Domain Modelling
A Foundation for Software Development

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

December 31, 2022: 09:03 am

Abstract
Domain modelling, as per the approach of this paper, offers the possibility of describing software application domains in a precise and comprehensive manner – well before requirements capture can take place. We endow domain modelling with appropriate analysis and description calculi and a systematic method for constructing domain models. The present paper is a latest exposé of the domain science & engineering as published in earlier papers and a book. It reports on our most recent simplifications to the domain analysis & description approach.

1 Introduction
This paper introduces the possibility of a new phase of software development, one that precedes requirements engineering, as well as a new way of looking at the world around us!

Today’s well-managed software development projects usually start with some form of requirements capture. Now the possibility arises to precede this phase of requirements engineering with an initial phase of domain engineering.

The present paper is an improvement over previously published accounts: builds upon a simpler domain ontology (Fig. 1 on page 4); has fewer domain concepts (Sects. 3 and 5); and presents a more rational way of “deriving” behaviours from parts (Sect. 6). Taken together the presentation is thus made shorter and more precise.

The approach to the modelling of domains put forward in this paper has two major phases: modelling external qualities of the world as we see it, as it manifests itself to us, or otherwise, and modelling the internal qualities, as we may not see it, but qualities that can be measured and/or spoken about. The modelling of external qualities has a few steps. The major step of modelling of external qualities is that of deciding upon the atomic-, Cartesian- and set-oriented parts. A minor step, following the major step, is that of identifying a notion of endurant state. The modelling of internal qualities has a few more steps. The modelling of unique identifiers; the modelling of mereologies; the modelling of attributes; and the modelling of ’intentional pull’. It is this structuring into manageable stages and steps that reassures us, i.e., me, that the approach is sound.
1.1 What is a Domain?

By a domain we shall understand a rationally describable segment of a discrete dynamics fragment of a human assisted reality, i.e., of the world: its endurants, i.e., solid and fluid entities: whether natural (“God-given”) or artefactual (“man-made”), their parts and living species entities: whether atomic or compound parts, respectively whether plant or animal living species, including humans — as well as its perdurants: the behaviours of parts and living species.

Clearly this characterisation does not possess the rigour that should be common in software development. Terms such as rationally describable, discrete dynamics and human assisted reality must be not just assumed, but must, below, be made more precise. Yet “ultimate” precision defies us: The domains we shall study, analyse and describe are not amenable to such precision. The world is not formal.

Thus the domain analysis & description methodology that we shall be concerned with is not directed at continuous dynamics systems such as we find them in for example aerospace applications. And we shall not, in this paper be concerned with the human assistance aspects.

By domain modelling we mean the study, analysis and description of a domain.

If the domain already exists, then the modelling amounts to a faithful rendering of that domain – involving no creative design – but such that the resulting model, i.e., description, “covers” as wide a spectrum of domain instances as is deemed reasonable.

If the domain does not already exists, then the modelling may involve creative design. We shall, in this paper, assume already existing domains.

By domain engineering we mean the construction of domain models.

1.2 Non-computable and Computable Specifications

When specifying software we usually make use of a formal language – one whose semantics can be expressed mathematically. And the specification had better be computable. Similarly for prescribing requirements: again a formal language can be deployed. And the specification had better be computable. Typically, when we derive a software specification, \( S \), from a requirements prescription, \( R \), the testing, model checking and proof of some form of correctness, \( D,S \models R \), of the software design relies on not only on relations between the two documents: the \( R \) and \( S \), but also on the domain description, \( D \). But in describing domains we cannot assume computability. It is the task of requirements engineering to “derive” computable requirements from domain models. Chapter 9 shows how. We refer to Sect. 7.2.3 on page 20 for summary comments.

1.3 Formal Method and Methodology

By a method we shall understand a set of principles for selecting and applying a number of procedures, techniques and tools for [effectively] constructing an artefact. By methodology we shall understand the study and knowledge of one or more methods. By a formal method we shall understand a method which uses one or more formal specification languages as per their intention: specification and verification (formal tests, model checks and proofs of properties of domains descriptions, requirement prescriptions and software designs. By a formal specification language we shall understand a language with a formal syntax, a formal semantics and a proof system with which

---

1 The term ‘design’ is used here in the sense of artistic design – such as used when expressing something being, for example, of ‘modern design’. “Philosophers seek to find the inescapable characteristic of any world. Scientists seek to determine how our world actually is and our situation in it. Artists seek to create objects for our experience. That is, what is necessary, real, respectively possible” [53] Sørlandet.

2 Thus a railway domain model should desirably cover such instances as the railways of Denmark and Norway and Sweden, each one individually as well as their combination.

3 [22, 2021], while using a different tool-oriented, proof, check and test approach to domain modelling, sets up a domain model for automobile assembly lines [see also [13]] and uses satisfiability modulo theory tools to fine-tune the layout of the automobile assembly line wrt. a number of optimality criteria.

4 The approach taken here can, however, also be used to “device” new domains.
Domain Modelling

to describe & validate domains, prescribe & validate requirements and specify (design) & validate software.

Our domain analysis & description method has been developed, over the years, with this understanding of formal methods.

1.4 From Programming Languages to Domains

Domain stakeholders, those whose primary work is in and of the domain, name the entities of the domain and use these names, nouns and verbs, in communicating with other stakeholders. These utterings constitute a language, albeit an informal one. In a domain model we give abstract syntax to (roughly speaking) the nouns, Sects. 3 and 5, and semantics to (roughly speaking) the verbs, Sect. 6.

When, in comparison, we define the syntax and semantics of a programming language, that syntax and semantics covers all well-formed instances of programs in that language. Similarly, when, in consequence, we define the abstract syntax and semantics, i.e., a model, of a domain, that syntax and semantics covers all well-formed instances — we mean it, the model, to cover all well-formed instances of domains.

1.5 A Review

We present a latest exposé of the domain science & engineering of [11, 14, 15, 2015–2021]. The first inklings of this applied science were first reported in [1, 1995–1997], Volume III, Part IV, Chapters 8–12, Pages 193–362 of [2, 2006] cover several aspects of domain engineering — but not what we now consider the most important contribution to the field: namely that of the analysis & description calculi. First developments of the proposed analysis and description calculi were reported in [7, 8, Kyiv 2010]. The recently published papers and book [11, 14, 15, 2015–2021] illustrates the fact that the details of the calculi may change. The present paper reports on our most recent simplification to the domain analysis & description approach and the few extensions, RSL+, to the RSL specification language [29]. The domain modelling approach presented here has been honed over the last 30 years in numerous experiments. Some of these are reported in [16, 13, 17, 18].

1.6 An Overview

1.6.1 A Domain Analysis & Description Ontology

Sections 3–6 represent the contribution of this paper. Figure 1 on the following page illustrates basic ideas of how we shall structure our domain analysis & description.

The domain analyser cum describer is confronted by a domain. How and where to start!

Figure 1 on the next page is intended to be read top-down, left-to-right. So it suggests that the domain analyser cum describer starts by looking “at the whole domain!” — call it φ. That is, at the • right under the term Universes, between the r and the s!

1.6.2 Step-wise Analysis and Description

Figure then suggests, by the two lines emerging from that •, that the domain analyser cum describer poses the question, of the domain, is it (more or less) rationally describable, i.e., is entity(φ), or not. If the domain analyser cum describer decides yes, it is so, then the analysis “moves” on to the Entity •. Now the question is, is the entity being observed, an endurant or a perdurant, (to be explained below), and so on. We now assume that the analysis proceeds along the left hand side dashed line (···- - - - - -) box labeled ‘Endurants’.

The so-called external quality analysis of endurants ends when reaching either of the Atomic, Cartesian or Part Set •. At this point the description proceeds to that of the internal qualities

---

5test, check and verify
6We shall, in this paper, not exemplify living species endurants.
of endurants. From Fig. [1] You observe seven vertical dashed lines, emanating downwards from endurant bullets to cross three horizontal (bottom of the figure) lines. They “call” for the domain analyser cum describer to now analyse and describe the internal qualities of endurants: their unique identification, their mereologies, and their attributes.

Then the domain analyser cum describer has “traversed” the left hand side of Fig. [1]. At this point a so-called transcendental deduction takes place: The domain analyser cum describer now “morphs” manifest endurant parts into behaviours. The focal point here are the part behaviour signatures and definitions. Figure [1]‘s right hand side hints at the issues to be covered and that the internal qualities are being a crucial element of behaviour definitions.

1.6.3 The Analysis and Description Prompts

Each ⋆ of Fig. [1] thus corresponds to an analysis or description prompt. There are two kinds of analysis prompts. Both are informal. The predicate analysis prompts – with 18 such prompts, and the function analysis prompts. There is two major kinds of description prompts. (α) external quality description prompts – with there being two such specific prompts: one for describing so-called Cartesian endurants (Sect. 3.4.1 on page 10), another for describing so-called Part Set endurants (Sect. 3.4.2 on page 10), and (β) internal quality description prompts with there being three such specific prompts: the unique identifier description prompt (Sect. 5.1.1 on page 13), the mereology description prompt (Sect. 5.2.1 on page 14), and the attribute description prompt (Sect. 5.3.2 on page 15). The predicate analysis prompts yield truth values. The function analysis prompts yield part endurants and the names of their type – which we shall call sorts. And the description prompts yield domain description texts – here in a slight extended version of the RAISE7

7Rigorous Approach to Industrial Software Engineering
1.7 RSL, RSL-text and RSL⁺

RSL is described in [29]. We use a subset of that RSL. Thus we shall not avail ourselves of the RSL module concepts of object, class and scheme. Basically, then, a specification expressed in RSL amounts to sequences of [alternating] type, value and axiom clauses – with, basically, a single channel clause:

\[
\begin{array}{cccc}
\text{type} & \text{type} & \text{channel} & \text{type} \\
\ldots & \ldots & \ldots & \ldots \\
\text{value} & \text{value} & \text{type} & \text{value} \\
\ldots & \ldots & \ldots & \ldots \\
\text{axiom} & \text{axiom} & \text{value} & \text{axiom} \\
\ldots & \ldots & \ldots & \ldots \\
\end{array}
\]

RSL-text is an addition to RSL. In describing domains in RSL we shall be introducing description prompts which are informal functions which yield values of type RSL-text, that is, proper RSL texts. Quoting an RSL text: “text”. shall denote an RSL-text.

RSL⁺ designate RSL-text plus, in this paper, one extension. That extension is that of the type and values of type names. If Τ denotes a type, i.e., a possibly infinite set of values, then ηΤ denotes a value, the name of type Τ, with φΤ denoting the type of type names.

The domain analysis & description method is informally explained in a mixture of English and RSL⁺. [10, 2014] attempts a formalisation of an early version of RSL⁺.

1.8 A Computer Science Philosophy

We shall base our domain analysis & description approach on the philosophy of Kai Sørlander. The issue here is: In studying, analysing & describing domains one is confronted with the basic [metaphysical] question[s]: which are the absolutely necessary conditions for describing any world ?, that is: what, if anything, is of such necessity, that it could under no circumstances be otherwise ?, or: which are the necessary characteristics of any possible world ? In his work Sørlander rationally argues that space, time, Newton’s laws, and a number of additional concepts are necessarily basic elements of any description of any domain.

