The DOMAIN Method
Analysis & Description Languages

Dines Bjørner
Fredsvej 11, DK-2840 Holte, Danmark
E–Mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

August 4, 2026: 22:57

Abstract

The DOMAIN Method entails two kinds of languages: the domain analysis and description prompt language, DAL, and the domain description language language, DDL. In this document we shall define the syntax of both – while informally & briefly explaining their purpose.

Contents

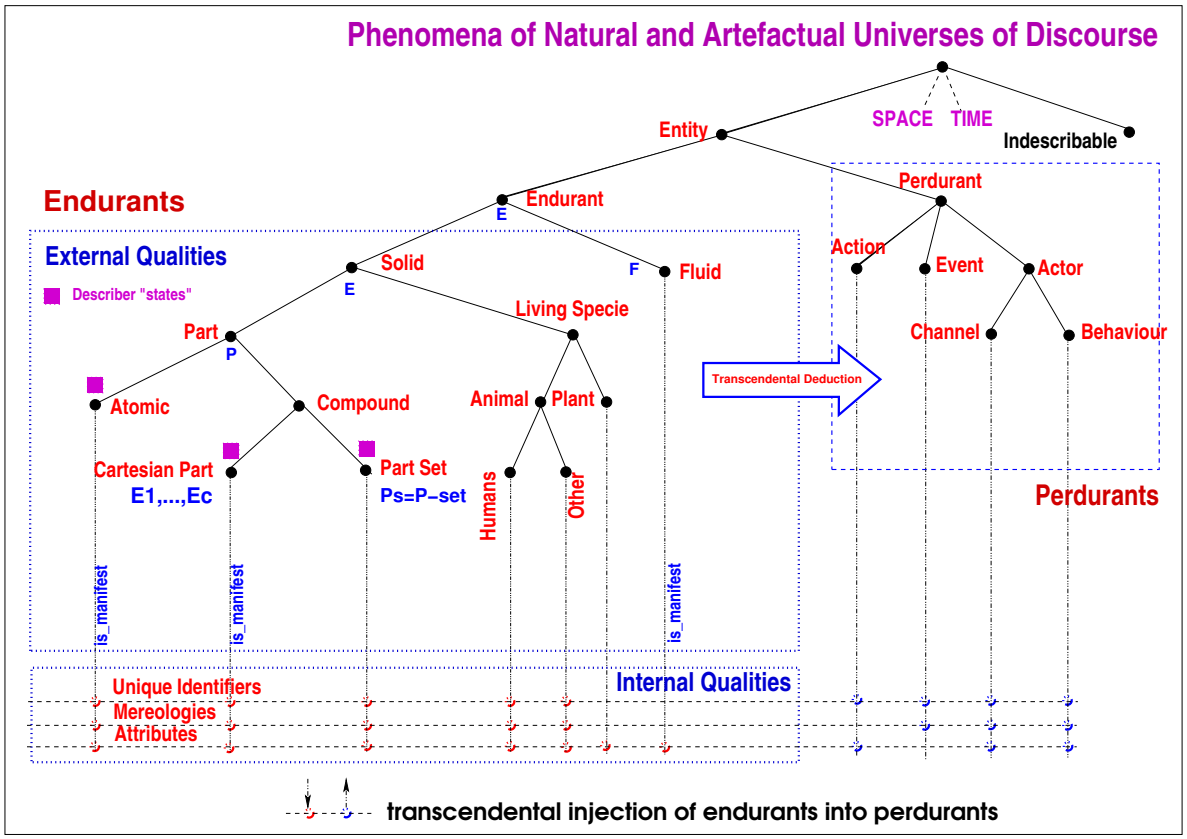

Figure 1: The DOMAIN Analysis & Description Ontology

Preamble

The DOMAIN Method "spring" from the following dogma:

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*.

An interpretation of the triptych dogma is interpreted in Fig. 1.

The present document is not intended for neither conference not journal publication. It is being / has been worked out by me, the author, as a means [i.e., the "working out"] to sort out / come to grips / better understand the very many issues that are hidden in the DOMAIN Method development process, [Bjø26g]. A final version is then intended to be posted on the Internet for readers

delectation ! More seriously, for consultation and, perhaps, or hopefully, for someone to implement [Bjø26g].

It is not a first attempt to describe syntaxes related to the 𝔻𝕆𝕄𝔸𝕀ℕ `Method`. Previous attempts were [Bjø26c, Bjø26b, Bjø26d, Bjø26e]. The present document is an extensively edited/revised/extended version of [Bjø26b].

In working out this document I have "amused" myself by going beyond a mere specification of the `DAL` and `DDL` syntaxes. I have, for most syntactic clauses related them to the Domain Analysis & Description Ontology of Fig. 1 on the preceding page, thereby also, sort of implicitly, related them, i.e.., the clauses, to the 𝔻𝕆𝕄𝔸𝕀ℕ `Method` [Bjø26g].

1 Introduction

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].

Over the years 2009 till present we have researched, experimentally applied [Bjø] and formulated [Bjø10a, Bjø08, Bjø10b, Bjø11a, Bjø09, Bjø11b, Bjø17, Bjø19c, Bjø20, Bjø, Bjø27, Bjø25b, Bjø25c, Bjø25d, Bjø25a, Bjø26a, Bjø26e, Bjø26c, Bjø26b, Bjø26d, Bjø16, Bjø26f, Bjø26g, Bjø26h] a rigorous method, 𝔻𝕆𝕄𝔸𝕀ℕ, for analysing & describing domains.

A major tool of this method is a pair of languages: `DAL` for analysis, `DDL` for description. Together we refer to them as `DADL`. In this document we shall define a syntax for both.

2 A Syntax of `DAL`

The domain analysis language consists of a set of prompts.[1]

[1] Prompts are here seen as

- *výzva, tágo* [Czech],
- *stikord* [Danish],
- *cues* [US English],
- *signals* [French],
- *stichwörter* [German],
- *richiesta, spunto* [Italian],
- *podpowiedź* or *replika* [Polish],
- *incitar, deixa* [Portuguese],
- 'signal' [Russian],
- *inmediato, señal* [Spanish].

Our use of the term 'prompt' pre-dates the current term *prompt engineering:* `the art and science of designing, testing, and refining text instructions to guide artificial intelligence models like large language models (LLMs) toward accurate, relevant, and safe outputs. Key components include clear instructions, proper context, and defining output formats` – but, as You can read: there are many similarities: whee the domain analyser cum describers use 𝔻𝕆𝕄𝔸𝕀ℕ analysis prompts when investigating "real", actual man-made domains, prompt-engineering is to be used in connection with the use of AIs LLMs.

See also `[https://www.promptingguide.ai/]`

There are three kinds of domain analysis & description prompts:

- **analysis predicates**,
  yield truth values
- **analysis functions**,
  yield other than truth values
- **description functions**,
  yield domain description units.

They are all "performed", i.e., "issued" and their outcome constructed/determined by the domain analyser cum describers. No computers are involved in this. Bit, perhaps, in some near future, some clever `generative AI`[2] may very well be involved ! We refer to Sect. 2.4 on page 9.

2.1 Analysis Predicate Prompts

Analysis predicate prompts yield either of three values: **true, false** or **chaos**. These value results are for the domain analyser cum describers' use. If **true** one **course direction of analysis** is chosen in he course of analysis. If **false** another. If **chaos** it effectively means that the domain analyser cum describer has decided to abandon, to give up, the [goal of] describing the chosen domain !

The concept of **course direction of analysis** is embodied in Fig. 1 on page 3. Another form of Fig. 1 on page 3 is shown in Fig. 2 on the following page.

Various nodes are now labeled with what we shall refer to as "states": [-1,-,1,2,...,8]. (We omit treatment of *living species*. *Perdurants* are not part of domain analysis. Their treatment is, as result of the *transcendental deduction* of manifest, behaviourable parts into behaviours, **synthetic**.)

The analysis predicates are:

- `is_describable` [-1]
- `is_entity` [0]
- `is_endurant` [1]
- `is_perdurant` [...]
- `is_solid` [2]
- `is_part` [3]
- `is_living ...` [...]
- `is_fluid` [4]
- `is_atomic` [5]
- `is_compound` [6]
- `is_Cartesian` [7]
- `is_part_set` [8]

plus the additional "silent operators"[3]:

- `is_manifest`
- `is_behaviourable`
- `is_structure`
- `is_mobile`
- `is_static`

Here we have indicated [...] "states" associated with the domain analysis direction. The domain analyser cum describer starts in state [-1]. Proceeds either to state [0] or abandons the domain analysis & description "undertaking" altogether !

The further analysis process can be informally, yet reasonably rigorously described by the following "pseudo-program".

Please **note** the use of "pseudo-program" comments shown as **[bracketed text]**.

We have "affixed" **state numbers** to the "pseudo program: .

value

1. uod:Uod,

[2] https://en.wikipedia.org/wiki/Generative_AI

[3] By "silent operators" we shall, informally, mean that the outcome of their application is recorded, but has no effect on the course of direction of the analysis !

