A Domain Analysis & Description Method

Principles, Techniques and Modelling Languages

Dines Bjørner

Technical University of Denmark, DK 2800 Kgs. Lyngby, Denmark

December 4, 2018: 11:23 am

Abstract

- We present a *method* for
  analysing and describing domains.
- By a **domain** we shall understand
  - a **rationally describable** segment of
  - a **human assisted** reality, i.e., of the world,
    - its **physical parts:**
      * **natural** [“God-given”] and
      * **artifactual** [“man-made”],
    - and **living species**:
      * **plants** and
      * **animals**
        including, notably, **humans**.

- These are
  - **endurants** (“still”),
  - as well as **perdurants** (“alive”).
- Emphasis is placed on **“human-assistedness”**,
  - that is, that there is *at least one (man-made)* **artifact**
  - and, therefore, that **humans** are a primary cause for
    - change of endurant **states**
    - as well as perdurant **behaviours**.

- By a **method** we shall mean
  - a set of **principles** of **analysis**
  - and for **selecting** and **applying**
  - a number of **techniques** and **tools**

  in the construction of some artifact, say a domain description.
- We shall present a method for constructing domain descriptions.
- Among the tools we shall only be concerned with are the analysis and synthesis languages.

- *Domain science & engineering* marks a new area of *computing science*.
  - Just as we are *formalising*
  - the *syntax and semantics of programming languages*,
  - so we are *formalising*
  - the *syntax and semantics of human-assisted domains*.

- Just as *physicists* are studying the *natural physical world*,
  - endowing it with *mathematical models*,
  - so we, *computing scientists*, are studying these *domains*,
  - endowing them with *mathematical models*,
- A difference between the endeavours of physicists and ours lies in the tools:
  - the physics models are based on *classical mathematics, differential equations* and *integrals,* etc.;
  - our models are based on *mathematical logic*, *set theory*, and *algebra*.
- Where physicists thus classically use a variety of *differential* and *integral calculi* to model the physical world,
- we shall be using the *analysis & description calculi* presented

in this paper to model primarily artifactual domains.

1. Introduction

1.1. Foreword

- Dear student!
  - You are about to embark on a journey.
  - The lectures in front of us are many !
  - But it is not the number of lecture slides, 437,
  - or duration of my unfolding the slides
    that I am referring to.

- It is the mind that should be prepared for a journey.
- It is a journey into a new realm.
- A realm where we confront the computer & computing scientists with a new universe:
  - a universe in which we build a bridge between the *informal* world,
    - that we live in,
    - the context for eventual, *formal* software,
  - and that *formal* software.
  - The bridge involves
    - a novel construction, new in computing science:
    - a **transcendental deduction**.

- We are going to present you, we immodestly claim,
  - with a new way of looking at the "origins" of software,
  - the domain in which it is to serve.
- We shall show a method,
  - a set of principles and techniques and a set of languages,
    - some formal, some "almost" formal,
    - and the informal language of usual computing science papers
  - for a systematic to rigorous way of
    - *analysing & describing domains.*
- We immodestly claim that
  such a method has not existed before.

1.2. An Engineering and a Science Viewpoint

1.2.1. A Triptych of Software Development

- It seems reasonable to expect that
  - before **software** can be designed
  - we must have a reasonable grasp of its **requirements**;
  - before **requirements** can be expressed
  - we must have a reasonable grasp of the underlying **domain**.

- It therefore seems reasonable to structure software development into:
  - **domain engineering**, in which "the underlying" domain is *analysed and described*[1];
  - **requirements engineering**, in which requirements are *analysed and prescribed* – such as we suggest it [**?**, **?**] – based on a domain description[2]; and
  - **software design**, in which the software is *rigorously "derived"* from a requirements prescription[3].
- Our interest, in this paper, lies sôlely in domain analysis & description.

---

[1] including the statement and possible proofs of properties of that which is denoted by the domain description

[2] including the statement and possible proofs of properties of that which is denoted by the requirements prescription with respect also to the domain description

[3] including the statement and possible proofs of properties of that which is specified by the software design with respect to both the requirements prescription and the domain description

1.2.2. Domain Science & Engineering:

- The present paper outlines a *methodology* for an aspect of software development.
- Domain analysis & description can be pursued in isolation, for example, without any consideration of any other aspect of software development.
- As such domain analysis & description represents an aspect of **domain science & engineering**.

- Other aspects are covered in:
  - [**?**, *Domain Facets*],
  - [**?**, *Requirements Engineering*],
  - [**?**, *An Analysis & Description Process Model*],
  - [**?**, *From Mereologies to Lambda-Expressions*] and in
  - [**?**, *A Philosophy Basis*].
- This work is over-viewed in [**?**, Domain Science & Engineering – A Review of 10 Years Work].
- They are all facets of an emerging **domain science & engineering**.
- *We consider the present paper to outline the basis for this science and engineering.*

1.3. Some Issues: Metaphysics, Epistemology, Mereology and Ontology

- But there is an even more fundamental issue "at play" here.
  - It is that of philosophy.
  - Let us briefly review some aspects of philosophy.
- **Metaphysics**
  - is a branch of *philosophy* that explores fundamental questions, including the nature of concepts like
  - *being, existence*, and *reality* ■[4]

[4] ■ is used to signal the end of a characterisation, a definition, or an example.

- Traditional metaphysics seeks to answer,
  - in a "suitably abstract and fully general manner",
  - the questions:
    - *What is there?* and
    - *And what is it like?* [5].