1.9 Previous Work

We refer to Sect. 1.5 on page 3.

Axel van Lamsweerde and Michael A. Jackson, as well as other requirements engineering researchers, do touch upon the issues of domains – such as that term is basically used here. But their requirements analysis and prescription “refer” to; they do not “put it center stage”, let alone mandate that the requirements engineer rely on an a priori established domain description. So they and others do not establish, as is the main focus of this contribution, calculi for the analysis & description of domains.

1.10 Structure of Paper

There are basically two parts to this paper. The main part consists of Sects. 3 and 5–6. They present a terse, comprehensive exposé of the domain analysis & description method of this paper.

---

8RSL: RAISE Specification Language
9Other formal specification languages are possible, f.ex.: VDM, Z, Alloy, or CafeOBJ.
An appendix, the other part, Appendix A brings an example. For the domain modelling approach to be believable the example must open up for a realistic domain, one that is not “small”.

We now explain the domain description ontology as a structured set of concepts for modelling domains, a set that shows their properties and the relations between them. In simple terms, ontology seeks the classification and explanation of entities. Figure 1 on page 4 is a graphical rendition of a structured set of concepts for modelling domains.

2 Universe of Discourse

Domain descriptions start with a terse sketch of the main facets of the domain followed by the naming of the domain.

<table>
<thead>
<tr>
<th>Narration:</th>
<th>Formalisation:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text</td>
<td>type UoD</td>
</tr>
</tbody>
</table>

1 Example. Universe of Discourse: We refer to Sect. A.1 on page 28

3 External and Internal Qualities

Characterisation 1: External qualities: External qualities of endurants of a domain are, in a simplifying sense, those properties of endurants that we can see, touch and which have spatial extent. They, so to speak, take form.

Characterisation 2: Internal qualities: Internal qualities of endurants of a domain are, in a less simplifying sense, those which we may not be able to see or “feel” when touching an endurant, but they can, as we now ‘mandate’ them, be reasoned about, as for unique identifiers and mereologies, or be measured by some physical/chemical means, or be “spoken of” by intentional deduction, and be reasoned about, as we do when we attribute and intentional pull properties to endurants.

3.1 Predicate Analysis of External Qualities of Endurants

Characterisation 3: Phenomenon: By a phenomenon we shall understand a fact that is observed to exist or happen. Examples of phenomena are: emotions of a human, the rivers, lakes, forests, mountains and valleys of mother nature; the railway tracks, their units, the locomotive of a railway system.

Domain Analysis Predicates: We shall define a number of domain analysis predicates. They are all referred to as prompts. Prompts are method tools. The domain analyser cum describer applies these to “real”, i.e., actual world phenomena, that is, not to formal values. In the next 18 paragraphs we shall “reveal” a number of such predicates. First with a reasonable definition (in slanted font), then with examples and some comments (in roman font).

Predicate Prompt 1: is_entity: By an entity we shall understand a phenomenon, i.e., something that can be observed, i.e., be seen or touched by humans, or that can be conceived as an
abstraction of an entity; alternatively, a phenomenon is an entity, if it exists, it is “being”, it is
that which makes a “thing” what it is: essence, essential nature [46, Vol. I, pg. 665]■ Some, but
not necessarily all aspects of a river can be rationally described, hence can be still be considered
types. Similarly, many aspects of a road net can be rationally described, hence will be considered
types. Domain endurants, when eventually modelled in software, typically become data. Hence the careful analysis
of domain endurants is a prerequisite for subsequent careful conception and analyses of data
structures for software, including data bases.

Predicate Prompt 2: is_endurant: Endurants are those quantities of domains that we can observe (see and touch), in space, as “complete” entities at no matter which point in time – “material” entities that persists, endures [46, Vol. I, pg. 656]■ Street segments [links], street intersections [hubs], automobiles standing still in an automobile show room are endurants. Domain endurants, when eventually modelled in software, typically become data. Hence the careful analysis of domain endurants is a prerequisite for subsequent careful conception and analyses of data structures for software, including data bases.

Predicate Prompt 3: is_perdurant: By a perdurant we shall understand an entity for which only a fragment exists if we look at or touch them at any given snapshot in time. Were we to freeze time we would only see or touch a fragment of the perdurant [46, Vol. II, pg. 1552]■ Automobiles in action, container vessels sailing on the 7 seas and loading and unloading containers in harbours are examples of perdurants. Domain perdurants, when eventually modelled in software, typically become processes.

Endurants are either solid endurants, or are fluid endurants.

Predicate Prompt 4: is_solid: By a solid endurant we shall understand an endurant which is separate, individual or distinct in form or concept, or, rephrasing: a body or magnitude of three-dimensional, having length, breadth and thickness [46, Vol. II, pg. 2046]■ Wells, pipes, valves, pumps, forks, joins, regulator, and sinks. of a pipeline are solids.

Predicate Prompt 5: is_fluid: By a fluid endurant we shall understand an endurant which is prolonged, without interruption, in an unbroken series or pattern; or, rephrasing: a substance (liquid, gas or plasma) having the property of flowing, consisting of particles that move among themselves [46, Vol. I, pg. 774]■ Fluids are otherwise liquid, or gaseous, or plasmatic, or granular[14] or plant products[15] et cetera. Specific examples of fluids are: water, oil, gas, compressed air, etc. A container, which we consider a solid endurant, may be conjoined with another, a fluid, like a gas pipeline unit may “contain” gas.

We analyse endurants into either of two kinds: parts and living species. The distinction between parts and living species is motivated in Kai Sørlander’s Philosophy [52, 53, 54, 55, 56].

Predicate Prompt 6: is_part: By a part we shall understand a solid endurant existing in time and space and subject to laws of physics, including the causality principle and gravitational pull [16]

Natural and man-made parts are either atomic or compound.

Predicate Prompt 7: is_atomic: By an atomic part we shall understand a part which the domain analyser considers to be indivisible in the sense of not meaningfully, for the purposes of the domain under consideration, that is, to not meaningfully consist of sub-parts.■ The wells, pumps, valves, pipes, forks, joins and sinks of a pipeline can be considered atomic.

Predicate Prompt 8: is_compound: Compound parts are those which are either Cartesian-product- or are set-oriented parts.

Predicate Prompt 9: is_Cartesian: Cartesian parts are those (compound parts) which consists of an "indefinite number" of two or more parts of distinctly named sorts.■ Some clarification may be needed. (i) In mathematics, as in RSL [29], a value is a Cartesian (“record”) value if it can be expressed, for example as (a, b, ..., c), where a, b, ..., c are mathematical (or, which is the same, RSL) values. Let the sort names of these be A, B, ..., C – with these being required to be distinct. We wrote “indefinite number”: the meaning being that the number is fixed, finite, but not specific. (ii) The requirement: ‘distinctly named’ is pragmatic. If the domain analyser cum

---

14 This is a purely pragmatic decision. “Of course” sand, gravel, soil, etc., are not fluids, but for our modelling purposes it is convenient to “compartmentalise” them as fluids!
15 I.e., chopped sugar cane, threshed, or otherwise. See footnote 14
16 This characterisation is the result of our study of relations between philosophy and computing science, notably influenced by Kai Sørlander’s Philosophy.
describer thinks that two or more of the components of a Cartesian part [really] are of the same sort, then that person is most likely confused and must come up with suitably distinct sort names for these “same sort” parts! (iii) Why did we not write “definite number”? Well, at the time of first analysing a Cartesian part, the domain analyser cum describer may not have thought of all the consequences, i.e., analysed, the compound part. Additional sub-parts, of the Cartesian compound, may be “discovered”, subsequently and can then, with the approach we are taking wrt. the modelling of these, be “freely” added subsequently! We refer to the road transport system example above. We there viewed (hubs, links and) automobiles as atomic parts. From another point of view we shall here understand automobiles as Cartesian parts: the engine train, the chassis, the car body, four doors (left front, left rear, right front, right rear), and the wheels. These may again be considered Cartesian parts.

**Predicate Prompt 10: is_part_set:** Part sets are those which, in a given context, are deemed to meaningfully consist of an indefinite number of sub-parts of the same sort. Examples of set parts are: the set of hubs of a road net hub aggregate, the set of links of a road net link aggregate, and the set of automobiles of an automobile aggregate – all of the road net transport that we are exemplifying.

**Predicate Prompt 11: is_living_species:** By a living species we shall understand a solid endurant, subject to laws of physics, and additionally subject to causality of purpose. Living species must have some form they can be developed to reach; a form they must be causally determined to maintain. This development and maintenance must further engage in exchanges of matter with an environment.

It must be possible that living species occur in two forms: plants, respectively animals. Although we have not yet come across domains for which the need to model the living species of plants were needed, we give some examples anyway: grass, tulip, rhododendron, oak tree. Similar for animals: dogs, cat, cows, butterflies, cod (fish), etc. Hence:

**Predicate Prompt 12: is_plant:** Plants are living species which are characterised by development, form and exchange of matters with the environment.

**Predicate Prompt 13: is_animal:** Animals are living species which are additionally characterised by the ability of purposeful movement.

Within animals we then have humans.

**Predicate Prompt 14: is_human:** A human (a person) is an animal, with the additional properties of having language, being conscious of having knowledge (of its own situation), and responsibility.

**Characterisation 4:** Manifest Part: By a manifest part we shall understand a part which ‘manifests’ itself either in a physical, visible manner, “occupying” an area or a volume and a position in space, or in a conceptual manner forms an organisation in Your mind! As we have already revealed, endurant parts can be transcendentally deduced into perdurant behaviours – with manifest parts indeed being so.

**Predicate Prompt 15: is_manifest:** is_manifest(e) holds if e is manifest.

**Characterisation 5:** Structure: By a structure we shall understand an endurant concept that allows the domain analyser cum describer to rationally decompose a domain analysis and/or its description into manageable, logically relevant sections, but where these abstract endurants are not further reflected upon in the domain analysis and description. Structures are therefore not transcendentally deduced into perdurant behaviours.

**Predicate Prompt 16: is_structure:** is_structure(e) holds if e is a structure.

**Predicate Prompt 17: is_stationary:** An endurant part is stationary if it never changes position in space.

**Predicate Prompt 18: is_mobile:** An endurant part is mobile if it may possibly change position in space.

We may need, occasionally, the distinction as now outline: Endurants are either natural endurants, or are artefactual endurants.

**Predicate Prompt 19: is_natural:** By a natural endurant we shall understand one which has been created by nature.
Predicate Prompt 20: *is* artefactual: By an artefactual endurant we shall understand one which has been created by humans.

Discrete Dynamic and Artefactual Domains: In our initial characterisation of domains, Page 2 an emphasis was put on their discrete dynamics and human assistedness. The analysis and description calculi and, hence, our domain modelling, are therefore “geared” in that direction.