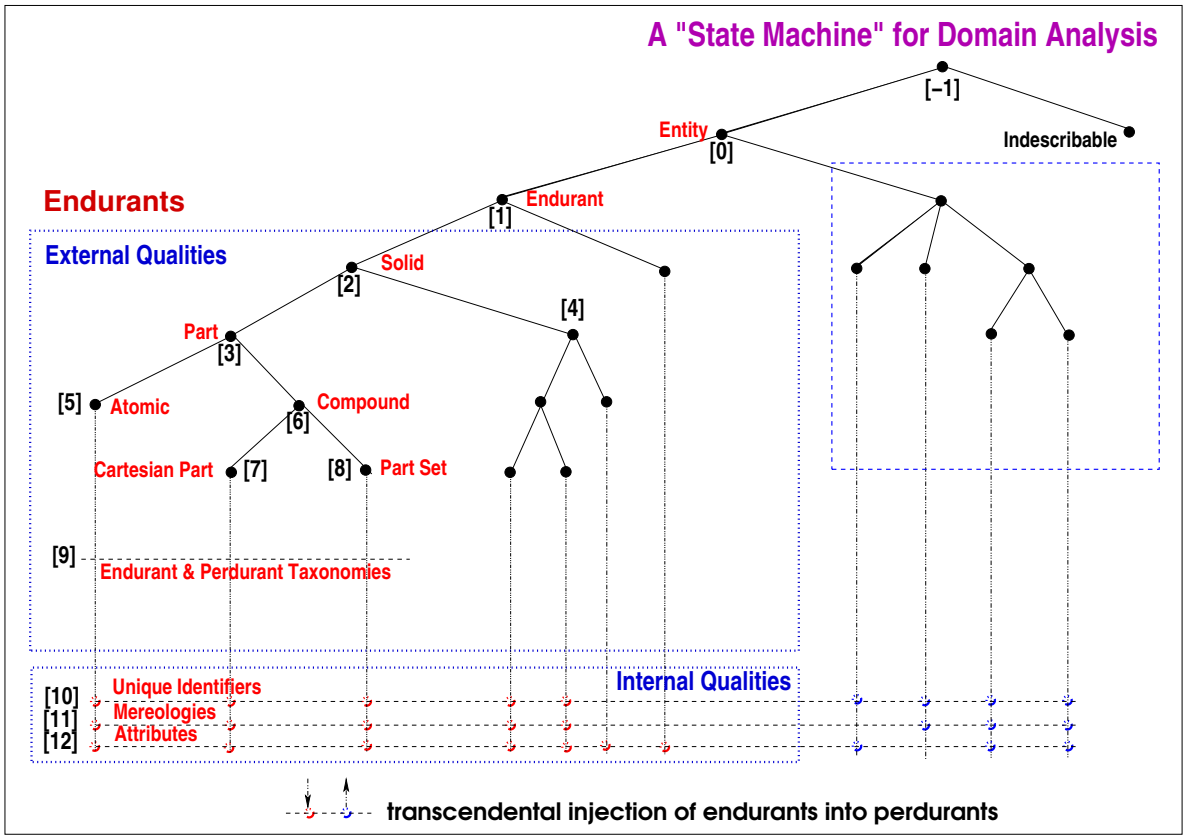

Figure 2: A DOMAIN Analysis "State Machine"

```
variable
2.  tbxa:ℰ-set := {uod};
variable
3.  tbia:ℰ-set := {}
value
4.  [ −1 ]  if ∼ is_describable(uod) then chaos else analysis() end
4.  [ 0 ]   [the domain analyser cum describers proceed on the]
4.  [ 0 ]   [assumption that  it is an endurant  being analysed ! ]
value
4.  analyse: Unit → RSL⁺text→ Unit
4.  analyse() ≡
5.    while tbxa ≠ {} do let e • e ∈ tbxa in  tbxa:= tbxa \ {e} ;
6.  [ 2 ]   if ∼ is_solid(e) then ... [is_fluid(e)] else [[3] is_part(e)]
7.  [ 5 ]   if is_atomic(e) then tbia := {tbia} ∪ {e} else [[6] is_compound(e)]
8.  [ 7 ]   if is_Cartesian(e) then describe_Cartesian(e)
9.  [ 8 ]   else describe_Part_Set(e)
4.   end end end end end
```

The type $\mathcal{E}$ is the type union of all "to be discovered" endurant types.

The **describe** functions are given below, see Sect. 2.3 on page 9.

The **analyse** function proceeds on the assumption that the domain analyser cum describers are analysing (and **describing**) endurants.

1. The domain analysis is of the `uod` universe of discourse – "set aside" as a global **value**.
2. A global **variable**, `tbxa`, holds the set of endurants `to be analysed` for e**x**ernal qualities. Initially `uod` is its only member. Once a member of this "`to be analysed`" set has been / is being analysed it is removed from the set.
3. As an endurant is being **analyse**d into its parts these are added to the set `tbxa` – these "additions" will be shown in the like-wise "pseud0-program" specifications of the **describe** "functions".
4. If the domain is indescribable then the analysis & description endeavor is abandoned !
5. For as long as there are endurants to be analysed & described an element to be so is selected (and removed from those to be analysed). We are in analysis etc. "state" [2].

   We refer to Fig. 2 on the facing page.
6. [2] If the endurant being examined is a fluid [3]then we skip that – we do not handle fluids in this document.

   then the examined endurant is either atomic or compound:
7. [5] if atomic then, if manifest, it is joined to the set of parts to be analysed and described for internal qualities.
8. [7] A compound is either Cartesian or a part set. If Cartesian it is to be **described** (see below) [itself have been removed from `tbxa`] with the **describe** function adding its the observed parts of the Cartesian to `tbxa`.

9. [8] If it is not a Cartesian then it is a part set and **described** as such, with examined and revealed parts removed from, respectively joined to `tbxa`.

   The next items do not relate to the present topic of analysis prompts:

10. When there are no more external qualities of endurants to be analysed and described the analysis proceeds to states [9]–[12]:

    - the "creation" of a description of an `endurant taxonomy` []

      and analysis and description of internal qualities of endurants:
    - [9] *unique identification*
    - [10] *perdurant taxonomy* and *mereology*
    - [11] *attributes* and possible/possibly
    - [12] *intentional pulls*.

2.2 Analysis Functions

𝔻𝕆𝕄𝔸𝕀ℕ `Method` analysis functions are applied as the result of prompts and yield other than]sortBoolean values. These values are noted by the domain analyser cum describers – say by being written down on a *domain analysis and description "dashboard"*!

11. We represent this dashboard as a global **variable**, the `dashboard`.

11. dashboard:... := {}

These are the 𝔻𝕆𝕄𝔸𝕀ℕ `Method` analysis functions:

12. `discover_Cartesian_part_sort_names`
13. `discover_Part_Set_sort_name`
14. `discover_Mereology_sort_name`
15. `discover_Attribute_type_mame[s]`
16. `discover_Intentional_Pull_name`

We shall now detail these:

12. `discover_Cartesian_part_sort_names:`
13. `discover_Part_Set_sort_name:`
14. `discover_Mereology_sort_name:`
15. `discover_Attribute_type_mame[s]:`
16. `discover_Intentional_Pull_name:`

2.3 Description functions

𝔻𝕆𝕄𝔸𝕀ℕ `Method` description functions are applied as the result of prompts and yield `domain description units`.

These domain description units are joined to a *domain description unit record:* `dsu-record`.

17. We represent this record as a global **variable**, the `dsu_record`.

11. dsu_record:DSU-**set** := {}

`DSU`s will be described in Sect. 3.

These are the 𝔻𝕆𝕄𝔸𝕀ℕ `Method` description functions:

18. `describe_Cartesian`
19. `describe_PartSets`
20. `describe_endurant_taxonomy`
21. `describe_uniqueness`
22. `describe_perdurant_taxonomy`
23. `describe_mereology`
24. `describe_attributes`
25. `describe_behaviours`
26. `describe_initialisation`

We shall now detail these:

18. `describe_Cartesian:`
19. `describe_PartSets:`
20. `describe_endurant_taxonomy:`
21. `describe_uniqueness:`
22. `describe_perdurant_taxonomy:`
23. `describe_mereology:`
24. `describe_attributes:`
25. `describe_behaviours:`
26. `describe_initialisation:`

2.4 Prompt Engineering

We have earlier, footnote 2 on page 5, indirectly referred to the concept of *prompt engineering*.

Prompt Engineering *"is the art and science of designing, testing, and refining text instructions to guide artificial intelligence models like large Language Models (LLMs) toward accurate, relevant, and safe outputs. Key components include clear instructions, proper context, and defining output formats"* `[https://cloud.google.com/discover/what-is-prompt-engineering]`.

We may very well consider this section, i.e., Sect. 2 (pages 4–10), a *"Prompt Engineering for the 𝔻𝕆𝕄𝔸𝕀ℕ `Method`"*:

- Not 'the', but just 'a'.
- Not for a large language machine, LLM.

- But for the analysis and description of actual, real domains.
- As such we can imagine the following kind of `Generative AI` LLM: **Domain_LLM** [4]
  - **Domain_LLM** embodies all the prompts mentioned in this section, i.e., pages 4–9.
  - **Domain_LLM** is then "trained" with the training data: examples, say taken from domain descriptions [Bjø].
  - Users of the **Domain_LLM** then "feed" **Domain_LLM** with what amounts to more-or-less perfect `Narratives`
  - and then expected to express their `Formalisation` answers in `RSL/CSP` and in the style of the `Domain Specification Units` to be covered in the [immediately] next section !

3 A Syntax of DDL

3.1 DOMAIN Specifications

27. A domain specification is a set of **domain specification unit**s, `dsu`.

    27. DS ::= DSU-set

Domain specification units, although abstracted as a set, is presented in some linear order – where it is clear, from the below syntax for these units, where it begins and where it ends. So we redefine DS:

28. A domain specification thus contains
    (a) exactly one universe of discourse `dsu`; UOD_DSU
    (b) one endurant taxonomy `dsu`, E_TAX_DSU
    (c) two or more endurant part `dsus`, EP_DSU
    (d) one endurant part global value DSU EPV_DSU, EPS_DSU
        i. for one or more endurant `dsus`,
        ii. and one "state" `dsu`,
    (e) one or more unique identification `dsus`, UID_DSU
        i. a unique identifier values `dsu`, UIDV_DSU

[4] **Generative AI** [`https://en.wikipedia.org/wiki/Generative_AI`]: (GenAI) is a sub-field of AI that uses generative models to generate text, images, videos, audio, software code or other forms of data. These models learn the underlying patterns and structures of their training data, and use them to generate new data in response to input, which often takes the form of natural language prompts.

Generative Model: Generative models are a class of computational models frequently used for classification. In machine learning, it typically models the joint distribution of inputs and outputs, such as $P(X, Y)$, or it models how inputs are distributed within each class, such as $P(X|Y)$ together with a class prior $P(Y)$. Because it describes a full data-generating process, a generative model can be used to draw new samples that resemble the observed data, a process often referred to as synthetic data generation. Generative models are used for density estimation, simulation, and learning with missing or partially labeled data. In classification, they can predict labels by combining $P(X|Y)$ and $P(Y)$ and applying Bayes' rule. Generative models are often contrasted with discriminative models, which focus on predicting outputs from inputs directly. Generative model approaches which uses a joint probability distribution instead, include naive Bayes classifiers, Gaussian mixture models, variational autoencoders, generative adversarial networks and others.

	ii. a uniqueness axiom `dsu`,	UNIQ_DSU
(f)	one perdurant taxonomy `dsu`,	P_TAX_DSU
(g)	one or more mereology `dsus`,	MER_DSU
(h)	one or more attribute `dsus`,	ATTR_DSU
(i)	zero or more intentional pull `dsu`,	IPULL_DSU
(j)	zero or more variable declaration `dsus`,	VAR_DSU
(k)	exactly one channel declaration `dsu`,	CH_DSU
(l)	one or more [informal, yet rigorous] action `dsus`,	ACT_DSU
(m)	two or more behaviour signature & definition `dsus`,	BEH_DSU
(n)	and one domain initialization `dsu`,	INIT_DSU

The use of $^+$ in DSU$^+$ does not mean *Kleene closure,* but is just part of the identifier name DSU$^+$.

27.	DS	::=	DSU$^+$	**Domain Specification**
28a.	DSU$^+$	::=	UOD_DSU	**Collection of DSUs**
28c.		×	EP_DSU-**set**$_2$	**Endurants**
28b.		×	E_TAX_DSU	**Edurant Taxonomy**
28(d)i.		×	EPV_DSU-**set**$_1$	**Global Part Values**
28(d)ii.		×	EPS_DSU-**set**$_1$	**State**
28e.		×	UID_DSU-**set**$_1$	**Unique Identification**
28(e)i.		×	UIDV_DSU-**set**$_1$	**Global UID Values**
28(e)ii.		×	UNIQ_DSU	**Part Uniqueness**
28f.		×	P_TAX_DSU	**Perdurant Taxonomy**
28g.		×	MER_DSU-**set**$_2$	**Mereology**
28h.		×	ATTR_DSU-**set**$_2$	**Attributes**
28i.		×	IPULL_DSU-**set**$_0$	**Intentional Pulls**
28j.		×	VAR_DSU-**set**$_0$	**Possible Global Variables**
28k.		×	CH_DSU	**Channel**
28l.		×	ACT_DSU	**Actions**
28m.		×	BEH_DSU-**set**$_2$	**Behaviours**
28n.		×	INIT_DSU	**Initialisation**

Concrete syntax-speaking, You may think of the elements of the `dsus` being separated by some syntactic marker, typically the display lines[5] mentioned [first] in the syntaxes given below – or as noted to the right ["fifth column"] in the above table.

