A Syntax for DDL
First Attempt

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 3, 2026: 10:27 am

Abstract

We present a syntax for the 𝔻𝕆𝕄𝔸𝕀ℕ analysis & description language DDL. An Appendix Example supports the presentation.

Contents

• • •

The Triptych Dogma

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

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

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

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].

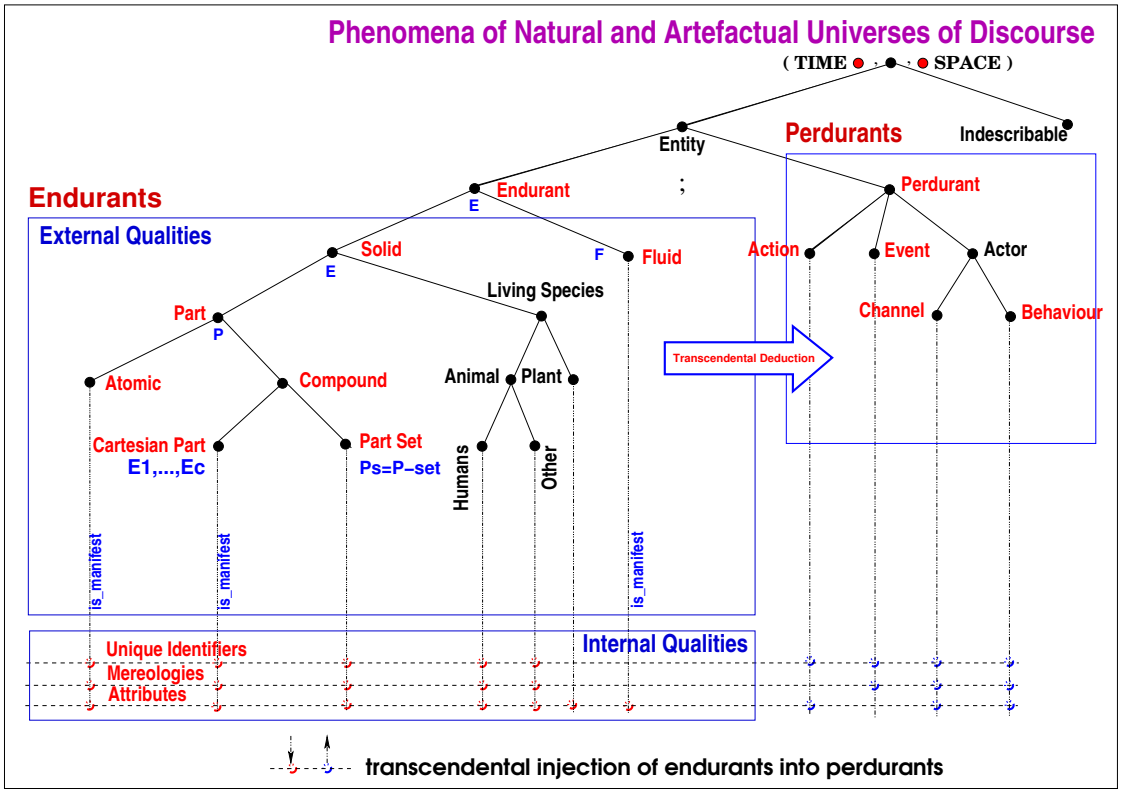

Figure 1: The DOMAIN Analysis & Description Ontology

Over the years 2009 till present we have researched, experimentally applied [1] and formulated [4, 2, 5, 6, 3, 7, 8, 9, 10, 1, 18, 12, 13, 11, 16, 15, 17] 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 DDL.

The DOMAIN method has, as a focal point, that of Fig. 1.

2 The Syntax of DDL

2.1 DOMAIN Specifications

1. A domain specification is a set of domain specification units, `dsu`.
   1. 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:

2. A domain specification thus contains
   (a) exactly one universe of discourse `dsu`,
   (b) two or more endurant part `dsus`, EP_DSU,
   (c) a taxonomy diagram `dsu`, TAX_DSU,
   (d) one or more endurant part value `dsus`, EPV_DSU,

(e) one or more unique identification `dsus`, UID_DSU,

(f) a unique identifier values `dsu`, UIDV_DSU,

(g) a uniqueness axiom `dsu`, UNIQ_DSU,

(h) one or more mereology `dsus`, MER_DSU,

(i) one or more attribute `dsus`, ATTR_DSU,

(j) zero or more intentional pull `dsu`, IPULL_DSU,

(k) zero or more variable declaration `dsus`, VAR_DSU,

(l) exactly one channel declaration `dsu`, CH_DSU,

(m) two or more behaviour definition `dsus`, BEH_DSU, and

(n) exactly one domain initialization `dsu`, INIT_DSU.

1.	DS	::=	DSU^+
2a.	DSU^+	::=	UOD_DSU
2b.		×	EP_DSU-**set**$_2$
2c.		×	TAX_DSU
2d.		×	EPV_DSU-**set**$_1$
2e.		×	UID_DSU-**set**$_1$
2f.		×	UIDV_DSU-**set**$_1$
2g.		×	UNIQ_DSU
2h.		×	MER_DSU-**set**$_1$
2h.		×	ATTR_DSU-**set**$_1$
2j.		×	IPULL_DSU-**set**$_0$
2k.		×	VAR_DSU-**set**$_0$
2l.		×	CH_DSU
2m.		×	BEH_DSU-**set**$_2$
2n.		×	INIT_DSU

Concrete syntax-speaking, You may think of the elements of a `dsu` set to be separated by some syntactic marker, typically the display lines[1] mentioned [first] in the syntaxes given below.

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.

2.2 Some Remarks

2.2.1 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[2] analysis & description state[3]: $\xi : \Xi$:

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

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

[2] nonuplet: group of nine!

