_path_length(ui),
90. ⟨ui⟩⌢p′ → retr_length(ui)+path_path_length(p′)
90. **end**

90. retr_path_length: UI → LEN
90. retr_path_length(ui) ≡ (is_EI(ui)→**attr_**LEN(retr_link(ui)),is_NI(ui)→0)

91. shortest_path: G → P-**set**
91. shortest_path(g) ≡
91. **let** ps = paths(g) **in**
91. { p | p:P • retr_len(p) ∧ ∀ p′:P•p′∈ps ∧ retr_path_len(p)≤retr_path_len(p′) }
91. **end**

B.13 Channel

92. There is a set of channels between hubs, links and automobiles.

93. These channels communicate messages, `M`. `M` will "transpire" frm the behaviour definitions.

channel
92. { ch[{ui,uj}] | {ui,ij}:(HI|LI|AI)-**set** • ui≠uj∧{ui,uj}⊆σ_{uids} } M
type
92. M .

B.14 Variables

B.15 Behaviours

There are three behaviours:

94. `automobile`, corresponding to endurants `a:A`, and with appropriate argument types,

95. `hub`, corresponding to endurants `h:H`, and with appropriate argument types, and

96. `link`, corresponding to endurants `l:L`, and with appropriate argument types.

value
94. automobile: AI → AM → ... → (Apos × AHist) → **Unit**
95. hub: HI → HM → (HΩ × ...) → (HΣ × HHist) → **Unit**
96. link: LI → LM → (LEN × LΩ × ...) → (LΣ × LHist) → **Unit**

97. The **automobile** behaviour is either *at a hub* or *on a link* – and **communicates** with the *hub* and *link* behaviours as to its entering, leaving, or remaining at the hub, respectively on the link.

98. Any **hub** behaviour is "passively" awaiting **communication** from automobile behaviours as to their entering, leaving, or remaining at the hub.

99. Any **link** behaviour is "passively" awaiting **communication** from automobile behaviours as to their entering, leaving, or remaining on the link.

97. automobile(ai)(ris)(...)(atH(hi),ahis)
97. automobile(ai)(ris)(...)(onL(li,(fhi,f,thi)),ahis)
98. hub(hi)(mh)(hω,...)(hσ,hhist)
99. link(li)(ml)(len,lω,...)(lσ,lhist)

100. We abstract automobile behaviour at a Hub (hi).
    (a) Either the automobile remains at the hub,
    (b) or, internally non-deterministically,
    (c) leaves the hub entering a link,
    (d) or, internally non-deterministically,
    (e) stops.

```
100  automobile(ai)(ris)(...)(atH(hi),ahis) ≡
100a      automobile_remain_at_hub(ai)(ris)(...)(atH(hi),ahis)
100b    ⌈⌉
100c      automobile_leaving_hub(ai)(ris)(...)(atH(hi),ahis)
100d    ⌈⌉
100e      automobile_stop(ai)(ris)(...)(atH(hi),ahis)
```

where we leave it to the reader to fill in the signature of these three behaviours.

101. [100a] The automobile **remains** at a hub:
    (a) time is recorded,
    (b) informing the hub behaviour, whereupon
    (c) the automobile remains at that hub, "idling",

```
101  automobile_remain_at_hub(ai)(ris)(...)(atH(hi),ahis) ≡
101a    let τ = record_𝕋𝕀𝕄𝔼() in
101b    ch[ {ai,hi} ] ! τ ;
101c    automobile(ai)(ris)(...)(atH(hi),⟨(τ,hi)⟩^ahis) end
```

102. [100c] 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.

```
102  automobile_leaving_b(ai)({li}∪ris)(...)(atH(hi),ahis) ≡
102a    let τ = record_𝕋𝕀𝕄𝔼() in
102b    (ch[ {ai,hi} ] ! τ ‖ ch[ {ai,li} ] ! τ) ;
102c    automobile(ai)(ris)(...)(onL(li,(hi,0,_)),⟨(τ,li)⟩^ahis) end
102     pre: [hub is not isolated]
```

103. [100e] Or the automobile **stop**s, "disappears — off the radar" !

```
103 automobile_stop(ai)(ris),(...)(atH(hi),ahis) ≡ stop .
```

