Domain Analysis & Description[⋆]

Dines Bjørner

Technical University of Denmark, DTU Compute.
Fredsvej 11, Holte 2840, Denmark. bjorner@gmail.com, June 22, 2025

[⋆] Invited Tutorial for ICTAC 2025, 24-28 November, Marrakesh, Marocco.

Table of 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.*

We present a systematic *method*, its *principles*, *procedures*, *techniques* and *tools*, for efficiently *analyzing* & *describing* domains. This paper is based on [13–15]. It simplifies the methodology of these considerably – as well as introduces some novel presentation and description language concepts.

• • •

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

High Light 1 *What The Paper is All About*: The *Triptych Dogma*, above, says it all: this paper is about a new area of computing science – that of *domains*. It is about what domains are. How to model them. And their role in software development. There are many "domain things" it is not about: it is not about 'derived' properties of domains – beyond, for example, *intentional pull* [Sect. 8.3]. Such are left for studies of domains based on the kind of formal domain descriptions such as those advocated by this paper •

High Light 2 *A Radically New Approach to Software Development*: The *Triptych Approach to Software Development*, calls for *software* to be developed on the basis of *requirements prescriptions*, themselves developed on the basis of *domain descriptions*. We furthermore advocate these specifications and their development be formal. That is: there are formal methods for the development of either of these three kinds of specifications:

- Development of domain descriptions is outlined in this paper.
- Development of requirements, from domain descriptions, is outlined in [15, *Chapter 9*].
- Development of software, from requirements prescriptions, is treated, extensively, in [8].

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

High Light 3 *Characterizations rather than Definitions*: The object of domain study, analysis and description, i.e., the domains, are, necessarily, informal. A resulting domain description is formal. So the domain items being studied and analyzed cannot be given a formal definition. Conventionally [so-called theoretical] computer scientists expect and can seemingly only operate in a world of clearly defined concepts. Not so here. It is not possible. Hence we use the term 'characterization' in lieu of 'definition' •

High Light 4 *Seemingly Fragmented Texts*: The text of this paper is a sequence of enumerated sections, subsections, sub-subsections and paragraphs, with short HIGHLIGHTS, CHARACTERIZATIONS, EXAMPLES, ONTOLOGICAL CHOICES, PROMPTS, SCHEMAS and ordinary short texts. The brevity is intentional. Each and all of these units outline important concepts. Each contain a meaning and can be read "in isolation" •

1 Domains

We start by delineating the informal concept of domain,[1]

1.1 What are They ?

What do we mean by 'domain' ?

Characterization 1. *Domain*: 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* •

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

Example 1. *Some Domain Examples*: A few, more-or-less self-explanatory examples:

- **Rivers** – with their natural sources, deltas, tributaries, waterfalls, etc., and their man-made dams, harbours, locks, etc. – and their conveyage of materials (ships etc.) [19, *Chapter B*].
- **Road nets** – with street segments and intersections, traffic lights and automobiles – and the flow of these [19, *Chapter E*].
- **Pipelines** – with their liquids (oil, or gas, or water), wells, pipes, valves, pumps, forks, joins and wells and the flow of fluids [19, *Chapter I*].
- **Container terminals** – with their container vessels, containers, cranes, trucks, etc. – and the movement of all of these[19, *Chapter K*] •

Characterization 1 relies on the understanding of the terms *'rationally describable'*, *'discrete dynamics'*, *'human assisted'*, *'solid'* and *'fluid'*. The last two will be explained later. By **rationally describable** we mean that what is described can be understood, including reasoned about, in a rational, that is, logical manner – in other words **logically tractable**.[2] By **discrete dynamics** we imply that we shall basically rule out such domain phenomena which have properties which are continuous with respect to their time-wise, i.e., dynamic, behaviour. By **human-assisted** we mean that the domains – that we are interested in modeling – have, as an important property, that they possess man-made entities.

1.2 Some Introductory Remarks

1.2.1 A Discussion of Our Characterization of a Concept of Domain.

Characterization 1 is our attempt to delineate the subject area. That is, "our" concept of *'domain'* is 'novel': *new and not resembling something formerly known or used*. As such it may be unfamiliar to most readers. So it takes time to digest that characterization. So the reader may have to return to the page, Page 4, to be reminded of the definition.

1.2.2 Formal Methods and Description Language.

The reader is assumed to have a reasonable grasp of formal methods – such as espoused in [21, 22, 8, 51].

The descriptions evolving from the modeling approach of this paper are in the abstract, model-oriented specification language `RSL` [28] of the **R**aise[3] **S**pecification **L**anguage. But other abstract specification languages could be used: `VDM` [21, 22], `Z` [51], `Alloy` [35], `CafeOBJ` [27], etc. We have chosen `RSL` since it embodies a variant of `CSP` [33] – being used to express domain behaviours.

1.2.3 Programming Languages versus Domain Semantics.

From around the late 1960s, spurred on by the works of John McCarthy, Peter Landin, Christopher Strachey, Dana Scott and others, it was not unusual to see publications of entire formal definitions of programming language semantics. Widespread technical reports were [3, 2, 1969, 1974] Notably so was [40, 1976]. There was the 1978 publication [21, *Chapter 5, Algol 60*, 1978]. Others were [22, *Chapters 6–7, Algol 60 and Pascal*, 1982] As late as into the 1980s there were such publications [4, 1980].

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

1.2.4 A New Universe.

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

[2] Another, "upside–down" – after the fact – [perhaps 'cheating'] way of defining 'describable' is: is it describable in terms of the method of this paper!

[3] **RAISE** stands for **R**igorous **A**pproach to **I**ndustrial **S**oftware **E**ngineering [29].

You will therefore be introduced to quite a universe of new concepts. You will find these concepts named in most display lines[4] and in Figs. 1 and 2.

2 Six Languages

This section is an artifice, an expedient.

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

2.1 The 6 Languages

There are 6 languages at play in this paper:

- (i) technical English, as in most papers;
- (ii) `RSL`, the `RAISE` Specification Language [28];
- (iii) an augmented `RSL` language;
- (iv) the domain modeling language – which we can view as the composition of clauses from two [sub-ordinate] languages:
  - (v) a domain analysis language; and
  - (vi) a domain specification

  language.

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

We presume that the reader is familiar with such languages as `RSL`. That is: `VDM` [21, 22], `Z` [51], `Alloy` [35], etc. They could all be use instead of, as here, `RSL`.

We summarize some of the language issues.

The Domain Analysis Language: We list a few, cf. Fig. 1, of the predicate prompts, i.e., language prompts: is_entity [pg 10], is_endurant [pg 11], is_perdurant [pg 11], is_solid [pg 13], is_fluid [pg 13], is_part [pg 14], aatomic [pg 14], is_compound [pg 14], is_Cartesian [pg 15], or is_part-set [pg 15]; and the extended `RSL` text yielding analysis prompts: record_Cartesian_type_names [pg 16], record_part_-set_type_names [pg 16] and record_attribute_type_names [pg 20].

The Domain Description Language: `RSL`. We shall us a subset of `RSL`. That subset is a simple, discrete mathematics, primarily functional specification language in the style of `VDM` [21, 22]. Emphasis is on sets, Cartesians, lists, and maps (i.e., finite definition set, enumerable functions).

Domain Description: A domain description consists of one or more domain specification units. A specification unit is of either of 10 kinds, all expressed in `RSL`. (1) a universe-of-discourse **type** clause [pg 12]; (2) a part **type** and **obs**_erver **value** clause [pg 16]; (3) a **value** clause; (4) a unique identifier **type** and (**uid**_) observer value (function) clause [pg 18]; (5) a mereology **type** and (**mereo**_) observer value (function) clause [pg 20]; (6) an attribute **type** and (**attr**_) observer value (function) definition clause [pg 21]; (7) an **axiom** clause; (8) a **channel** declaration clause [pg 27]; (9) a behaviour **value** (signature and definition) clause [pg 27 & pg 31]; and (10) a domain initialization clause [Sect. 10.6]. These clauses are often combined in 2-3 such clauses, and may, and usually do, include further `RSL` clauses.

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

[4] – that is, section, subsection, sub-subsection, paragraph and sub-paragraph lines
[5] Søren Kierkegaard: *Life is lived forwards but is understood backwards* [1843].

2.2 Semiotics

In *Foundations of the theory of signs* [41] defines semiotics as “consisting” of syntax, semantics and pragmatics.

- **Syntax:** The syntax of domain analysis and domain description clauses are simple atomic clauses consisting of a prompt (predicate or function) identifier, see above, and an identifier denoting a domain entity. The syntax of the domain modeling language prescribes a sequence of one or more domain analysis and domain description clauses.
- **Semantics:** The meaning of a domain analysis clause is that of a function from a domain entity to either a truth value or some augmented `RSL` text. The meaning of a domain description clause is that of a function from a domain entity to a domain specification unit.
- **Pragmatics:** The pragmatics of a domain analysis predicate clause, as applied to a domain entity e, is that of prompting the domain analyzer to a next domain analysis step: either that of applying a [subsequent, cf. Fig. 1] domain analysis predicate prompt to e; or applying a [subsequent, cf. Fig. 1] domain analysis function to e, and noting – as writing down on a “to remember board” – the result of the [latter] query; or applying a [subsequent, cf. Fig. 1] domain description function to e. The pragmatics of a domain description function is that of including the resulting `RSL` domain description text in the emerging domain description. There is no hint as to what to do next!

2.3 Speech Acts

The above explication of a pragmatics for the domain modeling language relates to the concepts of *speech acts*. We refer to [1, How to do things with words], [44, Speech Acts: An Essay in the Philosophy of Language] and [43, Brain mechanisms linking language and action]. A further study of the *illocutionary* and *locutionary* aspects of the domain analysis language seems in place.

3 Endurants and Perdurants, I

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

Characterization 2. *Endurants*: Endurants are those quantities of domains that we can observe (see and touch), in *space*, as “complete” entities at no matter which point in *time* – “material” entities that persists, endures – capable of enduring adversity, severity, or hardship [Merriam Webster] •

Endurants are either *natural* [“God-given”] or *artefactual* [“man-made”]. Endurants may be either solid (discrete) or fluid, and solid endurants, called parts, may be considered *atomic* or *compound* parts; or, as in this paper solid endurants may be further unanalysed *living species: plants* and *animals* – including *humans*.

Characterization 3. *Perdurants*: Perdurants are those quantities of domains for which only a fragment exists, in *space*, if we look at or touch them at any given snapshot in *time* •

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

• • •

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

4 A Domain Analysis & Description Ontology

4.1 The Chosen Ontology

Figure 1 expresses an ontology[6] for our analysis of domains. Not a taxonomy[7] for any one specific domain.