[3] 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.

(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[4];

(v) $\xi_{Es} : \Xi_{Es}$: a most recent last list[5] 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$^+$`text`s, and

(ix) $\xi_{ASN} : \Xi_{ASN}$: a set of unused, i.e., available sort names.[6]

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.[7]

2.2.2 No Recursively Defined Endurants

Based on both extensive domain modeling [1] and on rational reasoning [10] we can state that domains do not exhibit inherently recursively definable endurants[8].

2.2.3 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.[9]

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[10] "pleasing" name, usually a terse abbreviation thereof.

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

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

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

[7] The above was written 10-13 February 2026. As of March 3, 2026: 10:27 am it is a bit "unrehearsed": This whole text, pages 4–5, need by "cleaned up", better structured, etc.

[8] 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`.

[9] 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.

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

2.3 Universe of Discourse

See Example A.1 on page 19.

3. 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.

```
3.  UOD_DSU  ::=  UoD × Universe of Discourse.
3.               ×    Text
3.               ×    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''`.

2.4 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`.

2.4.1 External qualities

2.4.1.1 EP_DSU: Endurant Parts.

See Example A.2 on page 19.

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

```
4.   EP_DSU  ::=
4a.              C_EP_DSU
4b.           |  PS_EP_DSU
```

2.4.1.1.1 C_EP_DSU: A Cartesian Observer.

5. 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.

```
5.   C_EP_DSU  ::=  E × Cartesian Part.
5b.             ×    type
5c.             ×      E1, E2, ..., En
5d.             ×    value
5e.             ×      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 $\xi_{Val} : \Xi_{Val}$.

2.4.1.1.2 PS_EP_DSU: Endurant Part Set Parts.

6. 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.

```
6.   PS_EP_DSU  ::=  E × Parts Set Parts.
6b.              ×  type
6c.              ×      PS = P-set, P
6d.              ×  value
6e.              ×      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.

2.4.1.1.3 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:[11]

```
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_Anm_{n}:Anm_{n}
```

[11] This is a restriction of conventional RSL [21] – but is a construct of RSL$^+$text.

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[12]. 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.

2.4.1.2 TAX_DSU: An Endurant Part Taxonomy See Example A.4 on page 20. We refer to Appendix C on page 32. [We (also) refer to Carl von Linneaeus, the originator of the term Taxonomy[13].] 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.

7. 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.

7.	TAX_DSU	::=	E × **Taxonomy**
7a.		×	The taxonomy diagram resulting from 7a–7f.

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 $\xi_{Es} : \Xi_{Es}$.

2.4.1.3 EPV_DSU: Endurant Part Values: See Example A.3 on page 19.

The analyser cum describer decides to "apply" the EPV_DSU "prompt".[14] 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:

[12] See Sect. 2.4.2.5 on page 11.

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

[14] 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!

8. 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 = $\mathcal{E}_1$,
      - 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 18e page 15.

8.	EPV_DSU	::=	**Part Values**
8b.		×	**value**
8c.		×	e1:E1 = $\mathcal{E}_1$,
8c.		×	e2:E2 = $\mathcal{E}_2$,
8c.		×	...
8c.		×	em:Em = $\mathcal{E}_m$,
8d.		×	σ_{parts}
8e.		×	$\sigma_{atomic_{parts}}$
8f.		×	$\sigma_{behav_{parts}}$

The σ_{parts} is used by the uniqueness of part `dsu`, cf. Sect. 2.4.2.3 on the following page.

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 $\xi_{Es} : \Xi_{Es}$.

2.4.2 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.]

2.4.2.1 UID_DSU: Unique Identifiers

See Example A.5 on page 20.

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.

9. 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.

```
9.   UID_DSU  ::=
9a.                  E1, E2, ..., En × Unique Identification
9b.                  ×    type
9c.                  ×        E1I, E2I, ..., EnI
9d.                  ×    value
9e.                  ×        uid_E1: E1→E1I
9e.                  ×        uid_E2: E2→E2I
9e.                  ×        ...
9e.                  ×        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 $\xi_{Es} : \Xi_{Es}$.

2.4.2.2 UIDV_DSU: Unique Identifier Values:

See Example A.6 on page 21.

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. 2.4.1.3 on page 8 we introduced and defined the notion of endurant values and endurant value states (ξ_{Val} and $\sigma_{parts}, \sigma_{atomic_{parts}}$, etc.).