We summarise:

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>is_entity: $\Phi \rightarrow \text{Bool}$</td>
<td>is_living_species: $E \rightarrow \text{Bool}$</td>
</tr>
<tr>
<td>is_endurants: $E \rightarrow \text{Bool}$</td>
<td>is_human: $E \rightarrow \text{Bool}$</td>
</tr>
<tr>
<td>is_perdurant: $E \rightarrow \text{Bool}$</td>
<td>is_atomic: $E \rightarrow \text{Bool}$</td>
</tr>
<tr>
<td>is_solid: $E \rightarrow \text{Bool}$</td>
<td>is_compound: $E \rightarrow \text{Bool}$</td>
</tr>
<tr>
<td>is_fluid: $E \rightarrow \text{Bool}$</td>
<td>is_structure: $E \rightarrow \text{Bool}$</td>
</tr>
<tr>
<td>is_part: $E \rightarrow \text{Bool}$</td>
<td>is_cartesian: $E \rightarrow \text{Bool}$</td>
</tr>
<tr>
<td>is_living_species: $E \rightarrow \text{Bool}$</td>
<td>is_atomic: $E \rightarrow \text{Bool}$</td>
</tr>
<tr>
<td>is_compound: $E \rightarrow \text{Bool}$</td>
<td>is_structure: $E \rightarrow \text{Bool}$</td>
</tr>
<tr>
<td>is_solid: $E \rightarrow \text{Bool}$</td>
<td>is_cartesian: $E \rightarrow \text{Bool}$</td>
</tr>
<tr>
<td>is_fluid: $E \rightarrow \text{Bool}$</td>
<td>is_part_set: $E \rightarrow \text{Bool}$</td>
</tr>
<tr>
<td>is_part: $E \rightarrow \text{Bool}$</td>
<td>is_artefactual: $E \rightarrow \text{Bool}$</td>
</tr>
</tbody>
</table>

2 Example. Analysis Predicates: In the example of Appendix A on page 28 we do not [explicitly] show the “application” of analysis predicates. They are tacitly assumed.

3.2 On Interpreting the Analysis & Description Ontology

We interpret the kind of analysis & description methodology ontology diagrams of which Fig. 1 on page 4 is an example. The figure to the right illustrates a fragment of such diagrams. At node $A$, i.e., observing an endurant, $a$, of sort $A$, the downward diverging two lines express that $a$ is either of sort $B$ or of sort $C$; that is:

$$\text{pre: } \text{is}_B(a), \text{is}_C(a); \text{is}_A(a)$$

$$\text{is}_B(a) \Rightarrow \sim \text{is}_C(a) \land \sim \text{is}_A(a)$$

We summarise:

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>is_B</td>
<td>is_C</td>
</tr>
<tr>
<td>is_A</td>
<td></td>
</tr>
</tbody>
</table>

3.3 Functional Analysis of External Qualities of Endurants

Given a compound endurant, that is, either a Cartesian or a part set, we analyse that compound, at the two $\leftrightarrow$’s of Fig. 1 on page 4 into its constituent endurants, respectively parts, and the name of the sort:

3. determine Cartesian parts, determine part set

The above calculation function signatures and characterisations illustrate two extensions to RSL [29]: $\eta P$ expresses the name of a sort $P$, and $\eta \Phi$ expresses the type of sort names.

Again we emphasize that these calculations are performed by the domain analyser cum describer. They are used in subsequent schemas for describing external qualities of endurants.

$^{17}$Framed texts highlight domain analysis & description prompts.
3.4 Descriptions of External Qualities of Endurants

Similarly, again at the two ▶’s of Fig. 1 on page 4, we are now ready to describe respectively Cartesian parts and part set parts.

3.4.1 Describing Cartesian Parts

value

descr_Cartesian: P \rightarrow \text{RSL-Text}
descr_Cartesian(p) \equiv

"Narrative:
\[ s \] text on sorts
\[ o \] text on observers
\[ a \] text on axioms and/or proof obligations

Formalisation:
\[ s \] type
E_1, E_2, ..., E_n
\[ o \] value
obs_{E_1}: E_1 \rightarrow E_1, obs_{E_2}: E_2 \rightarrow E_2, ..., E_c \rightarrow E_c
\[ a \] axiom and/or proof obligation
A/P(...)"

3 Example. Cartesians: We refer to Sect. A.2.1 on page 28.

3.4.2 Describing Part Sets

value

descr_part-set: P \rightarrow \text{RSL-Text}
descr_part-set(p) \equiv

"Narrative:
\[ s \] text on sorts
\[ o \] text on observers
\[ a \] text on axioms and/or proof obligations

Formalisation:
\[ s \] type
P, Ps = P-set
\[ o \] value
obs_{Ps}: E \rightarrow Ps
\[ a \] axiom and/or proof obligation
A/P(...)"

4 Example. Part Sets: We refer to Sect. A.2.2 on page 28.

3.5 Endurant States
Characterisation 6: Endurant State: By an endurant state we shall understand any collection of endurant parts.

\[ \text{value } \Sigma = \text{P-set} \]

\[ \text{gen}_{\Sigma} : E \rightarrow \Sigma \]

\[ \text{gen}_{\Sigma}(e) \equiv \]

- if is_manifest(e) then
  - is_atom(e) \rightarrow \{e\},
  - is_Cartesian(node) \rightarrow
    let (p_1, p_2, ..., p_c) = \text{calc_Cartesian_parts_and_sorts}(e) in
    \{p_1, p_2, ..., p_c\} \cup \text{gen}_{\Sigma}(p_1) \cup \text{gen}_{\Sigma}(p_2) \cup \ldots \cup \text{gen}_{\Sigma}(p_c) \text{ end}
- is_part_set(e) \rightarrow
  let (\{p_1, p_2, ..., p_s\}, node) = \text{calc_part_sets_parts_and_sort}(e) in
  \{p_1, p_2, ..., p_s\} \cup \text{gen}_{\Sigma}(p_1) \cup \text{gen}_{\Sigma}(p_2) \cup \ldots \cup \text{gen}_{\Sigma}(p_s) \text{ end}
  \]

else \{\} end

5 Example. Endurant State Examples: We refer to Sect. A.2.3 on page 29.

3.6 A Proof-theoretic Explication, I

The concept of analysis predicates and part observer functions is due to McCarthy [50, Sect.12-13]. In [50] McCarthy introduces a notion of abstract syntax, Sect. 12, and semantics, Sect. 13. So far we have dealt, in our domain analysis, with syntax. There are three elements, according to McCarthy, to consider: the is_... predicates, the obs_... [“destructor”] functions, and, not shown, so far, in this paper, the mk_... constructor functions. For compound abstract syntactic entities they are related as follows:

\[ \text{is}_\text{Cartesian}(p) \equiv \]

- let (p_1, p_2, ..., p_c) = \text{calc_Cartesian_parts_and_sorts}(p) in
  \[ p = \text{mk}_\text{Cartesian}(\text{obs}_{P_1}(p), \text{obs}_{P_2}(p), \ldots, \text{obs}_{P_c}(p)) \text{ end} \]

\[ \text{is}_\text{part_set}(p) \equiv \]

- let (\{p_1, p_2, ..., p_s\}, node) = \text{calc_part_sets_parts_and_sort}(p) in
  \[ p = \text{mk}_\text{part_set}(\{p_1, p_2, ..., p_s\}) \text{ end} \]

The mk_... constructors were not introduced above. The reason is simple; a pragmatic decision: As the domain analyser does it in their work they may, when encountering Cartesian compounds, be free to leave some components (of the Cartesian) out, components that they may later introduce. So really, the first of the identities above ought be expressed as

\[ \text{is}_\text{Cartesian}(p) \equiv \]

- let (p_1, p_2, ..., p_c) = \text{calc_Cartesian_parts_and_sorts}(p) in
  \[ p = \text{mk}_\text{Cartesian}(\text{obs}_{P_1}(p), \text{obs}_{P_2}(p), \ldots, \text{obs}_{P_c}(p)) \text{ end} \]

We continue this explication in Sect. 5.5 on page 16.
4  Space and Time

The concepts of space and time can be *transcendentally deduced*, by rational reasoning, as has been shown in [52, 53, 54, 55, 56, Kai Sørlander], from the facts of *symmetry, asymmetry, transitivity* and *intransitivity* relations.

*They are therefore facts of every possible universe.*

4.1 Space

There is one given space. As a type we name it \( \text{SPACE} \). We do not bother, here, about textual representation of spatial locations, but here is an example that would work in or near this globe we call our earth: \( \text{Latitude 55.805600, Longitude 12.448160, Altitude 35m} \)\(^{18}\).

Also, in this paper, we do not present models of \( \text{SPACE} \). But we do introduce such notions as (i) \( \text{POINT} \): as \( \text{SPACE} \) being some dense and infinite collection of points; (ii) \( \text{LOCATION} \): as the location in space of some point;

\( \text{value record}\_\text{LOCATION}: \text{E} \rightarrow \text{LOCATION} \)

(iii) \( \text{CURVE} \): as an infinite collection of points forming a mathematical curve – having a (finite or infinite) \textbf{length}; (iv) \( \text{SURFACE} \): as an infinite collection of points forming a mathematical surface – having a (finite or infinite) \textbf{area}; and (v) \( \text{VOLUME} \): as an infinite collection of points forming a mathematical volume – having a (finite or infinite) \textbf{volume}. We suggest it, as a domain science & engineering research topic, that somebody studies a \textit{calculus or calculi of spatial modelling}.

4.2 Time

There is one given space. As a type we name it \( \text{TIME} \). We do not bother, here, about textual representation of time, but here is an example: \( \text{December 31, 2022: 09:03am} \)\(^{19}\). But we do introduce such crucial notions as \textit{time interval} \( \text{T} \) and operations on \( \text{TIME} \) and \( \text{T} \):

\( \text{value} \)

\[ -: \text{TIME} \times \text{TIME} \rightarrow \text{T} \]

\[ +: \text{TIME} \times \text{T} \rightarrow \text{TIME} \]

\[ *: \text{Real} \times \text{T} \rightarrow \text{T} \]

A crucial time-related operation is that of \textit{record\_TIME}. It applies to “nothing”: \textit{record\_TIME()} and yields \( \text{TIME} \).