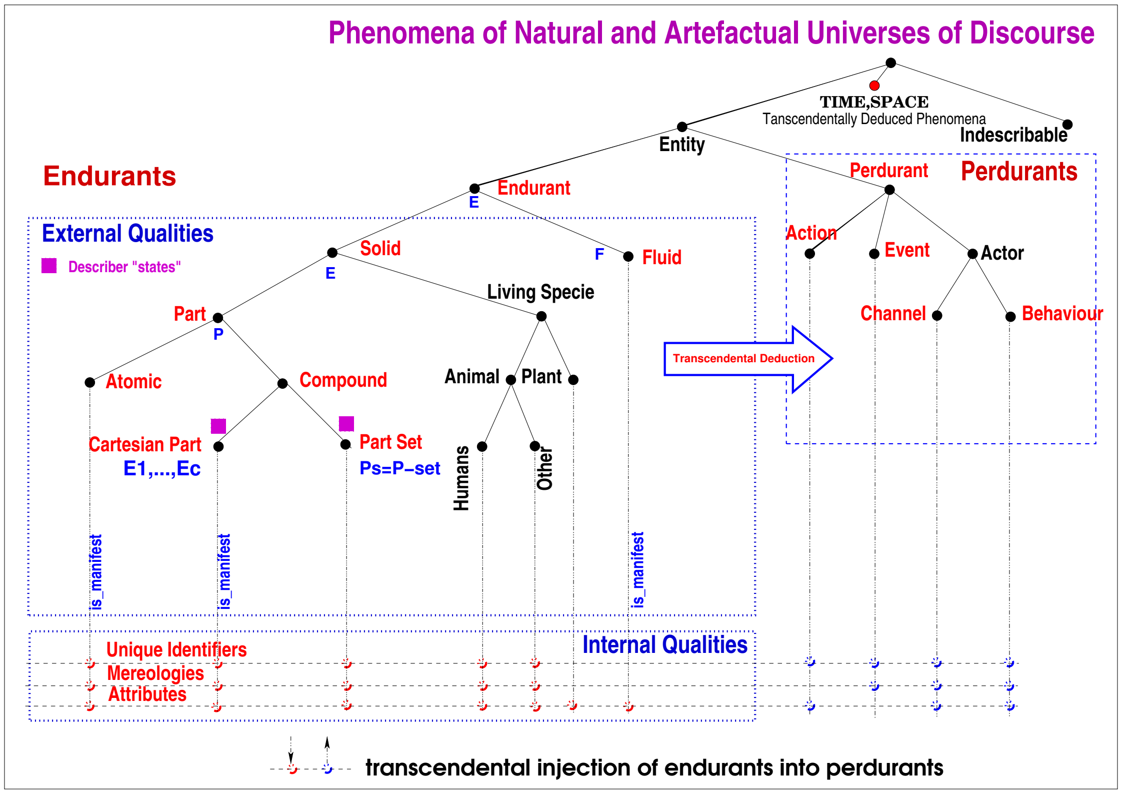

Fig. 1. A Domain Analysis & Description Ontology

The idea of Fig. 1 is the following:

- It presents a recipe for how to **analyze** a domain.
- You, the *domain* **analyzer** *cum describer*, are *'confronted'*[8] with, or by a domain.
- You have Fig. 1 in front of you, on a piece of paper, or in Your mind, or both.
- You are then asked, by the domain **analysis** & description method of this paper, to "start" at the uppermost •, just below and between the '**r**' and the first '**s**' in the main title, Phenomena of Natural and Artefactual Unive<u>rs</u>es of Discourse.

[6] An ontology is the philosophical study of being. It investigates what types of entities exist, how they are grouped into categories, and how they are related to one another on the most fundamental level (and whether there even is a fundamental level) [Wikipedia].

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

[8] By 'confronted' we mean: You are reading about it, in papers, in books, in postings on the Internet, visiting it, talking with domain stakeholders: professional people working "in" the domain; You may, yourself, "be an entity" of that domain!

- The **analysis** & description ontology of Fig. 1 then *directs* You to inquire as to whether the phenomenon – whichever You are ”looking at/reading about/...” – is either *rationally describable*, i.e., is an *entity* (`is_entity`) or is *indescribable*.
- That is, You are, in general, “positioned” at a bullet, •, labeled α, “below” which there may be two alternative bullets, one, β, to the right and one to the left, γ.
- It is Your decision whether the answer to the “query” that each such situation warrants, is yes, $\mathtt{is\_}\beta$, or no, $\mathtt{is\_}\gamma$.
- The characterizations of the concepts whose names, α, β, γ etc., are attached to the •s of Fig. 1 are given in the following sections.
- Whether they are precise enough to guide You in Your obtaining reasonable answers, “yes” or “no”, to the •ed queries is, of course, a problem. I hope they are.
- If Your answer is “yes”, then Your **analysis** is to proceed “down the tree”, usually indicated by “yes” or “no” answers.
- If one, or the other is a “leaf” of the ontology tree, then You have finished examining the phenomena You set out to **analyze**.
- If it is not a leaf, then further **analysis** is required.
- (We shall, in this paper, leave out the analysis and hence description of *living species*.)
- If an **analysis** of a phenomenon has reached one of the (only) two •’s, then the **analysis** at that • results in the domain describer **describing** some of the properties of that phenomenon.
- That **analysis** involves “setting aside”, for subsequent **analysis & description**, one or more [thus **analysis** etc.-pending] phenomena (which are subsequently to be tackled from the “root” of the ontology).

We do not [need to] prescribe in which order You analyze & describe the phenomena that has been “set aside”.

• • •

In Fig. 1 You will have noticed the positioning of the concepts of $\mathbb{TIME}$ and $\mathbb{SPACE}$ “right under” the *Phenomena* bullet •. These two concepts are neither endurants not perdurants. And they are not attributes of either. They can, however, as shown by Sørlander [48], be transcendentally deduced by rational reasoning.

4.2 Discussion of The Chosen Ontology

We shall in the following motivate the choice of the *ontological classification* reflected in Fig 1. We shall argue that this classification is not “an accidental choice”. In fact, we shall try justify the classification with reference to the philosophy of Kai Sørlander [45–48][9]. Kai Sørlander’s aim in these books is to examine *that which is absolutely necessary, inevitable, in any description of the world.* In [15, *Chapter 2*] we present a summary of Sørlander’s philosophy. In paragraphs, in the rest of this paper, marked Ontological Choice, we shall relate Sørlander’s philosophy’s “inevitability” to the ontology for studying domains.

5 The Name, Type and Value Concepts

Domain *modeling*, as well as *programming*, depends, in their *specification*, on *separation of concerns:* which kind of *values* are subjectable to which kinds of *operations*, etc., in order to achieve ease of *understanding* a model or a program, ease of *proving properties* of a model, or *correctness* of a program.

[9] The 2022 book, [47], is presently a latest in Kai Sørlander’s work. It refines and further develops the theme of the earlier, 1994–2016 books. [48] is an English translation of [47]

5.1 Names

We name things in order to refer to them in our speech, models and programs. Names of types and values in models and programs are usually not so-called "first-citizens", i.e., values that can be arguments in functions, etc. The "science of names" is interesting.[10] In `botanicalsociety.org.-za/the-science-of-names-an-introduction-to-plant-taxonomy` the authors actually speak of a "science of names" in connection with plant taxonomy: the "art" of choosing such names that reflect some possible classification of what they name.

5.2 Types

The type concept is crucial to programming and modeling.

Characterization 4. *Type*: A *type* is a class, i.e., a further undefined set, of values ("of the same kind") •

We name types.

Example 2. *Type Names*: Some examples of type names are:

- RT – the class of all road transport instances: the *Metropolitan London Road Transport*, the *US Federal Freeway System*, etc.
- RN – the class of all road net instances (within a road transport).
- SA – the class of all automobiles (within a road transport) •

You, the domain describer, choose type names. Choosing type names is a "serious affair". It must be done carefully. You can choose short (as above) or long names: Road_Transport, Road_Net, etc. We prefer short, but not cryptic names, like X, Y, Z, ... Names that are easy to *memorize*, i.e., *mnemonics*.

5.3 Values

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

6 Phenomena and Entities

Characterization 5. *Phenomena*: By a *phenomenon* we shall understand a fact that is observed to exist or happen •

Some phenomena are rationally describable – to some degree[11] – others are not.

Characterization 6. *Entities*: By an entity By an *entity* we shall understand a more-or-less rationally describable phenomenon •

Prompt 5 ***is_entity***: We introduce the informal presentation language predicate `is_entity`. It holds for phenomena ϕ if ϕ is describable •

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

[11] That is: It is up to the domain analyzer cum describer to decide as to how many rationally describable phenomena to select for analysis & description. Also in this sense one practices abstraction by "abstracting away" [the analysis & description of] phenomena that are irrelevant for the "current" (!) domain description.

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

Example 3. *Phenomena and Entities*: Some, but not necessarily all aspects of a river can be rationally described, hence can be still be considered entities. Similarly, many aspects of a road net can be rationally described, hence will be considered entities •

If You are not happy with this 'characterization', then substitute "rationally describable" with: *describable in terms of the endurants and perdurants brought forward in this paper: their external and internal qualities, unique identifiers, mereologies amd attributes, channels and behaviours!*

Ontological Choice 6 *Phenomena*: We choose to "initialize" our ontological "search" to a question of whether a phenomenon is rationally describable – based on the tenet of Kai Sørlander's philosophy, namely that "whatever" we postulate is either *true* or *false* and that *a principle of contradiction* holds: *whatever we so express can not both hold and not hold* •

Kai Sørlander then develops his inquiry – *as to what is absolutely necessary in any description of the world* – into the rationality of such descriptions necessarily be based on time and space and, from there, by a series of transcendental deductions, into a base in *Newton*'s physics. We shall, in a sense, stop there. That is, in the domain concept, such as we have delineated it, we shall not need to go into *Einsteinian* physics.

7 Endurants and Perdurants, II

We repeat our characterizations of endurants and perdurants.

7.1 Endurants

We repeat characterization 2.

Characterization 7. *Endurant*: Endurants are those quantities of domains that we can observe (see and touch), in *space*, as "complete" entities at no matter which point in *time* – "material" entities that persists, endures – capable of enduring adversity, severity, or hardship •

Example 4. *Endurants*: Examples of endurants are: a street segment [link], a street intersection [hub], an automobile •

Prompt 7 `is_endurant`: We introduce the informal presentation language predicate `is_endurant` to hold for entity `e` if `is_endurant(e)` holds •

7.2 Perdurants

We repeat characterization 3.

Characterization 8. *Perdurant*: Perdurants are those quantities of domains for which only a fragment exists, in *space*, if we look at or touch them at any given snapshot in *time* •

Example 5. *Perdurant*: A moving automobile is an example of a perdurant •

Prompt 8 `is_perdurant`: We introduce the informal presentation language predicate `is_perdurant` to hold for entity `e` if `is_perdurant(e)` holds •

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

7.3 Ontological Choice

The **ontological choice** of entities being “viewed” as either endurants or perdurants is motivated as follows: The concept of endurants can be justified in terms of Newton’s physics without going into kinematics, i.e., without including time considerations. The concept of perdurants can then, on one hand, be justified in terms of Newton’s physics now taking time into consideration, hence kinematics, and from there causality, etc.; and, on the other hand, and as we shall see, by transcendentally deducing perdurants from solid endurants •

8 External and Internal Endurant Qualities

The main contribution of this section is that of a calculus of domain analysis and description prompts. Two facets are being presented. Aspects of a domain science: of how we suggest domains can, and should, be viewed – ontologically. And aspects of a domain engineering: of how we suggest domains can, and should, be analyzed and described.

We begin by characterizing the two concepts: external and internal qualities.

Characterization 9. *External Qualities*: External qualities of endurants of a manifest domain are, in a simplifying sense, those we can see, touch and have spatial extent. They, so to speak, take form.

Characterization 10. *Internal Qualities*: Internal qualities are those properties [of endurants] that do not occupy *space* but can be measured or spoken about •

Perhaps we should instead label these two qualities tangible and intangible qualities.