[5] https://en.wikipedia.org/wiki/Metaphysics

- Topics of metaphysical investigation include
  - existence,
  - objects and their properties,
  - space and time,
  - cause and effect, and
  - possibility.
- **Epistemology**
  - is the branch of philosophy concerned with
  - the theory of knowledge[6] ■

[6] https://en.wikipedia.org/wiki/Epistemology

- *Epistemology* studies the nature of
  - knowledge, justification, and the rationality of belief.
  - Much of the debate in epistemology centers on four areas:
    - (1) the philosophical analysis of the nature of knowledge and how it relates to such concepts as truth, belief, and justification,
    - (2) various problems of skepticism,
    - (3) the sources and scope of knowledge and justified belief, and
    - (4) the criteria for knowledge and justification.
  - A central branch of *epistemology* is *ontology*.[7]

[7] https://en.wikipedia.org/wiki/Metaphysics

- **Ontology:** An *ontology* encompasses
  - a representation,
  - formal naming, and
  - definition
  - of the categories,
  - properties, and
  - relations
- of the entities that substantiate one, many, or all domains.[8].
- An *upper ontology* (also known as a top-level ontology or foundation ontology) is an ontology which consists of very general terms (such as *entity, endurant, attribute*) that are common across all domains[9] ■

[8] https://en.wikipeda.org/wiki/On-tology_(information_science)
[9] https://en.wikipedia.org/wiki/Upper_ontology

- **Mereology** (from the Greek $\mu\varepsilon\rho o\varsigma$ 'part') is the theory of part-hood relations:
  - of the relations of part to whole
  - and the relations of part to part within a whole [**?**][10] ■

[10] https://plato.stanford.edu/entries/mereology

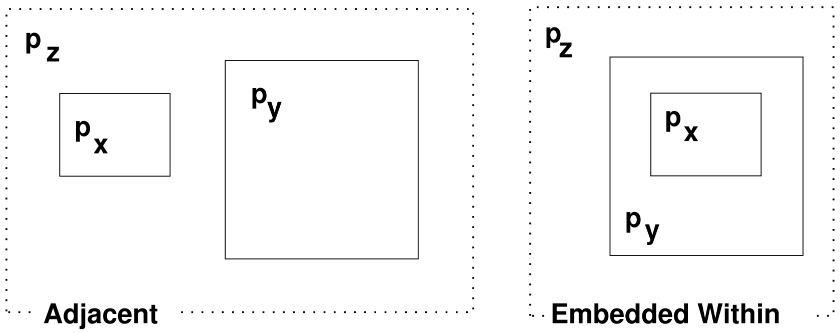


Figure 1: Immediately 'Adjacent' and 'Embedded Within' Parts

- Accordingly two parts, p_x and p_y, (of a same "whole")
  - are either "adjacent",
  - or are "embedded within", one within the other,

  as loosely indicated in Fig. 1.

- ‘Adjacent’ parts
  - are direct parts of a same third part, p_z,
  - i.e., p_x and p_y are “embedded within” p_z;
  - or one (p_x) or the other (p_y) or both (p_x and p_y) are parts of a same third part, p'_z “embedded within” p_z;
  - et cetera;

  as loosely indicated in Fig. 2 on the next slide,
- or one is “embedded within” the other — etc.
  as loosely indicated in Fig. 2 on the following slide.

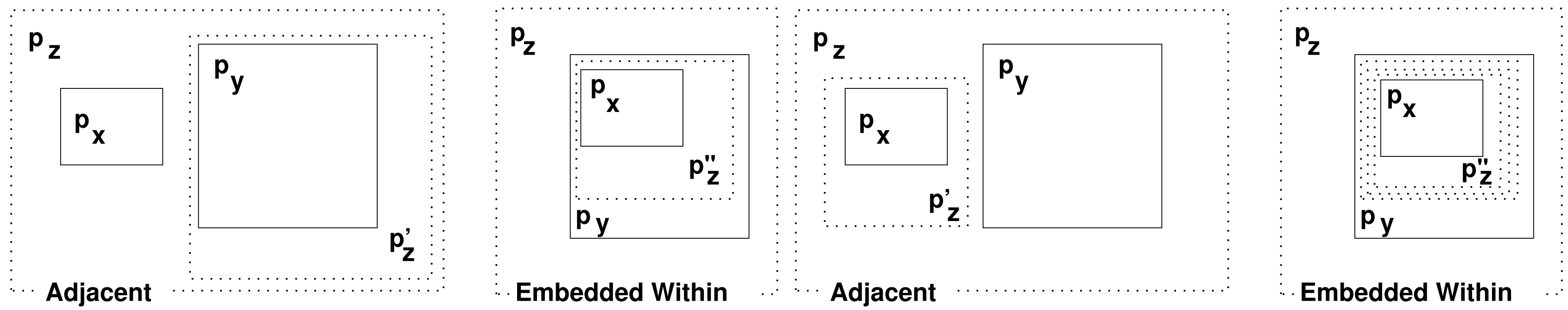
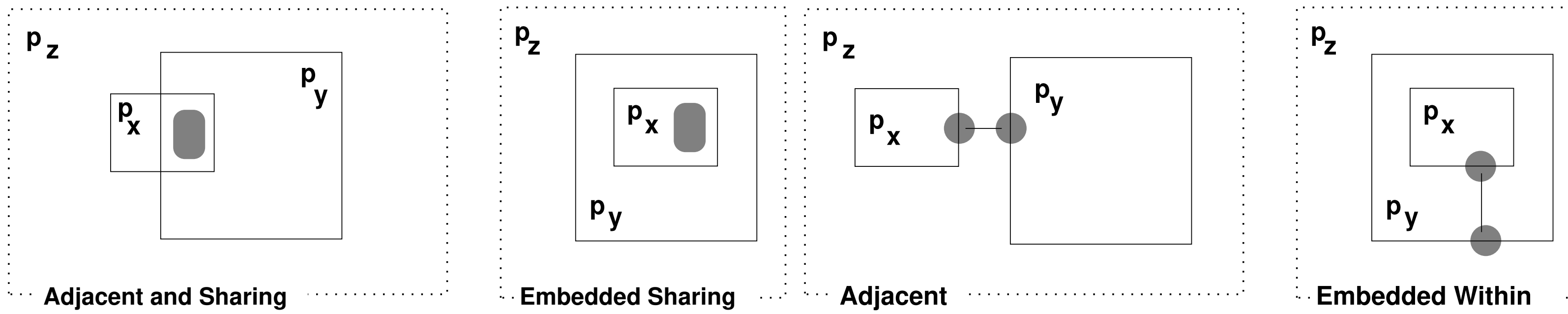