10. 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 8c on the previous page.
    (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.

```
10.   EPV_DSU  ::=  Unique Identifier Values.
10a.                 ×    value
10b.                 ×        e1_uid:E1I = uid_E1(e1),
10b.                 ×        e2_uid:E2I = uid_E2(e2),
10b.                 ×        ...
10b.                 ×        em_uid:EmI = uid_Em(em),
10c.                 ×        σ_uid_parts = { uid_E(p) | p:E ∈ σ_parts }
10d.                 ×        σ_uid_atomicparts = { uid_E(p) | p:E ∈ σ_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 $\xi_{Es} : \Xi_{Es}$.

2.4.2.3 UNIQ_DSU: Uniqueness of Parts

See Example A.7 on page 21.

As the result of the invocation of the EPV_DSU `dsu`, Sect. 2.4.1.3 on page 8, 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.

11. 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 $\sigma_{uid_{parts}}$,

11. UNIQ_DSU ::= **Uniqueness of Parts.**
11a. × **axiom**
11b. × **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 $\xi_{Es} : \Xi_{Es}$.

2.4.2.4 MER_DSU: Mereology. See Example A.8 on page 22.

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.

12. 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 9c– 9e on page 9),
    (c) the literal **value**; and
    (d) a list of corresponding mereology observers.

12. MER_DSU ::= **Mereologies**
12a. **type**
12b. × ME1 = $\mathcal{T}_1$(EI11,EI12,...,EI1n),
12b. × ME2 = $\mathcal{T}_2$(EI21,EI22,...,EI2n),
12b. × ...
12b. × MEm = $\mathcal{T}_m$(EIm1,EIm2,...,EImn)
12c. × **value**
12d. × **mereo_**E1: E1 → ME1,
12d. × **mereo_**E2: E2 → ME2,
12d. × ...
12d. × **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 $\xi_{Es} : \Xi_{Es}$.

2.4.2.5 ATTR_DSU: Attributes. See Example A.9 on page 22.

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.

13. 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.

13. ATTR_DSU ::= **Attributes**
13a. × **type**
13b. × $A1_1 = \mathcal{T}_{11}(T_{11_1},T_{11_2},...,T_{11_{m1}})$, ..., $A1_{m1} = \mathcal{T}_{1m1}(T_{1m1_1},T_{1m1_2},...,T_{1m1_{m1}})$
13b. × $A2_1 = \mathcal{T}_{21}(T_{21_1},T_{22_1},...,T_{21_{m2}})$, ..., $A2_{m2} = \mathcal{T}_{2m2}(T_{2m2_1},T_{2m2_2},...,T_{2m2_{m2}})$
13b. × ...
13b. × $An_1 = \mathcal{T}_{n1}(T_{n1_1},T_{n1_2},...,T_{n1_{m1n}})$, ..., $An_{mn} = \mathcal{T}_{nm}(T_{nm_1},T_{nm_2},...,T_{nm_{mnn}})$
13c. × **value**
13d. × **attr_**$A1_1$: E1 → $A1_1$
13d. × ...
13d. × **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 $\xi_{Es} : \Xi_{Es}$.

2.4.3 IPULL_DSU: Intentional Pull.

See Example A.10 on page 23.

14. 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.

14. IPULL_DSU ::=
14a. **Intentional Pulls**
14c. × **Name:** IPull_Name$_1$
14d. × text
14e. × **axiom** $\mathcal{A}_1(...)$
14c. × **Name:** IPull_Name$_2$
14d. × text
14e. × **axiom** $\mathcal{A}_2(...)$
14. × ...
14c. × **Name:** IPull_Name$_p$
14d. × text
14e. × **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 $\xi_{Es} : \Xi_{Es}$.

2.5 Perdurants

2.5.1 CH_DSU: Channel

See Example A.13 on page 26.

15. 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.

15a CH_DSU ::= **Channel Declaration**
15b × **channel**
15c × { ch[{$\{ui, uj\}$ | $\{ui, uj\} \in \sigma_{uid_{parts}}$]}: 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 $\xi_{Es} : \Xi_{Es}$.

2.5.2 BEH_DSU: Behaviours

See Example A.15 on page 26.

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!

16. 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:

16a CH_DSU ::= **Behaviours**
16b × **value**
16c × b_1: $UI_1 \to M_1 \to Sta_1 \to Mon_1 \to Prgr_1$ →**Unit**
16d × $b_1(ui_1)(mer_1)(sta_1)(mon_1)(prgr_1) \equiv \mathcal{B}_1(ui_1)(mer_1)(sta_1)(mon_1)(prgr_1)$ is
16d × ...
16c × b_2: $UI_2 \to M_2 \to Sta_2 \to Mon_2 \to Prgr_2$ →**Unit**
16d × $b_2(ui_2)(mer_2)(sta_2)(mon_2)(prgr_2) \equiv \mathcal{B}_2(ui_2)(mer_2)(sta_2)(mon_2)(prgr_2)$ is
16d × ...
16a × ...
16c × b_n:: $UI_n \to M_n \to Sta_n \to Mon_n \to Prgr_n$ →**Unit**
16d × $b_n(ui_n)(mer_n)(sta_n)(mon_n)(prgr_n) \equiv \mathcal{B}_n(ui_n)(mer_n)(sta_n)(mon_n)(prgr_n$ is)
16d × ...

We refer to Appendix Sect. B on page 29. 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:[15]

[15] Reference ι103a π29 designate Item 103a on page 29; et cetera.

```
value
ι103a π29.  B(bi)(mereo)(stat)(mon)(prg) ≡
ι103b π29.      pro_active_B(bai)(mereo)(stat)(mon)(prg)
ι103c π30.  ⫿ \ re_active_B(bai)(mereo)(stat)(mon)(prg)
```

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

```
value
ι104 π30.,ι103b π29.  pro_active_B(bi)(mereo)(stat)(mon)(prg) ≡
ι104a π30.              B_action_1(bi)(mereo)(stat)(mon)(prg)
ι104b π30.            ⫿ B_action_2(bi)(mereo)(stat)(mon)(prg)
ι104c π30.            ⫿ ...
ι104d π30.            ⫿ B_action_m(bi)(mereo)(stat)(mon)(prg)
```

17. . The `responding` behaviour ($\mathcal{B}$) 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
ι17a π14.,ι17 π14.  respond_B(bi)(mereo)(stat)(mon)(prg) ≡
ι17a π14.             let msg = [⫿] { comm[{bj,bi}] ?  | bj:BI • bj ∈ bis } in
ι17b π14.             react_behaviour_B(bi)(mereo)(stat)(mon)(prg)(msg) end
```

```
value
ι105 π30.  react_behaviour__B(bi)(mereo)(stat)(mon)(prg)(msg) ≡
ι107a π31.     is_action_i(msg) → B_action_i(bi)(mereo)(stat)(mon)(prg)(msg),
ι107b π31.     is_action_j(msg) → B_action_j(bi)(mereo)(stat)(mon)(prg)(msg),
ι107c π31.     ...,
ι107d π31.     is_action_k(msg) → B_action_k(bi)(mereo)(stat)(mon)(prg)(msg),
ι107e π31.     _ → 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 INIT_DSU: Initialisation

See Example A.16 on page 29.

18. 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.