Ontological Choice 9 *Rationality*: The rational, analytic philosophy issues of the inevitability of these qualities is this: (i) can they be justified as inevitable, and (ii) can they be suitably “separated”, i.e., both disjoint and exhaustive? Or are they merely of empirical nature? The choice here is also that we separate our inquiry into examining *both external and internal qualities* of endurants [not ‘either or’] •

8.1 External Qualities – Tangibles

Example 6. *External Qualities*: An example of external qualities of a domains is: the Cartesian[13] of sets of solid atomic street intersections, and of sets of solid atomic street segments, and of sets of solid automobiles of a road transport system where *Cartesian, sets, atomicity*, and *solidity* reflect external qualities •

8.1.1 The Universe of Discourse. The most immediate external quality of a domain is the “entire” domain – “itself” ! So any domain analysis starts by identifying that “entire” domain! By giving it a name, say UoD, for *universe of discourse*, Then describing it, in *narrative* form, that is, in natural language containing terms of professional/technical nature, the domain. And, finally, *formalizing* just the name: giving the name “status” of being a type name, that is, of the type of a class of domains whose further properties will be described subsequently.

Theorem 10. *The Universe of Discourse*:

Narration:

The name, and hence the type, of the domain is UoD
The UoD domain can be briefly characterized by ...

Formalization:

type UoD •

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

8.1.2 Solid and Fluid Endurants. Given then that there are endurants we now postulate that they are either [mutually exclusive] *solid* (i.e., discrete) or *fluid.*

Ontological Choice 11 *Solids vs. Fluids*: Here we [seem to] make a practical choice, not one based on a philosophical argument, one of logical necessity, but one based on empirical evidence. It is possible for endurants to either be solid or fluid; and here we shall not consider the case where solid [fluid] endurants, due to being heated [cooled], enters a fluid state [or vice versa] •

8.1.2.1 Solid cum Discrete Endurants.

Characterization 11. *Discrete cum Solid Endurants*: By a *solid* cum *discrete* endurant we shall understand an endurant which is separate, individual or distinct in form or concept, or, rephrasing, have body (or magnitude) of three-dimensions: length (or height), breadth and depth [39, *OED, Vol. II, pg. 2046*] •

Example 7. *Solid Endurants*: Pipeline system examples of solid endurants are *wells, pipes, valves, pumps, forks, joins* and *sinks* of pipelines. (These units may, however, and usually will, contain fluids, e.g., oil, gas or water.) •

Prompt 12 *`is_solid`*: We introduce the informal presentation language predicate `is_solid` to hold for endurant `e` if `is_solid(e)` holds •

8.1.2.2 Fluids.

Characterization 12. *Fluid Endurants*: By a *fluid endurant* we shall understand an endurant which is prolonged, without interruption, in an unbroken series or pattern; or, rephrasing: a substance (liquid, gas or plasma) having the property of flowing, consisting of particles that move among themselves [39, *OED, Vol. I, pg. 774*] •

Example 8. *Fluid Endurants*: Examples of fluid endurants are: *water, oil, gas, compressed air, smoke* •

Fluids are otherwise liquid, or gaseous, or plasmatic, or granular[14], or plant products, i.e., chopped sugar cane, threshed, or otherwise[15], et cetera. Fluid endurants will be analyzed and described in relation to solid endurants, viz. their "containers".

Prompt 13 *`is_fluid`*: We introduce the informal presentation language predicate `is_fluid` to hold for endurant `e` if `is_fluid(e)` holds •

8.1.3 Parts and Living Species Endurants. Given then that there are solid endurants we now postulate that they are either [mutually exclusive] *parts* or *living species.*

Ontological Choice 14 *Parts and Living Species*: With Sørlander, [48, *Sect. 5.7.1, pages 71–72*] we reason that one can distinguish between parts and living species •

8.1.3.1 Parts

Characterization 13. *Parts*: The non-living species solids are what we shall call parts •

Parts are the "work-horses" of man-made domains. That is, we shall mostly be concerned with the analysis and description of endurants into parts.

Example 9. *Parts*: Example 7, of solids, is an example of parts •

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

[15] See footnote 14.

Prompt 15 *`is_part`*: We introduce the informal presentation language predicate `is_part` to hold for solid endurants `e` if `is_part(e)` holds •

We distinguish between atomic and compound parts.

Ontological Choice 16 *Atomic and Compound Parts*: It is an empirical fact that parts can be composed from parts. That possibility exists. Hence we can [philosophy-wise] reason likewise •

— Atomic Parts.

Characterization 14. *Atomic Part*: By an *atomic part* we shall understand a part which the domain analyzer considers to be indivisible in the sense of not meaningfully consist of sub-parts •

Example 10. *Atomic Parts*: Examples of atomic parts are: hubs, H, i.e., street intersections; links, L, i.e., the stretches of roads between two neighbouring hubs; and automobiles, A:

type H, L, A •

Prompt 17 *`is_atomic`*: We introduce the informal presentation language predicate `is_atomic` to hold for parts `p` if `is_atomic(p)` holds •

— Compound Parts.

Characterization 15. *Compound Part*: Compound parts are those which are observed to [potentially] consist of several parts •

Example 11. *Compound Parts*: An example of a compound parts is: a road net consisting of a set of hubs, i.e., street intersections or "end-of-streets", and a set of links, i.e., street segments (with no contained hubs), is a Cartesian compound; and the sets of hubs and the sets of links are part set compounds •

Prompt 18 *`is_compound`*: We introduce the informal presentation language predicate `is_compound` to hold for parts `p` if `is_compound(p)` holds •

We, pragmatically, distinguish between Cartesian product- and set-oriented parts.

Ontological Choice 19 *Cartesians*: The Cartesian versus set parts is an empirical choice. It is not justified in terms of philosophy, but in terms of mathematics – of mathematical expediency ! •

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

Characterization 16. *Cartesians*: Cartesian parts are those compound parts which are observed to consist of two or more distinctly sort-named endurants (solids or fluids) •

Example 12. *Cartesians: Road Transport*: A road transport, rt:RT, is observed to consist of an aggregate of a road net, rn:RN, and a set of automobiles, SA, where the road net is observed, i.e., abstracted, as a Cartesian of a set of hubs, ah:AH, i.e., street intersections (or specifically designated points segmenting an otherwise "straight" street into two such), and a set of links, al:AL, i.e., street segments between two "neighbouring" hubs.

```
type
    RT, RN, SA, AH = H-set, AL = L-set
value
    obs_RN: RT → RN, obs_SA: RT → SA,, obs_AH: RN → AH, obs_AL: RN → AL  •
```

Prompt 20 *`is_Cartesian`*: We introduce the informal presentation language predicate `is_Cartesian` to hold for compound parts `p` if `is_Cartesian(p)` holds •

Once a part, say p:P, has been analyzed into a Cartesian, we inquire as to the type names of the endurants[16] of which it consists. The inquiry: `record_Cartesian_part_type_names(p:P)`, we decide, then yields the type of the constituent endurants.

Prompt 21 *record-Cartesian-part-type-names*:

value
  `record_Cartesian_part_type_names`: P → $\mathbb{T}$**-set**
  `record_Cartesian_part_type_names`(p) **as** {ηE1,ηE2,...,ηEn} •

Here $\mathbb{T}$ is the **name** of the type of all type names, and ηEi is the **name** of type Ei.

Please note the novel introduction of type names as values. Where a type identifier, say T, stands for, denotes, a class of values of that type, ηT denotes the name of type T.

Please also note that `record_Cartesian_part_type_names` is not a description language construct. It is an analysis language, i.e., an informal natural language, here English, construct. As such it is being used by the domain analyzer cum describer who "applies" it to an observed endurant and notes down, in her mind or jots it on a scratch of paper, her decision as to appropriate [new] type names.

Example 13. *Cartesian Parts*: The Cartesian parts of a road transport, rt:RT, is thus observed to consists of

- an aggregate of a road net, rn:RN, and
- an aggregate set of automobiles, sa:SA:

that is:

- `record_Cartesian_part_type_names(rt:RT)` = {ηRN,ηSA}

where the type name ηRT was – and the type names ηRN and ηSA are – coined, i.e., more-or-less freely chosen, by the domain analyzer cum describer •

— Part Sets.

Characterization 17. *Part Sets*: Part sets are those compound parts which are observed to consist of an indefinite number of zero, one or more parts •

Prompt 22 *`is_part_set`* : We introduce the informal presentation language predicate `is_part_set` to hold for compound parts `e` if `is_part_set(e)` holds •

Once a part, say e:E, has been analyzed into a part set we inquire as to the set of parts and their type of which it consists. The inquiry: `record_part_set_part_type_names`, we decide, then yields the (single) type of the constituent parts.

Prompt 23 *record-part-set-part-type-names*:

value
  `record_part_set_part_type_names`: E → $\mathbb{T}$Ps×$\mathbb{T}$P
  `record_part_set_part_type_names`(e:E) **as** (η Ps,η P) •

Here the name of the value, e, and the type names η Ps and η P are coined, i.e., more-or-less freely chosen, by the domain analyzer cum describer •

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

Please also note that `record_part_set_part_type_names` is not a description language construct. It is an analysis language, i.e., an informal natural language, here English, construct. As such it is being used by the domain analyzer cum describer who "applies" in to an observed endurant and notes down, in her mind or jots it on a scratch of paper, her decision as to appropriate [new] type names.

Example 14. *Part Sets: Road Transport*: The road transport contains a set of automobiles. The part set type name has been chosen to be SA. It is then determined (i.e., analyzed) that SA is a set of Automobile of type A

- `record_part_set_part_type_names(sa:SA)` = (η As,η A) •

— Compound Observers.

Once the domain analyzer cum describer has decided upon the names of atomic and compound parts, **obs**_erver functions can be applied to Cartesian and part set, e:E, parts:

Theorem 24. *Describe Cartesians and Part Set Parts*

```
value
    let {η P1,η P2,...,η Pn} = record_Cartesian_part_type_names(e:E) in
    "type
        P1, P2, ..., Pn;
    value
        obs_P1: E→P1, obs_P2: E→P2,...n obs_Pn: E→Pn "
                                                                    [respectively:]
    let (η Ps,η P) = record_part_set_part_type_names(e:E) in
    "type
        P, Ps = P-set,
    value
        obs_Ps: E→Ps "
    end end •
```

The "..." texts are the RSL texts "generated", i.e., written down, by the domain describer. They are *domain model specification units*. The "surrounding" RSL-like texts are not written down as phrases, elements, of the domain description. They are elements of the domain describers' "notice board", and, as such, elements of the development of domain models. We have introduced a core domain modeling tool the **obs**_... observer function, one to be "applied" mentally by the domain describer, and one that appears in (RSL) domain descriptions The **obs**_... observer function is "applied" by the domain describer, it is not a computable function.

Please also note that `Describe Cartesians and Part Set Parts` schema, 24, is not a description language construct. It is an analysis language, i.e., an informal natural language, here English, construct. As such it is being used by the domain analyzer cum describer who "applies" in to an observed endurant and notes down, but now in a final form, elements, that is *domain description units*.

• • •

A major step of the development of domain models has now been presented: that of the analysis & description of the external qualities of domains.

Schema 24 is the first manifestation of the domain analysis & description method leading to actual domain description elements.

From unveiling a *science of domains* we have "arrived" at an *engineering of domain descriptions*.

