Domain Modelling

Dines Bjørner
Technical University of Denmark

March 25, 2024

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*.

- Three days of morning lectures
  - **Lectures I: Endurants: Tangibles** 1–71
  - **Lectures II: Endurants: Intangibles** 72–104
  - **Lectures III: Perdurants** 105–147

- Each [set of] morning lecture[s] consists of two, 40-45 min. lecture.
- 10-15 min. breaks.
- And a max. 45 min. session in which we discuss a specific domain example, for example:
  - a pipeline system,
  - a customer/retailer/wholesaler market,
  - a railway system,
  - or other !

- We present a systematic *method*, its
  - *principles*,
  - *procedures*,
  - *techniques*
    and
  - *tools*,
- for efficiently *analyzing* & *describing* domains.
- This talk is based on [8, 9, 10].
- It simplifies the methodology of these
  – as well as introduces some novel presentation and description language concepts.

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 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.) [13, *Chapter B*].
- **Road nets** – with street segments and intersections, traffic lights and automobiles – and the flow of these [13, *Chapter E*].
- **Pipelines** – with their liquids (oil, or gas, or water), wells, pipes, valves, pumps, forks, joins and wells and the flow of fluids [13, *Chapter I*].
- **Container terminals** – with their container vessels, containers, cranes, trucks, etc. – and the movement of all of these[13, *Chapter K*] ■

- Characterization 1 on Slide 4 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 behaviour.
- By **human-assisted** we mean that the domains – that we are interested in modelling – have, as an important property, that they possess man-made entities.

[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 !

1.2 Some Introductory Remarks

1.2.1 A Discussion of Our Characterization of a Concept of Domain

- Characterization 1 on Slide 4 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 speaker says some words ...

1.2.3 Programming Languages versus Domain Semantics

... the speaker says some words ...

1.2.4 A New Universe

... the speaker says some words ...

2 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.
- Solid endurants, called parts, may be considered
  - eiher *atomic*
  - or *compound* parts;
- or, as in this talk 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* [Merriam Webster] ■

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

• • •

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

3 A Domain Analysis & Description Ontology

3.1 The Chosen Ontology

- Figure 1 expresses an ontology[3] for our analysis of domains.
- Not a taxonomy[4] for any one specific domain.

[3] 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].