```
18a.  INIT_DSU  ::=  Domain Initialisation
18b.                 domain_initialisation: Unit → Unit
18c.                 domain_initialisation() ≡
18d.                   ‖ { ℬ_i
18f.                       (uid_Ei(bi))
18g.                         (mereo_Ei(bi))
18h.                           (attr_StaA_p1(bi),...,attr_StaA_px(bi))
18i.                             (ηMonA_q1,...,ηMonA_qy)
18j.                               (attr_PrgA_r1(bi),...,attr_PrgA_rz(bi))
18e.                     | bi:Ei • bi ∈ σ_behav_parts ∧ ¬is_Part_set(bi) }
18k.                   ‖
18l.                   ‖ { ‖ { ℬ_p
18n.                           (uid_P(p))
18o.                             (mereo_P(p))
18p.                               (attr_StaA_p1(p),...,attr_StaA_px(p))
18q.                                 (ηMonA_q1,...,ηMonA_qy)
18r.                                   (attr_PrgA_r1(p),...,attr_PrgA_rz(p))
18m.                           | p:P • p∈obs_PS(ej) }
18e.                     | ej:Ei • ej ∈ σ_behav_parts ∧ is_Part_set(ej) }
```

Here $\sigma_{behav_{parts}}$ is defined in Item 8f page 9. 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 ?

Section 2, pages 3–15, presented, perhaps in a "chatty manner", a syntax for DDL.[16] With [17, *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 DDL syntax is required and that a tool support system implement that, but also that it provides interactive support for the domain analyser cum describers.

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.

[16] [16] presented that syntax 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 ! This document is one of a quadruplet, [14, 15, 16, 17], of documents that I worked on during the winter and spring of 2026.

References

[1] 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`

.

[2] 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.

[3] 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.

.

[4] 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.

[5] 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.

[6] 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.

[7] 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`.

[8] 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.

[9] Dines Bjørner. Domain Analysis & Description – Principles, Techniques and Modeling Languages. *ACM Trans. on Software Engineering and Methodology*, 28(2):66 pages, March 2019.

[10] Dines Bjørner. *Domain Science & Engineering – A Foundation for Software Development*. EATCS Monographs in Theoretical Computer Science. Springer, Heidelberg, Germany, January 2020. xii+346 pages[17].

[11] 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.

[12] 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.

[13] 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.

[14] 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 [16, 15, 17].

[15] 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 [16] and [17].

[16] 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 [15] and [17].

[17] 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 [16] and [15].

[18] Dines Bjørner. Domain Science & Engineering, a Primer[18] Technical University of Denmark. Significantly revised edition of [10]. xii+211 pages., February 2026 To be submitted [2026/2027].

[19] Dines Bjørner and Mikhail Chupilko. : . Translation of [18]. [Book], To be submitted [2026/2027].

[20] Dines Bjørner and Yang ShaoFa. 领域科学与工程导论. Translation of [18]. [Book], To be submitted [2026/2207].

[21] 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.

[22] 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.

[17] Due to copyright reasons no URL is given to this document's possible Internet location. A revised version of this book is [18]

[18] This book is currently being translated into Russian, [19], by Dr. Mikhail Chupilko and his colleagues, ISP/RAS (Institute of Systems Programming, Russian Academy of Sciences), Moscow and into Chinese, [20], by Dr. Yang ShaoFa, IoS/CAS (Institute of Software, Chinese Academy of Sciences), Beijing.

A Example

In this example we omit many of the keywords, etc., otherwise mentioned in Sects. 2.3–2.5.2, etc.

A.1 Universe of Discouse

19. The universe of discourse is that of
20. road traffic, RT –
21. whose narrative is then presented.

19. **universe of discourse**:
20. **type** RT
21. **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

22. In the road traffic domain we can observe a road net aggregate and an automobile aggregate.
23. In a road net aggregate we can observer an aggregate of hubs and an aggregate of links.
24. In an aggregate of automobiles we can observe a set of [atomic] automobiles.
25. Aggregates of hubs ad links are sets of [atomic] hubs and links.

type
22. RNA, AA
23. AH, AL
24. AS = A-**set**
25. HS = H-**set**, LS = L-**set**

value
22. **obs_**RNA: RT → RNA, **obs_**AA: RT → AA
23. **obs_**AH: RNA → AH, **obs_**AL: RNA → AL
24. **obs_**AS: AA → AS
25. **obs_**HS: AH → RS, **obs_**LS: AL → LS

A.3 Endurant Values

26. There is the road transport [universe of discourse].
27. There is the road net aggregate.
28. There is the automobile aggregate.
29. There is the hub aggregate.
30. There is the link aggregate.
31. There is the set of automobiles.

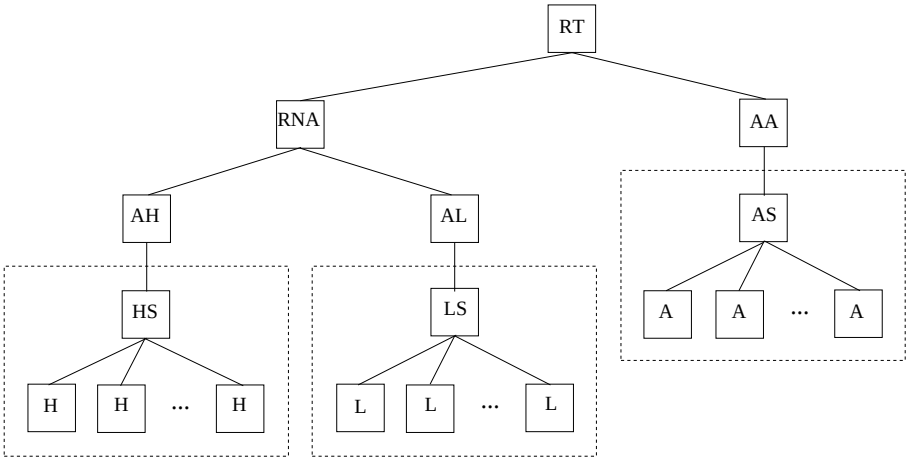

Figure 2: A Road Transport Taxonomy

32. There is the set of hubs.
33. There is the set of links.