In those syntaxes we may show [right hand side of ::=] elements of a syntactical [left hand side] category on several lines [separated by × symbols]. These 'several' elements could all be listed on same line as the '::='. Listing them on these separate lines is an informal suggestion that they may concretely, in actual `RSL` texts be shown on similarly separate lines.

[5] These display lines may be in the form of section, subsection, sub-subsection, paragraph and subparagraph display lines.

3.2 Some Remarks

3.2.1 On an Order of Analysis & Description

We remind the reader of how domain analysis & description ('examination') proceeds:

In those steps, of analysing a domain, where the analyser is analysing a compound endurant, E, that is either a Cartesian part or a part set parts, the analyser "discovers" a set of parts, p1, p2, ..., pn, and [decides] on their types and observers: E1, E2, ..., En, respectively PS and P.

In those steps the analyser "sets aside", for the case of analysing a Cartesian, p1:E1, p2:E2, ..., pn:En, and the case of part set parts, one of p1, p2, ..., pn, i.e., pi:P.

This "setting aside" is to a nonuplet[6] analysis & description state[7]: $\xi : \Xi$:

(i) $\xi_{USN} : \Xi_{USN}$: the set of used sort names;

(ii) $\xi_{BSN} : \Xi_{BSN}$: the set of sort names selected for internal quality analysis & description and for their transcendental deduction into behaviours;

(iii) $\xi_E : \Xi_E$: endurants (E1, E2, ..., En or P) to be further analysed and described;

(iv) $\xi_D : \Xi_D$: the domain description text "generated" so far, a list in the order described[8];

(v) $\xi_{Es} : \Xi_{Es}$: a most recent last list[9] of either "**mk_**Uod(uod)", "**mk_**C(E1,E2,...,En)" or "**mk_**S(PS,P)";

(vi) $\xi_{Val} : \Xi_{Val}$: a most recent last list of RSL$^+$`text` value definitions.

(vii) $\xi : \Xi$

(viii) $\xi_{temporary} : \Xi_{Temporary}$: a sequence of RSL$^+$`texts`, and

(ix) $\xi_{ASN} : \Xi_{ASN}$: a set of unused, i.e., available sort names.[10]

Once a compound endurant has been described the analyser cum describer selects, from $\xi_E : \Xi_E$ an endurant to be examined next ! The order, really, is, semantically, not important but we suggest an order: those endurant discovered first are to be examined next ! Initially $\xi_E : \Xi_E$ contains the universe-of-discourse ! and $\xi_D : \Xi_D$ is "empty". It is also advised to "append" the "most recently generated description texts" to $\xi_D : \Xi_D$ "to the end" of $\xi_D : \Xi_D$.

We shall refer to $\xi : \Xi$ as the analysis & description state.[11]

3.2.2 No Recursively Defined Endurants

Based on both extensive domain modeling [Bjø] and on rational reasoning [Bjø20] we can state that domains do not exhibit inherently recursively definable endurants[12].

[6] nonuplet: group of nine !

[7] The elements of the nonuplet are a kind of "dashboards": maintained by the human analyser cum describer during the domain analysis & description process – if not on paper or other media, then at least in the minds of the domain analyser cum describers.

[8] — appending most recent descriptions "at the end"

[9] extracted from $\xi_D : \Xi_D$

[10] We shall not attempt to further define $\xi : \Xi$ nor to detail operations on its components. We refer to [Bjø20, *Sect. 4.20, Sect. 5.8, and Sect. 7.12*] for details.

[11] The above was written 10-13 February 2026. As of August 4, 2026: 22:57 it is a bit "unrehearsed": This whole text, pages 12–12, need by "cleaned up", better structured, etc.

[12] Oh no, You – and my dear colleagues – might object: what about trees ? Well, I model trees as graphs of a special, i.e., restricted, kind `https://www.imm.dtu.dk/~dibj/2021/Graphs/nets.pdf`.

3.2.3 On [Choice of] Endurant Names

The "problem" here is *"how do the domain analyser cum describer choose names – for endurants, unique identifiers, mereologies, attributes, values, variables, channels and behaviours"*.

There are three issues here: (i) first there is the "problem" of whether an analysis of an endurant part leads to the possibility of having to choose an already defined sort name; secondly. (ii) there is the "problem" of choosing a sort name that has not been used before in the analysis & description; and (iii) thirdly, there is the issue of "message" to be conveyed to the reader of the [emerging] domain description.

As for (i) we have this to say: If You are confronted with the having to choose a sort or an attribute type name that, to You, appears to have been chosen before, then consider the possibility that the two or more such possibilities may actually [have to] be distinct: their "origin": the endurant from which they derive are distinctly named, so, perhaps, the two or more instances of names for "sub-parts" or attributes, should reflect that.[13]

As for (ii) we introduced, just above, the "dashboard" states $\xi_{USN} : \Xi_{USN}$ and $\xi_{ASI} : \Xi_{ASI}$. They "hold" used, respectively unused sort names.

With respect to (iii) we suggest that You choose a mnemomic'ally[14] "pleasing" name, usually a terse abbreviation thereof.

3.3 Universe of Discourse

See Example A.1 on page 30.

29. A universe of discourse `dsu` contains
    - A **Universe of Discourse.** display line,
    - a **type** literal,
    - a distinct universe of discourse "endurant" sort name
    - a reasonably precise text,
    - and, it is suggested, an **end** of the universe of discourse `dsu` literal.

29. UOD_DSU ::= UoD × **Universe of Discourse.**
29. × Text
29. × **end**

The meaning of UOD_DSU is then the text to the right of the ::= marker. Once UOD_DSU has been "performed" $\xi_{Es} : \Xi_{Es}$ is initialized to $\langle$**mk_**UoD(si)$\rangle$, UoD is "added" to $\xi_{USN} : \Xi_{USN}$, and $\xi_{Val} : \Xi_{Val}$ is initialised to the RSL$^+$`text: ``value uod:UoD''`.

3.4 Endurants

We present the syntax of endurant `dsus` in two steps. First the syntax of external quality `dsus`: EP_DSU and EPV_DSU, then the syntax of internal quality `dsus`.

[13] A case in point, for example, is this: the endurant parts *links* (street segments of a road net) have attribute *lengths*, and the endurant parts *automobiles* may be assigned *length* attributes. We would suggest to name these two lengths distinctly: LinkLength, respectively AutomobileLength.

[14] Mnemonic: a device such as a pattern of letters, ideas, or associations that assists in remembering something.

3.4.1 External qualities

3.4.1.1 EP_DSU: Endurant Parts. See Example A.2 on page 30.

30. An endurant part `dsu` observer is
    (a) either a Cartesian endurant part `dsu`
    (b) or a part set part `dsu`.

```
30.   EP_DSU  ::=
30a.              C_EP_DSU
30b.            | PS_EP_DSU
```

3.4.1.1.1 C_EP_DSU: A Cartesian Observer.

31. A Cartesian endurant part `dsu` contains
    (a) a endurant sort name, S , prefixing a display line: **Cartesian Part.**
    (b) a **type** literal,
    (c) a set of observed endurant sort names: E1, E2, ..., En– where E1, E2, ..., En are chosen from $\xi_{ASN} : \Xi_{ASN}$
    (d) a **value** literal,
    (e) and for each sort name, Ei, the signature of the corresponding **obs_**Ei observer function.

```
31.   C_EP_DSU  ::=  E × Cartesian Part.
31b.             ×  type
31c.             ×      E1, E2, ..., En
31d.             ×  value
31e.             ×      obs_E1:E→E1,obs_E2:E→E2,...,obs_En:E→En
```

The meaning of C_EP_DSU is then the text to the right of the ::= marker. Once C_EP_DSU has been "performed" the right-hand side of the ::= is appended to the end of $\xi_{Es} : \Xi_{Es}$ and E1, E2, ..., En are "added" to $\xi_{USN} : \Xi_{USN}$ – and removed from $\xi_{ASN} : \Xi_{ASN}$. The list of RSL$^+$`text` value definitions

- "**value** e1:E1 = **obs_**E1(e), e2:E2 = **obs_**E2(e), ..., en:En = **obs_**En(e)"

is appended to $\xi_{Val} : \Xi_{Val}$. Here e:E is a value [already] defined in $\xi_{Val} : \Xi_{Val}$.

3.4.1.1.2 PS_EP_DSU: Endurant Part Set Parts.

32. A part set parts `dsu` contains
    (a) an endurant sort name, S, prefixing a display line: **Part Set Part**;
    (b) a **type** literal;
    (c) a pair of the type name, P, of the set, PS, of parts observed from S, where PS is defined as sets of P elements – where PS, P are chosen from $\xi_{ASN} : \Xi_{ASN}$;
    (d) a **value** literal;

(e) and the signature, **obs_**PS, of the PS observer function.

32. PS_EP_DSU ::= E × **Parts Set Parts.**
32b. × **type**
32c. × PS = P-**set**, P
32d. × **value**
32e. × **obs_**PS: E→PS

The meaning of PS_EP_DSU is then the text to the right of the ::= marker. Once PS_EP_DSU has been "performed" the right-hand side of the ::= is appended to the end of $\xi_{Es} : \Xi_{Es}$ and PS, P are "added" to $\xi_{USN} : \Xi_{USN}$ – and removed from $\xi_{ASN} : \Xi_{ASN}$.

The list of RSL$^+$text value definitions

- "**value** ps:PS = { p | p:P :- p ∈ **obs_**PS(e) }"

is appended to $\xi_{Val} : \Xi_{Val}$. Here e:E is a value [already] defined in $\xi_{Val} : \Xi_{Val}$.

• • •

And: once all endurants have been analysed wrt. their possible non-atomicity, i.e., as to whether they embody further compound parts, the domain analyser cum describer decides on which endurant parts are to be transcendentally deduced, i.e., morphed, converted, into behaviours.

Let us refer to this subset of $\xi_{USN} : \Xi_{USN}$ as manifest, then $\xi_{BSN} : \Xi_{BSN}$ is initialised to manifest.

3.4.1.1.3 A Note on Part Set Parts The P in PS = P-**set** is considered atomic. That is: is restricted to be atomic !

Sometimes P is of the form:[15]

type
PS = P-**set**
P == P1 | P2 | ... | Pn
P1 :: **attr_**A11:A11 × **attr_**A11:A12 × ... × **attr_**A1m_{1}:A1m_{1}
P2 :: **attr_**A21:A12 × **attr_**A21:A22 × ... × **attr_**A2m_{2}:A2m_{2}
...
Pn :: **attr_**An1:An1 × **attr_**An2:An2 × ... × **attr_**Ann_{n}:Anm_{n}

Parts of type Pi, for i in the interval 1 to n, are expressed by **mk_**Pi(ai1,ai2,...,aim$_i$). aij's are attributes of Pi and hence of P. **attr_**Aij are attribute observers of Pi and P[16]. These are, of course, "still" atomic. Any attributes ascribed to P are also, inter alia, attributes of Pi – to which one may ascribe further attributes.