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

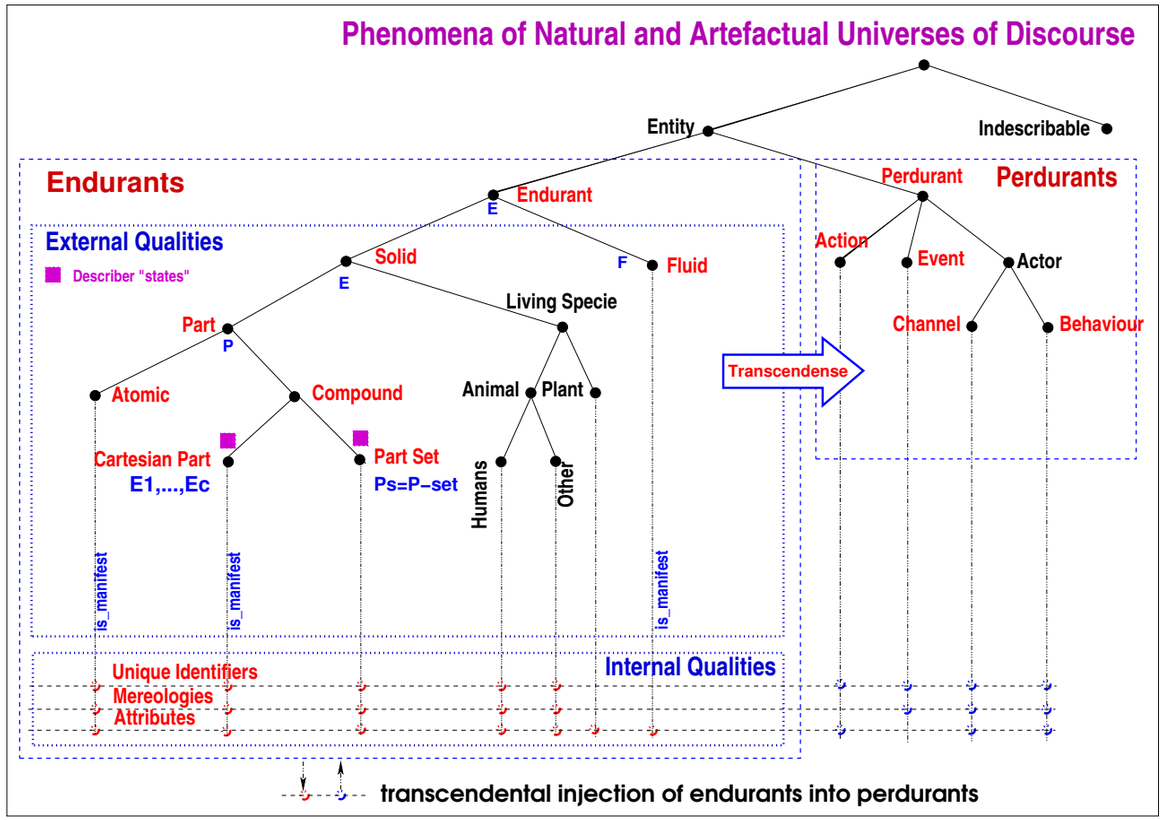

Figure 1: A Domain Analysis & Description Ontology

- The idea of Fig. 1 on the preceding slide is the following:
  - It presents a recipe for how to **analyze** a domain.
  - You, the *domain* **analyzer** *cum describer*, are *'confronted'*[5] with, or by a domain.
  - You have Fig. 1 on the previous slide 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 Univer**s**es of Discourse.
  - 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*.

[5] 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 !

– 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, `is_`β, or no, `is_`γ.
– 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".

3.2 Discussion of The Chosen Ontology

- We shall in the following motivate the choice of the *ontological classification* reflected in Fig 1 on Slide 13.
  - 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 [28, 29, 30, 31][6].
  - Kai Sørlander's aim in these books is to examine
  - *that which is absolutely necessary, inevitable, in any description of the world.*
- In [10, *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.

[6] The 2022 book, [30], is presently a latest in Kai Sørlander's work. It refines and further develops the theme of the earlier, 1994–2016 books. [31] is an English translation of [30]

4 The Name, Type and Value Concepts

- Domain *modelling*, 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.

4.1 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.[7]
- 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.

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

4.2 Types

- The type concept is crucial to programming and modelling.

Characterization 4 *Type*: A *type* is a class 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*.

4.3 Values

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

5 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[8] – others are not.

[8] 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.

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

We introduce the informal presentation language predicate `is_entity`. It holds for phenomena ϕ if ϕ is describable.

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: 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.

6 Endurants and Perdurants, II

We repeat our characterizations of endurants and perdurants.

6.1 Endurants

- We repeat characterization 2 on Slide 9.

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 [Merriam Webster]

■

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

- We introduce the informal presentation language predicate
- `is_endurant` to hold for entity `e` if `is_endurant(e)` holds.

6.2 Perdurants

- We repeat characterization 3 on Slide 11.

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* [Merriam Webster] ■

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

- We introduce the informal presentation language predicate `is_perdurant` to hold for entity `e` if `is_perdurant(e)` holds.

6.3 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 ■

7 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:

- 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'] ■

7.1 External Qualities – Tangibles

Example 6 *External Qualities*: An example of external qualities of a domains is:

- the Cartesian[9]
  - 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 ■

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

7.1.1 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.

Narration:

The name, and hence the type, of the domain] is UoD

The UoD domain can be briefly characterized by ...

Formalization:

type UoD

7.1.2 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:

- 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] ■

7.1.2.1 Solid [or Discrete] Endurants.

Characterization 11 *Discrete or Solid Endurants*: By a *solid* [or *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, breadth and depth [26, *OED, Vol. II, pg. 2046*] ■

Example 7 *Solid Endurants*: Pipeline system examples of solid endurants are

- *wells*,
- *valves*,
- *forks*,
- *sinks*
- *pipes*,
- *pumps*,
- *joins* and

of pipelines.
(These units may, however, and usually will, contain fluids, e.g., oil, gas or water.) ■

- We introduce the informal presentation language predicate `is_solid`
- to hold for endurant `e`
- if `is_solid(e)` holds.