104. We abstract automobile behaviour on a Link li:
    (a) Either (*internally non-deterministically*) the automobile **remains** on the link, advancing a fraction or halts [temporarily],
    (b) or, *internally non-deterministically*,
    (c) **leaves** the link [when reaching its end] and enters the connected hub,
    (d) or, *internally non-deterministically*,
    (e) **stops** [leaves the road net altogether (!)].

```
104  automobile(ai)(ris)(...)(onL(li(fhi,f,thi)),lhis) ≡
104a     automobile_remains_on_link(ai)(ris)(...)(onL(li(fhi,f,thi)),lhis)
104b   ⌈⌉
104c     automobile_leaving_linl(ai)(ris)(...)(onL(li(fhi,f,thi)),lhis)
104d   ⌈⌉
104e     automobile_stops_on_link(ai)(ris)(...)(onL(li(fhi,f,thi)),lhis)
```

We leave it to the reader to complete the definitions of `automobile_remains_on_link, automobile_leaving_link` and `automobile_stops_on_link`.

105. Hubs

106. external non-deterministically receives time-stamped, t, messages,ai, from automobiles as to their entering, remaining or leaving the hub.

107. They update their hub history accordingly and resume being a hub.

```
value
105. hub(hi)(hm)(hω,...)(hσ,hhist) ≡
106.     let (t,ai) = [] { ch[ {hi,ai} ]? | ai ∈ hm } in
107.     hub(hi)(hm)(hω,...)(hσ,⟨(t,ai)⟩^hhist) end
```

108. Links

109. external non-deterministically receives time-stamped, t, messages,ai, from automobiles as to their entering, remaining or leaving the link.

110. They update their link history accordingly and resume being a link.

```
value
108. link(li)(lm)(len,lω,...)(lσ,lhist) ≡
109.     let (t,ai) = [] { ch[ {li,ai} ]? | ai ∈ lm } in
110.     link(li)(lm)(len,lω,...)(lσ,⟨(t,ai)⟩^lhist) end
```

B.16 Initialization

111. Let us refer to the system initialization as parallel composition of behaviours:
    (a) All hubs are initialized,
    (b) and
    (c) all links are initialized,
    (d) and
    (e) all automobiles are initialized.

```
value
111.  initialisation: Unit → Unit
111.  initialisation() ≡
111a.     ‖ { hub(uid_H(h))
111a.               (mereo_H(h))
111a.                  (attr_HΩ(h),...)
111a.                     (attr_HΣ(h),attr_HΣ(h),attr_HHist(h))
111a.       | h:H • h ∈ hs }
111b.     ‖
111c.     ‖ { link(uid_L(l))
111c.               (mereo_L(l))
111c.                  (attr_LEN(l),...)
111c.                     (attr_LΣ(l),attr_LHist(l))
111c.       | l:L • l ∈ ls }
111d.     ‖
111e.     ‖ { automobile(uid_A(a))
111e.               (mereo_A(a))
111e.                  (attr_APos(a),attr_AHist(a))
111e.       | a:A • a ∈ as }
```