34. And there is the entire set of all of these.
35. And there is the entire set of just all automobiles, hubs and links.

value
26. rt:RT
27. rna:RNA = **obs_**RNA(rt)
28. aa:AA = **obs_**AA(rt)
29. ah:AH = **obs_**AA(rna)
30. al:AL = **obs_**AL(rna)
31. as:AS = **obs_**AS(ah)
32. hs:HS = **obs_**HS(ah)
33. ls:LS = **obs_**LS(al)
34. σ_{parts} = {rt,rna,aa,ah,al} ∪ aa ∪ as ∪ hs ∪ ls
35. 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:

36. The universe of discourse has a unique identifier.
37. The road net agregate has a unique identifier.
38. The automobile aggregate has a unique identifier.
39. The hub aggregate has a unique identifier.
40. The link aggregate has a unique identifier.
41. Each hub has a unique identifier.

42. Each link has a unique identifier.

43. Each automobile has a unique identifier.

type
36. RTI
37. RNAI
38. AAI
39. HAI
40. LAI
41. HI
42. LI
43. AI

value
36. **uid_**RT: RT → RTI
37. **uid_**RNA: RNA → RNAI
38. **uid_**AA: AA → AAI
39. **uid_**HA: HA → HAI
40. **uid_**LA: LA → LAI
41. **uid_**H: H → HI
42. **uid_**L: L → LI
43. **uid_**A: A → AI .

A.6 Unique Identifier Values

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

45. There is the unique identifier of the road net aggregate.

46. There is the unique identifier of the autombile aggregate.

47. There is the unique identifier of the hub aggregate.

48. There is the unique identifier of the link aggregate.

49. There are the unique identifiers of the automobiles of the set of autombiles.

50. There are the unique identifiers of the hubs of the set of hubs.

51. There are the unique identifiers of the links of the set of links.

52. And there is the entire set of all of these.

53. And there is the entire set of all identifiers of the automobiles, hubs and links.

value
44. rt_{uid}RT: RT = **uid_**RT(rt)
45. rna_{uid}RNA: RNA = **uid_**RNA(rt)
46. aa_{uid}AA: AA = **uid_**AA(rt)
47. ah_{uid}AH: AH = **uid_**AA(rna)
48. al_{uid}AL: AL = **uid_**AL(rna)
49. as_{uid}AS: AS = {**uid_**A(a)|a:A•a∈as}
50. hs_{uid}:HS = {**uid_**H(h)|h:H•h∈hs}
51. ls_{uid}:LS = {**uid_**L(l)|l:L•a∈ls}
52. $\sigma_{parts_{uid}}$ = {rt,rna,aa,ah,al} ∪ aa ∪ hs unon ls
53. $\sigma_{atoms_{uids}}$ = aa ∪ hs ∪ ls

A.7 Uniqueness of Endurants

54. Parts are uniquely identfied.

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

A.8 Mereology

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

55. The mereology of links is a 2 element set of hub identifiers.
56. The mereology of a hub is a possibly empty set of hub identifiers.
57. The mereology of an automobile is a set of hub and link identifiers

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

value
55. **mereo_**L: L → ML
56. **mereo_**H: H → MH
57. **mereo_**A: A → MA .

A.9 Attributes

Example attributes are:

- **Hubs:**
  58. 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.
  59. Hubs have state spaces, hω:HΩ: the set of hub states "signaling" which states are open/closed, i.e., **green**/**red**.
- **Links:**
  60. Links that have lengths, LEN;
  61. 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.
  62. Links have state spaces, lω:LΩ: the set of link states "signaling" which states are open/closed, i.e., **green**/**red**.
- **Automobiles:**
  63. 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.
  64. Automobiles have velocity;
  65. Automobiles have acceleration;
  66. Etc.[19]
- **Hubs, Links, Automobiles:**

[19]

67. 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.
68. Link positions have well-defined identifiers and fractions.

type
58. HΣ = (LI×LI)**-set** [**prg**]
59. HΩ = HΣ**-set** [**sta**]
60. LEN = **Nat** [**sta**] [m]
61. LΣ = (HI×HI)**-set** [**prg**]
62. LΩ = LΣ**-set** [**sta**]
63. APos = atH | onL [**prg**]
63a. atH :: HI
63b. onL :: LI × (fhi:HI × f:**Real** × thi:HI)
64. Veloc = **Real** [**mon**] [km/h]
65. Accel = **Real** [**mon**] [m/sec]
66. ...
67. HHis,LHis = ($\mathbb{TIME}$×AI)* [**prg**]
67. AHis = ($\mathbb{TIME}$×(HI|LI))* [**prg**]

value
58. attr_HΣ: H → HΣ
59. attr_HΩ: H → HΩ
60. attr_LEN: L → LEN
61. attr_LΣ: L → LΣ
62. attr_LΩ: L → LΩ
63. attr_APos: A → APos
64. attr_Veloc: A → Veloc
65. attr_Accel: A → Accel
67. attr_HHis: H → HHis
67. attr_LHis: L → LHis
67. attr_AHis: A → AHis

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

A.10 Intentional Pull

69. 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.
70. 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
69a. ∀ a:A • a ∈ as ⇒
69a. **let** ahist = attr_AHist(a) **in**
69a. ∀ ui:(LI|HI) • ui ∈ **dom** ahist ⇒
69b. ∀ τ:$\mathbb{TIME}$ • τ ∈ **elems** ahist(ui) ⇒
69b. **let** hist = is_LI(ui) → attr_LHist(retr_L(ui))(σ),
69b. _ → attr_HHist(retr_H(ui))(σ) **in**
69b. τ ∈ **elems** hist(uid_A(a)) **end end**
70. ∧
70c. ∀ u:(L|H) • u ∈ ls∪hs ⇒
70c. **let** uhist = attr(L|H)Hist(u) **in**
70d. ∀ ai:AI • ai ∈ **dom** uhist ⇒
70d. ∀ τ:$\mathbb{TIME}$ • τ ∈ **elems** uhist(ai) ⇒
70d. **let** ahist = attr_AHist(retr_A(ai))(σ) **in**
70d. τ ∈ **elems** uhist(ai) **end end**

A.11 Auxiliary Types

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

71. A path element identifier is either a link identifier or a hub identifier.
72. A path (of a road net) is a finite[20] sequence of one or more alternating hub and link identifiers
73. 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.
74. Given a hub [a link] identifier we can retrieve the identified hub [link].