8.1.4 States.

Characterization 18. *States*: By a *state* we shall mean any subset of the parts of a domain •

Example 15. *Road Transport State*:

variable
  hs:AH := **obs**_AH(**obs**_RN(rt)),
  ls:AL := **obs**_AL(**obs**_RN(rt)),
  as:SA := **obs**_SA(rt),
  σ:(H|L|A)**-set** := $hs \cup ls \cup as$ •

We have chosen to model domain states as **variable**s rather than as **value**s. The reason for this is that the values of monitorable, including biddable part attributes[17] can change, and that domains are often extended and "shrunk" by the addition, respectively removal of parts:

Example 16. *Road Transport Development*: adding or removing hubs, links and automobiles •

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

8.1.5 Validity of Endurant Observations. We remind the reader that the **obs**_erver functions, as all later such functions: **uid**_-, **mereo**_- and **attr**_-functions, are applied by humans and that the outcome of these "applications" is the result of human choices, and possibly biased by inexperience, taste, preference, bias, etc. How do we know whether a domain analyzer & describer's description of domain parts is valid? Whether relevantly identified parts are modeled reasonably wrt. being atomic, Cartesians or part sets Whether all relevant endurants have been identified? Etc. The short answer is: we never know. Our models are conjectures and may be refuted [42]. A social process of peer reviews, by domain stakeholders and other domain modelers is needed – as may a process of verifying[18] properties of the domain description held up against claimed properties of the (real) domain.

8.1.6 Summary of Endurant Analysis Predicates. Characterizations 6–17 imply the following analysis predicates (Char.: δ, Page π):

- is_entity, $\delta 6\, \pi\, 10$
- is_endurant, $\delta 7\, \pi\, 11$
- is_perdurant, $\delta 8\, \pi\, 11$
- is_solid, $\delta 11\, \pi\, 13$
- is_fluid, $\delta 12\, \pi\, 13$
- is_part, $\delta 13\, \pi\, 13$
- is_atomic, $\delta 14\, \pi\, 14$
- is_compound, $\delta 15\, \pi\, 14$
- is_Cartesian, $\delta 16\, \pi\, 14$
- is_part_set, $\delta 17\, \pi\, 15$

We remind the reader that the above predicates represent "formulas" in the presentation, **not** the description, language. They are not `RSL` clauses. They are in the mind of the domain analyzers cum describers. They are "executed" by such persons. Their result, whether **true**, **false** or **chaos**[19], are noted by these persons and determine their next step of domain analysis.

8.1.7 "Trees are Not Recursive". A 'fact', that seems to surprise many, is that parts are not "recursive". Yes, in all our domain modeling experiments, [19], we have not come across the need for recursively observing compound parts. Trees, for example, are not recursive in this sense. Trees have roots. Sub-trees not. Banyan trees[20] have several "intertwined trees". But it would be 'twisting' the modeling to try fit a description of such trees to a 'recursion wim'! Instead, trees are defined as nets, such as are road nets, where these nets then satisfy certain constraints [19, *Chapter B*] – usually modeled by a mereology, see Sect. 8.2.2.

[17] The concepts of monitorable, including biddable part attributes is treated in Sect. 8.2.3.2.
[18] testing, model checking and theorem proving
[19] The outcome of applying an analysis predicate of the prescribed kind may be **chaos** if the prerequisites for its application does not hold.
[20] https://www.britannica.com/plant/banyan

8.2 Internal Qualities – Intangibles

The previous section has unveiled an ontology of the external qualities of endurants. The unveiling consisted of two elements: a set of analysis predicates, predicates 6–17, and analysis functions, schemas 21–23, and a pair of description functions, schema 24.

The application of description functions result in RSL text.

That text conveys certain properties of domains: that they consists of such-and-such endurants, notably parts, and that these endurants "derive" from other endurants. But the RSL description texts do not *"give flesh & blood"* to these endurants. Questions like: *'what are their spatial extents ?'*, *'how much do the weigh ?'*, *'what colour do they have ?'*, et cetera, are left unanswered. In the present section we shall address such issues. We call them *internal qualities*.

Characterization 19. *Internal Qualities*: Internal qualities are those properties [of endurants] that do not occupy *space* but can be measured or spoken about •

Example 17. *Internal qualities*: Examples of internal qualities are the *unique identity* of a part, the *mereological relation* of parts to other parts, and the endurant *attributes* such as temperature, length, colour, etc. •

This section therefore introduces a number of domain description tools:

- **uid_**: the unique identifier observer of parts;
- **mereo_**: the mereology observer of parts;
- **attr_**: (zero,) one or more attribute observers of endurants; and
- **attributes_**: the attribute query of endurants.

8.2.1 Unique Identity.

Ontological Choice 25 *Unique Identity*: We postulate that separately discernible parts have unique identify. The issue, really, is a philosophical one. We refer to [15, *Sects. 2.2.2.3–2.2.2.4, pages 14–15*] for a discussion of the existence and uniqueness of entities •

Characterization 20. *Unique Identity*: A unique identity is an immaterial property that distinguishes any two *spatially* distinct solids[21] •

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

Theorem 26. *Describe-Unique-Identity-Part-Observer*

```
"type
      P,PI
  value
      uid_P: P → PI" •
```

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

Example 18. *Unique Road Transport Identifiers*: The unique identifierss of a road transport, rt:RT, consists of the unique identifiers of the

[21] For pragmatic reasons we do not have to speculate as to whether "bodies" of fluids can be ascribed unique identity. The pragmatics is that we, in our extensive modeling experiments have not found a need for such ascription !

- road transport – rti:RTI,
- (Cartesian) road net – rni:RNI,
- (set of) automobiles – sa:SAI,
- automobile, ai:AI,
- (set of) hubs, hai:AHI,
- (set of) links, lai:LAI,
- hub, hi:HI, and
- link, li:LI,

where the type names are all coined, i.e., more-or-less freely chosen, by the domain analyzer cum describer – though, as You can see, these names were here formed by "suffixing" Is to relevant part names •

We have thus introduced a core domain modeling tool the **uid**_... observer function, one to be "applied" mentally by the domain describer, and one that appears in (RSL) domain descriptions The **uid**_... observer function is "applied" by the domain describer, it is not a computable function.

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

Example 19. *Road Transport Uniqueness*:

variable
 hs_{uids}:HI-**set** := { **uid**_H(h) | h:H•u ∈ σ }
 ls_{uids}:LI-**set** := { **uid**_L(l) | l:L•u ∈ σ }
 as_{uids}:AI-**set** := { **uid**_A(a) | a:A•u ∈ σ }
 σ_{uids}:(HI|LI|AI)-**set** := { **uid**_(H|L|A)(u) | u:(H|L|A)•u ∈ σ }
axiom
 □ **card** σ = **card** σ_{uids} • For σ see Sect. 8.1.4.

We have chosen, for the same reason as given in Sect. 8.1.4, to model a unique identifier state. The □ [*always*] prefix in the **axiom** then expresses that changes of parts or addition of parts to and deletions of parts from the domain shall maintain their uniqueness over time (i.e., always).

8.2.2 Mereology. The concept of mereology is due to the Polish mathematician, logician and philosopher Stanisław Leśniewski (1886–1939) [50, 11].

Characterization 21. *Mereology*: Mereology is a theory of [endurant] part-hood relations: of the relations of an [endurant] parts to a whole and the relations of [endurant] parts to [endurant] parts within that whole •

Ontological Choice 27 *Mereology*: Stanisław Leśniewski was not satisfied with Bertrand Russell's "repair" of Gottlob Frege's axiom systems for set theory. Instead he put forward his axiom system for, as he called it, mereology. Both as a mathematical theory and as a philosophical reasoning •

Example 20. *Mereology*: Examples of mereologies are that a link is topologically *connected* to exactly one or, usually, two specific hubs, that hubs are *connected* to zero, one or more specific links, and that links and hubs are *open* to the traffic of specific subsets of automobiles •

Mereologies can be expressed in terms of unique identifiers.

Example 21. *Mereology Representation*: For our 'running road transport example' the mereologies of links, hubs and automobiles can thus be expressed as follows:

- **mereo**_L(l) = {hi′,hi″} where hi,hi′,hi″ are the unique identifiers of the hubs that the link connects, i.e., are in hs_{uids};
- **mereo**_H(h) = {li_1,li_2,...,li_n} where li_1,li_2,...,li_n are the unique identifiers of the links that are imminent upon (i.e., emanates from) the hub, i.e., are in ls_{uids}; and
- **mereo**_A(a) = {ri_1,ri_2,...,ri_m} where ri_1,ri_2,...,ri_m are unique identifiers of the road (hub and link) elements that make up the road net, i.e., are in $hs_{uids} \cup ls_{uids}$ •

Once the unique identifiers of all parts of a domain has been described we can analyses and describe their mereologies. The inquiry: **mereo_P**(p) yields a mereology type (name), say PMer, and its description[22]:

Theorem 28. *Describe-Mereology*

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

where $\mathcal{M}$(PI1,PI2,...,PIm) is a type expression over unique identifier types of the domain; **mereo**_P is the mereology observer function for parts p:P; and $\mathcal{A}$(pm:PMer) is an axiom that secures that the unique identifiers of any part are indeed of parts of the domain.

8.2.3 Attributes.

Attributes are what finally gives "life" to endurants: The external qualities "only" named and gave structure to their atomic or compound types. The internal qualities of uniqueness and mereology are intangible quantities. The internal quality of attributes gives "flesh & blood" to endurants: they let us express endurant properties that we can more easily, i.e., concretely, relate to.

8.2.3.1 General

Characterization 22. *Attributes*: Attributes are properties of endurants that can be measured either physically (by means of length (ruler) and spatial quantity measuring equipment, electronically, chemically, or otherwise) or can be objectively spoken about •

Ontological Choice 29 *Attributes*: First some empirical observation: in reasoning about "the world around us" we express its properties in terms of predicates. These predicates, for example: *"that building's wall is red"*, *building* refers to an endurant part whereas *wall* and *red* refers to attributes. Now the "rub": endurant attributes is what give *"flesh & blood"* to domains •

Attributes are of types and, accordingly have values.

We postulate an informal domain analysis function, `record_attribute_type_names`: The domain analyzer, in observing a part, p:P, analyzes it into the set of attribute names of parts p:P

Theorem 30. *record-attribute-type-names*

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

Example 22. *Road Net Attributes, I*: Examples of attributes are: 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; and hubs have state spaces, hω:HΩ: the set of hub states "signaling" which states are open/closed, i.e., green/red; links that have lengths, LEN; and automobiles have road net positions, APos, either *at a hub*, atH, or *on a link*, onL, some fraction, f:**Real**, down a link, identified by li, from a hub, identified by fhi, towards a hub, identified by thi. Hubs and links have *histories:* time-stamped, chronologically ordered sequences of automobiles entering and leaving links and hubs, with automobile histories similarly recording hubs and links entered and left.