7.1.2.2 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 [26, *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[10], or
  - plant products, [11],
  - et cetera.
- Fluid endurants will be analyzed and described in relation to solid endurants, viz. their “containers”.

[10] This is a purely pragmatic decision. “Of course” sand, gravel, soil, etc., are not fluids, but for our modelling purposes it is convenient to “compartmentalise” them as fluids !
[11] i.e., chopped sugar cane, threshed, or otherwise; see footnote 10.

- We introduce the informal presentation language predicate
  - `is_fluid`
  - to hold for endurant `e`
  - if `is_fluid(e)` holds.

7.1.3 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:

- With Sørlander, [31, *Sect. 5.7.1, pages 71–72*]
- we reason that one can distinguish between parts and living species ■

7.1.3.1 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 on Slide 39, of solids, is an example of parts ■

- 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:**
  - It is an empirical fact that parts can be composed from parts.
  - That possibility exists.
  - Hence we can [philosophy-wise] reason likewise ■

7.1.3.1.1 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 ■

- We introduce the informal presentation language predicate `is_atomic`
- to hold for parts `p`
- if `is_atomic(p)` holds.

7.1.3.1.2 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 ■

- 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:

- 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 ! ■

7.1.3.1.3 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) ■
- We introduce the informal presentation language predicate `is_Cartesian`
- to hold for compound parts `p`
- if `is_Cartesian(p)` holds.

Example 12 *Cartesians: Road Transport*:

1. A road transport, rt:RT, is observed to consist of
2. an aggregate of a road net, rn:RN,
3. and a set of automobiles, SA.

   Here the road net is observed, i.e., abstracted, as a Cartesian of
4. a set of hubs, ah:AH[12], and
5. a set of links, al:AL[13]

type

1,2,3. RT, RN, SA,

4,5. AH = H**-set**, AL = L**-set**

value

2. **obs**_RN: RT → RN,
3. **obs**_SA: RT → SA,
4. **obs**_AH: RN → AH,
5. **obs**_AL: RN → AL ■

[12] i.e., street intersections (or specifically designated points segmenting an otherwise "straight" street into two such)

[13] i.e., street segments between two "neighbouring" hubs.

- Once a part, say p:P, has been analyzed into a Cartesian, we inquire as to the type names of the endurants[14] of which it consists.
- The inquiry: `record_Cartesian_part_type_names(p:P)`, we decide, then yields the type of the constituent endurants.

Schema 1 *record-Cartesian-part-type-names*

value

`record_Cartesian_part_type_names`: P $\rightarrow$ $\mathbb{T}$**-set**
`record_Cartesian_part_type_names`(p) **as** $\{\eta$E1,ηE2,...,ηEn$\}$ ■

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

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

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 ■

7.1.3.1.4 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 ■

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.

Schema 2 *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 ■

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) ■

7.1.3.1.5 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:

Schema 3 *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 "
end ■

respectively:

```
value
  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 ■
```

- 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 modelling 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, 3, 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*.

7.1.4 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[15] 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.

[15] The concepts of monitorable, including biddable part attributes is treated in Sect. 7.2.3.2.

7.1.5 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 [27].
- A social process of peer reviews, by domain stakeholders and other domain modelers is needed –
- as may a process of verifying[16] properties of the domain description held up against claimed properties of the (real) domain.