Figure 2: Transitively 'Adjacent' and 'Embedded Within' Parts

- Parts, whether 'adjacent' or 'embedded within', can share properties.
  - For adjacent parts this sharing seems, in the literature, to be diagrammatically expressed by letting the part rectangles "intersect".
  - Usually properties are not spatial hence 'intersection' seems confusing.

– We refer to Fig. 3.

[L] 

[R]

Figure 3: Two models, [L,R], of parts sharing properties

– Instead of depicting parts sharing properties as in Fig. 3[L]eft,
  - where shaded, dashed rounded-edge rectangles stands for ‘sharing’,
– we shall (eventually) show parts sharing properties as in Fig. 3[R]ight
  - where •—• connections connect those parts.

We refer to [**?**, *From Mereologies to Lambda-Expressions*].

- Mereology is basically the contribution [**?**, **?**] of the Polish philosopher, logician and mathematician Stanisław Leśniewski (1886–1939).

1.3.1. Kai Sørlander's Philosophy:

- We shall base some of our modelling decisions of Kai Sørlander's Philosophy [?, ?, ?, ?].
- A main contribution of Kai Sørlander is, on the philosophical basis of the *possibility of truth*
  (in contrast to Kant's *possibility of self-awareness*),
  - to *rationally* and *transcendentally deduce*
  - **the absolutely necessary conditions for describing any world**.

- These conditions presume a *principle of contradiction* and lead to the *ability*
  - to *reason* using *logical connectives* and
  - to *handle asymmetry, symmetry* and *transitivity*.
  - *Transcendental deductions* then lead to
  - *space* and *time*,
  - not as priory assumptions, as with Kant,
  - but derived facts of any world.

- From this basis Kai Sørlander then, by further transcendental deductions, arrive at
  - kinematics,
  - dynamics and
  - the bases for Newton's Laws.
- And so forth.
- We build on Sørlander's basis to argue
  - that the domain analysis & description calculi are necessary and sufficient
  - for the analysis & description of domains and
  - that a number of relations between domain entities
  - can be understood transcendentally and
  - as "variants" of laws of physics, biology, etc. !

1.4. The Precursor

- The present lectures are based on a revision of the published [?].
- The major revision that prompts this complete rewrite is due to a serious study of Kai Sørlander's Philosophy.
- As a result we extend [?]'s ontology of endurants: describable phenomena to not only cover those of **physical phenomena**, but also those of **living species**, notably **humans**, and, as a result of that, our understanding of discrete endurants is refined into those of **natural parts** and **artifacts**.
- A new contribution is that of **intentional "pull"** akin to the *gravitational pull* of physics.

- Both these lectures and [**?**] are the result of extensive “non-toy” example case studies, see the example: *Universes of Discourse* – on Page 50.
- The last half of these were carried out in the years since [**?**] was first submitted (i.e., 2014).
- The present lectures omit the extensive introduction[11] and closing of [**?**, Sects. 1 and 5].
- Most notably, however, is a clarified view on the transition from **parts** to **behaviours**, a **transcendental deduction** from *domain space* to *domain time*.

[11] **Note added in proof:** Omitted from the extensive, five page, literature survey of [**?**] was [**?**, Section 5.3]. It is an interesing study of the domain of geographics.

1.5. What are these Lectures About ?

- We present a *method* for *analysing &*[12] *describing domains*.

[12] By *A&B* we mean one topic, the confluence of topics *A* and *B*.

Definition 1 Domain:

- By a **domain** we shall understand
  - a **rationally describable** segment of
  - a **human assisted** reality, i.e., of the world,
    - its **physical parts**,
      - **natural** [“God-given”] and
      - **artifactual** [“man-made”],
    - and **living species**:
      - **plants** and
      - **animals**

        including, predominantly, **humans**.

- These are
  - **endurants** (“still”)
  - as well as **perdurants** (“alive”).
- Emphasis is placed on **“human-assistedness”**,
  - that is, that there is *at least one (man-made)* **artifact**
  - and that **humans** are a primary cause for
    - change of endurant **states**
    - as well as perdurant **behaviours** ■

Definition 2 **Domain Description:** By a **domain description** we shall understand

- a combination of **narration** and **formalisation** of a domain.
- A **formal specification** is a collection of
  - *sort*, or *type* definitions,
  - *function* and *behaviour* definitions,
  - together with *axiom*s and *proof obligation*s constraining the definitions.