[22] Cf. Sect. 8.1.3.1

```
type
    HΣ = (LI×LI)-set
    HΩ = HΣ-set
    LEN = Nat m
    APos = atH | onL
    atH :: HI
    onL :: LI × (fhi:HI × f:Real × thi:HI)
    HHis,LHis = (𝕋𝕀𝕄𝔼×AI)*
    AHis = (𝕋𝕀𝕄𝔼×(HI|LI))*
value
    attr_HΣ: H → HΣ
    attr_HΩ: H → HΩ
    attr_LEN: L → LEN
    attr_APos: A → APos
    attr_HHis: H → HHis
    attr_LHis: L → LHis
    attr_AHis: A → AHis
axiom
    ∀ (li,(fhi,f,thi)):onL • 0<f<1
        ∧li∈ls_uids∧{fhi,thi}⊆hs_uids∧... •
```

Theorem 31. *Describe-endurant-attributes(e:E)*

```
let {ηA1,ηA2,...,ηAn} = record_attribute_type_names(e:E) in
"type
     A1, A2, ..., An
  value
     attr_A1: E → A1, attr_A2: E → A2, ..., attr_An: E → An
  axiom
     ∀ a1:A1, a2:A2, ..., an:An: 𝒜(a1,a2,...,an) "
end •
```

8.2.3.2 Michael A. Jackson's Attribute Categories Michael A. Jackson [36] has suggested a hierarchy of attribute categories:from *static* (is_static[23]) to *dynamic* (is_dynamic[24]) values – and within the dynamic value category: *inert* values (is_inert[25]), *reactive* values (is_reactive[26]), *active* values (is_active[27]) – and within the dynamic active value category: *autonomous* values (is_autonomous[28]), *biddable* values (is_biddable[29]), and *programmable* values (is_programmable[30]) . We postulate informal domain analysis predicates, "performed" by the domain analyzer:

```
value
    is_static,is_autonomous,is_biddable,is_programmable [etc.]: ηT→Bool
```

We refer to [36] and [15] [*Chapter 5, Sect. 5.4.2.3*] for details. We suggest a minor revision of Michael A. Jackson's attribute categorization, see left side of Fig. 2. We single out the inert from the ontology of Fig. 2, left side. Inert attributes seem to be "set externally" to the endurant. So we now distinguish between is_external and is_internal dynamic attributes. We summarize Jackson's attribute and our revised categorization in Fig. 2.

This distinction has [pragmatic] consequences for how we treat arguments of the behaviours of parts, cf. Sect. 10.5.1 (page 28).

Example 23. *Road Net Attributes, II*: The link length and hub state space attributes are static, hub states and automobile positions programmable. Automobile speed and acceleration attributes, which we do not model, are monitorable •

The attributes categorization determines, in the next major section on perdurants, the treatment of hub, link and automobile behaviours.

[23] static: values are constants, cannot change
[24] dynamic: values are variable, can change
[25] inert: values can only change as the result of external stimuli where these stimuli prescribe new values
[26] reactive: values, if they vary, change in response to external stimuli, where these stimuli either come from outside the domain of interest or from other endurants.
[27] active: values can change (also) on their own volition
[28] autonomous: values change only "on their own volition"; the values of an autonomous attributes are a "law onto themselves and their surroundings".
[29] biddable: values are prescribed but may fail to be observed as such
[30] programmable: values can be prescribed

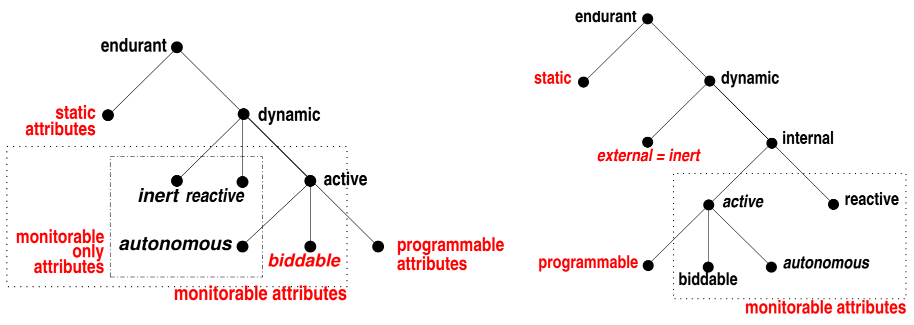

Fig. 2. Michael Jackson's [Revised] Attribute Categories

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

```
value
    p:P
    tns = record_attribute_type_names(p)

    static_attributes: $\eta T$-set → $\eta T$-set
    static_attributes(tns) ≡ { $\eta$tn | $\eta$tn:$\eta T$ • $\eta$tn ∈ tns ∧ is_static(tn) }

    inert_attributes: $\eta T$-set → $\eta T$-set
    inert_attributes(tns) ≡ { $\eta$tn | $\eta$tn:$\eta T$ • $\eta$tn ∈ tns ∧ is_inert(tn) }

    monitorable_attributes $\eta T$-set → $\eta T$-set
    monitorable_attributes(tns) ≡ { $\eta$tn | $\eta$tn:$\eta T$ • $\eta$tn ∈ tns ∧ is_monitorable(tn) }

    programmable_attributes $\eta T$-set → $\eta T$-set
    programmable_attributes(tns) ≡ { $\eta$tn | $\eta$tn:$\eta T$ • $\eta$tn ∈ tns ∧ is_programmable(tn) }

    is_monitorable: T → Bool
    is_monitorable(t) ≡ ∼is_static(t) ∧ ∼is_inert(t) ∧ ∼is_programmable(t)
```

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

8.3 Intentional Pull

Ontological Choice 32 *Intentional Pull*: In [46, *pages 167–168*] Sørlander argues wrt. *"how can entities be the source of forces ?"* and thus reasons for *gravitational pull*. That same kind of reasoning, with proper substitution of terms, leads us to the concept of *intentional pull* •

Two or more parts of different sorts, but with overlapping sets of intents[31] may excert 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*

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

intent, ι. *Manifestations* of these, their common intent must somehow be *subject to constraints*, and these must be *expressed predicatively*.

When a compound artifact models "itself" as put together with a number of other endurants then it does have an intentionality and the components' individual intentionalities does, i.e., shall relate to that.

Example 24. *Road Transport Intentionality*: *Automobiles* include the *intent:* `transport`, and so do *hubs* and *links*. *Manifestations* of `transport` are reflected in *hubs, links* and *automobiles* having the *history* attribute. The *intentional "pull"* of these manifestations is this: For every automobile, if it records being in some hub or on some link at time τ, then the designated hub, respectively link, records exactly that automobile; and vice versa: for every hub [link], if it records the visit of some automobile at time τ, then the designated automobile records exactly that hub [link]. We leave the formalization of the above to the reader •

Example 25. *Double-entry Bookkeeping*: Another example of intentional "pull" is that of double-entry bookkeeping. Here the income/expense ledger must balance the actives/passives ledger •

Example 26. *The Henry George Theorem.*: The Henry George theorem states that under certain conditions, aggregate spending by government on public goods will increase aggregate rent based on land value (land rent) more than that amount, with the benefit of the last marginal investment equaling its cost •[32,33]

8.4 Summary of Endurants

We have completed our treatment of endurants. That treatment was based on an ontology for the observable phenomena of domains – such as we have delineated the concept of domains. The treatment was crucially based on an ontology for the structure of domain phenomena, and, in a sense, "alternated" between analysis predicates, analysis functions, and description functions. We have carefully justified this ontology in **'Ontological Choice'** paragraphs

9 Facets

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

[32] Stiglitz, Joseph (1977). "The Theory of Local Public Goods". In Feldstein, M.S.; Inman, R.P. (eds.). The Economics of Public Services. Palgrave Macmillan, London. pp. 274333. doi:10.1007/978-1-349-02917-4_12. ISBN 978-1-349-02919-8.

[33] Henry George (September 2, 1839 – October 29, 1897) was an American political economist and journalist. His writing was immensely popular in 19th-century America and sparked several reform movements of the Progressive Era. He inspired the economic philosophy known as Georgism, the belief that people should own the value they produce themselves, but that the economic value of land (including natural resources) should belong equally to all members of society. George famously argued that a single tax on land values would create a more productive and just society.

We leave it to [15, *Chapter 8, pages 205–240*] to detail the principles, procedures, techniques and tool for describing facets.

These are the facets that we have so far identified:

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

9.1 Intrinsics

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

9.2 Support Technology

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

9.3 Rules & Regulations

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

9.4 Scripts

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

A special "subclass" of scripts are those of commands.

Commands are syntactic entities. Semantically they denote state changes. The state referred to is the state of the domain. Domain facets, as a wider concept than just commands, were first treated in [16, *Chapter 8*] which places facets in the wider context of domain modeling. Commands are but just one of the many kinds of script facets.

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

9.5 License Languages

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

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

9.6 Management & Organization

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

9.7 Human Behaviour

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

10 Perdurant Concepts

The main contribution of this section is that of *transcendentally deducing* perdurants from endurant parts, in particular *behaviours* "of" parts.

Major perdurants are those of actions, events and behaviours with behaviours generally being sets of sequences of actions, events and behaviours.

10.1 "Morphing" Parts into Behaviours

As already indicated we shall transcendentally deduce (perdurant) behaviours from those (endurant) parts which we, as domain analyzers cum describers, have endowed with all three kinds of internal qualities: unique identifiers, mereologies and attributes. We shall use the `CSP` [33] constructs of `RSL` (derived from `RSL` [28]) to model concurrent behaviours.

10.2 Transcendental Deduction

Characterization 23. *Transcendental*: By transcendental we shall understand the philosophical notion: the a priori or intuitive basis of knowledge, independent of experience •

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

Characterization 24. *Transcendental Deduction*: By a transcendental deduction we shall understand the philosophical notion: a transcendental "conversion" of one kind of knowledge into a seemingly different kind of knowledge •

Example 27. *Transcendental Deductions – Informal Examples*: We give some intuitive examples of transcendental deductions. They are from the "domain" of programming languages. There is the syntax of a programming language, and there are the programs that supposedly adhere to this syntax. Given that, the following are now transcendental deductions.

The software tool, *a syntax checker*, that takes a program and checks whether it satisfies the syntax, including the statically decidable context conditions, i.e., the *statics semantics* – such a tool is one of several forms of transcendental deductions.

The software tools, *an automatic theorem prover* and *a model checker*, for example `SPIN` [34], that takes a program and some `theorem`, respectively a `Promela` statement, and proves, respectively checks, the program correct with respect the theorem, or the statement.

A *compiler* and an *interpreter* for any programming language.

Yes, indeed, any *abstract interpretation* [25] reflects a transcendental deduction: firstly, these examples show that there are many transcendental deductions; secondly, they show that there is no single-most preferred transcendental deduction •

Ontological Choice 33 *Transcendental Deduction of Behaviours from Parts*: So this, then, is, in a sense, our "final" ontological choice: that of transcendentally deducing behaviours from parts •

10.3 Actors – A Synopsis

This section provides a summary overview.

Characterization 25. *Actors*: An actor is anything that can initiate an **action**, **event** or **behaviour** •

10.3.1 Action.

Characterization 26. *Actions*: An action is a function that can purposefully change a state •

Example 28. *Road Net Actions*: These are some road transport actions: an automobile leaving a hub, entering a link; leaving a link, entering a hubs; entering the road net; and leaving the road net •

10.3.2 Event.

Characterization 27. *Events*: An event is a function that surreptitiously changes a state •

Example 29. *Road Net Events*: These are some road net events: The blocking of a link due to a mud slide; the failing of a hub traffic signal due to power outage; an automobile failing to drive; and the blocking of a link due to an automobile accident •

We shall not formalize events.

10.3.3 Behaviour.

Characterization 28. *Behaviours*: Behaviours are sets of sequences of actions, events and behaviours •

Concurrency is modeled by the ***sets*** of sequences. Synchronization and communication of behaviours are effected by CSP ***output/inputs***: ch[{i,j}] !value/ch[{i,j}] ?.

Example 30. *Road Net Traffic*: Road net traffic can be seen as a behaviour of all the behaviours of automobiles, where each automobile behaviour is seen as sequence of start, stop, turn right, turn left, etc., actions; of all the behaviours of links where each link behaviour is seen as a set of sequences (i.e., behaviours) of "following" the link entering, link leaving, and movement of automobiles on the link; of all the behaviours of hubs (etc.); of the behaviour of the aggregate of roads, viz. *The Department of Roads*, and of the behaviour of the aggregate of automobiles, viz, *The Department of Vehicles*.