[16] testing, model checking and theorem proving

7.1.6 Summary of Analysis Predicates

Characterizations 6–17 imply the following analysis predicates (Defn.: δ, page π):

- **Endurant Ontology:**
  - is_entity, $\delta 6\, \pi\, 24$
  - is_entity, $\delta 6\, \pi\, 24$
  - is_entity, $\delta 6\, \pi\, 24$
  - is_endurant, $\delta 7\, \pi\, 27$
  - is_perdurant, $\delta 8\, \pi\, 28$
  - is_solid, $\delta 11\, \pi\, 38$
  - is_fluid, $\delta 12\, \pi\, 41$
  - is_part, $\delta 13\, \pi\, 45$
  - is_atomic, $\delta 14\, \pi\, 47$
  - is_compound, $\delta 15\, \pi\, 50$
  - is_Cartesian, $\delta 16\, \pi\, 53$
  - is_part_set, $\delta 17\, \pi\, 57$

- We remind the student 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**[17],
  - are noted by these persons
  - and determine their next step of domain analysis.

[17] The outcome of applying an analysis predicate of the prescribed kind may be **chaos** if the prerequisites for its application does not hold.

7.1.7 "Trees are Not Recursive"

- A 'fact', that seems to surprise many, is that parts are not "recursive".
  - Yes, in all our domain modelling experiments, [13],
  - 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[18] have several "intertwined trees".
  - But it would be 'twisting' the modelling 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 [13, *Chapter B*].

[18] https://www.britannica.com/plant/banyan

7.2 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 1–2, and
    - a pair of description functions, schema 3 on Slide 61.
  - 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.

7.2.1 Unique Identity

Ontological Choice:

- We postulate that separately discernible parts have unique identify.
- The issue, really, is a philosophical one.
- We refer to [10, *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[19] ■
- The unique identity of a part p of type P is obtained by the postulated observer **uid**_P:

Schema 4 *Describe-Unique-Identity-Part-Observer*

"**type**
  PI
value
  uid_P: P → PI" ■

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

[19] 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 modelling experiments have not found a need for such ascription !

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

- 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" ls to relevant part names ■

- We have thus introduced a core domain modelling 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.

7.2.1.1 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. 7.1.4 on Slide 65.

We have chosen, for the same reason as given in Sect. 7.1.4, 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).

7.2.2 Mereology

- The concept of mereology is due to the Polish mathematician, logician and philosopher Stanisław Leśniewski (1886–1939) [32, 6].
- **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:

- The Polish mathematician and philosopher 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[20]:

Schema 5 *Describe-Mereology*

"**type**
  PMer = $\mathscr{M}$(PI1,PI2,...,PIm)
value
  mereo_P: P → PMer
axiom
  $\mathscr{A}$(pm:PMer)" ■

[20] Cf. Sect. 7.1.3.1.5

7.2.3 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.

7.2.3.1 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:

- 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 ■[21]

[21] **Editorial remark:** I am not yet satisfied with this reasoning. The issue is: to force the concept of attributes to be justified philosophically, as an inevitable element of any world description, and not being forced upon us solely from empirical evidence.

- 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

Schema 6 *record-attribute-type-names*

value

`record_attribute_type_names`: P $\rightarrow$ $\eta\mathbb{T}$-**set**

`record_attribute_type_names(p:P)` **as** ηT-**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.

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}∧... ■

- Generally:

Schema 7 *Describe-endurant-attributes(e:E)*

let {η A1,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: $\mathscr{A}$(a1,a2,...,an) "
end ■

7.2.3.2 Michael A. Jackson's Attribute Categories.

- Michael A. Jackson [25] has suggested a hierarchy of attribute categories:
  - from *static* (`is_static`[22])
  - to *dynamic* (`is_dynamic`[23]) values – and within the dynamic value category:
    - *inert* values (`is_inert`[24]),
    - *reactive* values (`is_reactive`[25]),
    - *active* values (`is_active`[26]) – and within the dynamic active value category:
      - *autonomous* values (`is_autonomous`[27]),
      - *biddable* values (`is_biddable`[28]), and
      - *programmable* values (`is_programmable`[29]) .

[22] `static:` values are constants, cannot change
[23] `dynamic:` values are variable, can change
[24] `inert:` values can only change as the result of external stimuli where these stimuli prescribe new values
[25] `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.
[26] `active:` values can change (also) on their own volition
[27] `autonomous:` values change only "on their own volition"; the values of an autonomous attributes are a "law onto themselves and their surroundings".
[28] `biddable:` values are prescribed but may fail to be observed as such
[29] `programmable:` values can be prescribed

– We postulate informal domain analysis predicates, “performed” by the domain analyzer:

value
is_static,is_autonomous,is_biddable,is_programmable [etc.]: η T→**Bool**

- We refer to [25] and [10] [*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 on the following slide.
  - We single out the inert from the ontology of Fig. 2 on the next slide, 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.