3.4.1.2 TAX_DSU: An Endurant Part Taxonomy See Example A.4 on page 31. [We refer to Carl von Linneaeus, the originator of the term Taxonomy[17].] A visual domain model 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.

[15] This is a restriction of conventional RSL [GHH$^+$92] – but is a construct of RSL$^+$text.

[16] See Sect. 3.4.2.5 on page 20.

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

33. A taxonomy `dsu` 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.

33.	TAX_DSU	::=	E × **Taxonomy**
33a.		×	The taxonomy diagram resulting from 33a–33f.

The meaning of TAX_DSU is then the text to the right of the ::= marker. Once EPV_DSU has been "performed" the right-hand side of the ::= is appended to the end of $\xi_{Es} : \Xi_{Es}$.

3.4.1.3 EPV_DSU: Endurant Part Values: See Example A.3 on page 31.

The analyser cum describer decides to "apply" the EPV_DSU "prompt".[18] To generate the ... we make use of the information "kept in" $\xi_{Val} : \Xi_{Val}$. To recall, $\xi_{Val} : \Xi_{Val}$ contains an ordered list of value definitions of the form:

- **value** e:E = $\mathcal{E}(...)$

The RSL$^+$`text` to be generated as a result of the invocation of the EPV_DSU `dsu` is now, first, this ordered list of value definitions. Then the domain analyser cum describer decides on yet some endurant part value, for example, σ_{parts}, the union of all of the values defined in $\xi_{Val} : \Xi_{Val}$ – where those which are single value definitions are made into single element sets. The domain analyser cum describer may then decide to identify one or more subsets of σ_{parts}, for example that, $\sigma_{atomic_{parts}}$, which contains only atomic parts. Thus:

34. An endurant part values `dsu` contains:
    (a) the [display line] keyword **Part Values**
    (b) the literal **value**,
    (c) a list of value definitions – as kept in ξ_{Val}, where ξ_{Val} is of the form:
        - e1:E1 = $\mathcal{E}_1$,

[18] There is no "node" in Fig. 1 on page 3 which prescribes so. So the decision is really when the $\xi_D : \Xi_E$ is first empty !

- e2:E1 = $\mathcal{E}_2$,
- ...
- em:E1 = $\mathcal{E}_m$,

(d) the definition of σ_{parts} the union of all of the above values (e1, e2, ..., em) where, if ei is a part set, then the distributed union of all its elements;

(e) then $\sigma_{atomic_{parts}}$, a subset of σ_{parts} where its elements are atoms;

(f) and, finally, $\sigma_{behav_{parts}}$, the set of parts for which the domain analyser cum describer seek their behaviours, see Item 44e page 24.

34.	EPV_DSU	::=	**Part Values**
34b.		×	**value**
34c.		×	e1:E1 = $\mathcal{E}_1$,
34c.		×	e2:E2 = $\mathcal{E}_2$,
34c.		×	...
34c.		×	em:Em = $\mathcal{E}_m$,
34d.		×	σ_{parts}
34e.		×	$\sigma_{atomic_{parts}}$
34f.		×	$\sigma_{behav_{parts}}$

The σ_{parts} is used by the uniqueness of part `dsu`, cf. Sect. 3.4.2.3 on page 19.

The meaning of EPV_DSU is then the text to the right of the ::= marker. Once EPV_DSU has been "performed" the right-hand side of the ::= is appended to the end of $\xi_{Es} : \Xi_{Es}$.

3.4.2 Internal Qualities

Analysing and describing the internal qualities of endurants is indicated by the the **red** "bullets" on the dashed, horizontal lines at the bottom of Fig. 1 on page 3. [The **blue** bullets indicate that they are used by perdurants.]

3.4.2.1 UID_DSU: Unique Identifiers

See Example A.5 on page 31.

Invocation of the UID_DSU `dsu` is only meaningful after the domain analyser cum describer has completed analysis & description of all endurants. At this point in development the domain analyser cum describer must decide which endurants the "elevate" to perdurant-hood", i.e., to be transcendentally deduced to behaviour. The result of this decision is manifest Invocation is then suggested to be the very first next domain modeling step.

35. A unique identifier [or: unique identification] `dsu` contains

(a) the endurant sort names, E1, E2, ..., En, for which the unique identification is sought, and a display line **Unique Identification** – the E1, E2, ..., En are those listed in $\xi_{BSN} : \Xi_{BSN}$ [manifest];

(b) a **type** literal

(c) and a list of suitably chosen unique identifier sort names, say "I" suffixing Ei, i.e. EiI;

(d) a **value** literal

(e) and a list of the signatures of the unique identifier observers, **uid_**Ei.

35. UID_DSU ::=
35a. E1, E2, ..., En × **Unique Identification**
35b. × **type**
35c. × E1I, E2I, ..., EnI
35d. × **value**
35e. × **uid_**E1: E1→E1I
35e. × **uid_**E2: E2→E2I
35e. × ...
35e. × **uid_**En: En→EnI

Two or more sort unique identifications are "lumped" together – corresponding to the display line E1, E2, ..., En **Unique Identification**. By choosing to name the unique identifier sort as the sort name suffixed with I we can guarantee, due to the distinctness of endurant sort names, that the unique identifier sort names are also distinct.

The unique identifier analysis and description of endurants is a "mechanical" effort as illustrated by the above: the text of the UID_DSU `dsu` can, in a sense, be generated automatically.
The meaning of UID_DSU is then the text to the right of the ::= marker. Once UID_DSU has been "performed" the right-hand side of the ::= is appended to the end of $\xi_{Es} : \Xi_{Es}$.

3.4.2.2 UIDV_DSU: Unique Identifier Values:

See Example A.6 on page 32.

The analyser cum describer decides to "apply" the UIDV_DSU "prompt". The invocation of UIDV_DSU can take place right after completion of the invocation of UID_DSU.

In Sect. 3.4.1.3 on page 16 we introduced and defined the notion of endurant values and endurant value states (ξ_{Val} and $\sigma_{parts}, \sigma_{atomic_{parts}}$, etc.).

36. The unique identifier values `dsu` contains the [display line] keyword **Unique Identifier Values**.
    (a) The literal **value**.
    (b) A list of unique identifier value definitions corresponding to the endurant value definitions of lines 34c on page 16.
    (c) It is then suggested, as the definition will be used subsequently, i.e., next (!), to define the set of the unique identifiers of all parts, and
    (d) of those of only the atomic parts.

36. EPV_DSU ::= **Unique Identifier Values**.
36a. × **value**
36b. × $e1_{uid}$:E1I = **uid_**E1(e1),
36b. × $e2_{uid}$:E2I = **uid_**E2(e2),
36b. × ...
36b. × em_{uid}:EmI = **uid_**Em(em),
36c. × $\sigma_{uid_{parts}}$ = { **uid_**E(p) | p:E ∈ σ_{parts}}
36d. × $\sigma_{uid_{atomicparts}}$ = { **uid_**E(p) | p:E ∈ $\sigma_{atomic_{parts}}$}

The meaning of UIDV_DSU is then the text to the right of the ::= marker. Once UIDV_DSU has been "performed" the right-hand side of the ::= is appended to the end of $\xi_{Es} : \Xi_{Es}$.

3.4.2.3 UNIQ_DSU: Uniqueness of Parts

See Example A.7 on page 33.

As the result of the invocation of the EPV_DSU `dsu`, Sect. 3.4.1.3 on page 16, we have recorded in σ_{parts}, the set of all parts, and in $\sigma_{atomic_{parts}}$ the set of all atomic parts, and as the result of the invocation of the UIDV_DSU `dsu` we have defined their corresponding unique identifier values.

37. The uniqueness of parts is expressed in the [display line] keyword **Uniqueness of Parts.**
    (a) followed by the literal **axiom**
    (b) followed by the axiom that all parts are uniquely identified, i.e., the cardinality of σ_{parts} equals the cardinality of $\sigma_{uid_{parts}}$,

37.	UNIQ_DSU	::=	**Uniqueness of Parts.**
37a.		×	**axiom**
37b.		×	**card** σ_{parts} = **card** $\sigma_{uid_{parts}}$

The meaning of UNIQ_DSU is then the text to the right of the ::= marker. Once UNIQ_DSU has been "performed" the right-hand side of the ::= is appended to the end of $\xi_{Es} : \Xi_{Es}$.

3.4.2.4 MER_DSU: Mereology.

See Example A.8 on page 33.

The "generation" of unique identifier definitions was, sort of, "automatic". Not so for the "generation" of mereology definitions. Here the specific mereology for each [manifest] part requires that the domain analyser cum describer thinks carefully about how each part "relates" to other parts, where "relate" can be taken in the sense of how each part behaviour communicates with other part behaviours.

38. The mereologies of parts `dsu` contains the [display line] keyword **Mereologies**,
    (a) the literal **type**;
    (b) a list of mereology type definitions of the form ME = $\mathcal{T}$(EI1,EI2,...,EIn), where $\mathcal{T}$(EI1,EI2,...,EIn) is a type expression and EI1, EI2, ..., EIn are types of part identifiers (defined in items 35c– 35e on page 17),
    (c) the literal **value**; and
    (d) a list of corresponding mereology observers.

38.	MER_DSU	::=	**Mereologies**
38a.			**type**
38b.		×	ME1 = $\mathcal{T}_1$(EI11,EI12,...,EI1n),
38b.		×	ME2 = $\mathcal{T}_2$(EI21,EI22,...,EI2n),
38b.		×	...
38b.		×	MEm = $\mathcal{T}_m$(EIm1,EIm2,...,EImn)
38c.		×	**value**
38d.		×	**mereo_**E1: E1 → ME1,
38d.		×	**mereo_**E2: E2 → ME2,
38d.		×	...
38d.		×	**mereo_**Em: Em → MEm

The meaning of MEREO_DSU is then the text to the right of the ::= marker. Once MEREO_DSU has been "performed" the right-hand side of the ::= is appended to the end of $\xi_{Es} : \Xi_{Es}$.

3.4.2.5 ATTR_DSU: Attributes.

See Example A.9 on page 34.

As for MEREO_DSU, the decision as to which attributes (type names and, especially, attibute type definitions) to ascribe to parts depends very much on the analytic skills of the domain analyser cum describer.

39. The attributes of parts `dsu` contains the [display line] keyword **Attributes**,
    (a) the literal **type**,
    (b) a list of attribute type definitions,
    (c) the literal **value**, and
    (d) a list of signatures of attribute type observers.