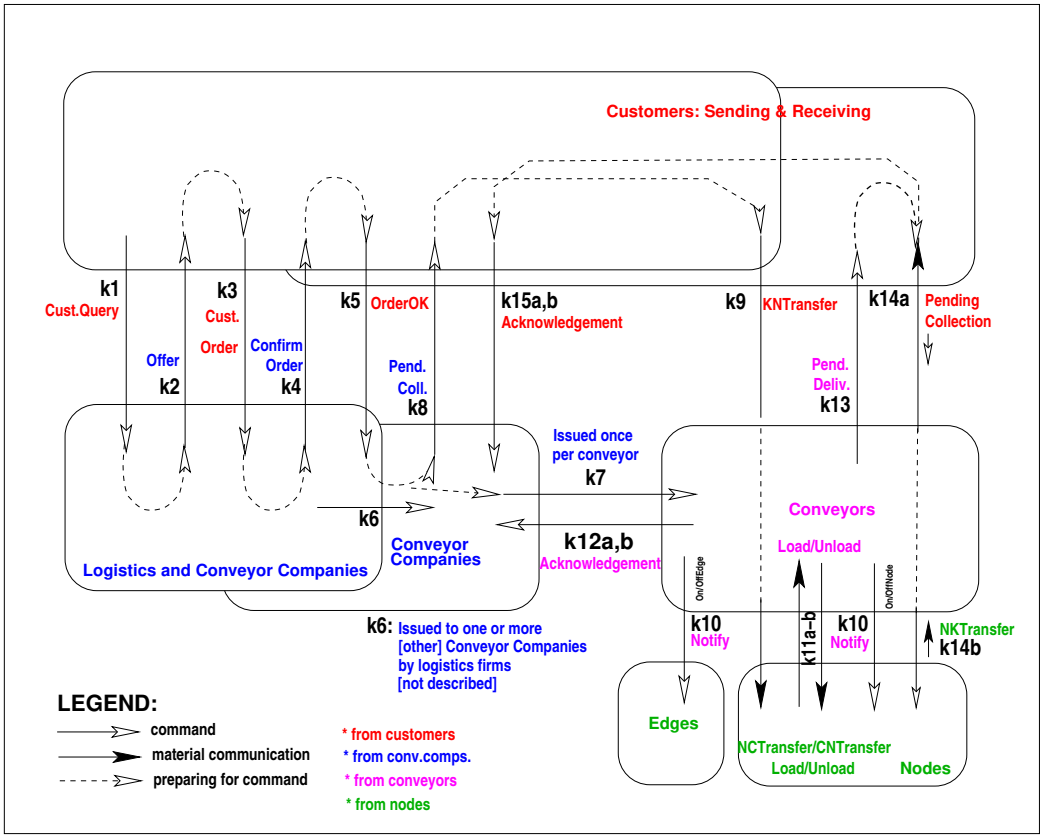

Figure 3: A Road Transport Behavioural Taxonomy

C Example Behavioural Taxonomies

C.1 A Multi-mode Transport Behavioural Taxonomy

See Fig. 3.

The various rounded boxes designate 1, 2, or more instances. That is: there are many customers, many logistics and conveyor companies, many [truck, ship, aircraft] conveyors, many hubs [road side loading/unloading places, harbours, airports], and many links [streets, sea lanes, airlanes].

C.2 – more to come –

TO BE WRITTEN

C.3 – more to come –

TO BE WRITTEN

D Behaviour Patterns

D.1 Behaviour Definitions

A typical, informal rendition of abstracted behaviours, BA, BC, BD, ... is shown in Fig. D.1.

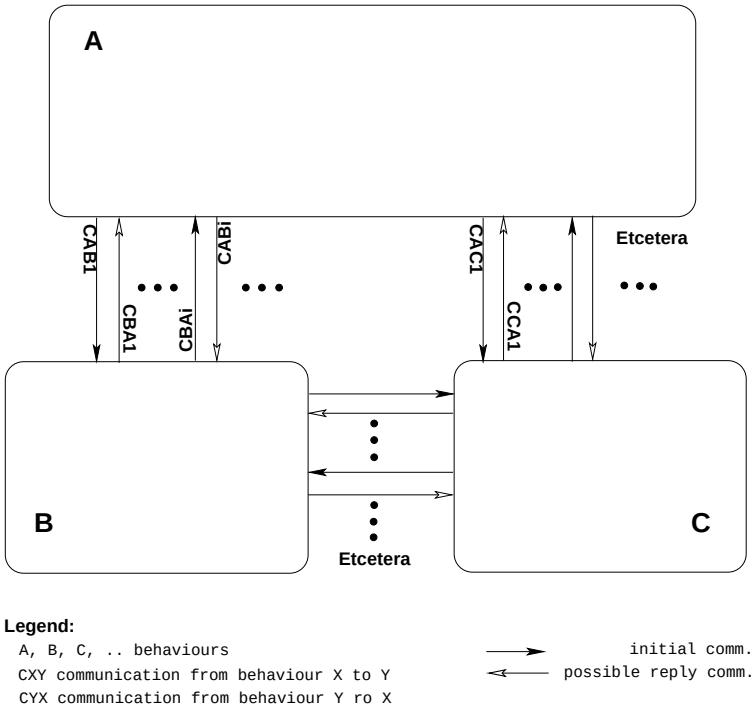

Figure 4: Communicating Behaviours

Figure D.1 should be understood as follows:[37] The **bold faced** labels **A, B, C, ...** are meant to designate behaviours. The **black arrows**, from behaviour **X** to behaviour **Y** are meant to designate CSP-like *communications* from **X** to **Y**. The **open arrows** ("white"), from behaviour **X** to behaviour **Y** are meant to designate possible CSP-like *communications* from **X** to **Y**. 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 **X** to **Y**.