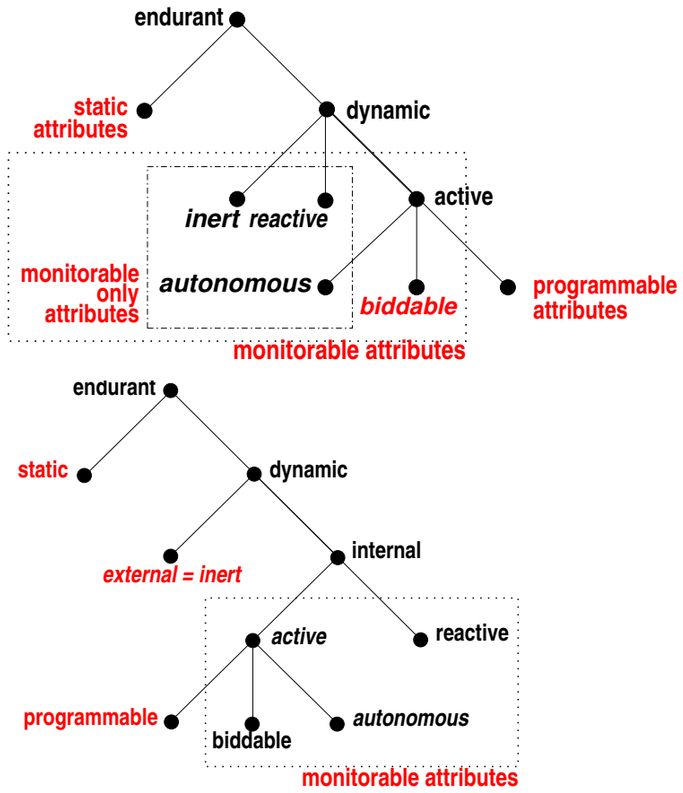

Figure 2: Michael Jackson's [Revised] Attribute Categories

This distinction has [pragmatic] consequences for how we treat arguments of the behaviours of parts, cf. Sect. 8.5.1 (page 120).

Example 23 *Road Net Attributes, II*:

- The link length and hub state space attributes are static,
- hub states and automobile positions are programmable attributes.
- 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.

7.2.3.3 Analytic Attribute Extraction Functions:.

value
  p:P
  tns = record_attribute_type_names(p)

  static_attributes: ηT-**set** → ηT-**set**
  static_attributes(tns) ≡ { ηtn | ηtn:ηT · ηtn ∈ tns ∧ is_static(tn) }

  inert_attributes: ηT-**set** → ηT-**set**
  inert_attributes(tns) ≡ { ηtn | ηtn:ηT · ηtn ∈ tns ∧ is_inert(tn) }

  monitorable_attributes ηT-**set** → ηT-**set**
  monitorable_attributes(tns) ≡ { ηtn | ηtn:ηT · ηtn ∈ tns ∧ is_monitorable(tn) }

  programmable_attributes ηT-**set** → ηT-**set**
  programmable_attributes(tns) ≡ { ηtn | ηtn:ηT · ηtn ∈ tns ∧ is_programmable(tn) }

  is_monitorable: T → **Bool**
  is_monitorable(t) ≡ ∼is_static(t) ∧ ∼is_inert(t) ∧ ∼is_programmble(t)

7.3 Intentional Pull

Ontological Choice:

- In [29, *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[30]
  - may excert an intentional "pull" on one another.

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

- This *intentional "pull"* may take many forms.
  - Let $p_x : X$ and $p_y : Y$
  - be two parts of *different sorts* (X, Y),
  - and with *common intent*, 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* of `'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 listener ■

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 ■[31,32]

[31]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.

[32]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.

7.4 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

8 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.**

8.1 "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 [24] constructs of RSL (derived from RSL [20]) to model concurrent behaviours.