- A **specification narrative** is a natural language text which in terse statements introduces
  - the names of (in this case, the domain),
  - and, in cases, also the definitions, of
  - sorts (types), functions, behaviours and axioms;
  - not anthropomorphically, but by emphasizing their properties ■

- *Domain descriptions* are (to be) void of any reference to future, contemplated software, let alone IT systems, that may support entities of the domain.
  - As such *domain models*[13]
    - can be studied separately, for their own sake, for example as a basis for investigating possible domain theories, or
  - can, subsequently, form the basis for requirements engineering
  - with a view towards development of ('future') software, etc.
- *Our aim is to provide a method for the precise analysis and the formal description of domains.*

[13] We use the terms *'domain descriptions'* and *'domain models'* interchangeably.

2. Entities: Endurants and Perdurants
2.1. A Generic Domain Ontology – A Synopsis

- Figure 4 on Slide 40 shows an *upper ontology* for domains such a defined in Defn. 1 on Slide 33.
- Kai Sørlander's Philosophy justifies our organising the *entities* of any describable domain, for example[14], as follows:
  - We shall review Fig. 4 on Slide 40 by means of a top-down, left-traversal of the tree (whose root is at the top).

[14] We could organise the ontology differently: entities are either naturals, artifacts or living species, et cetera. If an upper node (•) satisfies a predicate $\mathscr{P}$ then all descendant nodes do likewise.

- There are
  - *describable* phenomena and there are
  - phenomena that we cannot describe.
  - The former we shall call *entities*.
- The *entities* are
  - either *endurants*, “still” entities – existing in *space*,
  - or *perdurants*, “alive” entities – existing also in *time*.

•

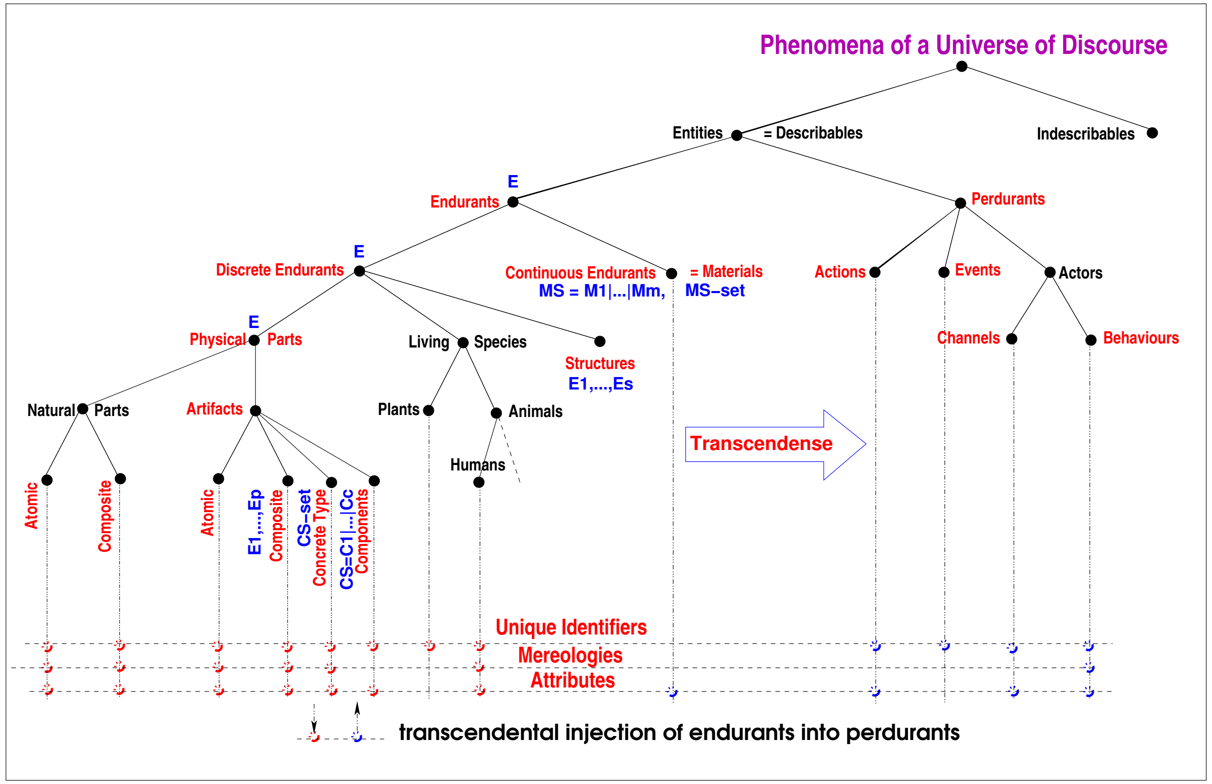


Figure 4: An Upper Ontology for Domains

- *Endurants* are
  - either *discrete*
  - or *continuous* –
  - in which latter case we call them *materials*[15].
- *Discrete* endurants are
  - *physical parts*,
  - *living species*, or are
  - *structures*.

[15] Please observe that *materials* were either *natural* or *artifactual*, but that we do not "bother" in this paper. You may wish to slightly change the ontology diagram to reflect a distinction.

- *Physical parts* are
  - either *naturals*,
  - *artifacts*, i.e. man-made.
- Natural and man-made parts are either
  - *atomic* or
  - *composite*.

- We additionally analyse artifacts into
  - either *components*[16],
  - or *sets* of *parts*.
- That additional analysis could also be expressed for natural parts but as we presently find no use for that we omit such further analysis.

[16] Whether a discrete endurant as we shall soon see, is treated as a part or a component is a matter of pragmatics. Again cf. Footnote 15.

- *Living Species* are
  - either *plants*
  - or *animals*.
- Among animals we have the *humans*.