39. ATTR_DSU ::= **Attributes**
    39a. × **type**
    39b. × $A1_1 = \mathcal{T}_{11}(T_{11_1},T_{11_2},...,T_{11_{m1}})$, ..., $A1_{m1} = \mathcal{T}_{1m1}(T_{1m1_1},T_{1m1_2},...,T_{1m1_{m1}})$
    39b. × $A2_1 = \mathcal{T}_{21}(T_{21_1},T_{22_1},...,T_{21_{m2}})$, ..., $A2_{m2} = \mathcal{T}_{2m2}(T_{2m2_1},T_{2m2_2},...,T_{2m2_{m2}})$
    39b. × ...
    39b. × $An_1 = \mathcal{T}_{n1}(T_{n1_1},T_{n1_2},...,T_{n1_{m1n}})$, ..., $An_{mn} = \mathcal{T}_{nm}(T_{nm_1},T_{nm_2},...,T_{nm_{mnn}})$
    39c. × **value**
    39d. × **attr_**$A1_1$: E1 → $A1_1$
    39d. × ...
    39d. × **attr_**An_{mn}: En → An_{mn}

The above formulation may seem "involved", but it really is not. Just read the individual Ai_j as attribute type names for endurant Ei and the type expressions $\mathcal{T}_i(T_{i_1},T_{i_2},...,T_{i_n})$ as simple type expressions over RSL types.

The meaning of ATTR_DSU is then the text to the right of the ::= marker. Once ATTR_DSU has been "performed" the right-hand side of the ::= is appended to the end of $\xi_{Es} : \Xi_{Es}$.

3.4.3 IPULL_DSU: Intentional Pull.

See Example A.10 on page 35.

40. The intentional pull `dsu` is very simple: it contains
    (a) a **Intentional Pulls** display line
    (b) and a set of one more separate intentional pull formulations, each of the form:
    (c) an intentional pull **Name** IPull_Name$_i$
    (d) enumerated text and a likewise
    (e) enumerate formalization in the form of an axiom.

40. IPULL_DSU ::=
40a. **Intentional Pulls**
40c. × **Name:** IPull_Name$_1$
40d. × text
40e. × **axiom** $\mathcal{A}_1(...)$
40c. × **Name:** IPull_Name$_2$
40d. × text
40e. × **axiom** $\mathcal{A}_2(...)$
40. × ...
40c. × **Name:** IPull_Name$_p$
40d. × text
40e. × **axiom** $\mathcal{A}_p(...)$

The meaning of IPULL_DSU is then the text to the right of the ::= marker. Once IPULL_DSU has been "performed" the right-hand side of the ::= is appended to the end of $\xi_{Es} : \Xi_{Es}$.

3.5 Perdurants

3.5.1 CH_DSU: Channel

See Example A.13 on page 39.

41. The channel dsu is very simple: it contains
    (a) a **Channel Declaration** display line,
    (b) the literal **channel** line, and
    (c) the channel declaration: naming the channel (as an array of channels), say ch and the index set of the channel array: The set $\{ui, uj\}$ of all pairs of unique identifier parts – and the type, MSG, of the messages communicated over the channel.

41a CH_DSU ::= **Channel Declaration**
41b × **channel**
41c × { ch[{$\{ui, uj\} \mid \{ui, uj\} \in \sigma_{uid_{parts}}$]}: MSG

The type MSG "transpires" from the action definitions, i.e., from the messages communicated between part behaviours ($\{ui, uj\}$).

The domain analyser cum describer can, at this stage of the development of the domain model, analyse which such output/input messages/values are sent/received (ch[$\{ui, uj\}$] ! expr/ch$\{ui, uj\}$[$\{ui, uj\}$] ?) by behaviours. But we suggest to postpone that to the treatment of behaviour actions. See the next section.

The meaning of CH_DSU is then the text to the right of the ::= marker. Once CH_DSU has been "performed" the right-hand side of the ::= is appended to the end of $\xi_{Es} : \Xi_{Es}$.

3.5.2 BEH_DSU: Behaviours

See Example A.15 on page 39.

The behaviour dsu is "the real thing": Now all lines of analysis, i.e., of external and internal qualities of endurants "meet". The domain analyser cum describer can be give some guidelines, but for the details of the actions referred to below, the domain analyser cum describer must work hard !

42. The behaviour `dsu` is "extensive". It contains:
    (a) a **Behaviours** display line;
    (b) a **value** literal;
    (c) and one, usually [many] more, behaviour signatures
    (d) and behaviour [body] definitions.

Below we hint at the above. The detailing of the individual $\mathcal{B}_i$ is suggested further below, and their invocation of usually non-deterministic choice pro_- and re_active behaviours and the structure of their [body] definitions are also suggested below – and are included in the behaviour `dsu`:

```
42a  CH_DSU  ::=  Behaviours
42b          ×      value
42c          ×        b1: UI1 → M1 → Sta1 → Mon1 → Prgr1 →Unit
42d          ×        b1(ui1)(mer1)(sta1)(mon1)(prgr1)≡B1(ui1)(mer1)(sta1)(mon1)(prgr1) is
42d          ×          ...
42c          ×        b2: UI2 → M2 → Sta2 → Mon2 → Prgr2 →Unit
42d          ×        b2(ui2)(mer2)(sta2)(mon2)(prgr2)≡B2(ui2)(mer2)(sta2)(mon2)(prgr2) is
42d          ×          ...
42a          ×          ...
42c          ×        bn:: UIn → Mn → Stan → Monn → Prgrn →Unit
42d          ×        bn(uin)(mern)(stan)(monn)(prgrn)≡Bn(uin)(mern)(stan)(monn)(prgrn is)
42d          ×          ...
```

We refer to Appendix Sect. B on page 42. Behaviours $\mathcal{B}_i$ are of a variety of forms – shown next (as $\mathcal{B}$). Typically behaviour $\mathcal{B}$ non-deterministically internal choice chooses between pro_active and re_active behaviours:[19]

```
value
ι129a π42.  B(bi)(mereo)(stat)(mon)(prg) ≡
ι129b π42.      pro_active_B(bai)(mereo)(stat)(mon)(prg)
ι129c π43.  ⌈⌉ \ re_active_B(bai)(mereo)(stat)(mon)(prg)
```

The pro_active non-deterministically internal choice choose between a number, m, of actions

```
value
ι130 π43.,ι129b π42.  pro_active_B(bi)(mereo)(stat)(mon)(prg) ≡
ι130a π43.              B_action_1(bi)(mereo)(stat)(mon)(prg)
ι130b π43.            ⌈⌉ B_action_2(bi)(mereo)(stat)(mon)(prg)
ι130c π43.            ⌈⌉ ...
ι130d π43.            ⌈⌉ B_action_m(bi)(mereo)(stat)(mon)(prg)
```

43. . The `responding` behaviour ($\mathcal{B}$) reacts to a number of such initiating actions by
    (a) external non-deterministically ($\square$) offering to accept messages from responding behaviours,

[19] Reference ι129a π42 designate Item 129a on page 42; et cetera.

(b) and then performing corresponding actions.

```
value
ι43a π22.,ι43 π22.  respond_ℬ(bi)(mereo)(stat)(mon)(prg) ≡
ι43a π22.           let msg = [] { comm[{bj,bi}] ?  | bj:BI • bj ∈ bis } in
ι43b π23.           react_behaviour_ℬ(bi)(mereo)(stat)(mon)(prg)(msg) end


value
ι131 π43.  react_behaviour__ℬ(bi)(mereo)(stat)(mon)(prg)(msg) ≡
ι133a π44.     is_action_i(msg) → ℬ_action_i(bi)(mereo)(stat)(mon)(prg)(msg),
ι133b π44.     is_action_j(msg) → ℬ_action_j(bi)(mereo)(stat)(mon)(prg)(msg),
ι133c π44.     ...,
ι133d π44.     is_action_k(msg) → ℬ_action_k(bi)(mereo)(stat)(mon)(prg)(msg),
ι133e π44.     _ → ℬ(bi)(mereo)(stat)(mon)(prg)
```

Once BEH_DSU has been "performed" the right-hand side of the ::= is appended to the end of $\xi_{Es} : \Xi_{Es}$.

3.5.3 INIT_DSU: Initialisation

See Example A.16 on page 41.

44. The domain initialisation dsu is "repetitively" straightforward. It contains:
    (a) a **Domain Initialisation** display line;
    (b) a domain_initialisation signature;
    (c) a domain_initialisation() invocation followed by ≡;
    (d) then, in distributed parallel, the invocation of behaviour $\mathcal{B}_i$, bi,
    (e) for all values, bi, in $\sigma_{behav_{parts}}$ which are not part sets, their initialisation:
    (f) their unique identifier,
    (g) their mereology,
    (h) their static attributes,
    (i) their monitorable attributes – by attribute names, and
    (j) their programmable attributes;
    (k) and, in parallel,
    (l) the similarly distributed parallel composition of behaviours, $\mathcal{B}_p$,
    (m) of those values, p, for which there are values, ej, in $\sigma_{behav_{parts}}$ where ej are part set parts, and where p is in ej,
    (n) their unique identifier,
    (o) their mereology,
    (p) their static attributes,
    (q) their monitorable attributes – by attribute names, and

(r) their programmable attributes.

```
44a.  INIT_DSU  ::=  Domain Initialisation
44b.                 domain_initialisation: Unit → Unit
44c.                 domain_initialisation() ≡
44d.                   ‖ { ℬ_i
44f.                       (uid_Ei(bi))
44g.                         (mereo_Ei(bi))
44h.                           (attr_StaA_p1(bi),...,attr_StaA_px(bi))
44i.                             (ηMonA_q1,...,ηMonA_qy)
44j.                               (attr_PrgA_r1(bi),...,attr_PrgA_rz(bi))
44e.                     | bi:Ei • bi ∈ σ_behav_parts ∧ ¬is_Part_set(bi) }
44k.                   ‖
44l.                   ‖ { ‖ { ℬ_p
44n.                         (uid_P(p))
44o.                           (mereo_P(p))
44p.                             (attr_StaA_p1(p),...,attr_StaA_px(p))
44q.                               (ηMonA_q1,...,ηMonA_qy)
44r.                                 (attr_PrgA_r1(p),...,attr_PrgA_rz(p))
44m.                       | p:P • p∈obs_PS(ej) }
44e.                   | ej:Ei • ej ∈ σ_behav_parts ∧ is_Part_set(ej) }
```

Here $\sigma_{behav_{parts}}$ is defined in Item 34f page 17. Once INIT_DSU has been "performed" the right-hand side of the ::= is appended to the end of $\xi_{Es} : \Xi_{Es}$ – and the domain has been described ! Finito !

4 What is Next ? – Tool Support !

Sections 2–3, pages 4–24, presented, perhaps in a "chatty manner", syntaxes for DAL and DDL.[20] With [Bjø26f, *Methodology*] we present a number of 𝔻𝕆𝕄𝔸𝕀ℕ *method principles, procedures, techniques* and *tools*.

Together these documents illuminate the question: *tool support* for 𝔻𝕆𝕄𝔸𝕀ℕ development ? !

Not only does that mean that a rigorous specification of the DAL and DDL syntaxes is required and that a tool support system implement that, but also that it provides interactive support for the domain analyser cum describers.

I hope in a next document to present a proposal in the form of a rigorous specification of such a 𝔻𝕆𝕄𝔸𝕀ℕ Method **Tool Support**.

5 Conclusion

So, who is/are game ! I have not, since 20 years ago, had MSc students whose MSc thesis work could tackle various aspects of such a tool support.

[20] [Bjø26e] presented those syntaxes in another manner.