\( \text{value record\_TIME: Unit} \rightarrow \text{TIME} \)

5  Internal Qualities

We refer to the \textit{Internal Qualities} characterisation on Page\(^{8}\). We can justify the grouping of internal endurant qualities into three kinds: \textit{unique identifiers}, cf. Sect. 5.1 \textit{mereologies}, cf. Sect. 5.2 and \textit{attributes}, cf. Sect. 5.3. To this we add the concept of \textit{intentional pull}, cf. Sect. 5.4

5.1 Unique Identification

On the basis of \textit{philosophical reasoning}, within \textit{metaphysics}, we [can] argue that parts are uniquely identifiable \([52, 53, 54, 55, 56, \text{Kai Sørlander}]\).
5.1.1 Calculate Unique Identifiers

<table>
<thead>
<tr>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>descr_unique_identifier: P → RSL-Text</td>
</tr>
<tr>
<td>descr_unique_identifier(p) ≜</td>
</tr>
<tr>
<td>&quot; Narrative:</td>
</tr>
<tr>
<td>[s] text on unique identifier sort</td>
</tr>
<tr>
<td>[o] text on unique identifier observer</td>
</tr>
<tr>
<td>[a] text on axioms and/or proof obligations</td>
</tr>
<tr>
<td>Formalisation:</td>
</tr>
<tr>
<td>[s] type</td>
</tr>
<tr>
<td>PI</td>
</tr>
<tr>
<td>[o] value</td>
</tr>
<tr>
<td>uid_P: P → PI</td>
</tr>
<tr>
<td>[a] axiom and/or proof obligation</td>
</tr>
<tr>
<td>A/P(...) &quot;</td>
</tr>
</tbody>
</table>

6 Example. Unique Identifiers: We refer to Sect. A.3.1 on page 29

5.1.2 Endurant Identifier States

Given the endurant state values, for the whole domain or for respective, manifest part sorts, one can define corresponding unique identifier values.

7 Example. Unique Identifier State: We refer to Sect. A.3.2 on page 29

5.1.3 Axioms

The number of manifest parts is the same as the number of manifest part unique identifiers.

8 Example. Unique Identifier Axiom: We refer to Sect. A.3.3 on page 30

5.1.4 Endurant Retrieval

Given a unique identifier, \( \pi \), of a manifest part, \( p \), of an endurant state, \( \sigma \), of a domain one can retrieve that part:

<table>
<thead>
<tr>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \sigma: \Sigma = \text{gen}_\Sigma(\text{uod}) )</td>
</tr>
<tr>
<td>retr_P: ( \Pi \rightarrow \Sigma \rightarrow P )</td>
</tr>
<tr>
<td>retr_P(( \pi ))(( \sigma )) ≜ let ( p:P \cdot p \in \sigma \land \text{uid}_P(p)=\pi ) in ( p ) end</td>
</tr>
</tbody>
</table>

5.2 Mereology

Mereology is the study and knowledge of parts and part relations. It was first put forward, around 1916, by the Polish logician Stanisław Leśniewski [48, 23].

Which are the relations that can be relevant for “endurant-hood”? There are basically two relations: (i) physical ones, and (ii) conceptual ones. (i) Physically two or more endurants may be topologically either adjacent to one another, like rails of a line, or within an endurant, like links and hubs of a road net, or an atomic part is conjoined to one or more fluids, or a fluid is conjoined to one or more parts. The latter two could also be considered conceptual “adjacencies.”
(ii) Conceptually some parts, like automobiles, “belong” to an embedding endurant, like to an automobile club, or are registered in the local department of vehicles, or are intended to drive on roads.

5.2.1 Calculate Mereologies

<table>
<thead>
<tr>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>descr.mereology: P → RSL-Text</td>
</tr>
<tr>
<td>descr.mereology(p) ≡</td>
</tr>
<tr>
<td>&quot;Narrative:</td>
</tr>
<tr>
<td>[s] text on mereology type</td>
</tr>
<tr>
<td>[o] text on mereology observer</td>
</tr>
<tr>
<td>[a] text on axioms and/or proof obligations</td>
</tr>
<tr>
<td>Formalisation:</td>
</tr>
<tr>
<td>[s] type MT = M(p)</td>
</tr>
<tr>
<td>[o] value mereo_P: P → MT</td>
</tr>
<tr>
<td>[a] axiom and/or proof obligation A/P(...) &quot;</td>
</tr>
</tbody>
</table>

$M(p)$ is usually a type expression over unique identifiers of mereology-related parts.

9 Example. Mereology: We refer to Sect. A.4 on page 30.

Given the definition of external qualities of a domain, and its unique identifier and mereology internal qualities one can analyse and describe many properties of that domain. The routes subsection (Page 31) of the mereology example, Example 9 illustrates one such property.

5.3 Attributes

Parts and fluids are typically recognised because of their spatial form and are otherwise characterised by their intangible, but measurable attributes. That is, whereas endurants, whether solid (as are parts) or fluids, are physical, tangible, in the sense of being spatial [or being abstractions, i.e., concepts, of spatial endurants], attributes are intangible: cannot normally be touched, or seen, but can be objectively measured. Thus, in our quest for describing domains where humans play an active rôle, we rule out subjective “attributes”: feelings, sentiments, moods. Thus we shall abstain, in our domain science also from matters of psychology and aesthetics.

5.3.1 Functional Analysis of Attributes

Given a manifest part, $p$, that is, either an atom, or a Cartesian, or a part set, we calculate from that part, its constituent attributes values and types:

<table>
<thead>
<tr>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>determine_attributes: P → (a1×ηA1) × (a2×ηA2) × ... × (aa×ηAa)</td>
</tr>
</tbody>
</table>
5.3.2 Describe Attributes

10. descr_attributes

value
descr_attributes: P → RSL-Text
let ((\_ηA1),(\_ηA2),...,(\_ηAa)) = determine_attributes(p:P) in
descr_attributes(p) ≡

"Narrative:

[s] text on attribute types
[o] text on attribute observers
[a] text on axioms and/or proof obligations

Formalisation:

[s] type
A1 [ = ... ], A2 [ = ... ], ..., Aa [ = ... ],
[o] value
attr_A1: P → A1, attr_A2: P → A2, ..., attr_Aa: P → Aa,
[a] axiom and/or proof obligation
A/P(...)"

The domain analyser cum describer has thus determined/decided that A1, A2, ..., Aa are the “interesting” attributes of of parts of sort P. Attributes are often given a “concrete” form, hence the [ = ... ] where the ... is some type expression.

10 Example. Attributes: We refer to Sect. A.5 on page 31.

5.3.3 Attribute Categories

Michael A. Jackson has proposed a structure of attributes [41].

Attribute Category 1: Static: By a static attribute we shall understand an attribute whose values are constants, i.e., cannot change.

Attribute Category 2: Dynamic: By a dynamic attribute we shall understand an attribute whose values are variable, i.e., can change. Dynamic attributes are either inert, reactive or active attributes.

Attribute Category 3: Inert: By an inert attribute we shall understand a dynamic attribute whose values only change as the result of external stimuli where these stimuli prescribe new values.

Attribute Category 4: Reactive: By a reactive attribute we shall understand a dynamic attribute whose values, if they vary, change in response to external stimuli, where these stimuli either come from outside the domain of interest or from other endurants.

Attribute Category 5: Active: By an active attribute we shall understand a dynamic attribute whose values change (also) of its own volition. Active attributes are either autonomous, or biddable or programmable attributes.

Attribute Category 6: Autonomous: By an autonomous attribute we shall understand a dynamic active attribute whose values change only “on their own volition”. The values of an autonomous attributes are a “law onto themselves and their surroundings”.

Attribute Category 7: Biddable: By a biddable attribute we shall understand a dynamic active attribute whose values are prescribed but may fail to be observed as such.

Attribute Category 8: Programmable: By a programmable attribute we shall understand a dynamic active attribute whose values can be prescribed.

We modify Jackson’s categorisation. This is done in preparation for our exposé of behaviour signatures, cf. Sect. 6.4.1 on page 19. Figure 2 on the following page shows groupings of some of M. A. Jackson’s six basic categories.
Instead of M. A. Jackson’s six basic categories () we shall, as indicated in Fig. 2, make use of the combined attribute categories of monitorable and monitorable only attributes.

5.4 Intentional Pull

5.4.1 Characterisations

Intentionality as a philosophical concept is defined by the Stanford Encyclopedia of Philosophy\(^{20}\) as “the power of minds to be about, to represent, or to stand for, things, properties and states of affairs.”

Intent is then a usually clearly formulated or planned intention. An example of intent is that of roads made for automobiles and automobiles meant for roads.

Intentional Pull\(^{21}\): Two or more artefactual parts of different sorts, but with overlapping sets of intents may exert an intentional “pull” on one another. This intentional “pull” may take many forms. Let \( p_x : X \) and \( p_y : Y \) be two parts of different sorts \((X, Y)\), and with common intent, \( i \). Manifestations of these, their common intent, must somehow be subject to constraints, and these must be expressed predicatively. When a composite or conjoin artefact models “itself” as put together with a number of other endurants then it does have an intentionality and the components’ individual intentionalities does, i.e., shall relate to that. The composite road transport system has intentionality of the road serving the automobile part, and the automobiles have the intent of being served by the roads, across “a divide”, and vice versa, the roads of serving the automobiles.

11 Example. Intentional Pull: Road Transport: We refer to Sect. A.6 on page 33.

12 Example. Double-entry Bookkeeping: Double-entry bookkeeping, also known as double-entry accounting, is a method of bookkeeping that relies on a two-sided accounting entry to maintain financial information. Every entry to an account requires a corresponding and opposite entry to a different account. The double-entry system has two equal and corresponding sides known as debit and credit. A transaction in double-entry bookkeeping always affects at least two accounts, always includes at least one debit and one credit, and always has total debits and total credits that are equal\(^{22}\).

5.5 A Proof-theoretic Explication, II

We remind You of Sect. 3.6 on page 11

With the introduction of analysis functions and observers for unique identifiers, mereology and attributes we can now augment the is..., uid..., mereo..., attr_A... observers introduced since Page[11]


\(^{21}\) The term intentional pull is chosen so as to connote with the term gravitational pull.

\(^{22}\) https://en.wikipedia.org/wiki/Double-entry_bookkeeping

© Dines Bjørner. December 31, 2022: 09:03 am
is\_manifest(p:P) \equiv 
\text{let } ((\_\_\_\eta A1) , (\_\_\_\eta A2) , (\_\_\_\eta Aa)) = \text{calc\_attributes}(p) \text{ in}
\text{p} = \text{mk\_P} (\text{uid\_P}(p) , \text{mero\_P}(p) , (\text{attr\_A1}(p) , \text{attr\_A2}(p) , ... , \text{attr\_Aa}(p))) \text{ end}

6 Perdurants

A key point of our domain science & engineering approach is this: to every manifest part we transcendentially deduce a unique behaviour.

By transcendental we shall understand the philosophical notion: the a priori or intuitive basis of knowledge, independent of experience.

By a transcendental deduction we shall understand the philosophical notion: a transcendental ‘conversion’ of one kind of knowledge into a seemingly different kind of knowledge.

6.1 Channels

Part behaviours may communicate with one another. To express behaviours and their communication we use Hoare’s CSP [35, 36, 37]. One may question this choice. In [5, 8, 12, 2009–2017] we show “that to every mereology there is a CSP expression”. On that background we maintain that CSP is a reasonable choice — but invite the reader to suggest more appropriate mechanisms for handling behaviours and their communication.

So, in general, we declare a RSL/CSP channel:

\begin{verbatim}
11. channel declaration

channel { ch[\{ui,uj\}] | ui,uj:UI • \{ui,uj\} \subseteq uis } : M
\end{verbatim}

Here \text{ch} is the name of the indexed array of channels and the indexes are, in general, any two element set of unique part identifiers. \text{M} is the type of the messages communicate between behaviours of index \text{ui,uj}.