- *Structures* consist of one or more endurants.
  - Structures and components really are parts, but for pragmatic reasons we choose to not model them as [full fledged] parts.

- The categorisation into
  - structures,
  - natural parts,
  - artifactual parts,
  - plants,
  - animals, and
  - components

  is thus partly based in Sørlander's Philosophy, partly pragmatic.

- The distinction between endurants and perdurants,
  are necessitated by Sørlander
  as being in space, respectively in space **and** time;
  - discrete and continuous are motivated by arguments of natural sciences;
  - structures and components are purely pragmatic;
  - plants and animals, including humans,
    are necessitated by Kai Sørlander's Philosophy.
- The distinction between natural, physical parts, and artifacts
  is not necessary in Sørlander's Philosophy,
  but, we claim, necessary, philosophically,
  in order to perform the *intentional "pull"*, a transcendental deduction.

- **On Pragmatics:**
  - We have used the term ‘pragmatic’ a few times.
  - On one hand there is philosophy’s need for absolute clarity.
  - On the other hand, when applying the
  - natural part, artifactual part, and living species,
  - concepts in practice, there can be a need for “loosening” up.
  - As for example: a structure really is a collection of parts and relations between them.
  - As we shall later see, parts are transcendentally to be understood as behaviours.
  - We know that modelling is imperative when we model a domain,
  - but we may not wish to model a discrete endurant as a behaviour
  - so we decide, pragmatically, to model it as a structure.

- Our reference, here, to Kai Sørlander's Philosophy, is very terse.
  - We refer to a detailed research report:
  - *A Philosophy of Domain Science & Engineering*[17]
  - for carefully reasoned arguments.
- That report is under continued revision:
  - It reviews the domain analysis & description method;
  - translates many of Sørlander's arguments
  - and relates, in detail, the "options"
  - of the domain analysis & description approach
    to Sørlander's Philosophy.

[17] http://www.imm.dtu.dk/˜dibj/2018/philosophy/filo.pdf

2.2. Universes of Discourse

- By a **universe of discourse** we shall understand
  - the same as the **domain of interest**,
  - that is, the *domain* to be *analysed & described* ■

Example 0: Example 0: Universes of Discourse

- **railways** [?, ?, ?],
- **container shipping** [?],
- **stock exchange** [?],
- **oil pipelines** [?],
- **"The Market"** [?],
- **Web systems** [?],
- **weather information** [?],
- **credit card systems** [?],
- **document systems** [?],
- **urban planning** [?],
- **swarms of drones** [?],
- **container terminals** [?]

■

- It may be a **"large" domain**, that is, consist
  - of many, as we shall see, *endurants* and *perdurants*,
  - of many *parts*, *components* and *materials*,
  - of many *humans* and *artifacts*,
  - and of many*actors*, *actions*, *events* and *behaviours*.
- Or it may be a **"small" domain**, that is, consist
  - of a few such entities.

- The choice of "boundaries", that is,
  - of how much or little to include, and
  - of how much or little to exclude
- is entirely the choice of the domain engineer cum scientist:
  - the choice is crucial, and is not always obvious.
  - The choice delineates an *interface*,
    - that is, that which is within the boundary, i.e., is in the domain,
    - and that which is without, i.e., outside the domain, i.e., is the **context of the domain**,
    - that is, the **external domain interfaces**.
  - Experience helps set reasonable boundaries.

- There are two "situations":
  - Either a domain analysis & description endeavour is pursued in order to
    - prepare for a subsequent development of *requirements modelling*,
    - in which case one tends to choose a **"narrow" domain**,
    - that is, one that "fits", includes, but not much more,
    - the domain of interest for the requirements.

- Or a domain analysis & description endeavour is pursued in order to research a domain.
  - *Either* one that can form the basis
  - for subsequent engineering studies
  - aimed, eventually at requirements development;
  - in this case “wider” boundaries may be sought.
- *Or* one that experimentally “throws a larger net”,
  - that is, seeks a “large” domain
  - so as to explore interfaces
  - between what is thought of as **internal system interfaces**.

- Where, then, to start the *domain analysis & description* ?
  - Either one can start "bottom-up", that is,
    - with atomic entities: endurants or perdurants,
    - one-by-one, and work one's way "out",
    - to include composite entities, again endurants or perdurants,
    - to finally reach some satisfaction:
    - *Eureka*, a goal has been reached.
  - Or one can start "top-down", that is, "casting a wide net".
  - The choice is yours.
- Our presentation, however, is "top down":
  most general domain aspects first.

Example 1: Example 1: Universe of Discourse

- The universe of discourse is *road transport systems*.
  - We analyse & describe not the class of all road transport systems
  - but a representative subclass, UoD, is *structured* into such notions as
    - a road net, RN, of hubs, H, (intersections) and links, L, (street segments between intersections);
    - a fleet of vehicles, FV, *structured* into companies, BC, of buses, B, and pools, PA, of private automobiles, A (et cetera);
    - et cetera.
  - See Fig. 5 on Slide 147.