10.4 Channel

Characterization 29. *Channel*: A channel is anything that allows synchronization and communication of values between behaviours •

Theorem 34. *Channel*

We suggest the following schema for describing channels:

"**channel** { ch[{ui,uj}] | ui,ij:UI • ... } M

where ch is the describer-chosen name for an array of channels, ui,uj are channel array indices of the unique identifiers, UI, of the chosen domain •

Example 31. *Road Transport Interaction Channel*:

channel { ch[{ui,uj}] | {ui,ij}:(HI|LI|AI)-**set** • ui≠uj∧{ui,uj}$\subseteq\sigma_{uids}$ } M

Channel array ch is indexed by a "pair" of distinct unique part identifiers of the domain. We shall later outline M, the type of the "messages" communicated between behaviours •

10.5 Behaviours & Actions

We single out the perdurants of behaviours – as they relate directly to the parts of Sect. 8. The treatment is "divided" into three sections.

10.5.1 Behaviour Signature.

Theorem 35. *Behaviour Signature*

By the ***behaviour signature***, for a part p, we shall understand a pair: the name of the behaviour, B_p, and a function type expression as indicated:

value

B_p: $\mathsf{Uid}_p\rightarrow$[34] $\mathsf{Mereo}_p\rightarrow\mathsf{Sta\_Vals}_p\rightarrow\mathsf{Inert\_Vals}_p\rightarrow\mathsf{Mon\_Refs}_p\rightarrow\mathsf{Prgr\_Vals}_p\rightarrow$ { ch[{i,j}] | ... } **Unit**

We explain:

[34] We have Schönfinckel'ed `https://en.wikipedia.org/wiki/Moses_Schönfinkel#Further_reading` (Curried `https://en.wikipedia.org/wiki/Currying`) the function type

- Uid_p is the type of unique identifiers of part p, **uid**_P(p) = Uid_p;
- Mereo_p is the type of the mereology of part p, **mereo**_P(p) = Mereo_p;
- Sta_Vals_p is a Cartesian of the type of inert attributes of part p. Given `record_attribute_type_names(p) static_attributes(record_attribute_type_names(p))` yields Sta_Vals_p;
- Inert_Vals_p is a Cartesian of the type of static attributes of part p. Given `record_attribute_type_names(p) inert_attributes(record_attribute_type_names(p))` yields Inert_Vals_p;
- Mon_Refs_p is a Cartesian of the **attr**_ibute observer functions of the types of monitorable attributes of part p. Given `record_attribute_type_names(p)` analysis function `monitorable_attributes(record_attribute_type_names(p))` yields Mon_Vals_p;
- Prgr_Vals_p is a Cartesian of the type of programmable attributes of part p. Given `record_attribute_type_names(p)` analysis function `programmable_attributes(record_attribute_type_names(p)).` yields Prgr_Vals_p;
- { ch[{i,j}] | ... } specifies the channels over which part p behaviours, B_p, may communicate; and
- **Unit** is the type name for the () value[35] •

The Cartesian arguments may "degenerate" to the non-Cartesian of no, or just one type identifier, In none, i.e., (), then () may be skipped. If one, e.g., (a), then (a) is listed.

Example 32. *Road Transport Behaviour Signatures*:

value
hub: HI→MereoH→(HΩ×...)→(...)→(HHist×...)
  →{ch[{**uid**_H(p),ai}]|ai:AI•ai∈as_{uid}} **Unit**
link: LI→MereoL→(LEN×...)→(...)→(LHist×...)
  →{ch[{**uid**_L(p),ai}]|ai:AI•ai∈as_{uid}} **Unit**
automobile: AI→MereoA→(...)→(**attr**_AVel×**attr**_HAcc×...)→(APos×AHist×...)
  →{ch[{**uid**_H(p),ri}]|ri:(HI|LI)•ri∈hs_{uid}∪ls_{uid}} **Unit**

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

10.5.2 Inert Arguments: Some Examples.

Let us give some examples of inert attributes of automobiles. (i) Driving uphill, one a level road, or downhill, excert some `inert` "drag" or "pull". (ii) Velocity can be treated as a reactive attribute – but it can be [approximately] calculated on the basis of, for example, these `inert` attributes: drag/pull and accelerator pedal pressure, and the `static` engine power attribute.

10.5.3 Behaviour Definitions.

A typical, informal rendition of abstracted behaviours, `BA, BC, BD, ...` is shown in Fig. 10.5.3.

Figure 10.5.3 should be understood as follows:[36] The **bold faced** labels **BA, BB, BC, ...** are meant to designate behaviours. The **black arrows**, from behaviour **Bx** to behaviour **By** are meant to designate CSP-like *communications* from **Bx** to **By**. The **open arrows** ("white"), from behaviour **Bx** to behaviour **By** are meant to designate

[35] – You may "read' () as the value yielded by a statement, including a never-terminating function
[36] The explanation of Fig. 10.5.3 is in now way an attempt to explain the semantics of behaviours. That is left to the $\mathtt{RSL}^+$ formalization's.

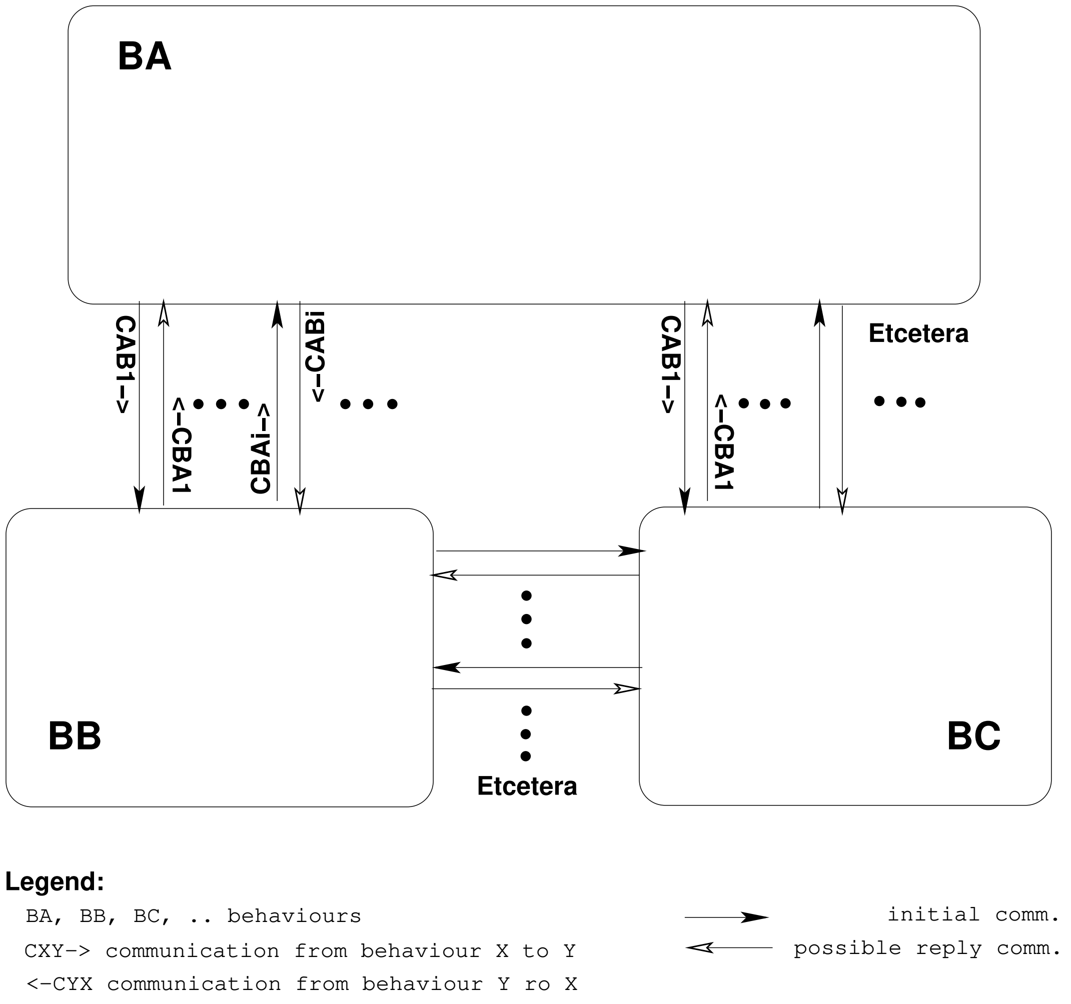
BA
CAB1–>
<–CBA1
CBAi–>
<–CABi
CAB1–>
<–CBA1
Etcetera
BB
BC
Etcetera
Legend:
BA, BB, BC, .. behaviours
CXY-> communication from behaviour X to Y
<-CYX communication from behaviour Y ro X
initial comm.
possible reply comm.

Fig. 3. Communicating Behaviours

possible CSP-like *communications* from **Bx** to **By**. These latter communications, the "possible" ones, are then thought of as *in response* to the "earlier", in the figure: "immediately prior, next to" communication from **Bx** to **By**.

Figure 10.5.3 is now given a more precise "meaning" – with this "meaning" suggesting a general "pattern" for behaviour definitions:

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

```
value
1a.  B(bi)(mereo)(stat)(mon)(prg) ≡
1b.        pro_active_B(bai)(mereo)(stat)(mon)(prg)
1c.    ⌈⌉ re_active_B(bai)(mereo)(stat)(mon)(prg)
```

2. ι1b π29 The pro-active behaviour (B) internal deterministically ($\sqcap$) choosing between a number of initiating actions:
   (a) action 1,
   (b) action 2,
   (c) ...,
   (d) action n.

value

```
2., ι1b π29. pro_active_B(bi)(mereo)(stat)(mon)(prg) ≡
2a.               B_action_1(bi)(mereo)(stat)(mon)(prg)
2b.             ⊓ B_action_2(bi)(mereo)(stat)(mon)(prg)
2c.             ⊓ ...
2d.             ⊓ B_action_m(bi)(mereo)(stat)(mon)(prg)
```

3. ι1b π29. The `responding` behaviour (B) reacts to a number of such initiating actions by
   (a) external non-deterministically ($\lceil\!\rceil$) offering to accept messages from responding behaviours,
   (b) and then performing corresponding actions.

value

```
3a., ι1b π29. respond_B(bi)(mereo)(stat)(mon)(prg) ≡
3a.               let msg = ⌈⌉{ ch[{bj,bi}] ? | bj:BI • bj ∈ bis } in
3b.               react_behaviour_B(bi)(mereo)(stat)(mon)(prg)(msg) end
```

4. The `react_behaviour_B` inquires as to the type of the message, say, a command, received (?): if it is:
   (a) of type `Cmd_i` then it performs action `act_cmd_i`,
   (b) of type `Cmd_j` then it performs action `act_cmd_j`,
   (c) ..., or
   (d) of type `Cmd_k` then it performs action `act_cmd_k`.
   (e) If it is of neither of these types then it "skips" treatment of that response by resuming to be the behaviour `B`.

value

```
4.  react_behaviour_B(bi)(mereo)(stat)(mon)(prg)(msg) ≡
4a.     is_action_i(msg) → B_action_i(bi)(mereo)(stat)(mon)(prg)(msg),
4b.     is_action_j(msg) → B_action_j(bi)(mereo)(stat)(mon)(prg)(msg),
4c.     ...,
4d.     is_action_k(msg) → B_action_k(bi)(mereo)(stat)(mon)(prg)(msg),
4e.     _ → B(bi)(mereo)(stat)(mon)(prg)
```

10.5.4 Action Definitions. *"Actions are what makes behaviours meaningful"* We remind the reader that our function (incl. behaviour) definitions are expressed in a functional, "applicative", style. [that is, there are no assignable variables] The actions elaborate to **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.