6.2 Actors

By an action we shall understand either an action, or an event, or a behaviour.

6.2.1 Actions

By an action of a behaviour we shall understand something which is local to a behaviour, and, when applied, potentially changes the state. Generally action clauses are expressed in RSL [29].

6.2.2 Events

By an event of a behaviour we shall understand something that involves two behaviours, and, when applied, potentially changes the state of both behaviours. Event clauses are expressed using the CSP elements of RSL. That is, the CSP output “!” and input events “?”:

\text{ch[\{ui,uj\}]} ! \text{expr}
\text{let } \text{val} = \text{ch[\{ui,uj\}]} ? ... \text{end}

13 Example. Road Transport Actions an Events: We refer to Sect. A.7.2 on page 35.

\text{23} Please bear in mind that the use, here, of CSP, is in the following context: the CSP clauses are not to be “interpreted” on a computer where this “computerisation” has to be “shared” with other computations; hence CSP synchronisation & communication is “ideal” and reflects reality.
6.3 State Access and Updates

We need define two functionals: one for changing the mereology of a part and another for changing the attribute value of a part. We therefore informally define the following functionals:

6.3.1 Update Mereologies

- **part_update_mereology** is a functional: it takes the following arguments: a part \( p \) of type \( P \) and a mereology value and yields a part of type \( P \).
- The yielded result, \( p' \), has the same unique identifier, as the argument part \( p \),
- a new, the argument, mereology, as the argument part \( p \),
- and the same attribute values for all attributes, as the argument part \( p \).

\[
\text{value} \quad \text{part_update_mereology}: P \rightarrow M \rightarrow P
\]
\[
\text{part_update}(p)(m) \equiv \\
\text{let } ((\_\_\eta A_1),(\_\_\eta A_2),...,(\_\_\eta A_a)) = \text{determine_attributes}(p) \text{ in} \\
\text{let } p':P \cdot \text{uid}_P(p')=\text{uid}_P(p)\wedge \text{mereo}_P(p')=m \wedge \\
\forall \_\_\eta A:\_\_\Phi \cdot \eta A\in\{\eta A_1,\eta A_2,....,\eta A_a\} \Rightarrow \text{attr}_A(p')=\text{attr}_A(p) \text{ in} \\
p' \text{ end end}
\]

6.3.2 Update Attributes

- **part_update_attribute** is a functional: it takes the following arguments: a part \( p \) of type \( P \) and a pair of an attribute name and value, and yields a part \( p' \) of type \( P \).
- The argument attribute name must be that of an attribute of the part.
- The yielded result \( p' \) has the same unique identifier and mereology as the argument part \( p \),
- and the same attribute values for all attributes, as the argument part \( p \), except for argument attribute (name) for which it now yields the argument attribute value.

\[
\text{value} \quad \text{part_update_attribute}: P \rightarrow \Phi A \times A \rightarrow P
\]
\[
\text{part_update_attribute}(p)(\eta A,a) \equiv \\
\text{let } ((\_\_\eta A_1),(\_\_\eta A_2),...,(\_\_\eta A_a)) = \text{determine_attributes}(p) \text{ in} \\
\text{assert: } \eta A\in\{\eta A_1,\eta A_2,....,\eta A_a\} \\
\text{let } p':P \cdot \text{uid}_P(p')=\text{uid}_P(p)\wedge \text{mereo}_P(p')=\text{mereo}_P(p) \wedge \\
\forall \_\_\eta A:\_\_\Phi \cdot \eta A\in\{\eta A_1,\eta A_2,....,\eta A_a\} \setminus \eta A \Rightarrow \text{attr}_A(p')=\text{attr}_A(p) \text{ in} \\
p' \text{ end end}
\]

Monitorable attributes usually change their values surreptitiously. That is, “behind the back”, so-to-speak, of the part behaviour.

6.4 Behaviours

By a **behaviour** we shall understand a set of sequences of actions, events and behaviours.
6.4.1 Behaviour Signatures:

We now come to a crucial point in our unrolling the *domain science & engineering method*. It is that of explaining the signature of behaviours, that is, the arguments ascribed to part behaviours. The general form of part \( p \) behaviour signatures is as follows.

\[
\text{value} \\
p\_\text{behaviour} : p : P \rightarrow \text{in,out} \{ \text{ch[} \{ \text{uid}_P(p),ui\} ] | \text{ui:UI} \} \cup \{ \text{Mereo}(p) \} \quad \text{Unit}
\]

Yes, that is it! The behaviour of a[ny] (manifest) part, \( p \), is a function whose only argument is that part! The signature informs of the channels that \( p\_\text{behaviour} \) may communicate with. The literal \text{Unit} informs that the behaviour may not yield any value, but, for example, go on “forever” having possibly effected a state change!

6.4.2 Behaviour Definitions:

Behaviours, besides their signatures, are defined. That is, a *behaviour definition ‘body’* describes, in, for us, using RSL [29] with its embodiment of a variant of CSP [37], basically CSP clauses how it interacts with other behaviours, and, in basically RSL’s functional specification (read: programming) clauses, how it otherwise “goes about its business”!

In fragment I the focus is on the possible [action] update of either biddable or programmable attributes.

\[
\text{p\_behaviour}(p) \equiv \\
\text{let } p' = \text{possible_update_of_biddable_and_programmable_attributes}(p) \text{ in} \\
p\_\text{behaviour}(p') \quad \text{end}
\]

In fragment II the focus is on the possible [action] value access to any attributes.

\[
p\_\text{behaviour}(p) \equiv \ldots \text{attr}_A(p) \ldots p\_\text{behaviour}(p)
\]

In fragment III the focus is on the possible interaction with other behaviours, hence illustrates two events as seen from one behaviour.

\[
p\_\text{behaviour}(p) \equiv \\
\text{let } (\text{val,ui}) = \mathcal{E}(p) \text{ in ch[} \{ \text{uid}_P(p),ui\} ] ! \text{val} \quad \text{end} ; \\
\ldots \\
\text{let } ui = \mathcal{I}(p) \text{ in let } (\text{val}',ui) = \text{ch[} \{ \text{uid}_P(p),ui\} ] ? \text{ in} \\
\ldots \\
p\_\text{behaviour}(p) \quad \text{end end end}
\]

14 Example. Road Transport Behaviour Definitions: *We refer to Sect. A.7.4 on page 35*
6.5 Domain Initialisation

By domain initialisation we mean the “start-up” of a behaviour for all manifest parts.

15 Example. Road Transport Domain Initialisation: We refer to Sect. A.8 on page 38.

6.6 End of Domain Modelling Presentation

This concludes the four sections, Sects. 2, 3, 4 and 6, on domain modelling.

7 Closing

7.1 The Current Calculi

The treatment of behaviours of Sect. 6.4.2 differs very much from that of Sects. 7.6–7.7 of [15]. The present one is very short, but results in a repeated use of the part_update functional. Our domain modelling approach allows a wide spectrum, in-between these behaviour signature and definition styles, for expressing behaviours. What remains fixed in the treatment of endurants: both of their external qualities, and of their internal qualities.

7.2 Some Issues

A number of issues need be addressed.

7.2.1 A New View of Software Development?

Yes, we [somewhat immodestly] claim that this paper presents a new view of software development! Aircraft designers and manufacturers employ professionally educated aeronautics engineers having state-of-the-art insight into aerodynamics. But, we claim, software companies do not, today, exhibit the same professionalism in their staffing. Software for health care (hospitals, etc.) are often developed by programmers with no previous professional insight into that area. Likewise for domains such as law, public administration, health care and tax administration. With sound methods for “deriving” requirements from domain models, cf. Sect. 7.2.7 on the next page these software houses now have a possibility of becoming professional.

7.2.2 From Programming Language Semantics to Domain Models

Domain models give semantics to the nouns (endurants) and verbs (perdurants) spoken by domain workers. Just like the development of compilers for programming languages were based on formal models of their semantics, so we can now give semantics to the nouns and verbs spoken by domain workers, and, from these, using rigorous development methods, similar to those used for compiler development [21, 25], develop trustworthy domain software.

7.2.3 Correctness: Verification, Checking, Testing

This paper has not dealt with the issue of correctness of domain models. A number of endurant and perdurant Description prompts have indicated that axioms and assertion need be expressed. For domain assertions their correctness must, of course, be shown – using whichever (testing, model checking and proof) techniques are adequate. The axioms and assertions carry over into Rrequirements prescriptions and, from there, into software Specifications. Now the full-blown force of testing, model checking and proofs must be applied. As indicated in formula D, S |= R, Sect. 1.2 on page 2 domain models now make proof obligations more clear.

\[24\] i.e., proof obligations
7.2.4 No Recursive Domains!

Surprise, surprise! Yes, there are no recursively defined endurant sorts. Domains do not contain “recursive endurants”.

7.2.5 Domain Facets

There is more to domain engineering than this paper can cover. A main element of domain modelling is that of modelling also other than the intrinsics of domains – as so far covered. By a domain facet we shall understand one amongst a finite set of generic ways of analysing a domain: a view of the domain, such that the different facets cover conceptually different views – and these views together cover the domain. Chapter 8 covers methods for modelling additional facets – such as support technology, rules & regulations, scripts (or contracts), license languages, management & organisation, and human behaviour.

7.2.6 Algorithmics

Algorithms are the hallmark and corner-stone of computing. So where is “algorithmics” in all this? So where, in all this, does algorithmics fit? The straight answer is: algorithm concerns are not concerns of domain modelling!

Domain models focus on expressing properties. They do so using abstraction in general, and simple combinations of proof theoretic and model theoretic means such as defining abstract types, here called sorts, comprehension over sets, sequences and maps \( \{ f(i) : D \cdot P(f,i) \} \), \( \{ f(i) : D \cdot Q(f,i) \} \), and \( f(i) \mapsto g(i) \cdot D \cdot R(f,g,i) \). The predicates, \( P \), \( Q \) and \( R \) further “raise” the abstraction. It is in the efficient rendering of these abstractions that algorithms play a crucial role.

7.2.7 Requirements

In Chapter 9, 2021 we show how to “derive”, in a systematic manner, requirements prescriptions from domain descriptions. Requirements are for a machine. The machine is the hardware upon which the software to be developed is to be executed – as well as the auxiliary software “under which” that new software is performing (operating system, database system, data communications software, etc.). First requirements development proceeds in three stages: (i) a domain requirements stage in which requirements that can be expressed solely using terms from the domain are developed; (ii) an interface requirements stage in which requirements that can be express using terms from both the domain and the machine are developed; and (iii) a domain requirements stage in which requirements that can be expressed solely using terms from the machine are developed. Chapter 9 shows how domain requirements stage can be decomposed, sequentially, into projection, initialisation, determination, extension and fitting steps. For details on this and more we refer to [15].

7.2.8 Software Design

[2, 2005-2006] shows how to further develop software from their requirements prescriptions.