■

2.3. Entities

Characterisation 1 **Entity:**

- By an **entity** we shall understand a **phenomenon**, i.e., something
  - that can be *observed*, i.e., be
    - seen or touched by humans,
    - *or* that can be *conceived*
    - as an *abstraction* of an entity;
  - alternatively,
    - a phenomenon is an entity, *if it exists, it is* **"being"**,
    - *it is that which makes a "thing" what it is: essence, essential nature* ■

Analysis Prompt 1 *is_entity:*

- *The domain analyser analyses "things" (θ) into entities or non-entities.*
- *The method can thus be said to provide the domain analysis prompt:*
  - *is_entity* – *where* is_entity(θ) *holds if* θ *is an entity*[18] ■
- is_entity *is said to be a prerequisite prompt for all other prompts.*

[18]Analysis prompt definitions and description prompt definitions and schemes are delimited by ■

- *To sum up:*
  - *An entity is what we can analyse and describe*
    - *using the analysis & description prompts*
    - *outlined in these lectures.*
- *The entities that we are concerned with*
  - *are those with which Kai Sørlander's Philosophy is likewise concerned.*
  - *They are the ones that are unavoidable in any*
  - *description of any possible world.*

- *And then, which are those entities ?*
  - *In both [**?**] and [**?**]*
  - *Kai Sørlander rationally deduces that these entities*
  - *must be in space and time,*
  - *must satisfy laws of physics – like those of Newton and Einstein,*
  - *but among them are also living species: plants and animals and hence humans.*
  - *The living species, besides still*
  - *being in space and time, and satisfying laws of physics,*
  - *must satisfy further properties – which we shall outline later.*

2.4. Endurants and Perdurants

- The concepts of endurants and perdurants
  - are not present in,
  - that is, are not essential

  to Sørlander's Philosophy.
- Since our departure point is that of *computing science*
  - where, eventually, conventional computing performs operations on, i.e. processes data,
  - we shall, however, introduce these two notions:
  - *endurant* and *perdurant*.
  - The former, in a rough sense, "corresponds" to data;
  - the latter, similarly, to processes.

***Characterisation 2* Endurant:**

- By an **endurant** we shall understand an entity
  - that can be observed, or conceived and described, as a "complete thing" at no matter which given snapshot of time;
  - alternatively an entity is endurant if it is capable of *enduring*, that is *persist*, *"hold out"*.

  Were we to "freeze" time
  - we would still be able to observe the entire endurant ■

Example 2: Example 2: Endurants Geography Endurants:

- *The geography of an area, like some island, or a country, consists of*
  - *its geography – "the lay of the land",*
  - *the geodetics of this land,*
  - *the meteorology of it,*
  - *et cetera.*

Railway System Endurants:

- *Example railway system endurants are:*
  - *a railway system,*
  - *its net,*
  - *its individual tracks,*
  - *switch points,*
  - *trains,*
  - *their individual locomotives,*
  - *et cetera.*

Analysis Prompt 2 *is_endurant:*

- *The domain analyser analyses an entity, ϕ, into an endurant as prompted by the domain analysis prompt:*
  - *is_endurant* – ϕ *is an endurant if* *is_endurant*(ϕ) *holds.*
- *is_entity is a prerequisite prompt for is_endurant* ■

***Characterisation 3* Perdurant:**

- By a **perdurant** we shall understand an entity
  - for which only a fragment exists
    if we look at or touch them
    at any given snapshot in time.
  - Were we to freeze time we would only see or touch
    a fragment of the perdurant,
  - alternatively
    - an entity is perdurant
    - if it endures continuously, over time, persists, lasting ■

Example 3: Example 3: Perdurants Geography:

- *Example geography perdurants are:*
  - *the continuous changing of the weather (meteorology);*
  - *the erosion of coast lines;*
  - *the rising of some land and the "sinking" of other land areas;*
  - *volcano eruptions;*
  - *earth quakes;*
  - *et cetera.*

Railway Systems:

- *Example railway system perdurants are:*
  - *the ride of a train from one railway station to another; and*
  - *the stop of a train at a railway station*
    from some arrival time to some departure time.

Analysis Prompt 3 `is_perdurant:`

- *The domain analyser analyses an entity e into perdurants as prompted by the domain analysis prompt:*
  - `is_perdurant` – *e is a perdurant if* `is_perdurant(e)` *holds.*
- `is_entity` *is a prerequisite prompt for* `is_perdurant` ■

- *Occurrent* is a synonym for perdurant.

3. Endurants: Analysis of External Qualities

3.1. Discrete and Continuous Endurants

Characterisation 4 **Discrete Endurant:**

- By a **discrete endurant** we shall understand an endurant which is
  - separate,
  - individual or
  - distinct

  in form or concept ■

- *To simplify matters we shall allow separate elements of a discrete endurant to be continuous !*

Example 4: Example 4: Discrete Endurants

- *The individual endurants of the above example of railway system endurants were all discrete.*
- *Here are examples of discrete endurants of pipeline systems.*
  - *A pipeline and*
  - *its individual units:*
    - *pipes,*
    - *valves,*
    - *pumps,*
    - *forks,*
    - *etc.*

Analysis Prompt 4 *is_discrete:*

- *The domain analyser analyses endurants e into discrete entities as prompted by the domain analysis prompt:*
  - *is_discrete* – *e is discrete if is_discrete(e) holds* ■

Characterisation 5 Continuous Endurant:

- By a **continuous endurant** we shall understand an endurant which is
  - prolonged, without interruption,
  - in an unbroken series or pattern ■
- *We shall prefer to refer to continuous endurants as materials*
- *and otherwise cover materials in Sect. on Slide 125.*

Example 5: Example 5: Materials

- *Examples of materials are:*
  - *water, oil, gas, compressed air, etc.*
- *A container, which we consider a discrete endurant,*
  - *may contain a material,*
  - *like a gas pipeline unit may contain gas.*

Analysis Prompt 5 *is_continuous*:

- *The domain analyser analyses endurants e into continuous entities as prompted by the domain analysis prompt:*
  - *is_continuous* – *e is continuous if is_continuous(e) holds*

■

- Continuity shall here not be understood in the sense of mathematics.
  - Our definition of 'continuity' focused on
    - *prolonged,*
    - *without interruption,*
    - *in an unbroken series or*
    - *pattern.*
  - In that sense
    materials
    shall be seen as 'continuous'.
- The mathematical notion of 'continuity' is an abstract one.
- The endurant notion of 'continuity' is physical one.