```
type
71. PEI = LI | HI
72. Path = PEI*
73. axiom [Well-formed Paths]
72.   ∀ path:Path •
73a.     ∀ {i,i+1}⊆inds path ⇒
73a.       ( (is_HI(path[i])∧is_LI(path[i+1]) ∨ is_LI(path[i])∧is_HI(path[i+1]))
73b.       ∧ (path[i]∈hs_uis⇒path[i+1]∈ls_uis
73c.         ∧ uid_H(retr_hub(path[i]))∈mereo_L(retr_hub(path[i])))
73d.       ∧ (path[i]∈ls_uis⇒path[i+1]∈hs_uis
73e.         ∧ uid_L(retr_link(path[i])∈mereo_H(retr_link(path[i]))))  )
value
74. retr_hub: HI → H, retr_link: LI → L, retr_unit: UI → U
74. retr_hub(hi) as h • h ∈ hs ∧ uid_H(h)=hi
74. retr_link(li) as l • l ∈ ls ∧ uid_L(l)=li
74. retr_unit(ui) as u • u ∈ hs∪ls ∧ uid_U(u)=ui
74. 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.

75. The paths[21] 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.

    We define the paths function in two ways.

76. Either axiomatically

[20] We shall only consider finite paths. The `paths` function, Item 75 below, can easily be modified to yield also infinite length paths!

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

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

78. or inductively[22]:

    (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
75. paths: Unit → Path-infset
76. paths() as ps
77.     such that: ∀ p:ps satisfy the wellformedness of Sect. A.11 on the preceding page
```

We can also express the `paths` functions explicitly:

```
value
75. paths: Unit → Path-infset
78. paths() ≡
78a.    let ps = {⟨ni⟩ | ni:NI ∈ hs_uis}∪{⟨ei⟩ | ei:EI ∈ ls_uis}
78(b)iv.        ∪ { pi⌢⟨ui⟩⌢⟨uj⟩⌢pj | pi⌢⟨ui⟩:Path-set, ⟨uj⟩⌢pj:Path-infset •
78b.              ∧ ({pi⌢⟨ui⟩,⟨uj⟩⌢pj}⊆ps)
78(b)i.             ∧ (ui∼ ∈ elems pj ∧ uj∼ ∈ elems pi)
78(b)iii.            ∧ (ui ∈ mereo_U(retr_unit(uj)))
78(b)ii.             ∧ (uj ∈ mereo_U(retr_unit(ui)))} in
78c.    ps end
type
75. U = H|L
```

Solution to the equation, lines 78a–78(b)i, is "'obtained' by a smallest set fix-point reasoning.

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

```
value
79. 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.

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

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

theorem: [All finite paths have finite reverse paths]
80. ∀ g:G,p:Path•p ∈ paths(g) ⇒ rev_path(p) ∈ paths(g)
value
80. rev_path: P → P
80. rev_path(p) ≡
80. **case** p **of**
80. ⟨⟩ → ⟨⟩,
80. ⟨ui⟩ → ⟨ui⟩,
80. ⟨ui⟩⌢p′⌢⟨uj⟩ → ⟨uj⟩⌢rev_path(p′)⌢⟨ui⟩
80. **end**

We can define further functions. For example:

81. `path_length`,
82. `shortest_path`, etc.

value
81. path_length: P → LEN
81. path_length(p) ≡
81. **case** p **of**
81. ⟨⟩ → 0
81. ⟨ui⟩ → retr_path_length(ui),
81. ⟨ui⟩⌢p′ → retr_length(ui)+path_path_length(p′)
81. **end**
81. retr_path_length: UI → LEN
81. retr_path_length(ui) ≡ (is_EI(ui)→**attr_**LEN(retr_link(ui)),is_NI(ui)→0)

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

A.13 Channel

83. There is a set of channels between hubs, links and automobiles.
84. These channels communicate messages, `M`. `M` will "transpire" frm the behaviour definitions.

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

A.14 Variables

A.15 Behaviours

There are three behaviours:

85. `automobile`, corresponding to endurants `a:A`, and with appropriate argument types,
86. `hub`, corresponding to endurants `h:H`, and with appropriate argument types, and

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

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

88. 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.

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

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

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

91. 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**.