Figure D.1 is now given a more precise "meaning" – with this "meaning" suggesting a general "pattern" for behaviour definitions:

112. There are behaviours B, ... with identities bi, ...
    (a) These behaviours,typically, have the form of internal, $\sqcap$, non-deterministically "choosing" between
    (b) *pro-actively* initiating communications with other behaviors
    (c) and *re-actively* responding to such initiatives.

[37] The explanation of Fig. D.1 is in now way an attempt to explain the semantics of behaviours. That is left to the RSL$^+$ formalization's.

```
value
112a.  B(bi)(mereo)(stat)(mon)(prg) ≡
112b.      pro_active_B(bai)(mereo)(stat)(mon)(prg)
112c.  ⌈⌉ re_active_B(bai)(mereo)(stat)(mon)(prg)
```

113. The pro-active behaviour (B) internal deterministically (⌈⌉) choosing between a number of initiating actions (B_action_i):
   (a) action 1,
   (b) action 2,
   (c) ...,
   (d) action n.

114. where B_action_i is typically of the form shown next.

```
value
113.  pro_active_B(bi)(mereo)(stat)(mon)(prg) ≡
113a.      B_action_1(bi)(mereo)(stat)(mon)(prg)
113b.   ⌈⌉ B_action_2(bi)(mereo)(stat)(mon)(prg)
113c.   ⌈⌉ ...
113d.   ⌈⌉ B_action_m(bi)(mereo)(stat)(mon)(prg)
where:
114.  B_action_i(bi)(mereo)(stat)(mon)(prg) ≡
114.    let prgr′ = Action(bi)(mereo)(stat)(mon)(prg) in
114.    B(bi)(mereo)(stat)(mon)(prg′) end
```

115. The `reactive` behaviour reacts to a number of such initiating actions by
   (a) external non-deterministically ([]) offering to accept messages from responding behaviours,
   (b) and then performing corresponding actions.

```
value
115.  re_active_B(bi)(mereo)(stat)(mon)(prg) ≡
115a.    let msg = [] { comm[{bj,bi}]? | bj:BI • bj ∈ bis } in
115b.    respond_behaviour_B(bi)(mereo)(stat)(mon)(prg)(msg) end
```

116. The `responding _behaviour_B` inquires as to the type of the message, say, a command, received (?): if it is:
   (a) of type `action_i` then it performs action `B_action_i`,
   (b) of type `action_j` then it performs action `B_action_j`,
   (c) ..., or
   (d) of type `action_k` then it performs action `B_action_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
116. react_behaviour_B(bi)(mereo)(stat)(mon)(prg)(msg) ≡
116a.     is_action_i(msg) → B_action_i(bi)(mereo)(stat)(mon)(prg)(msg),
116b.     is_action_j(msg) → B_action_j(bi)(mereo)(stat)(mon)(prg)(msg),
116c.     ...,
116d.     is_action_k(msg) → B_action_k(bi)(mereo)(stat)(mon)(prg)(msg),
116e.     _ → B(bi)(mereo)(stat)(mon)(prg)
```

Et cetera !

D.2 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 **value**s. These values may be Booleans, numbers, sets, Cartesians, lists, maps and functions (over these), or the values by 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.

117. 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.

```
117.  action_i(bi)(mereo)(stat)(mon)(prg) ≡
117a.     let v = evaluate_i(bi)(mereo)(stat)(mon)(prg) in
117b.     let (bj,prg′) = elaborate_i(v)(bi)(mereo)(stat)(mon)(prg) in
117c.     comm[{bi,bj}] ! ℰ(prg′) ;
117d.     behaviour(bi)(mereo)(stat)(mon)(prg′)
117.      end end
```

Variants of Item ι117c π54 are also used:

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

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

• • •

I refer to a recent paper [39].