5. Actions, as semantic quantities,
   (a) evaluate some values,
   (b) typically change some programmable attributes,
   (c) and may communicate, "issue" or inform, to some other behaviours, some requests, respectively information –
   (d) whereupon the "revert", "tail-recursively" to the activating Behaviour.

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

Variants of Item ι5c π31 are also used:

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

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

10.5.5 Behaviour Invocation.

Theorem 36. *Behaviour Invocation*

Behaviours are invoked as follows:

```
"B_p(uid_p(p))[37]
     (mereo_P(p))
          (attr_staA_1(p),...,attr_staA_s(p))
               (attr_inertA_1(p),...,attr_inertA_i(p))
                    (attr_monA_1,...,attr_monA_m)
                         (attr_prgA_1(p),...,attr_prgA_p(p))"
```

- All arguments are passed *by value*.
- The *uid* value is never changed.
- The *mereology* value is usually not changed.
- The *static attribute* values are fixed, never changed.

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

- The *inert attribute* values are fixed, but can be updated by receiving explicit input communications.
- The *monitorable attribute* values are functions, i.e., it is as if the "actual" monitorable values are passed *by name*!
- The *programmable attribute* values are usually changed, "updated", by actions described in the behaviour definition •

10.5.6 Argument References. Within behaviour descriptions, see next section, references are made to the behaviour arguments. References, a, to *unique identifier, mereology, static* and *progammable attribute* arguments yield their value. References, a, to *monitorable attribute* arguments also yield their value. This value is an **attr**_A observer function. To yield, i.e., read, the monitorable attribute value this function is applied to that behaviour's uniquely identified part, p_{uid}, in the global part state, σ. To update,, i.e., write, say, to a value v, for the case of a *biddable, monitorable* attribute, that behaviour's uniquely identified part, p_{uid}, in the global part state, σ, shall have part p_{uid}'s A attribute changed to v – with all other attribute values of p_{uid} unchanged. Common to both the read and write functions is the *retrieve part* function:

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

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

You may think of the functions being illustrated in this section, Sect. 10.5.6, retr_part, read_A_from_P and update_P_with_A, as "belonging" to the description language, but here suitably expressed for any domain, that is, with suitable substitutions for A and P.

10.5.6.1 Evaluation of Monitorable Attributes.

6. Let pi:PI be the unique identifier of any part, p, with monitorable attributes, let A be a monitorable attribute of p, and let ηA be the name of attribute A.
7. Evaluation of the [current] attribute A value of p is defined by function read_A_from_P.

value
6. pi:PI, a:A, ηA:$\eta\mathbb{T}$
7. read_A_from_P: PI × $\mathbb{T}$ → **read** σ A
7. read_A(pi,ηA) ≡ attr_A(retr_part(pi))

10.5.6.2 Update of Biddable Attributes

8. The update of a monitorable attribute A, with attribute name ηA of part p, identified by pi, to a new value **write**s to the global part state σ.
9. Part p is retrieved from the global state.
10. A new part, p′ is formed such that p′ is like part p:
    (a) same unique identifier,
    (b) same mereology,
    (c) same attributes values,
    (d) except for A.
11. That new p' replaces p in σ.

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

8.    update_P_with_A: PI $\times$ A $\times$ $\eta\mathbb{T}$ $\rightarrow$ write $\sigma$
8.    update_P_with_A(pi,a,$\eta$A) $\equiv$
9.        let p = retr_part(pi) in
10.         let p′:P •
10a.              uid_P(p′)=pi
10b.              $\wedge$ mereo_P(p)=mereo_P(p′)
10c.              $\wedge$ $\forall$ $\eta$A′ $\in$ record_attribute_type_names(p)\{$\eta$A}
10c.                  $\Rightarrow$ attr_A′(p)=attr_A′(p′)
10d.              $\wedge$ attr_A(p′)=a in
11.         $\sigma$ := c$\sigma$ \ {p} $\cup$ {p′}
8.        end end
9.     pre: $\exists$ p:P • p $\in$ c$\sigma$ $\wedge$ uid_P(p)=pi
```

10.5.7 Behaviour Description – Examples. Behaviour descriptions rely strongly on CSPs' [33] expressivity. Leaving out some details (_, '...'), and *without "further ado"*, we exemplify.

Example 33. *Automobile Behaviour at Hub*:

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

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

13. [12a] The automobile remains in the hub:
    (a) time is recorded,
    (b) the automobile remains at that hub, “idling”,
    (c) informing (“first”) the hub behaviour.

```
13  automobile_remains_in_hub(ai)(ris)(...)(atH(hi),ahis,_) ≡
13a     let τ = record_𝕋𝕀𝕄𝔼 in
13c     ch[{ai,hi}] ! τ ;
13b     automobile(ai)(ris)(...)(atH(hi),⟨(τ,hi)⟩^ahis,_) end
```

14. [12c] The automobile leaves the hub entering link li:
    (a) time is recorded;
    (b) hub is informed of automobile leaving and link that it is entering;
    (c) “whereupon” the vehicle resumes (i.e., “while at the same time” resuming) the vehicle behaviour positioned at the very beginning (0) of that link.