Acknowledgements

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, though, I am very happy to acknowledge, again, and perhaps taken out of context, the inspiration, over the years, from the work of Tony Hoare and Michael A. Jackson, from my IBM Vienna Laboratory 1973–1975 colleagues: Peter Lucas, Hans Bekič and Cliff B. Jones, and from the late Søren Prehn and Chris W. George.

MORE TO COME

6 Bibliography

I have taken the liberty of generously citing my own work:[Bjø17, Bjø19c, Bjø20, Bjø25b, Bjø27, BC27, BS07, Bjø16, Bjø26h, Bjø26g, Din25, Bjø26a, Bjø26e, Bjø26b, Bjø26f, Bjø26c, Bjø26d, Bjø25a, Bjø25d, Bjø25c, Bjø23b, Bjø23a, Bjø22b, Bjø23c, Bjø19b, Bjø22a, Bjø, Bjø15, Bjø19a, BH19, Bjø18] !

This document is one of a quadruplet, [Bjø26a, Bjø26b, Bjø26e, Bjø26f], of documents that I worked on during the winter and spring of 2026.[Bjø10a, Bjø08, Bjø10b, Bjø11a, Bjø09, Bjø11b, Bjø17, Bjø19c, Bjø20, Bjø, Bjø27, Bjø25b, Bjø25c, Bjø25d, Bjø25a, Bjø26a, Bjø26e, Bjø26c, Bjø26b, Bjø26d, Bjø16, Bjø26f, Bjø26g, Bjø26h]

References

[BC27] Dines Bjørner and Mikhail Chupilko. Моделирование предметной области: основы. Translation of [Bjø27]. [Book], To be submitted [2026/2027].

[BH19] Dines Bjørner and Klaus Havelund. 45 Years of Formal Methods – Challenges and Trends. www.imm.dtu.dk/˜dibj/2019/45years/45years.pdf. Technical report, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, Denmark, September 2019.

[Bjø] 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*, July 2023 https://www.imm.dtu.dk/˜dibj/2023/nupopl/nupopl.pdf
- 2021: *Shipping*, April 2021 https://www.imm.dtu.dk/˜dibj/2021/ral/ral.pdf
- 2021: *Rivers and Canals – Endurants – A Technical Note*, March 2021 https://www.imm.dtu.dk/˜dibj/2021/Graphs/Rivers-and-Canals.pdf
- 2021: *A Retailer Market*, January 2021 https://www.imm.dtu.dk/˜dibj/2021/Retailer/BjornerHeraklit27January2021.pdf

- 2019: *Container Terminals*, ECNU, Shanghai, China, Sept. 2018
  https://www.imm.dtu.dk/~dibj/2018/yangshan/maersk-pa.pdf
- 2017: *Documents*, TongJi Univ., Shanghai, China, Sept. 2017
  https://www.imm.dtu.dk/~dibj/2017/docs/docs.pdf
- 2017: *Urban Planning*, TongJi Univ., Shanghai, China, Sept. 2017
  https://www.imm.dtu.dk/~dibj/2017/urban-planning.pdf
- 2017: *Swarms of Drones*, Inst. of Softw., Chinese Acad. of Sci., Peking, China, November 2017
  https://www.imm.dtu.dk/~dibj/2017/swarms/swarm-paper.pdf
- 2013: *Road Transport*, Techn. Univ. of Denmark
  https://www.imm.dtu.dk/~dibj/2013/road/road-p.pdf
- 2012: *Credit Cards*, Uppsala, Sweden
  https://www.imm.dtu.dk/~dibj/2016/credit/accs.pdf
- 2012: *Weather Information*, Bergen, Norway
  https://www.imm.dtu.dk/~dibj/2016/wis/wis-p.pdf
- 2010: *Web-based Transaction Processing*, Techn. Univ. of Vienna, Austria, April 2010
  https://www.imm.dtu.dk/~dibj/wfdftp.pdf
- 2010: *The Tokyo Stock Exchange*, Tokyo Univ., Japan, November 2009
  https://www.imm.dtu.dk/~db/todai/tse-1.pdf,
  https://www.imm.dtu.dk/~db/todai/tse-2.pdf
- 2009: *Pipelines*, Techn. Univ. of Graz, Austria, November 2008
  https://www.imm.dtu.dk/~dibj/pipe-p.pdf
- 2007: *A Container Line Industry Domain*, Techn. Univ. of Denmark, June 2007
  https://www.imm.dtu.dk/~dibj/container-paper.pdf
- 2002: *The Market*, Techn. Univ. of Denmark
  https://www.imm.dtu.dk/~dibj/themarket.pdf
- 1995–2004: *Railways*, Techn. Univ. of Denmark - a compendium
  https://www.imm.dtu.dk/~dibj/train-book.pdf

This is not a single document. It is a [printable] collection of older case studies. Some, i.e., the later ones, adhere to the emerging DDL: Domain Description Language, [Bjø26e]. Earlier ones may not. *Urban Planning*, for example, https://www.imm.dtu.dk/~dibj-/2017/urban-planning.pdf, do not. It treats 𝕋𝕀𝕄𝔼 in an erroneous way – and should be corrected.

[Bjø08] 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.

[Bjø09] Dines Bjørner. *Domain Engineering: Technology Management, Research and Engineering*. Research Monograph (# 4); JAIST Press, 1-1, Asahidai, Nomi, Ishikawa 923-1292 Japan; https://www.imm.dtu.dk/~dibj/jaistmono.pdf, 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.

.

[Bjø10a] 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. www.imm.dtu.dk/~dibj/facs-domain.pdf, https://www.booksamillion.com/p/Formal-Methods/Paul-Boca/9781848827356.

[Bjø10b] Dines Bjørner. Domain Science & Engineering – *From Computer Science to The Sciences of Informatics, Part I: The Engineering Part*. *Kibernetika i sistemny analiz*, 2(4):100–116, May 2010. www.imm.dtu.dk/~db/kiev-p1.pdf.

[Bjø11a] Dines Bjørner. Domain Science & Engineering – *From Computer Science to The Sciences of Part II: The Science Part*. *Kibernetika i sistemny analiz*, 2(3):100–120, June 2011. www.imm.dtu.dk/~db/kiev-p2.pdf.

[Bjø11b] 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.

[Bjø15] Dines Bjørner. *ProCoS: How It All Began – as Seen from Denmark www.imm.dtu.dk/ dibj/2015/procos-bjorner.pdf*. NASA Monographs in Systems and Software Engineering. Springer, March 9–10 2015. BCS-FACS ProCoS Workshop on Provably Correct Systems, London, UK, printed in Provably Correct Systems. Eds. Hinchey M., Bowen J., Olderog E-R.

[Bjø16] Dines Bjørner. Domains: Ontologies and Taxonomies: $\in+\mathcal{N}$. Technical report, DTU Compute, Techn.Univ.of Denmark, Fredsvej 11, DK-2840, Denmark, March 2016. Experimental Researchf [https://www.imm.dtu.dk/ dibj/2026/OntoTaxo/main.pdf].

[Bjø17] 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.

[Bjø18] Dines Bjørner. Container Terminals. www.imm.dtu.dk/~dibj/2018/yangshan/maersk-pa.pdf. Technical report, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, Denmark, September 2018. An incomplete draft report; currently 60+ pages.

[Bjø19a] Dines Bjørner. A Philosophy Basis for Domain Science & Engineering? www.imm.dtu.dk/~dibj/2019/filo/philosophy1.pdf. Technical report, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, Denmark, September 2019.

[Bjø19b] Dines Bjørner. An Assembly Plant Domain – Analysis & Description. Technical report, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, Denmark, September 2019. `www.imm.dtu.dk/~dibj/2021/assembly/assembly-line.pdf`.

[Bjø19c] Dines Bjørner. Domain Analysis & Description – Principles, Techniques and Modeling Languages. *ACM Trans. on Software Engineering and Methodology*, 28(2):66 pages, March 2019. www.imm.dtu.dk/~dibj/2018/tosem/Bjorner-TOSEM.pdf.

[Bjø20] Dines Bjørner. *Domain Science & Engineering – A Foundation for Software Development*. EATCS Monographs in Theoretical Computer Science. Springer, Heidelberg, Germany, January 2020. xii+346 pages. A revised version of this book is [Bjø27]. https:///doi.org/10.1007/978-3-030-73484-8.

[Bjø22a] Dines Bjørner. A Domain Science & Engineering Interpretation of Sørlander's Philosophy, http://www.imm.dtu.dk/ dibj/2022/sorlander/Sorlander.pdf. Technical report, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, Denmark, September 2022. Currently incomplete.

[Bjø22b] Dines Bjørner. Documents: A Basis for Government. In *United Nations Inst., Festschrift for Tomas Janowski and Elsa Estevez*. Guimaraes, Portgal, October, `www.imm.dtu.dk/~dibj/2022/janowski/docs.pdf`, 2022.

[Bjø23a] Dines Bjørner. Domain Modelling. In Jonathan Bowen et al., editor, *Theories of Programming and Formal Methods: Essays Dedicated to Jifeng He on the Occasion of His 80th Birthday*, Lecture Notes in Computer Science, Festschrift. Spriger, Heidelberg, Germany, August 2023. https://www.imm.dtu.dk/~dibj/2023/FEA/hjf.pdf and https://www.imm.dtu.dk/ dibj/2023/final/HeJiFeng.pdf.

[Bjø23b] Dines Bjørner. Double-entry Bookkeeping. Research, Institute of Mathematics and Computer Science. Technical University of Denmark, DK-2800 Kgs.Lyngby, Denmark, August 2023. `http://www.imm.dtu.dk/~dibj/2023/doubleentry/dblentrybook.pdf`.

[Bjø23c] Dines Bjørner. Pipelines: A Domain Science & Engineering Description. In *FSEN 2023: Fundamentals of Software Engineering, Teheran, Iran, May 3–5*. `www.imm.dtu.dk/ dibj/2023/tehran/tehran.pdf`, 2023.

[Bjø25a] 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.

[Bjø25b] 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.

[Bjø25c] Dines Bjørner. Domain Analysis & Description. Technical report, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, September 2025.

[Bjø25d] 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.

[Bjø26a] 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`.

[Bjø26b] 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`.

[Bjø26c] Dines Bjørner. DADL: A DOMAIN Analysis and Description Language. Technical report, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, February 2026. Incomplete draft. `https://www.imm.dtu.dk/~dibj/2026/ddl/ddl.pdf`.

[Bjø26d] Dines Bjørner. DADL: A Domain Analysis and Description Language. Technical report, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, February 2026. Incomplete draft. `www.imm.dtu.dk/~dibj/2026/ddl/ddl.pdf`.

[Bjø26e] Dines Bjørner. DADL: An 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`.

[Bjø26f] 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`.

[Bjø26g] Dines Bjørner. The DOMAIN Method. Technical report, DTU Compute, Technical University of Denmark, March 2026. Submitted.

[Bjø26h] Dines Bjørner. **Wolt** A DOMAIN Decription. Technical report, DTU Compute, Technical University of Denmark, August 2026. Incomplete draft. `[https://www.imm.dtu.dk/~dibj/2026/wolt/main.pdf]`.

[Bjø27] Dines Bjørner. Domain Science & Engineering, a Primer[21] Technical University of Denmark. Significantly revised edition of [Bjø20]. xii+211 pages., February 2026 To be submitted [2026/2027].