---

25 Some readers may object, but we insist! If trees are brought forward, as an example of a recursively definable domain, then we argue: Yes, trees can be recursively defined. Trees can, as well, be defined as a variant of graphs, and you wouldn't claim, would you, that graphs are recursive? We shall consider the living species of trees (that is, plants), as atomic. In defining attribute types You may wish to model certain attributes as ‘trees’. Then, by all means, You may do so recursively. But natural trees, having roots and branches cannot be recursively defined, since proper “sub-trees” of trees would then have roots!

26 At an IFIP WG2.2 meeting in Kyoto, August 1978, John McCarthy "waking up" from deep thoughts, asked, in connection with my presentation of abstract models of various database models, "is there any recursion in all this?", to which I replied, "No! – whereupon he resumed his interrupted thoughts."

27 This characterisation clearly lacks sufficient formality. We refer to Sect. 7.2.16 on page 23 below.

28 As suggested by Michael A. Jackson [41].
7.2.9 Continuity

As remarked in Sect. 3.1 on page 9 the calculi of this paper do not address the issue of modelling continuous dynamic phenomena. This is clearly a weakness. The Integrated Formal Methods conferences \[43\] initially set out to spur research aimed at amalgamating continuous and discrete specifications. Not much progress has been made. We do refer, however, to \[58, 59\].

7.2.10 Modelling Concurrency

We have used Hoare’s CSP \[37\] to model concurrency. There are other, in this case, graphical languages for modelling concurrency. We refer to Chapters 12–15 of \[3\]. In these chapters I treat the modelling of four graphical specification languages: Petri Nets \[51\], Message Sequence Charts \[38, 39\], State Charts \[32\] and Live Sequence Charts \[26, 34\]. All of them are fascinating. Their graphics appeal to many of us – so I recommend to use them informally, aside, for the textual modelling shown in this paper. But they do not “merge” into formal, textual specification languages, like VDM-SL, RSL, Z, Alloy.

7.2.11 Modelling Temporality

Although time is modelled, as part of internal attribute properties, we have not shown the modelling of temporality of behaviours. In Chapter 15 of \[3\] I show how to merge Duration Calculus, DC \[60\] with RSL-Text. Another fascinating such formal specification language is Leslie Lamport’s TLA+: Temporal Logic of Actions \[44\].

7.2.12 Domain Specific Languages

A domain specific language, DSL, is a computer programming language specialised to a particular application domain. What we have shown here is not a DSL. Examples of DSLs could be programming languages for expressing calculations for railways or financial services or hospitals or other. \[24\] Actulus reports on an actuarial programming language for life insurance and pensions. To give semantics for a specific DSL one invariably specifies a domain model. So that, then, is a rôle for domain modelling.

7.2.13 Three Rôles for Domain Models

There are three rôles for domain models: (i) to just simply study and understand a domain – irrespective of any ensuing software for that domain; (ii) to serve as a basis for the development of a DSL; and (iii) to serve as a basis for the development of [other] software for the domain.

7.2.14 How Comprehensive should a Domain Model be?

Clearly domain models for any reasonable domain can be potentially be very large in terms of pages of description. So the question is: how much of the “domain at large” should be included in a domain description? We cannot, of course, give a general answer to that question. But we can say that the domain model must at least encompass those domain entities that will, or might, be referred to in a requirements prescription. That is, if it is found when developing a domain requirements\[29\] of a requirements prescription, that terms thought to be of the domain was not covered by the domain description, then, obviously, that description must be augmented.

We do expect there to be, eventually, available for general use, a few, domain models for selected domains.

For physics Newton and Leibniz\[30\] has given us a calculus with which to – more or less quickly – establish a model for some physical phenomenon. When control engineers then wish to set up some automatic control system for a phenomenon they first apply the Newton/Leibniz calculi to

---

\[29\] Cf. Sect. 7.2.7 on the previous page

\[30\] https://en.wikipedia.org/wiki/Leibniz%E2%80%93Newton_calculus_controversy
model the phenomenon, then, from that, somehow derive a control model. We advocate a similar approach, as already hinted at in our expressing the Triptych Dogma (Page 1).

The road transport domain modelled in Appendix A is one such domain. It has here been expressed in a way, devoid of any specific orientation. Based on the model of Appendix A we can envisage some such orientations as a road pricing domain, a cadastral map domain, a road development domain, a road maintenance domain, et cetera.

7.2.15 Domain Laws

Physics has excelled in our understanding the world we live in by its laws and by the calculi it has spawned – calculi that enables us to explain what has happened and to predict what will or might happen. Domain modelling has already lead to some domain laws – such as illustrated by for example intentional pulls, cf. Sect. 5.4 on page 16 (approx. half a page) and Appendix A.6 on page 33 (two pages). The study of intentional pull in domains has just started! Its counterpart in physics, gravitational pull, is “behind” many laws of physics.

7.2.16 A Domain Modelling Science?

A science of domain modelling systematically builds and organizes knowledge about the ways and means of modelling domains such that that knowledge can explain what these models express. As an example of there not yet being a sufficient scientific knowledge of domains we refer to our informal coverage of the concept of domain facets, cf. footnote 27 on page 21. A formal understanding of domains and what “facet” distinguishes them, could help sharpen the characterisation of Sect. 7.2.5 on page 21. Such a formal understanding was first reported in [10, 2014]. Of more specific nature we suggest, next, studies of some specific issues.

(i) An “integrated” form of use of differential equations with the present RSL+, i.e., the extension of our approach to domain modelling to cover more specifically issues of continuity.

(ii) A “further detailed” understanding of the concept of intentional pull.

(iii) A study of a possible Calculus of Perdurants.

(iv) A study of examples of domain models with an emphasis on human interaction.

(v) Formal models of the analysis predicates and functions and the description functions, cf. [10].

7.3 Acknowledgments

I gratefully acknowledge the opportunity given to me, to write this paper, during my PhD lectures, October–November 2022, at the TU Wien Informatics32 Vienna, Austria, by Prof. Laura Kovacs. I also gratefully acknowledge comments by Klaus Havelund, Kazuhiro Ogata and Wolfgang Reisig.

8 Bibliography

8.1 Bibliographical Notes

I plan to remove some references

8.2 References


31 https://eng.gst.dk/danish-cadastre-office/cadastral-map
32 https://informatics.tuwien.ac.at/


Domain Modelling


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


Domain Modelling


A  A Road Transport Domain Example

A.1 Naming and Sketch of Domain

We refer to Sect. 2 on page 6.

Narration:

1. The domain is referred to as RTD, the road transport domain.

2. The road transport domain comprises a set of automobiles and a road net of street intersections, called hubs, and [uninterrupted] street segments, called links. Automobiles drive in and out of hubs and links.

Formalisation:

\[
\text{type} \quad \mathbf{1. RTD}
\]

A.2 Endurants: External Qualities

A.2.1 Cartesian Examples

We refer to Sect. 3.4.1 on page 10.

3. There is a road transport domain.

From road transport domains we can observe

4. a road net aggregate and

5. an automobile aggregate.

From the road net aggregate we can observe

6. an aggregate of hubs, i.e., street intersections, and

7. an aggregate of links, i.e., street segments (with no hubs).

\[
\begin{align*}
\text{type} & : \mathbf{3. RTD} & \mathbf{4. RNA} & \mathbf{5. AA} & \mathbf{6. HA} & \mathbf{7. LA} \\
\text{value} & : \mathbf{1. obs_{RNA}: RTD \to RNA} & \mathbf{5. obs_{AA}: RTD \to AA} & \mathbf{6. obs_{HA}: RNA \to HA} & \mathbf{7. obs_{LA}: RNA \to LA}
\end{align*}
\]

A.2.2 Part Sets

We refer to Sect. 3.4.2 on page 10.

8. There are hubs; from aggregate of hubs one can observe sets of hubs.

9. There are links; from aggregate of links one can observe sets of links.

10. There are automobiles; from aggregate of automobiles one can observe sets of automobiles.

\[
\begin{align*}
\text{type} & : \mathbf{8. H, Hs = H-set} & \mathbf{9. L, Ls = L-set} & \mathbf{10. A, As = A-set} \\
\text{value} & : \mathbf{8. obs_{Hs}: HA \to Hs} & \mathbf{9. obs_{Ls}: LA \to Ls} & \mathbf{10. obs_{As}: AA \to As}
\end{align*}
\]
A.2.3 **Endurant States**

We refer to Sect. 3.5 on page 10.

11 The singleton value $rtd$ represents a road transport [domain] state.
12 The set value $hs$ represents a state of all hubs of that road transport domain.
13 The set value $ls$ represents a state of all links of that road transport domain.
14 The set value $as$ represents a state of all automobiles of that road transport domain.

\[
\begin{align*}
\text{value} & \\
11 & rtd: \text{RTD}, \\
12 & hs: \text{H-set} = \text{obs}_H(\text{obs}_{\text{HA}}(\text{obs}_{\text{RNA}}(rtd))), \\
13 & ls: \text{L-set} = \text{obs}_L(\text{obs}_{\text{LA}}(\text{obs}_{\text{RNA}}(rtd))), \\
14 & as: \text{A-set} = \text{obs}_A(\text{obs}_{\text{AA}}(rtd))
\end{align*}
\]

A.3 **Unique Identifiers**

We refer to Sect. 5.1 on page 12.

A.3.1 **Unique Identification**

We shall only consider hubs, links and automobiles.

15 Hubs have unique identifiers.
16 Links have unique identifiers.
17 We define also a unique identifier observer for hubs and links.
18 Automobiles have unique identifiers.

\[
\begin{align*}
\text{type} & \\
15 & \text{HI} \\
16 & \text{LI} \\
18 & \text{AI}
\end{align*}
\]

\[
\begin{align*}
\text{value} & \\
15 & \text{uid}_H: H \to \text{HI} \\
16 & \text{uid}_L: L \to \text{LI} \\
17 & \text{uid}_{HL}: (H|L) \to (\text{HI}|\text{LI}), \ \text{uid}_{HL}(hl) \equiv \text{is}_H(hl) \rightarrow \text{uid}_H(hl), \_ \rightarrow \text{uid}_L(hl) \\
18 & \text{uid}_A: A \to \text{AI}
\end{align*}
\]

A.3.2 **Unique Identifier State**

19 The variable $his$ contains all unique hub identifiers of the road transport domain on the preceding page.
20 The variable $lis$ contains all unique link identifiers of the road transport domain on the facing page.
21 The variable $ais$ contains all unique automobile identifiers of the road transport domain on the preceding page.

\[
\begin{align*}
\text{variable} & \\
14 & his = \{ \text{uid}_H(h) \mid h: H \cdot h \in hs \}. \\
19 & lis = \{ \text{uid}_L(l) \mid L \cdot l \in ls \}. \\
20 & ais = \{ \text{uid}_A(a) \mid a: A \cdot a \in as \}. 
\end{align*}
\]
A.3.3  Unique Identifier Axiom

22 No two hubs, links and automobiles have the same unique identifier.

23 \( ps \) is the set of all hubs, links and automobiles.