8.2 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.
    - The software tools,
      an automatic theorem prover and
      a model checker.
    - A **compiler** and
      an **interpreter**.

- Yes, indeed, any **abstract interpretation** [18]
  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:

- So this, then, is, in a sense, our “final” ontological choice:
- that of transcendentally deduce behaviours from, or of, parts ■

8.3 Actors – A Synopsis

This section provides a summary overview.

Characterization 25 *Actors*:

- An actor is anything that can initiate an **action**, **event** or **behaviour** ■

8.3.1 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 ■

8.3.2 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, in these lectures, not exemplify formalization of events.

8.3.3 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* ■

8.4 Channel

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

Schema 8 *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}⊆σ_{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 ■

8.5 Behaviours

- We single out the perdurants of behaviours
  - as they relate directly to the parts of Sect. 7.
  - The treatment is “divided” into three sections.
    - (i) behaviour signatures,
    - (ii) behaviour invocation, and
    - (iii) behaviour definition.

8.5.1 Behaviour Signature

Schema 9 *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

$$\begin{aligned}
&\mathsf{B}_p\text{: } \mathsf{Uid}_p \to^{33} \\
&\quad \mathsf{Mereo}_p \to \\
&\quad\quad \mathsf{Sta_Vals}_p \to \\
&\quad\quad\quad \mathsf{Inert_Vals}_p \to \\
&\quad\quad\quad\quad \mathsf{Mon_Refs}_p \to \\
&\quad\quad\quad\quad\quad \mathsf{Prgr_Vals}_p \to \\
&\quad\quad\quad\quad\quad\quad \{\ \mathrm{ch}[\{\mathrm{i,j}\}] \mid \ldots\ \}\ \textbf{Unit}
\end{aligned}$$

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

We explain:

- 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;
- $\mathsf{Sta\_Vals}_p$ is a Cartesian of the type of inert attributes of part p. Given `record_attribute_type_names(p)` `static_attributes(record_attribute_types(p))` yields $\mathsf{Sta\_Vals}_p$;
- $\mathsf{Inert\_Vals}_p$ is a Cartesian of the type of static attributes of part p. Given `record_attribute_type_names(p)` `inert_attributes(record_attribute_types(p))` yields $\mathsf{Inert\_Vals}_p$;
- $\mathsf{Mon\_Refs}_p$ 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 $\mathsf{Mon\_Vals}_p$;

- $\mathsf{Prgr\_Vals}_p$ 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 $\mathsf{Prgr\_Vals}_p$;
- { ch[{i,j}] | ... } specifies the channels over which part p behaviours, B_p, may communicate;

and:

- **Unit** is the type name for the () value[34] ■

• • •

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

[34] – You may “read’ () as the value yielded by a statement, including a never-terminating function

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 ■

8.5.1.1 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.

8.5.2 Behaviour Invocation

Schema 10 *Behaviour Invocation*

- Behaviours are invoked as follows:

“$\mathsf{B}_p(\mathbf{uid}_{\_p}(\mathsf{p}))$[35]
$(\mathbf{mereo}\_\mathsf{P}(\mathsf{p}))$
$(\mathbf{attr}\_\mathsf{staA}_1(\mathsf{p}),\ldots,\mathbf{attr}\_\mathsf{staA}_s(\mathsf{p}))$
$(\mathbf{attr}\_\mathsf{inertA}_1(\mathsf{p}),\ldots,\mathbf{attr}\_\mathsf{inertA}_i(\mathsf{p}))$
$(\mathbf{attr}\_\mathsf{monA}_1,\ldots,\mathbf{attr}\_\mathsf{monA}_m)$
$(\mathbf{attr}\_\mathsf{prgA}_1(\mathsf{p}),\ldots,\mathbf{attr}\_\mathsf{prgA}_p(\mathsf{p}))$”

[35] We show the arguments of the invocation on separate lines only for readability. That is: normally we show the invocation arguments as $\mathsf{B}(\ldots)(\ldots)(\ldots)(\ldots)(\ldots)$.