[BS07] Dines Bjørner and Yang ShaoFa. 领域科学与工程导论. Translation of [Bjø27]. [Book], To be submitted [2026/2207].

[21] This book is currently being translated into Russian, [BC27], by Dr. Mikhail Chupilko and his colleagues, ISP/RAS (Institute of Systems Programming, Russian Academy of Sciences), Moscow and into Chinese, [BS07], by Dr. Yang ShaoFa, IoS/CAS (Institute of Software, Chinese Academy of Sciences), Beijing.

[Din25] Dines Bjørner. Formal Methods: My 50+ Years as an Engineer, Researcher and Scientist. *Formal Aspects of Computing [ACM]*, 2025.

[GHH+92] 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. ISBN 0137528337 and 9780137528332.

[Low25] 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.

A Example

In this example we omit many of the keywords, etc., otherwise mentioned in Sects. 3.3–3.5.2, etc.

A.1 Universe of Discouse

45. The universe of discourse is that of
46. road traffic, RT –
47. whose narrative is then presented.

45. **universe of discourse**:
46. **type** RT
47. **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.

A.2 Compound Endurants

48. In the road traffic domain we can observe a road net aggregate and an automobile aggregate.
49. In a road net aggregate we can observer an aggregate of hubs and an aggregate of links.
50. In an aggregate of automobiles we can observe a set of [atomic] automobiles.
51. Aggregates of hubs ad links are sets of [atomic] hubs and links.

type
48. RNA, AA
49. AH, AL
50. AS = A-**set**
51. HS = H-**set**, LS = L-**set**

value

48. **obs_**RNA: RT → RNA, **obs_**AA: RT → AA
49. **obs_**AH: RNA → AH, **obs_**AL: RNA → AL
50. **obs_**AS: AA → AS
51. **obs_**HS: AH → RS, **obs_**LS: AL → LS

A.3 Endurant Values

52. There is the road transport [universe of discourse].
53. There is the road net aggregate.
54. There is the automobile aggregate.
55. There is the hub aggregate.
56. There is the link aggregate.
57. There is the set of automobiles.
58. There is the set of hubs.
59. There is the set of links.
60. And there is the entire set of all of these.
61. And there is the entire set of just all automobiles, hubs and links.

value
52. rt:RT
53. rna:RNA = **obs_**RNA(rt)
54. aa:AA = **obs_**AA(rt)
55. ah:AH = **obs_**AA(rna)
56. al:AL = **obs_**AL(rna)
57. as:AS = **obs_**AS(ah)
58. hs:HS = **obs_**HS(ah)
59. ls:LS = **obs_**LS(al)
60. σ_{parts} = {rt,rna,aa,ah,al} ∪ aa ∪ as ∪ hs ∪ ls
61. σ_{atoms} = aa ∪ hs ∪ ls

A.4 Taxonomy

A.5 Unique Identification

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

62. The universe of discourse has a unique identifier.
63. The road net agregate has a unique identifier.
64. The automobile aggregate has a unique identifier.

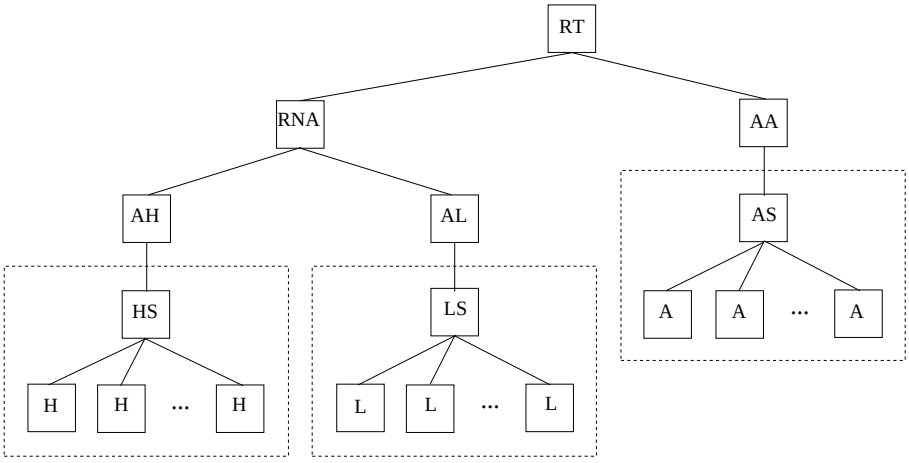

Figure 3: A Road Transport Taxonomy

65. The hub aggregate has a unique identifier.
66. The link aggregate has a unique identifier.
67. Each hub has a unique identifier.
68. Each link has a unique identifier.
69. Each automobile has a unique identifier.

type

62. RTI
63. RNAI
64. AAI
65. HAI
66. LAI
67. HI
68. LI
69. AI

value

62. **uid_**RT: RT → RTI
63. **uid_**RNA: RNA → RNAI
64. **uid_**AA: AA → AAI
65. **uid_**HA: HA → HAI
66. **uid_**LA: LA → LAI
67. **uid_**H: H → HI
68. **uid_**L: L → LI
69. **uid_**A: A → AI .

A.6 Unique Identifier Values

70. There is the unique identifier of the road transport.
71. There is the unique identifier of the road net aggregate.
72. There is the unique identifier of the autombile aggregate.
73. There is the unique identifier of the hub aggregate.
74. There is the unique identifier of the link aggregate.
75. There are the unique identifiers of the automobiles of the set of autombiles.

76. There are the unique identifiers of the hubs of the set of hubs.
77. There are the unique identifiers of the links of the set of links.
78. And there is the entire set of all of these.
79. And there is the entire set of all identifiers of the automobiles, hubs and links.

value
70. rt_{uid}RT: RT = **uid_**RT(rt)
71. rna_{uid}RNA: RNA = **uid_**RNA(rt)
72. aa_{uid}AA: AA = **uid_**AA(rt)
73. ah_{uid}AH: AH = **uid_**AA(rna)
74. al_{uid}AL: AL = **uid_**AL(rna)
75. as_{uid}AS: AS = {**uid_**A(a)|a:A•a∈as}
76. hs_{uid}:HS = {**uid_**H(h)|h:H•h∈hs}
77. ls_{uid}:LS = {**uid_**L(l)|l:L•a∈ls}
78. $\sigma_{parts_{uid}}$ = {rt,rna,aa,ah,al} ∪ aa ∪ hs unon ls
79. $\sigma_{atoms_{uids}}$ = aa ∪ hs ∪ ls

A.7 Uniqueness of Endurants

80. Parts are uniquely identfied.

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

A.8 Mereology

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

81. The mereology of links is a 2 element set of hub identifiers.
82. The mereology of a hub is a possibly empty set of hub identifiers.
83. The mereology of an automobile is a set of hub and link identifiers

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

value
81. **mereo_**L: L → ML
82. **mereo_**H: H → MH
83. **mereo_**A: A → MA .

A.9 Attributes

Example attributes are:

- **Hubs:**

  84. Hubs have states, hσ:HΣ: the set of pairs of link identifiers, (*f*li,*t*li), of the links *f*rom and *t*o which automobiles may enter, respectively leave the hub.
  85. Hubs have state spaces, hω:HΩ: the set of hub states "signaling" which states are open/closed, i.e., **green**/**red**.

- **Links:**

  86. Links that have lengths, LEN;
  87. Links have states, lσ:LΣ: the set of pairs of link identifiers, (*f*li,*t*li), of the links *f*rom and *t*o which automobiles may enter, respectively leave the hub.
  88. Links have state spaces, lω:LΩ: the set of link states "signaling" which states are open/closed, i.e., **green**/**red**.

- **Automobiles:**

  89. 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.
  90. Automobiles have velocity;
  91. Automobiles have acceleration;
  92. Etc.[22]

- **Hubs, Links, Automobiles:**

  93. 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.
  94. Link positions have well-defined identifiers and fractions.

[22]

type
84. HΣ = (LI×LI)-**set** [**prg**]
85. HΩ = HΣ-**set** [**sta**]
86. LEN = **Nat** [**sta**] [m]
87. LΣ = (HI×HI)-**set** [**prg**]
88. LΩ = LΣ-**set** [**sta**]
89. APos = atH | onL [**prg**]
89a. atH :: HI
89b. onL :: LI × (fhi:HI × f:**Real** × thi:HI)
90. Veloc = **Real** [**mon**] [km/h]
91. Accel = **Real** [**mon**] [m/sec]
92. ...
93. HHis,LHis = ($\mathbb{TIME}$×AI)* [**prg**]
93. AHis = ($\mathbb{TIME}$×(HI|LI))* [**prg**]

value
84. attr_HΣ: H → HΣ
85. attr_HΩ: H → HΩ
86. attr_LEN: L → LEN
87. attr_LΣ: L → LΣ
88. attr_LΩ: L → LΩ
89. attr_APos: A → APos
90. attr_Veloc: A → Veloc
91. attr_Accel: A → Accel
93. attr_HHis: H → HHis
93. attr_LHis: L → LHis
93. attr_AHis: A → AHis

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

A.10 Intentional Pull

95. 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.
96. 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
95a. ∀ a:A • a ∈ as ⇒
95a. **let** ahist = attr_AHist(a) **in**
95a. ∀ ui:(LI|HI) • ui ∈ **dom** ahist ⇒
95b. ∀ τ:$\mathbb{TIME}$ • τ ∈ **elems** ahist(ui) ⇒
95b. **let** hist = is_LI(ui) → attr_LHist(retr_L(ui))(σ),
95b. _ → attr_HHist(retr_H(ui))(σ) **in**
95b. τ ∈ **elems** hist(uid_A(a)) **end end**
96. ∧
96c. ∀ u:(L|H) • u ∈ ls∪hs ⇒
96c. **let** uhist = attr(L|H)Hist(u) **in**
96d. ∀ ai:AI • ai ∈ **dom** uhist ⇒
96d. ∀ τ:$\mathbb{TIME}$ • τ ∈ **elems** uhist(ai) ⇒
96d. **let** ahist = attr_AHist(retr_A(ai))(σ) **in**
96d. τ ∈ **elems** uhist(ai) **end end**

A.11 Auxiliary Types

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

97. A path element identifier is either a link identifier or a hub identifier.
98. A path (of a road net) is a finite[23] sequence of one or more alternating hub and link identifiers
99. 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.
100. Given a hub [a link] identifier we can retrieve the identified hub [link].

type
97. PEI = LI | HI
98. Path = PEI*
99. **axiom** [Well-formed Paths]
98. ∀ path:Path •
99a. ∀ {i,i+1}⊆**inds** path ⇒
99a. ((is_HI(path[i])∧is_LI(path[i+1]) ∨ is_LI(path[i])∧is_HI(path[i+1]))
99b. ∧ (path[i]∈hs_{uis}⇒path[i+1]∈ls_{uis}
99c. ∧ **uid_**H(retr_hub(path[i]))∈**mereo_**L(retr_hub(path[i])))
99d. ∧ (path[i]∈ls_{uis}⇒path[i+1]∈hs_{uis}
99e. ∧ **uid_**L(retr_link(path[i])∈**mereo_**H(retr_link(path[i]))))
value
100. retr_hub: HI → H, retr_link: LI → L, retr_unit: UI → U
100. retr_hub(hi) **as** h • h ∈ hs ∧ **uid_**H(h)=hi
100. retr_link(li) **as** l • l ∈ ls ∧ **uid_**L(l)=li
100. retr_unit(ui) **as** u • u ∈ hs∪ls ∧ **uid_**U(u)=ui
100. **uid_**U(u) ≡ is_L(u)→**uid_**L(u),is_H(u)→**uid_**H(u)

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.