24 \( uis \) is the set of all unique hub, link and automobile identifiers.

\[
\text{axiom}
\]

\[
\begin{align*}
\text{card } hs &= \text{card } his, \\
\text{card } ls &= \text{card } lis, \\
\text{card } as &= \text{card } ais, \\
\text{card } hs + \text{card } ls + \text{card } as &= \text{card } his + \text{card } lis + \text{card } ais
\end{align*}
\]

\[
\text{value}
\]

\[
\begin{align*}
\text{ps} &= hs \cup ls \cup as \\
\text{uis} &= his \cup lis \cup ais
\end{align*}
\]

\[
\text{axiom}
\]

\[
\text{card } ps = \text{card } uis
\]

A.4  Mereology

We refer to Sect. 5.2 on page 13.

25 The mereology of any hub is a pair: the possibly empty set of the unique identifiers of links leading into and/or out from the hub, and the set of the unique identifiers of automobiles that are allowed to drive in the hub.

26 The mereology of any link is a pair: the two element set of the unique identifiers of the two hubs that are connected by the link, and the set of the unique identifiers of automobiles that are allowed to drive on the link.

27 The mereology of any automobile is the set of the unique identifiers of hubs in and links on which the automobile may be driving.

\[
\begin{align*}
\text{type} & \quad \text{value} \\
25 & \quad \text{H Mer}=\text{LI-set}\times\text{AI-set} & 25 & \quad \text{mero}_H: \text{H}\to\text{H Mer} \\
26 & \quad \text{L Mer}=\text{HI-set}\times\text{AI-set} & 26 & \quad \text{mero}_L: \text{L}\to\text{L Mer} \\
27 & \quad \text{A Mer}=\text{(HI|LI)-set} & 27 & \quad \text{mero}_A: \text{A}\to\text{A Mer}
\end{align*}
\]

28 Link and automobile identifiers of hub mereologies must be of the road transport domain.

29 Hub and automobile identifiers of hub mereologies must be of the road transport domain and there must be exactly two hub identifiers of those mereologies.

30 Hub and links identifiers of automobile mereologies must be of the road transport domain.

\[
\text{axiom}
\]

\[
\begin{align*}
\forall (lis,ais): & \quad \text{H Mer}\cdot\text{lis}\subseteq\text{lis}\land\text{ais}\subseteq\text{ais} \\
\forall (his,ais): & \quad \text{L Mer}\cdot\text{his}\subseteq\text{his}\land\text{ais}\subseteq\text{ais}\land\text{card } his=2 \\
\forall ris: & \quad \text{A Mer}\cdot\text{ris}\subseteq\text{his}\land\text{his}
\end{align*}
\]
A.4.1 Routes

31 By a route (of a road net) we shall understand

a. an alternating sequence of one or more hub and link identifiers

32 such that

a. basis clause 0: the empty list is a route;

b. basis clause 1: a singleton list of a hub or a link identifier of the road net is a route;

c. inductive clause: the concatenation of a route, \( r \), and the tail of a route \( r' \) where the last element of \( r \) is identical to the first element of \( r' \) is a route; and

d. extremal clause: and only such routes that can be formed using the above clauses are routes.

```
type
31 R' = (HI|LI)*
31a R = { | r:R' | wf_R(r)(rtd) |}
value
31a wf_R: R' → RTD → Bool
31a wf_R(r)(rtd) ≡
31a ∀ i,i+1:Nat • {i,i+1}⊆index(r) ⇒
31a let (ri,ri') = (r[i],r[i+1]) in
31a is_HI(ri)&is_LI(ri')∧ ...  
31a is_HI(ri)&is_LI(ri')∧ ...
31a end

32 routes: RTD×HI-set×LI-set → R-infset
32 routes(rtd,his,lis) ≡
32 let rs = { ⟨⟩ } 
32 ∪ { ⟨hi⟩ | hi:HI • hi ∈ his } 
32 ∪ { ⟨li⟩ | li:LI • li ∈ lis } 
32 ∪ { ⟨r'[tl,r'] | {r,r'}⊆rs ∧ r[len r]=hd r' ⟩ in
32c rs end
32c pre: his={uid_H(h)|h:H • h ∈ obs_Hs(obs_AH(obs_RN(rtd)))} ∧ 
32c lis={uid_L(l)|l:L • l ∈ obs_Ls(obs_AL(obs_RN(rtd)))}
```

A.5 Attributes

We refer to Sect. 5.3 on page 14.

A.5.1 Hubs, Links and Automobiles

Hub Attributes

33 Hubs have [traffic signal] states which are set of pairs, \( li, lj \), of identifiers of the mereology links “signaling” that automobiles can connect from link \( li \) to link \( lj \).

34 Hubs have [traffic signal] state spaces – designating the set of all possible hub states.

35 Hubs have a history; see Item 46 on page 33

Link Attributes

36 Links have lengths.
37. Links have a history; see Item [47 on the next page].

**Automobile Attributes**

38. Automobiles have positions on the road net:
   a. either *at a hub*,
   b. or *on a link*, some fraction
   c. down from an entry hub towards the exit hub.

39. Automobiles have a history; see Item [48 on the facing page].

We postpone treatment of hub, link and automobile histories till Sect. A.6.1.

type

33. \( \mathcal{H}_\Sigma = (LI \times LI)\)-set
34. \( \mathcal{H}_\Omega = \mathcal{H}_\Sigma\)-set
35. \( \mathcal{H}_\text{Hist} = \ldots \)
36. \( \mathcal{L}_\text{Hist} = \ldots \)
38. \( \mathcal{A}_\text{Pos} = \text{At}_\text{Hub} \mid \text{On}_\text{Link} \)
38a. \( \text{At}_\text{Hub} : \rightarrow \mathcal{H}_\text{Hist} \)
38b. \( \text{On}_\text{Link} : LI \times \mathcal{H}_\text{Hist} \times F \times \mathcal{H}_\text{Hist} \)
38c. \( F = \textbf{Real axiom} \ \forall f : 0 < f < 1 \)
39. \( \mathcal{A}_\text{Hist} = \ldots \)

value

33. \( \text{attr}_\mathcal{H}_\Sigma : H \rightarrow \mathcal{H}_\Sigma \)
34. \( \text{attr}_\mathcal{H}_\Omega : H \rightarrow \mathcal{H}_\Omega \)
35. \( \text{attr}_\mathcal{H}_\text{Hist} : A \rightarrow \mathcal{H}_\text{Hist} \)
36. \( \text{attr}_\mathcal{L}_\text{Hist} : A \rightarrow \mathcal{L}_\text{Hist} \)
37. \( \text{attr}_\mathcal{A}_\text{Pos} : A \rightarrow \mathcal{A}_\text{Pos} \)
38. \( \text{attr}_\mathcal{A}_\text{Hist} : A \rightarrow \mathcal{A}_\text{Hist} \)

We omit treatment of such automobile attributes as speed, acceleration, engine temperature, energy (gas, oil, electricity) level, mileage and trip counters, GPS (map) position, road surface temperature, gear position (reverse, neutral, forward (1, 2, 3, 4, 5), hand brake position, clutch position, accelerator pressure, brake pedal position, etc.

40. The link identifiers of a hub state must be of the mereology of that hub.
41. A hub state must be in the hub state space.
42. The automobile position must be on the road net.

axiom

40. \( \forall h : H \cdot h \in hs \cdot \textbf{let} \ h \sigma = \text{attr}_\mathcal{H}_\Sigma(h), (lis,_) = \text{mero}_H(h) \textbf{ in} \)
40. \( \forall (li,lj):(LI \times LI) \cdot (li,lj) \in h \sigma \Rightarrow \{li,lj\} \subseteq lis \)
41. \( \forall h : H \cdot h \in hs \cdot \text{attr}_\mathcal{H}_\Sigma(h) \in \text{attr}_\mathcal{H}_\Omega(h) \)
42. \( \forall a : A \cdot a \in as \cdot \textbf{let} \ apos = \text{attr}_\mathcal{A}_\text{Pos}(a) \textbf{ in} \)
42. \( \textbf{cases} \ apos \ \textbf{of} \)
42. \( \text{At}_\text{Hub}(hi) \rightarrow hi \in his, \)
42. \( \text{On}_\text{Link}(li,fhi,thi) \rightarrow \)
42. \( \textbf{let} \ (his,ais) = \text{mero}_L(\text{retr}_L(li,ls)) \textbf{ in} \)
42. \( \{fhi,thi\} \subseteq his \land \text{uid}_A(a) \in ais \textbf{ end} \)
42. \( \textbf{end} \)
These were some well-formedness axioms. In Sect. A.6.1 we shall treat well-formedness of hub, link and automobile histories.

A.5.2 Attribute Category Examples

Attribute categories are: \(H\Sigma\) (Item 33) is a programmable attribute; \(H\Omega\) (Item 34 on page 31) is a static attribute; LEN (Item 36 on page 31) is a static attribute; A\_Pos (Item 38 on the facing page) is a programmable attribute; GPS\_Map is an inert attribute; Speed is a biddable attribute; Road\_Surface\_Temperature is an autonomous attribute; etcetera.

A.6 Intentional Pull

We refer to Sect. 5.4 on page 16.

A.6.1 Further Attributes

We start by formulating the hub, link and automobile history attribute definitions.

43 Hubs and links are entered and left by automobiles, i.e., marked by corresponding events.

44 Automobile enters and leaves hubs, i.e., marked by corresponding events.

45 Automobile enters and leaves links, i.e., marked by corresponding events.

46 Hub histories are time-stamped sequences of automobile enter/leave events – in decreasing order (most recent events are listed first),

47 Link histories are time-stamped sequences of automobile enter/leave events – in decreasing order (most recent events are listed first),

48 Automobile histories are time-stamped sequences of hub and link enter/leave events – in decreasing order (most recent events are listed first),

49 For convenience we “lump” hub and link histories into hub-link histories.

\[
\text{type} \\
43. \text{HL\_OnOff} = \text{mkEnter}(ai:\text{AI}) \mid \text{mkLeave}(ai:\text{AI}) \\
44. \text{A\_OnOff\_H} = \text{mkEnterHub}(s:\text{HI}) \mid \text{mkLeaveHub}(s:\text{HI}) \\
45. \text{A\_OnOff\_L} = \text{mkEnterLink}(s:\text{LI}) \mid \text{mkLeaveLink}(s:\text{LI}) \\
46. \text{H\_Hist} = (s.t:\text{TIME} \times s.oo:\text{HL\_OnOff})^* \\
47. \text{L\_Hist} = (s.t:\text{TIME} \times s.oo:\text{HL\_OnOff})^* \\
48. \text{A\_Hist} = (s.t:\text{TIME} \times s.oo:(\text{OnOff\_H}(\text{OnOff\_L})\text{)})^* \\
49. \text{HL\_Hist} = \text{H\_Hist} \mid \text{L\_Hist} \\
\text{value} \\
49. \text{attr\_HL\_Hist}: (\text{H} \rightarrow \text{H\_Hist}) \mid (\text{L} \rightarrow \text{L\_Hist})
\]

50 Automobile histories

a. alternate between being on hubs and being on links.

b. such that the enter hub event time is identical to the immediately “prior” leave link event time,

c. and such that these events are otherwise ordered in decreasing order of time.
We leave the (narrative and formal) expression of the well-formedness of hub and link histories to the reader!