- 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.
- 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 ■

8.5.2.1 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:
  6. Given a unique part identifier, pi, assumed to be that of an existing domain part,
  7. retr_part *read*s the global [all parts] variable σ to retrieve that part p whose unique part identifier is pi.

value

7. retr_part: PI → P **read**
7. retr_part(pi) ≡ **let** p:P · p ∈ **c** σ ∧ uid_P(p)=pi **in** p **end**
6. **pre**: ∃ p:P · p ∈ **c** σ ∧ uid_P(p)=pi

- You may think of the functions being illustrated in this section, Sect. 8.5.2.1,
  - 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.

8.5.2.2 Evaluation of Monitorable Attributes.

8. 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.
9. Evaluation of the [current] attribute A value of p is defined by function read_A_from_P.

value

8. pi:PI, a:A, ηA:$\eta\mathbb{T}$
9. read_A_from_P: PI $\times$ $\mathbb{T}$ $\rightarrow$ **read** σ A
9. read_A(pi,ηA) $\equiv$ attr_A(retr_part(pi))

8.5.2.3 Update of Biddable Attributes.

10. 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.
11. Part p is retrieved from the global state.
12. 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.
13. That new p' replaces p in σ.

```
value
10.  σ, a:A, pi:PI, ηA:ηT

10.  update_P_with_A: PI × A × ηT → write σ
10.  update_P_with_A(pi,a,ηA) ≡
11.      let p = retr_part(pi) in
12.      let p′:P ·
12a.         uid_P(p′)=pi
12b.         ∧ mereo_P(p)=mereo_P(p′)
12c.         ∧ ∀ ηA′ ∈ record_attribute_type_names(p)\{ηA} ⇒ attr_A′(p)=attr_A′(p′)
12d.         ∧ attr_A(p′)=a in
13.      σ := c σ \ {p} ∪ {p′}
10.      end end
11.  pre: ∃ p:P · p ∈ c σ ∧ uid_P(p)=pi
```

8.5.3 Behaviour Description – Examples

- Behaviour descriptions rely strongly on CSPs' [24] expressivity.
- Leaving out some details (_, '...'), and *without "further ado"*, we exemplify.

Example 33 *Automobile Behaviour at Hub*:

14. 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.

```
14  automobile(ai)(ris)(...)(atH(hi),ahis,_) ≡
14a     automobile_remains_in_hub(ai)(ris)(...)(atH(hi),ahis,_)
14b     ⌈⌉
14c     automobile_leaving_hub(ai)(ris)(...)(atH(hi),ahis,_)
14d     ⌈⌉
14e     automobile_stop(ai)(ris)(...)(atH(hi),ahis,_)
```

15. [14a] The automobile remains in the hub:

    (a) time is recorded,

    (b) the automobile remains at that hub, “idling”,

    (c) informing (“first”) the hub behaviour.

```
15  automobile_remains_in_hub(ai)(ris)(...)(atH(hi),ahis,_) ≡
15a     let τ = record_𝕋𝕀𝕄𝔼 in
15c     ch[{ai,hi}] ! τ ;
15b     automobile(ai)(ris)(...)(atH(hi),⟨(τ,hi)⟩^ahis,_) end
```

16. [14c] 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.

```
16  automobile_leaving_hub(ai)({li}∪ris)(...)(atH(hi),ahis,_) ≡
16a     let τ = record_𝕋𝕀𝕄𝔼 in
16b     (ch[{ai,hi}] ! τ ‖ ch[{ai,li}] ! τ) ;
16c     automobile(ai)(ris)(...)(onL(li,(hi,0,_)),⟨(τ,li)⟩^ahis,_) end
16      pre: [hub is not isolated]
```

- The choice of link entered is here expressed (16) as a non-deterministic choice[36].
- One can model the leave hub/enter link otherwise.

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

17. [14e] Or the automobile “disappears — off the radar” !

17 automobile_stop(ai)(ris),(...)(atH(hi),ahis,_) ≡ **stop** ■

8.6 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.