3.2. Discrete Endurants

- We analyse discrete endurants into
  - *physical parts*,
  - *living species* and
  - *structures*.
- Physical parts and living species can be identified as separate entities – following Kai Sørlander's Philosophy.
- To model discrete endurants as structures represent a pragmatic choice which relieves the domain describer from transcendentally considering structures as behaviours.

3.2.1. Physical Parts

***Characterisation 6* Physical Parts:**

- By a *physical part* we shall understand
  - a discrete endurant existing in time and
  - subject to laws of physics,
  - including the *causality principle* and
  - *gravitational pull*[19] ■

[19]This characterisation is the result of our study of relations between philosophy and computing science, notably influenced by Kai Sørlander's Philosophy. We refer to our research report [**?**, www.-imm.dtu.dk/˜dibj/2018/philosophy/filo.pdf].

Analysis Prompt 6 `is_physical_part:`

- *The domain analyser analyses "things" (η) into physical part.*
- *The method can thus be said to provide the domain analysis prompt:*
  - `is_physical_part` – *where* `is_physical_part(η)` *holds if* η *is a physical part* ■
- *Section continues our treatment of physical parts.*

3.2.2. Living Species

Definition 3 Living Species, I:

- By a *living species* we shall understand
  - a discrete endurant,
  - subject to laws of physics, and
  - additionally subject to *causality of purpose.*[20]
- [Defn. 9 on Slide 109 elaborates further on this point] ■

[20] See Footnote 19 on Slide 77.

Analysis Prompt 7 *is_living_species:*

- *The domain analyser analyses "things" (e) into living species.*
- *The method can thus be said to provide the domain analysis prompt:*
  - *is_living_species* – *where is_living_species(e) holds if e is a living species* ■

- Living species
  - have a *form* they can *develop* to reach;
  - they are *causally* determined to *maintain* this form;
  - and they do so by *exchanging matter* with an *environment*.
  - We refer to [**?**] for details.
- Section continues our treatment of living species.

3.2.3. Structures

Definition 4 Structure: By a **structure** we shall understand

- a discrete endurant
- which the domain engineer chooses
- to describe as consisting of one or more endurants,
- whether discrete or continuous,
- but to **not** endow with **internal qualities**:
  - unique identifiers,
  - mereology or
  - attributes ■

- *Structures* are "conceptual endurants".
  - A *structure* "gathers" one or more endurants under "one umbrella",
  - often simplifying a presentation of some elements of a domain description.
- Sometimes, in our domain modelling, we choose
  - to model an endurant as a *structure*,
  - sometimes as a *physical part*;
  - it all depends on what we wish to focus on
  - in our domain model.

- As such structures are “compounds”
  – where we are interested
  – only in the (external and internal) qualities
  – of the elements of the compound,
  – but not in the qualities
  – of the structure itself.

Example 6: Example 6: Structures

- *A transport system is modelled as structured into*
  - *a road net structure and*
  - *an automobile structure.*
- *The road net structure is then structured as a pair:*
  - *a structure of hubs and*
  - *a structure of links.*
- *These latter structures are then modelled as set of hubs, respectively links.*

Example 7: Example 7: Structures – Contd.

- *We could have modelled the road net structure*
  - *as a composite part*
  - *with unique identity, mereology and attributes*
  - *which could then serve to model*
  - *a `road net authority`.*
- *We could have modelled the automobile structure*
  - *as a composite part*
  - *with unique identity, mereology and attributes*
  - *which could then serve to model*
  - *a `department of vehicles`.*

- The concept of *structure* is new.
- Whether to analyse & describe a discrete endurant into a structure or a physical part is a matter of choice.
- If we choose to analyse a discrete endurant into a *physical part* then it is because we are interested in endowing the part with *qualities*, the
  - unique identifiers,
  - mereology and
  - one or more attributes.
- If we choose to analyse a discrete endurant into a *structure* then it is because we are **not** interested in endowing the endurant with *qualities*.

- When we choose that an endurant sort should be modelled as a part sort
  - with unique identification, mereology and proper attributes,
  - then it is because we eventually shall consider the part sort
  - as being the basis for transcendentally deduced behaviours.

Analysis Prompt 8 *is_structure:*

- *The domain analyser analyse endurants, e, into structure entities as prompted by the domain analysis prompt:*
  - *is_structure e is a structure if is_structure(e) holds* ■

- We shall now treat the external qualities of discrete endurants:
  - *physical parts* (Sect.) and
  - *living species* (Sect.).
- After that we cover
  - *components* (Sect.),
  - *materials* (Sect.) and
  - *artifacts* (physical man-made parts, Sect.) .

- We remind the listener
  - that in this section, i.e. Sect. , we cover only the *analysis calculus* for *external qualities*;
  - the *description calculus* for *external qualities* is treated in Sect. .
- The analysis and description calculi for internal qualities is covered in Sect. .

3.3. Physical Parts

- Physical parts are
  - either *natural parts*,
  - or *components*,
  - or *sets of parts* of the same type,
  - or are *artifacts* i.e. man-made parts.