The above indicates that one has to be very careful concerning well-formedness.

But we have not captured all of the constraints, i.e., well-formedness of the history attributes. Next we secure full care!

A.6.2 An Intentional Pull

51 For all automobiles,

- if their traffic history records that the automobile was entering [leaving] a hub (link) at a certain time,
- then that hub’s (link’s) traffic history shall record that that automobile entered [left] that hub (link) at exactly that time;

52 and vice versa, for all hubs an links:

- if a hub or link traffic history records that an automobile was leaving that hub (link) at a certain time,
- then that automobile’s traffic history shall record that that automobile left that hub (link) at exactly that time.

axiom

51 a:A • a ∈ as ⇒
    ⏞
    ∀ a_hist: A_Hist • let a_hist=attr_a_Hist(a) in 
          ⏞
          ∀ (t,on_off) • (t,on_off) ∈ elems a_hist ⇒ 
          ⏞
          ∀ hl • s(on_off) ⇒ 
            ⏞
            ∀ hli: H[L] • hli ∈ hs∪ls ⇒ 
              ⏞
              mkEnter(hli) ⇒ 
              ⏞
              mkLeave(hli) ⇒ 

            ⏞
            ⏞
            ⏞

          ⏞

51a ⏞
      ∀ ai:AI • ai ∈ dom hli ⇒ 
      ⏞
      let a:A • a ∈ as • uid_A(a)=ai ⇒ 
    ⏞
    ∀ (t,on_off) • (t,on_off) ∈ elems hli_hist(•) ⇒ 
      ⏞
      ⏞
      ⏞

The above formalisation is currently being checked

A.7 Perdurants

A.7.1 Channels

We refer to Sect. 6.1 on page 17.

channel \{ ch[\{ui,uj\}] | ui,uj:(HI|LI|AI) \cdot \{ui,uj\} \subseteq hislisais \} : \mathcal{M}

\(\mathcal{M}\) is presently left undefined.

A.7.2 Domain Actions and Events

A.7.2.1 Domain Actions

Automobile actions are here simplified to be those of remaining (staying) in a hub (Item 56a on the next page) and remaining (staying) on a link (Item 57a on the following page).

A.7.2.2 Domain Events

Automobile events are here simplified to be those of leaving a hub in order to enter a link (Item 58d on the next page and Item 62 on page 37) and leaving a link in order to enter a hub (Item 59c on page 37 and Item 67 on page 37).

16 Example. Domain Actions and Events: We refer to Sect. A.7.2

A.7.3 Behaviour Signatures

We refer to Sect. 6.4.1 on page 19.

value

\begin{align*}
\text{hub} & : h:H \rightarrow \text{in}, \text{out} \{ \text{ch}[\{hi,ui\}] | ui:(LI|AI) \cdot \{ui,uj\} \subseteq \text{his}, \text{ais} \} \rightarrow \text{Unit}, \\
\text{link} & : l:L \rightarrow \text{in}, \text{out} \{ \text{ch}[\{li,ui\}] | ui:(LI|AI) \cdot \{ui,uj\} \subseteq \text{lis}, \text{ais} \} \rightarrow \text{Unit}, \\
\text{auto} & : a:A \rightarrow \text{in}, \text{out} \{ \text{ch}[\{ai,ui\}] | ui:(LI|HI)-\text{set} \cdot \{ui,uj\} \subseteq \text{his} \} \rightarrow \text{Unit}.
\end{align*}

A.7.4 Behaviour Definitions

We refer to Sect. 6.4.2 on page 19

Automobile Behaviour

We omit consideration of the monitorable GPS_Map, Speed and Road_Surface_Temperature attributes.

53 One interpretation of an automobile, \(a\), focuses on its road position.
54 Either the automobile is at a hub,
55 or it is on a link.

value

\begin{align*}
\text{auto} & (a) \equiv \text{auto_pos}(a)(\text{attr}_A \text{Pos}(p), \text{attr}_A \text{His}(a)) \\
\text{auto_pos}(a)(\text{At}_\text{Hub}(hi), a, \text{hist}) & \equiv \\
& \text{traversing}_\text{hub}(a)(\text{At}_\text{Hub}(hi), a, \text{hist}) \\
& \text{pre}: \text{attr}_A \text{Pos}(a) = \text{At}_\text{Hub}(hi) \land \text{attr}_A \text{His}(a) = a, \text{hist} \\
\text{auto_pos}(a)(\text{On}_\text{Link}(li, fhi, f, thi), a, \text{hist}) & \equiv \\
& \text{traversing}_\text{link}(a)(\text{On}_\text{Link}(li, fhi, f, thi), a, \text{hist}) \\
& \text{pre}: \text{attr}_A \text{Pos}(a) = \text{On}_\text{Link}(li, fhi, f, thi) \land \text{attr}_A \text{His}(a) = a, \text{hist}
\end{align*}
56 In traversing a hub an automobile

a is either, internal non-deterministically, \( [\) , moving on inside the hub
b or, internal non-deterministically, entering a link from the hub.

\[
\text{value}
\]

\begin{align*}
\text{traversing\_hub(a)(At\_Hub(hi),a\_hist)} & \equiv \\
\text{staying\_at\_H(a)(At\_Hub(hi),a\_hist)} & \\
\text{entering\_L(a)(At\_Hub(hi),a\_hist)} & \quad \text{pre: attr\_A\_Pos(a)=At\_Hub(hi) \land attr\_A\_Hist(a)=a\_hist}
\end{align*}

57 In traversing a link an automobile

a is either, internal non-deterministically, \( ]\), moving on inside the link
b – possibly advancing a bit, i.e., increasing its fraction position “down” the link,
c or, internal non-deterministically, entering a hub from the link.

\[
\text{value}
\]

\begin{align*}
\text{traversing\_link(a)(On\_Link(li,fhi,f,thi),a\_hist)} & \equiv \\
\text{staying\_on\_L(a)(On\_Link(li,fhi,f,thi),a\_hist)} & \\
\text{entering\_H(a)(On\_Link(li,fhi,f,thi),a\_hist)} & \quad \text{pre: attr\_A\_Pos(a)=On\_Link(li,fhi,f,thi) \land attr\_A\_Hist(a)=a\_hist}
\end{align*}

58 In entering a link

a the automobile internal non-deterministically selects the link to be entered, and thus
b records the time,
c updates its history and automobile position accordingly,
d so informs the behaviour of the hub being left and the link being entered,
while resuming being an automobile – with the updated history.

\[
\text{value}
\]

\begin{align*}
\text{entering\_L(a)(At\_Hub(hi),a\_hist)} & \equiv \\
\text{let li:LI • li \in lis \land li \in mereo\_H(retr\_H(fhi)(\sigma)),thi:HI•thi \in his \land thi \in mereo\_L(retr\_L(li)(\sigma))\{fhi\}} & \\
\quad \tau = \text{record\_TIME}[ai=uid\_A(a) \text{ in}]
\end{align*}
In entering a hub

a. the time is recorded,
b. the automobile history and position is updated,
c. and the behaviours of the link left link and hub entered are being so informed
while the automobile resumes being an automobile – in the updated state.

```wolfram
value
59. entering_H(a)(On_Link(li,fhi,f,thi),a_hist) ≡
   let \( \tau = \text{record}_\text{TIME}, \)
   ai = uid_A(a),
   a_pos = at_Hub(thi) in
   let a_hist' = \langle (a_pos,\tau) \rangle \_a_hist in
   let a' = part_update(a)(\( \eta A_{Hist},(\tau,a_hist') \)) in
   let a'' = part_update(a')(\( \eta A_{Pos},a_pos \)) in
   (ch[ai,li] ! mk_leave_L(ai,\tau) \mid ch[ai,thi] ! mk_enter_H(ai,\tau) \mid \text{auto}(a''))
end end end end
59. pre: attr_A_{Pos}(a) = On_Link(li,fhi,f,thi) \land attr_A_{Hist}(a) = a_hist

Hub Behaviour

60. The hub behaviour
61. externally non-deterministically (\[\]) offers
62. to accept, non-deterministically, a leave message,
63. from any automobile in its mereology;
64. it prepares for proper insertion of this event into its traffic history
65. updating to an augmented traffic history, and, hence, hub state;
66. resuming to be the hub behaviour in the updated state;
67. or to accept, non-deterministically, an enter message,
68. again from any automobile in its mereology;
69. updating to an augmented traffic history, and, hence, hub state;
70. resuming to be the hub behaviour in the updated state.

value
60. hub(h) ≡
   [] \{ let mk\_leave_H(ai,\tau) = ch[\{hi,ai\}] ? in
       let h_hist' = \langle (\tau,mkEnter(ai)) \rangle \_attr \_H Hist(h) in
       let h' = part_update(\( \eta H_{Hist},h_hist' \)) in
       hub(h')
       | ai:AI \_ ai \_ ais end end end
   \}
61. [] \{ let mk\_enter_H(ai,\tau) = ch[\{hi,ai\}] ? in
       let h_hist' = \langle (\tau,mkLeave(ai)) \rangle \_attr \_H Hist(h) in
       let h' = part_update(\( \eta H_{Hist},h_hist' \)) in
       hub(h')
       | ai:AI \_ ai \_ ais end end end \}

\[33\] For retr… see Sect. 5.1.4 on page 13
\[34\] For record\_TIME see Sect. 4.2 on page 12
The above formalisation is currently being checked

We leave the definition of link behaviours as an exercise!

A.8 Domain Initialisation

We refer to Sect. 6.5 on page 20.

We initialise a domain behaviour for all atomic endurants: hubs, links and automobiles.

71 The domain behaviour is the parallel composition of
72 the distributed parallel composition of all hub behaviours, with
73 the distributed parallel composition of all link behaviours, with
74 the distributed parallel composition of all automobile behaviours.

\[
\begin{align*}
72 & \parallel \{ \text{hub}(b) \mid h:H \cdot h \in hs \} \\
73 & \parallel \{ \text{link}(l) \mid l:L \cdot l \in ls \} \\
74 & \parallel \{ \text{auto}(a) \mid a:A \cdot a \in as \}
\end{align*}
\]

A.9 Verification

It remains to verify that the automobile, hub and link behaviours and the road transport domain initialisation satisfy the appropriate axioms and the intentional pull.

End of Example

B Method Tool Index

Analysis Predicates:

- is_Cartesian
- is_animal
- is_atomic
- is_compound
- is_endurant
- is_entity
- is_fluid
- is_human
- is_living_species
- is_manifest
- is_mobile
- is_part_set
- is_part
- is_perdurant
- is_plant
- is_solid
- is_stationary
- is_structure

Analysis Functions:

- determine_Cartesian_parts
- determine_attributes
- determine_part_set

Description Functions:

- descr_Cartesian
- descr_ Universe_of_Discourse
- descr_attributes
- descr_mereology
- descr_part_set
- descr_unique_identifier
- record_LOCATION
- record_TIME