91 automobile(ai)(ris)(...)(atH(hi),ahis) ≡
91a automobile_remain_at_hub(ai)(ris)(...)(atH(hi),ahis)
91b ⌈⌉
91c automobile_leaving_hub(ai)(ris)(...)(atH(hi),ahis)
91d ⌈⌉
91e automobile_stop(ai)(ris)(...)(atH(hi),ahis)

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

92. [91a] The automobile **remains** at a hub:
    (a) time is recorded,
    (b) informing the hub behaviour, whereupon
    (c) the automobile remains at that hub, "idling",

92 automobile_remain_at_hub(ai)(ris)(...)(atH(hi),ahis) ≡
92a **let** τ = **record_**𝕋𝕀𝕄𝔼() **in**
92b ch[{ai,hi}] ! τ ;
92c automobile(ai)(ris)(...)(atH(hi),⟨(τ,hi)⟩^ahis) **end**

93. [91c] 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.

```
93  automobile_leaving_b(ai)({li}∪ris)(...)(atH(hi),ahis) ≡
93a     let τ = record_𝕋𝕀𝕄𝔼() in
93b     (ch[ {ai,hi} ] ! τ ‖ ch[ {ai,li} ] ! τ) ;
93c     automobile(ai)(ris)(...)(onL(li,(hi,0,_)),⟨(τ,li)⟩^ahis) end
93      pre: [hub is not isolated]
```

94. [91e] Or the automobile **stop**s, "disappears — off the radar" !

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

95. 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 (!)].

```
95  automobile(ai)(ris)(...)(onL(li(fhi,f,thi)),lhis) ≡
95a     automobile_remains_on_link(ai)(ris)(...)(onL(li(fhi,f,thi)),lhis)
95b   ⌈⌉
95c     automobile_leaving_linl(ai)(ris)(...)(onL(li(fhi,f,thi)),lhis)
95d   ⌈⌉
95e     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`.

96. Hubs
97. external non-deterministically receives time-stamped, t, messages,ai, from automobiles as to their entering, remaining or leaving the hub.
98. They update their hub history accordingly and resume being a hub.

```
value
96. hub(hi)(hm)(hω,...)(hσ,hhist) ≡
97.     let (t,ai) = ⌊⌈ { ch[ {hi,ai} ] ? | ai ∈ hm } in
98.     hub(hi)(hm)(hω,...)(hσ,⟨(t,ai)⟩^hhist) end
```

99. Links
100. external non-deterministically receives time-stamped, t, messages,ai, from automobiles as to their entering, remaining or leaving the link.
101. They update their link history accordingly and resume being a link.

```
value
99.  link(li)(lm)(len,lω,...)(lσ,lhist) ≡
100.     let (t,ai) = ⌊⌈ { ch[ {li,ai} ] ? | ai ∈ lm } in
101.     link(li)(lm)(len,lω,...)(lσ,⟨(t,ai)⟩^lhist) end
```

A.16 Initialization

102. 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
102. initialisation: Unit → Unit
102. initialisation() ≡
102a.    ‖ { hub(uid_H(h))
102a.              (mereo_H(h))
102a.                 (attr_HΩ(h),...)
102a.                    (attr_HΣ(h),attr_HΣ(h),attr_HHist(h))
102a.      | h:H • h ∈ hs }
102b.    ‖
102c.    ‖ { link(uid_L(l))
102c.              (mereo_L(l))
102c.                 (attr_LEN(l),...)
102c.                    (attr_LΣ(l),attr_LHist(l))
102c.      | l:L • l ∈ ls }
102d.    ‖
102e.    ‖ { automobile(uid_A(a))
102e.              (mereo_A(a))
102e.                 (attr_APos(a),attr_AHist(a))
102e.      | 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 following page.

Figure B.1 on the next page should be understood as follows:[23] 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 following page is now given a more precise "meaning" – with this "meaning" suggesting a general "pattern" for behaviour definitions:

103. There are behaviours `B, ...` with identities `bi, ...` .
    (a) These behaviours,typically, have the form of internal, ⌈⌉, non-deterministically "choosing" between
    (b) *pro-actively* initiating communications with other behaviors

[23] 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.

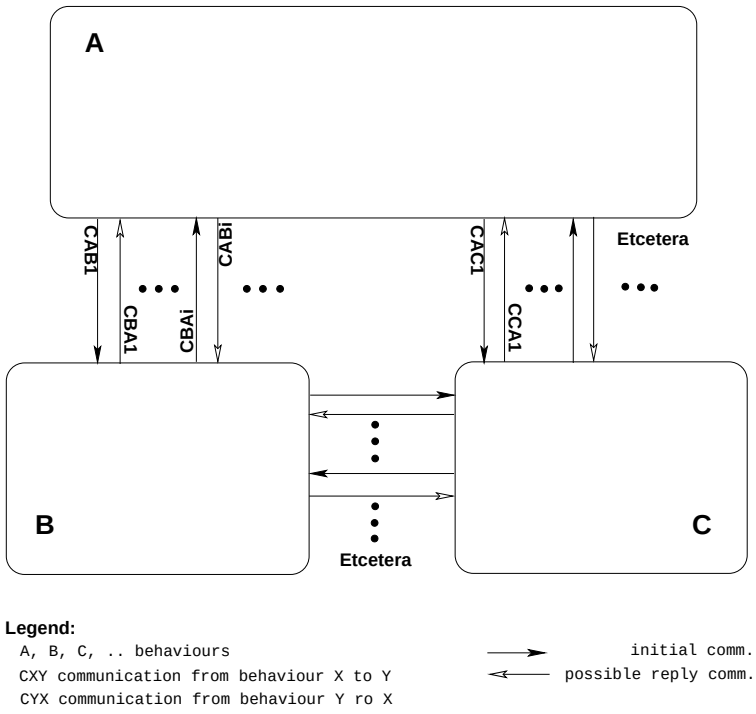

Figure 3: Communicating Behaviours

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

value
103a. B(bi)(mereo)(stat)(mon)(prg) ≡
103b. pro_active_B(bai)(mereo)(stat)(mon)(prg)
103c. ⌈⌉ re_active_B(bai)(mereo)(stat)(mon)(prg)

104. 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.

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

value
104. pro_active_B(bi)(mereo)(stat)(mon)(prg) ≡
104a. B_action_1(bi)(mereo)(stat)(mon)(prg)
104b. ⌈⌉ B_action_2(bi)(mereo)(stat)(mon)(prg)
104c. ⌈⌉ ...
104d. ⌈⌉ B_action_m(bi)(mereo)(stat)(mon)(prg)
where:
105. B_action_i(bi)(mereo)(stat)(mon)(prg) ≡
105. **let** prgr′ = Action(bi)(mereo)(stat)(mon)(prg) **in**
105. B(bi)(mereo)(stat)(mon)(prg′) **end**

106. 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
106.  re_active_B(bi)(mereo)(stat)(mon)(prg) ≡
106a.     let msg = ⫿{ comm[{bj,bi}]?  | bj:BI • bj ∈ bis } in
106b.     respond_behaviour_B(bi)(mereo)(stat)(mon)(prg)(msg) end
```