```
14  automobile_leaving_hub(ai)({li}∪ris)(...)(atH(hi),ahis,_) ≡
14a      let τ = record_𝕋𝕀𝕄𝔼   in
14b      (ch[{ai,hi}] ! τ ∥ ch[{ai,li}] ! τ) ;
14c      automobile(ai)(ris)(...)(onL(li,(hi,0,_)),⟨(τ,li)⟩^ahis,_) end
14       pre: [hub is not isolated]
```

The choice of link entered is here expressed (14) as a non-deterministic choice[38]. One can model the leave hub/enter link otherwise.

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

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

rm

10.6 Behaviour Initialization.

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

Example 34. *Road Transport Initialization*: We “wrap up” the main example of this paper: We omit treatment of monitorable attributes.

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

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

We have here omitted possible monitorable attributes. For hs, ls, as we refer to Sect. 8.1.4 •

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

11 Conclusion

We have summarized a method to be used by [human] domain analyzers cum describers in studying and modeling domains. Our previous publications [13–15] have, with this paper, found its most recent, we risk to say, for us, final form.

Of course, domain models can be developed without the calculi presented in this paper. And was for many years. From the early 1990s a number of formal models of railways were worked out [30, 5, 7, 20, 6]. The problem, though, was still, between 1992 and 2016, *"where to begin, how to proceed and when to end"*. The domain analysis & description ontology and, hence calculus, of this paper shows how. The systematic approach to domain modeling of this ontology and calculus has stood its test of time. The `Internet` 'publication' `https://www.imm.dtu.dk/~dibj/2021/-dd/dd.pdf` include the following domain models[39] from the 2007–2024 period. Their development has helped hone the method of the present paper.

11.1 Previous Literature

To the best of my knowledge there is no prior, comparable publications in the field of domain science and engineering. Closest would be Michael A. Jackson's [38]. Well, most computer scientists working in the field of correctness of programs, from somewhat "early on", stressed the importance of making proper assumptions about the domain, They would then express these "in-line", as appropriate predicates, with their proofs. Michael A. Jackson, lifted this, to a systematic treatment of the domain in his triplet 'Problem Frame Approach': *program, machine, domain* [37]. But Jackson did not lift his problem frame concern into a proper study of domains.

11.2 The Method

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

11.3 Specification Units

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

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

There are basically three forms of value specification units: (i) ("simple") naming of values, (ii) signature of functions: function name and function type, and (iii) signature of (endurant **obs_**, unique identifier **uid_**, mereology, **mereo_**, and attribute **attr_**) observer functions.

39

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

11.4 Object Orientation

So far we have not used the term 'object' !

We shall now venture the following:

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

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

That is, an object concept solely based on essentially inescapable world description facts – as justified by Sørlander's Philosophy [45–48] ! No "finicky" programming language "tricks" !

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

11.5 Other Domain Modeling Approaches

[49] shows fragments of a number of expertly expressed domain models. They are all expressed in `RAISE`.[40] But they are not following the method of this paper. In other words, it is possible to develop domain models not using the method ! This author has found, however, that following the method – developed after the projects reported in [49] – leads to far less problematic situations – in contrast to my **not** adhering strictly to the method. In other words, based on this subjective observation, we advice using the method.

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

But we do take the fact that we can justify the method, cf. Fig. 1, on the basis on the inevitability of describing the world as per philosophy of Kai Sørlander [45–48], and that that may have a bearing on the experienced shorter domain description development efforts.

11.6 How Much ? How Little ?

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

11.7 Correctness

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

11.8 Domain Facets

There is more to *domain modeling* than covered in this paper. In [10] and in [15, *Chapter 8*] we cover the concept of *domain facets*. General examples of domain facets are *support technologies*, *rules & regulations*, *scripts*, *license languages*, *management & organization*, and *human behaviour*.

[40] Other approaches could also be used: `VDM` [21, 22], `Z` [51], `Alloy` [35], `CafeOBJ` [27], etc.

11.9 Perspectives

Domain models can be developed for either of a number of reasons:

- (i) in order to *understand* a human-artifact domain;
- (ii) in order to *re-engineer the business processes* of a human-artifact domain; or
- (iii) in order to develop *requirements prescriptions* and, subsequently *software application* "within" that domain.

[(ii)] We refer to [31, 32] and [8, *Vol. 3, Chapter 19, pages 404–412*] for the concept of *business process engineering*. [(iii)] We refer to [15, *Chapter 9*] for the concept of *requirements engineering*.

11.10 The Semantics of Domain Models

The meaning of domain models, such as we describe them in this paper, is, "of course", the actual, real domain "out there" ! One could, and, perhaps one should, formulate a mathematical semantics of the models, that is, of the **is_...**, **obs_...**, **uid_...**, **mereo_...** and **attr_...** analysis and description functions and what they entail (e.g., the type name labels: $\eta\mathbb{T}$'s; etc.). An early such semantics description is given in [12].

11.11 Further on Domain Modeling

Additional facets of domain modeling are covered in [9] and [15, *Chapter 8: Domain Facets.*]

11.12 Software Development

[9] and [15, *Chapter 9 Requirements*] show how to develop $\mathcal{R}$equirements prescriptions from $\mathcal{D}$omain descriptions. [8] shows how to develop $\mathcal{S}$oftware designs from $\mathcal{R}$equirements prescriptions.

11.13 Modeling

Domain descriptions, such as outlined in this paper, are models of domains, that is, of some reality. They need not necessarily lead to or be motivated by possible development of software for such domains. They can be experimentally researched and developed just for the sake of understanding domains in which man has had an significantly influence. They are models. We refer to [26] for complementary modeling based on Petri nets. The current author is fascinated by the interplay between graphical and textual descriptions of `HERAKLIT`, well, in general `Petri Nets`.

11.14 Philosophy of Computing

The Danish philosopher Kai Sørlander [45–48] has shown that there is a foundation in philosophy for domain analysis and description. We refer to [17, *Chapter 2*] for a summary of his findings.

11.15 A Manifesto

So there is no excuse, anymore! Of course we have developed interpreters and compilers for programming languages by first developing formal semantics for those languages [23, 24]. Likewise we must now do for the languages of domain stakeholders, at least for the domains covered by this paper. There really is no excuse!

References

1. J. L. Austin. *How to Do Things with Words*. Harvard University Press, Cambridge, Mass., 2 edition, 1975. (William James Lectures).
2. H. Bekič, D. Bjørner, W. Henhapl, C.B. Jones, and P. Lucas. A Formal Definition of a PL/I Subset. Technical Report 25.139, Vienna, Austria, December 1974.
3. Hans Bekič, Peter Lucas, Kurt Walk, and Many Others. Formal Definition of PL/I, ULD Version III. IBM Laboratory, Vienna, 1969.
4. D. Bjørner and O. Oest. *Towards a Formal Description of Ada*, volume 98 of *LNCS*. Springer–Verlag, 1980.
5. Dines Bjørner. Formal Software Techniques in Railway Systems. In Eckehard Schnieder, editor, *9th IFAC Symposium on Control in Transportation Systems*, pages 1–12, Technical University, Braunschweig, Germany, 13–15 June 2000. VDI/VDE-Gesellschaft Mess– und Automatisieringstechnik, VDI-Gesellschaft für Fahrzeug– und Verkehrstechnik. Invited talk.
6. Dines Bjørner. Dynamics of Railway Nets: On an Interface between Automatic Control and Software Engineering. In *CTS2003: 10th IFAC Symposium on Control in Transportation Systems*, Oxford, UK, August 4-6 2003. Elsevier Science Ltd. Symposium held at Tokyo, Japan. Editors: S. Tsugawa and M. Aoki. www2.imm.dtu.dk/ dibj/ifac-dynamics.pdf.
7. Dines Bjørner. New Results and Trends in Formal Techniques for the Development of Software for Transportation Systems. In *FORMS2003: Symposium on Formal Methods for Railway Operation and Control Systems*. Institut für Verkehrssicherheit und Automatisierungstechnik, Techn.Univ. of Braunschweig, Germany, 15–16 May 2003. Conf. held at Techn.Univ. of Budapest, Hungary. Editors: G. Tarnai and E. Schnieder, Germany. www2.imm.dtu.dk/ dibj/dines-amore.pdf.
8. Dines Bjørner. *Software Engineering, Vol. 1: Abstraction and Modelling; Vol. 2: Specification of Systems and Languages; Vol. 3: Domains, Requirements and Software Design*. Texts in Theoretical Computer Science, the EATCS Series. Springer, Heidelberg, Germany, 2006.
9. 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.
10. 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.
11. Dines Bjørner. A Rôle for Mereology in Domain Science and Engineering. In *Mereology and the Sciences*, Synthese Library (eds. Claudio Calosi and Pierluigi Graziani), pages 323–357, Amsterdam, The Netherlands, October 2014. Springer. https://www.imm.dtu.dk/ dibj/2011/urbino/urbino-colour.pdf.
12. Dines Bjørner. Domain Analysis: Endurants – An Analysis & Description Process Model www.imm.dtu.dk/ dibj/2014/kanazawa/kanazawa-p.pdf. In Shusaku Iida and José Meseguer and Kazuhiro Ogata, editor, *Specification, Algebra, and Software: A Festschrift Symposium in Honor of Kokichi Futatsugi*. Springer, Heidelberg, Garmany, May 2014.
13. Dines Bjørner. Manifest Domains: Analysis & Description www.imm.dtu.dk/ dibj/2015/faoc/faoc-bjorner.pdf. *Formal Aspects of Computing*, 29(2):175–225, March 2017. Online: 26 July 2016.
14. Dines Bjørner. Domain Analysis & Description – Principles, Techniques and Modeling Languages. www.imm.dtu.dk/ dibj/2018/tosem/Bjorner-TOSEM.pdf. *ACM Trans. on Software Engineering and Methodology*, 28(2):66 pages, March 2019.
15. Dines Bjørner. *Domain Science & Engineering – A Foundation for Software Development*. EATCS Monographs in Theoretical Computer Science. Springer, Heidelberg, Germany, 2021. A revised version of this book is [18].
16. Dines Bjørner. *Domain Science & Engineering – A Foundation for Software Development*. EATCS Monographs in Theoretical Computer Science. Springer, Heidelberg, Germany, 2021. A revised version of this book is [**?**].
17. Dines Bjørner. Domain Modelling – A Primer. A short and significantly revised version of [15]. xii+202 pages[41], May 2023.
18. Dines Bjørner. Domain Science & Engineering – A Foundation for Software Development. Revised edition of [15]. xii+346 pages[42], January 2023.

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

[42] Due to copyright reasons no URL is given to this document's possible Internet location. A primer version, omitting certain chapters, is [17]

19. Dines Bjørner. Domain Models – A Compendium. Internet: `http://www.imm.dtu.dk/~dibj/2024/models/domain-models.pdf`, March 2024. This is a very early draft. 19 domain models are presented.
20. Dines Bjørner, Chris W. George, and Søren Prehn. Computing Systems for Railways — A Rôle for Domain Engineering. Relations to Requirements Engineering and Software for Control Applications. In *Integrated Design and Process Technology. Editors: Bernd Kraemer and John C. Petterson*, P.O.Box 1299, Grand View, Texas 76050-1299, USA, 24–28 June 2002. Society for Design and Process Science. www2.imm.dtu.dk/ dibj/pasadena-25.pdf.
21. Dines Bjørner and Cliff B. Jones, editors. *The Vienna Development Method: The Meta-Language*, volume 61 of *LNCS*. Springer, Heidelberg, Germany, 1978.
22. Dines Bjørner and Cliff B. Jones, editors. *Formal Specification and Software Development*. Prentice-Hall, London, England, 1982.
23. Dines Bjørner and Ole N. Oest, editors. *Towards a Formal Description of Ada*, volume 98 of *LNCS*. Springer, Heidelberg, Germany, 1980.
24. Geert Bagge Clemmensen and Ole N. Oest. Formal specification and development of an Ada compiler – a VDM case study. In *Proc. 7th International Conf. on Software Engineering, 26.-29. March 1984, Orlando, Florida*, pages 430–440, New York, USA, 1984. IEEE.
25. Patrick Cousot. *Principles of Abstract Interpretation*. The MIT Press, 2021.
26. Peter Fettke and Wolfgang Reisig. *Understanding the Digital World – Modeling with HERAKLIT*. Springer, 2024. To be published.
27. K. Futatsugi, A.T. Nakagawa, and T. Tamai, editors. *CAFE: An Industrial–Strength Algebraic Formal Method*, Sara Burgerhartstraat 25, P.O. Box 211, NL–1000 AE Amsterdam, The Netherlands, 2000. Elsevier. Proceedings from an April 1998 Symposium, Numazu, Japan.
28. 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.
29. Chris W. George, Anne Elisabeth Haxthausen, Steven Hughes, Robert Milne, Søren Prehn, and Jan Storbank Pedersen. *The RAISE Development Method*. The BCS Practitioner Series. Prentice-Hall, Hemel Hampstead, England, 1995.
30. Chris W. George, Hung Dang Van, Tomasz Janowski, and Richard Moore. *Case Studies using The RAISE Method*. FACTS (Formal Aspects of Computing: Theory and Software) and FME (Formal Methods Europe). Springer–Verlag, London, 2002. This book reports on a number of case studies using RAISE (Rigorous Approach to Software Engineering). The case studies were done in the period 1994–2001 at UNU/IIST, the UN University's International Institute for Software Technology, Macau (till 20 Dec., 1997, Chinese Teritory under Portuguese administration, now a Special Administrative Region (SAR) of (the so–called People's Republic of) China).
31. Michael Hammer and James A. Champy. *Reengineering the Corporation: A Manifesto for Business Revolution*. HarperCollins*Publishers*, 77–85 Fulham Palace Road, Hammersmith, London W6 8JB, UK, May 1993. 5 June 2001, Paperback.
32. Michael Hammer and Stephen A. Stanton. *The Reengineering Revolutiuon: The Handbook*. HarperCollins*Publishers*, 77–85 Fulham Palace Road, Hammersmith, London W6 8JB, UK, 1996. Paperback.
33. Charles Anthony Richard Hoare. *Communicating Sequential Processes*. C.A.R. Hoare Series in Computer Science. Prentice-Hall International, London, England, 1985. Published electronically: `usingcsp.com/cspbook.pdf` (2004).
34. Gerard J. Holzmann. *The SPIN Model Checker, Primer and Reference Manual*. Addison-Wesley, Reading, Massachusetts, 2003.
35. Daniel Jackson. *Software Abstractions: Logic, Language, and Analysis*. The MIT Press, Cambridge, Mass., USA, April 2006. ISBN 0-262-10114-9.
36. Michael A. Jackson. *Software Requirements & Specifications: a lexicon of practice, principles and prejudices*. ACM Press. Addison-Wesley, Reading, England, 1995.
37. Michael A. Jackson. *Problem Frames — Analyzing and Structuring Software Development Problems*. ACM Press, Pearson Education. Addison-Wesley, England, 2001.
38. Michael A. Jackson. Program Verification and System Dependability. In Paul Boca, Jonathan Bowen, and Jawed Siddiqi, editors, *Formal Methods: State of the Art and New Directions*, pages 43–78, London, UK, December 2009. Springer.
39. W. Little, H.W. Fowler, J. Coulson, and C.T. Onions. *The Shorter Oxford English Dictionary on Historical Principles*. Clarendon Press, Oxford, England, 1973, 1987. Two vols.
40. R. Milne and C. Strachey. *A Theory of Programming Language Semantics*. Chapman and Hall, London, Halsted Press/John Wiley, New York, 1976.

41. Charles W. Morris. *Foundations of the theory of signs*, volume I of *International encyclopedia of unified science*. The University of Chicago Press, 1938.
42. Karl R. Popper. *Conjectures and Refutations. The Growth of Scientific Knowledge*. Routledge and Kegan Paul Ltd. (Basic Books, Inc.), 39 Store Street, WC1E 7DD, London, England (New York, NY, USA), 1963,. . . ,1981.
43. F. Pulvermüller. Brain mechanisms linking language and action. *Nature Reviews: Neuroscience*, 6:576582, 2005. https://doi.org/10.1038/nrn1706.
44. John R. Searle. *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press, 1969.
45. Kai Sørlander. *Det Uomgængelige – Filosofiske Deduktioner [The Inevitable – Philosophical Deductions, with a foreword by Georg Henrik von Wright]*. Munksgaard · Rosinante, Copenhagen, Denmark, 1994. 168 pages.
46. Kai Sørlander. *Indføring i Filosofien [Introduction to The Philosophy]*. Informations Forlag, Copenhagen, Denmark, 2016. 233 pages.
47. Kai Sørlander. *Den rene fornufts struktur [The Structure of Pure Reason]*. Ellekær, Slagelse, Denmark, 2022. See [48].
48. Kai Sørlander. *The Structure of Pure Reason*. Springer, February 2025. This is an English translation of [47] – done by Dines Bjørner in collaboration with the author.
49. Hung Dang Van, Chris George, Tomasz Janowski, and Richard Moore, editors. *Specification Case Studies in RAISE*. Springer, 2002.
50. Achille C. Varzi. *On the Boundary between Mereology and Topology*, pages 419–438. Hölder-Pichler-Tempsky, Vienna, 1994.
51. James Charles Paul Woodcock and James Davies. *Using Z: Specification, Proof and Refinement*. Prentice Hall International Series in Computer Science, London, England, 1996.