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

```
value
18. rts_initialisation: Unit → Unit
18. rts_initialisation() ≡
19.     ‖ { hub(uid_H(h))(mereo_H(h))(attr_HΩ(h),...)(attr_HΣ(h),...)| h:H · h ∈ hs }
20.     ‖ ‖ { link(uid_L(l))(mereo_L(l))(attr_LEN(l),...)(attr_LΣ(l),...)| l:L · l ∈ ls }
21.     ‖ ‖ { 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. 7.1.4 ■

9 Conclusion

- We have summarized a method
  - to be used by [human] domain analyzers cum describers
  - in studying and modelling domains.
  - Our previous publications [8, 9, 10] 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 [21, 1, 3, 15, 2].
  - 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 modelling of this ontology and calculus has stood its test of time.

– The `Internet` ‘publication’ [14] presents 19 domain models from the 2007–2024 period.

9.1 Previous Literature

... the speaker says some words ...

9.2 Domain Facets

... the speaker says some words ...

9.3 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 [22, 23] and [4, *Vol. 3, Chapter 19, pages 404–412*] for the concept of *business process engineering*.

(iii) We refer to [10, *Chapter 9*] for the concept of *requirements engineering*.

9.4 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 [7].

9.5 Further on Domain Modelling

- Additional facets of domain modelling are covered in
  - [5] and
  - [10, *Chapter 8: Domain Facets.*]

9.6 Software Development

- [5] and [10, *Chapter 9 Requirements*]
  - show how to develop $\mathscr{R}$equirements prescriptions from $\mathscr{D}$omain descriptions.
- [4] shows how to develop $\mathscr{S}$oftware designs from $\mathscr{R}$equirements prescriptions.

9.7 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 [19] 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`.

9.8 Philosophy of Computing

- The Danish philosopher Kai Sørlander [28, 29, 30, 31]
  - has shown that there is a foundation in philosophy
  - for domain analysis and description.
- We refer to [11, *Chapter 2*] for a summary of his findings.

9.9 A Manifasto

- 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 [16, 17].
  - 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] 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.

[2] 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`.

[3] 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`.

[4] 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.

[5] 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.

[6] 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`.

[7] 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.

[8] 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.

[9] 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.

[10] 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 [12].

[11] Dines Bjørner. Domain Modelling – A Primer. A short version of [12]. xii+202 pages[37], May 2023.

[12] Dines Bjørner. Domain Science & Engineering – A Foundation for Software Development. Revised edition of [10]. xii+346 pages[38], January 2023.

[37] This book is currently being translated into Chinese by Dr. Yang ShaoFa, IoS/CAS, Beijing and into Russian by Dr. Mikhail Chupilko, ISP/RAS, Moscow

[38] Due to copyright reasons no URL is given to this document's possible Internet location. A primer version, omitting certain chapters, is [11]

[13] 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.

[14] 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.

[15] 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`.

[16] Dines Bjørner and Ole N. Oest, editors. *Towards a Formal Description of Ada*, volume 98 of *LNCS*. Springer, Heidelberg, Germany, 1980.

[17] 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.

[18] Patrick Cousot. *Principles of Abstract Interpretation*. The MIT Press, 2021.

[19] Peter Fettke and Wolfgang Reisig. *Understanding the Digital World – Modeling with HERAKLIT*. Springer, 2024. To be published.

[20] 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.

[21] 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).

[22] 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.

[23] 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.

[24] 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).

[25] Michael A. Jackson. *Software Requirements & Specifications: a lexicon of practice, principles and prejudices*. ACM Press. Addison-Wesley, Reading, England, 1995.

[26] 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.

[27] 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.

[28] 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.

[29] Kai Sørlander. *Indføring i Filosofien [Introduction to The Philosophy]*. Informations Forlag, Copenhagen, Denmark, 2016. 233 pages.

[30] Kai Sørlander. *Den rene fornufts struktur [The Structure of Pure Reason]*. Ellekær, Slagelse, Denmark, 2022. See [31].

[31] Kai Sørlander. *The Structure of Pure Reason*. Publisher to be decided, 2023. This is an English translation of [30] – done by Dines Bjørner in collaboration with the author.

[32] Achille C. Varzi. *On the Boundary between Mereology and Topology*, pages 419–438. Hölder-Pichler-Tempsky, Vienna, 1994.