107. 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
107.  react_behaviour_B(bi)(mereo)(stat)(mon)(prg)(msg) ≡
107a.     is_action_i(msg) → B_action_i(bi)(mereo)(stat)(mon)(prg)(msg),
107b.     is_action_j(msg) → B_action_j(bi)(mereo)(stat)(mon)(prg)(msg),
107c.     ...,
107d.     is_action_k(msg) → B_action_k(bi)(mereo)(stat)(mon)(prg)(msg),
107e.     _ → 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.

108. 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.

```
108.  action_i(bi)(mereo)(stat)(mon)(prg) ≡
108a.     let v = evaluate_i(bi)(mereo)(stat)(mon)(prg) in
108b.     let (bj,prg′) = elaborate_i(v)(bi)(mereo)(stat)(mon)(prg) in
108c.     comm[{bi,bj}] ! ℰ(prg′) ;
108d.     behaviour(bi)(mereo)(stat)(mon)(prg′)
108.      end end
```

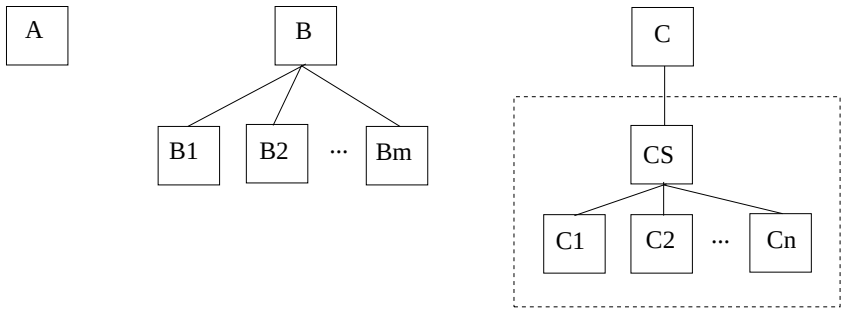

Figure 4: Taxonomy Elements

Variants of Item $\iota 108c\,\pi 31$ are also used:

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

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

• • •

I refer to a recent paper [22].

C Taxonomy

The Internet[24] describes taxonomy as follows: *Taxonomy is the science of defining, naming, and classifying organisms into hierarchical groups (taxa) based on shared characteristics and evolutionary relationships.* It uses a standardized system of ranks: *Domain, Kingdom, Phylum, Class, Order, Family, Genus, Species* to organize life, often utilizing binomial nomenclature for unique identification[25]

We shall instantiate this description into: A domain taxonomy is syntactically a tree structure – to reflect a conceived hierarchy. Tree nodes are here shown as square boxes. Its descendant nodes are of three kinds:

A: A single, suitably labeled, A, square box.

B: A square, suitably labeled square ("root") box, B, with m suitably labeled (B1, B2, ..., Bm) descendants shown as square boxes connected to the root box by simple lines.

C: A square, suitably labeled square ("root") box, C, with a subordinate labeled box, CS, which again has n suitably labeled (C1, C2, ..., Cn) square boxes. Appropriate boxes connected by simple lines.

Figure 4 illustrates these three cases:
The dashed line around the CS, C1, C2, ..., Cn sub-tree are usually omitted in our taxonomy diagrams.

A [full] domain taxonomy now displays

- The universe of discourse as a root node.
- It is "just" a node depicting an endurant.

[24] https://www.google.com/search?client=ubuntu-sn&channel=fs&q=Taxonomy

[25] This characterization, in terms of *Domain, Kingdom, Phylum, Class, Order, Family, Genus, Species* reveals the origin of the term 'taxonomy', namely the Swedish biologist Carl Linnaeus `https://en.wikipedia.org/wiki/Carl_Linnaeus`.

- If an endurant is atomic, A, then that endurant is just a square box.
- If the endurant is Cartesian, then that endurant is as the B tree with Bi being elements of the Cartesian.
- If the endurant is a part set, then that endurant is as the C tree with CS being a sort name for the set {C1,C2,...,Cn}.

Taxonomies are not ontologies.

- Ontology is *the branch of metaphysics dealing with the nature of being – a set of concepts and categories in a subject area or domain that shows their properties and the* ***relations*** *between them.*

We refer to Appendix A.4 on page 20 for an example.