A.12 Simple Function Values

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

[23] We shall only consider finite paths. The `paths` function, Item 101 below, can easily be modified to yield also infinite length paths !

101. The paths[24] 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. A.11 on the facing page.

    We define the paths function in two ways.

102. Either axiomatically

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

104. or inductively[25]:

    (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
101. paths: **Unit** → Path-**infset**
102. paths() **as** ps
103. **such that**: ∀ p:ps satisfy the wellformedness of Sect. A.11 on the preceding page

We can also express the `paths` functions explicitly:

value
101. paths: **Unit** → Path-**infset**
104. paths() ≡
104a. **let** ps = {⟨ni⟩ | ni:NI ∈ hs_{uis}}∪{⟨ei⟩ | ei:EI ∈ ls_{uis}}
104(b)iv. ∪ { pi⌢⟨ui⟩⌢⟨uj⟩⌢pj | pi⌢⟨ui⟩:Path-**set**, ⟨uj⟩⌢pj:Path-**infset** •
104b. ∧ ({pi⌢⟨ui⟩,⟨uj⟩⌢pj}⊆ps)
104(b)i. ∧ (ui∼ ∈ **elems** pj ∧ uj∼ ∈ **elems** pi)
104(b)iii. ∧ (ui ∈ **mereo_**U(retr_unit(uj)))
104(b)ii. ∧ (uj ∈ **mereo_**U(retr_unit(ui)))} **in**
104c. ps **end**
type
101. U = H|L

Solution to the equation, lines 104a–104(b)i, is "'obtained' by a smallest set fix-point reasoning.

[24] **Alarm !** Check that this function indeed generates only finite length paths !
[25] https://www.cs.odu.edu/ toida/nerzic/content/recursive_def/more_ex_rec_def.html

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

value
105. $paths$: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.

106. 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]
106. ∀ g:G,p:Path•p ∈ paths(g) ⇒ rev_path(p) ∈ paths(g)
value
106. rev_path: P → P
106. rev_path(p) ≡
106. **case** p **of**
106. ⟨⟩ → ⟨⟩,
106. ⟨ui⟩ → ⟨ui⟩,
106. ⟨ui⟩⌢p′⌢⟨uj⟩ → ⟨uj⟩⌢rev_path(p′)⌢⟨ui⟩
106. **end**

We can define further functions. For example:

107. `path_length`,
108. `shortest_path`, etc.

value
107. path_length: P → LEN
107. path_length(p) ≡
107. **case** p **of**
107. ⟨⟩ → 0
107. ⟨ui⟩ → retr_path_length(ui),
107. ⟨ui⟩⌢p′ → retr_length(ui)+path_path_length(p′)
107. **end**
107. retr_path_length: UI → LEN
107. retr_path_length(ui) ≡ (is_EI(ui)→**attr_**LEN(retr_link(ui)),is_NI(ui)→0)

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

A.13 Channel

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

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

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

A.14 Variables

A.15 Behaviours

There are three behaviours:

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

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

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

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

114. 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.

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

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

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

117. We abstract automobile behaviour at a Hub (hi).
    (a) Either the automobile **remain**s at the hub,
    (b) or, **internally non-deterministically**,
    (c) **leaves** the hub entering a link,

(d) or, **internally non-deterministically**,

(e) **stops**.

```
117 automobile(ai)(ris)(...)(atH(hi),ahis) ≡
117a   automobile_remain_at_hub(ai)(ris)(...)(atH(hi),ahis)
117b   ⌈⌉
117c   automobile_leaving_hub(ai)(ris)(...)(atH(hi),ahis)
117d   ⌈⌉
117e   automobile_stop(ai)(ris)(...)(atH(hi),ahis)
```

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

118. [117a] The automobile **remains** at a hub:

(a) time is recorded,

(b) informing the hub behaviour, whereupon

(c) the automobile remains at that hub, "idling",

```
118 automobile_remain_at_hub(ai)(ris)(...)(atH(hi),ahis) ≡
118a   let τ = record_𝕋𝕀𝕄𝔼() in
118b   ch[ {ai,hi} ] ! τ ;
118c   automobile(ai)(ris)(...)(atH(hi),⟨(τ,hi)⟩^ahis) end
```

119. [117c] 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.

```
119 automobile_leaving_b(ai)({li}∪ris)(...)(atH(hi),ahis) ≡
119a   let τ = record_𝕋𝕀𝕄𝔼() in
119b   (ch[ {ai,hi} ] ! τ ‖ ch[ {ai,li} ] ! τ) ;
119c   automobile(ai)(ris)(...)(onL(li,(hi,0,_)),⟨(τ,li)⟩^ahis) end
119    pre: [hub is not isolated]
```

120. [117e] Or the automobile **stop**s, "disappears — off the radar" !

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

121. 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 (!)].

```
121 automobile(ai)(ris)(...)(onL(li(fhi,f,thi)),lhis) ≡
121a    automobile_remains_on_link(ai)(ris)(...)(onL(li(fhi,f,thi)),lhis)
121b    ⌈⌉
121c    automobile_leaving_linl(ai)(ris)(...)(onL(li(fhi,f,thi)),lhis)
121d    ⌈⌉
121e    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`.

122. Hubs

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

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

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

125. Links

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

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

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

A.16 Initialization

128. 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
128. initialisation: Unit → Unit
128. initialisation() ≡
128a.    ‖ { hub(uid_H(h))
128a.              (mereo_H(h))
128a.                 (attr_HΩ(h),...)
128a.                    (attr_HΣ(h),attr_HΣ(h),attr_HHist(h))
128a.      | h:H • h ∈ hs }
128b.    ‖
128c.    ‖ { link(uid_L(l))
128c.              (mereo_L(l))
128c.                 (attr_LEN(l),...)
128c.                    (attr_LΣ(l),attr_LHist(l))
128c.      | l:L • l ∈ ls }
128d.    ‖
128e.    ‖ { automobile(uid_A(a))
128e.              (mereo_A(a))
128e.                 (attr_APos(a),attr_AHist(a))
128e.      | a:A • a ∈ as }
```

B Behaviour Patterns

B.1 Behaviour Definitions

A typical, informal rendition of abstracted behaviours, `BA, BC, BD, ...` is shown in Fig. B.1 on the facing page.

Figure B.1 on the next page should be understood as follows:[26] 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 B.1 on the facing page is now given a more precise "meaning" – with this "meaning" suggesting a general "pattern" for behaviour definitions:

129. 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

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

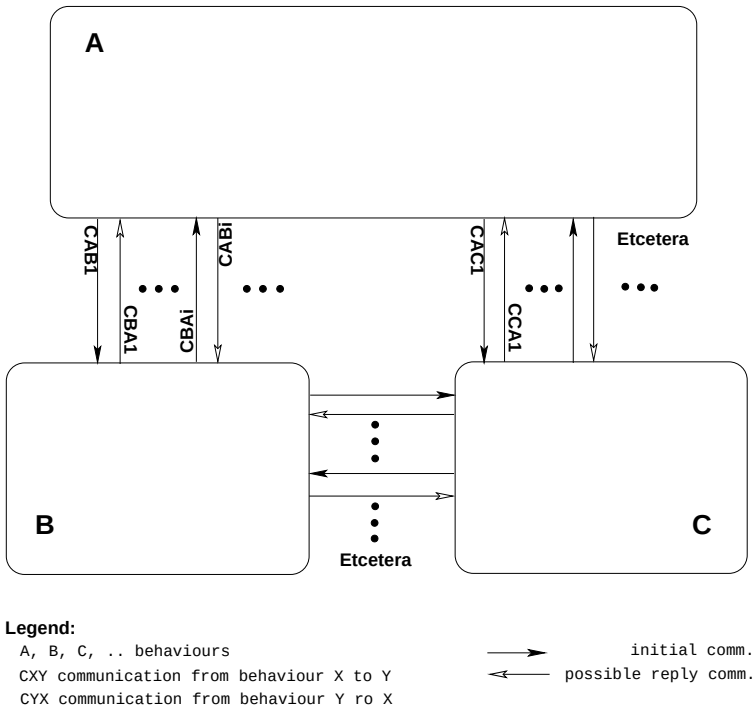

Figure 4: Communicating Behaviours

(c) and *re-actively* responding to such initiatives.

value
129a. B(bi)(mereo)(stat)(mon)(prg) ≡
129b. pro_active_B(bai)(mereo)(stat)(mon)(prg)
129c. ⌈⌉ re_active_B(bai)(mereo)(stat)(mon)(prg)

130. 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.

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

value
130. pro_active_B(bi)(mereo)(stat)(mon)(prg) ≡
130a. B_action_1(bi)(mereo)(stat)(mon)(prg)
130b. ⌈⌉ B_action_2(bi)(mereo)(stat)(mon)(prg)
130c. ⌈⌉ ...
130d. ⌈⌉ B_action_m(bi)(mereo)(stat)(mon)(prg)
where:

131. B_action_i(bi)(mereo)(stat)(mon)(prg) ≡
131. **let** prgr′ = Action(bi)(mereo)(stat)(mon)(prg) **in**
131. B(bi)(mereo)(stat)(mon)(prg′) **end**

132. 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
132. re_active_B(bi)(mereo)(stat)(mon)(prg) ≡
132a. **let** msg = [] { **comm**[{bj,bi}] ? | bj:BI • bj ∈ bis } **in**
132b. respond_behaviour_B(bi)(mereo)(stat)(mon)(prg)(**msg**) **end**

133. 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
133. react_behaviour_B(bi)(mereo)(stat)(mon)(prg)(**msg**) ≡
133a. is_action_i(msg) → B_action_i(bi)(mereo)(stat)(mon)(prg)(**msg**),
133b. is_action_j(msg) → B_action_j(bi)(mereo)(stat)(mon)(prg)(**msg**),
133c. ...,
133d. is_action_k(msg) → B_action_k(bi)(mereo)(stat)(mon)(prg)(**msg**),
133e. _ → B(bi)(mereo)(stat)(mon)(prg)

Et cetera !

B.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.

134. 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.

```
134. action_i(bi)(mereo)(stat)(mon)(prg) ≡
134a.   let v = evaluate_i(bi)(mereo)(stat)(mon)(prg) in
134b.   let (bj,prg′) = elaborate_i(v)(bi)(mereo)(stat)(mon)(prg) in
134c.   comm[{bi,bj}] ! ℰ(prg′) ;
134d.   behaviour(bi)(mereo)(stat)(mon)(prg′)
134.    end end
```

Variants of Item ι134c π45 are also used:

{ comm[{bi,bj}] ! ℰ(prg′) | bj ∈ bis } ;

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

• • •

I refer to a recent paper [Low25].