- The categorisation of physical parts into these four is pragmatic.
  - *Physical parts* follow from Kai Sørlander's Philosophy.
  - *Natural parts* are what Sørlander's Philosophy is initially about.
  - *Artifacts* follow from *humans* acting according to their *purpose* in making "physical parts".
  - *Components* is a simplification of natural and man-made parts.
  - *Set of parts* is a simplification of composite natural and composite man-made parts as will be made clear in Sect. .

3.3.1. Natural Parts

Characterisation 7 **Natural Parts:**

- Natural parts
  - are in *space* and *time*;
  - are subject to the *laws of physics*,
  - and also subject to
    - the *principle of causality*
    - and *gravitational pull* ■
- *The above is a factual characterisation of natural parts.*
- *The below is our definition – such as we shall model natural parts.*

***Definition 5* Natural Part:**

- By a **natural part** we shall understand
  - a *physical part*
  - which the domain engineer chooses
  - to endow with all three **internal qualities**:
    - unique identification,
    - mereology, and
    - one or more attributes ■

3.3.2. Artifacts

***Characterisation 8* Man-made Parts: Artifacts:**

- Artifacts are man-made either discrete or continuous endurants.
  - In this section we shall only consider discrete endurants.
  - Man-made continuous endurants are not treated separately but are "lumped" with [natural] materials.
  - Artifacts are
    - subject to the *laws of physics* ■

- *The above is a factual characterisation of discrete artifacts.*
- *The below is our definition – such as we shall model discrete artifacts.*

Definition 6 **Artifact:**

- By an **artifact** we shall understand
  - a *man-made physical part*
  - which, like for *natural parts*, the domain engineer chooses
  - to endow with all three **internal qualities**:
    - unique identification,
    - mereology, and
    - one or more attributes ■

3.3.3. Parts

We revert to our treatment of parts.

Example 8: Example 8: Parts

- *The geography examples (of Page 63) of are all natural parts.*
- *The railway system examples (of Page 63) are all artifacts*

- Except for the *intent* attribute of artifacts,
  we shall, in the following, treat

  - *natural* and
  - *artifactual*

  parts on par, i.e., just as physical parts.

Analysis Prompt 9 *is_part*:

- *The domain analyser analyse endurants, e, into part entities as prompted by the domain analysis prompt:*
  - *is_part e is a part if is_part(e) holds* ■

3.3.4. Atomic and Composite Parts:

- A distinguishing quality
  - of natural and artifactual parts
  - is whether they are
    - atomic or
    - composite.
- Please note that we shall,
  - in the following,
  - examine the concept of parts
  - in quite some detail.
- That is, parts become the domain endurants of main interest, whereas components, structures and materials become of secondary interest.

- This is a choice.
  - The choice is based on pragmatics.
  - It is still the domain analyser cum describers' choice whether to consider a discrete endurant
    - a part
    - or a component,
    - or a structure.
  - If the domain engineer wishes to investigate
    - the details of a discrete endurant
    - then the domain engineer chooses to model[21]
    - the discrete endurant as a part
    - otherwise as a component.

[21] We use the term *to model* interchangeably with the composite term *to analyse & describe*; similarly *a model* is used interchangeably with *an analysis & description*.

3.3.5. Atomic Parts

Definition 7 Atomic Part:

- **Atomic part**s are those which,
  - in a given context,
  - are deemed to *not* consist of meaningful, separately observable proper *sub-part*s.
- A **sub-part** is a *part* ■

Analysis Prompt 10 *is_atomic:*

- *The domain analyser analyses a discrete endurant, i.e., a part p into an atomic endurant:*
  - *is_atomic: p is an atomic endurant if* is_atomic(p) *holds* ■

Example 9: Example 9: Atomic Road Net Parts

- *From one point of view all of the following can be considered atomic parts:*
  - *hubs, links*[22]*, and*
  - *automobiles.*

[22] Hub ≡ street intersection; link ≡ street segments with no intervening hubs.

3.3.6. Composite Parts

Definition 8 Composite Part:

- Composite parts are those which,
  - in a given context,
  - are deemed to *indeed* consist of
    meaningful, separately observable proper *sub-part*s ■

Analysis Prompt 11 *is_composite:*

- *The domain analyser analyses a discrete endurant, i.e., a part p into a composite endurant:*
  - *is_composite: p is a composite endurant if is_composite(p) holds* ■
- `is_discrete` is a prerequisite prompt of both `is_atomic` and `is_composite`.

Example 10: Example 10: Composite Automobile Parts

- *From another point of view all of the following can be considered composites parts:*
  - *an automobile,*
    consisting of, for example, the following parts:
    - *the engine train,*
    - *the chassis,*
    - *the car body,*
    - *the doors and*
    - *the wheels.*
  - *These can again be considered composite parts.*

3.4. Living Species

- We refer to Sect.
  for our first characterisation (Slide 79) of the concept of *living species*[23]:
  - a discrete endurant existing in time,
  - subject to laws of physics, and
  - additionally subject to *causality of purpose*[24]

[23] See analysis prompt 7 on Slide 80.
[24] See Footnote 19 on Slide 77.

Definition 9 Living Species, II:

- Living species
  - must have some *form they can be developed to reach*;
  - which they must be *causally determined to maintain*.
  - This *development and maintenance* must further in an *exchange of matter with an environment*.

- It must be possible that living species occur in one of two forms:
  - one form which is characterised by *development, form and exchange*;
  - another form which, **additionally**, can be characterised by the *ability to purposeful movement*
- The first we call **plants**,
- the second we call **animals** ■

Analysis Prompt 12 *is_living_species:*

- *The domain analyser analyse discrete endurants, ℓ, into living species entities as prompted by the* domain analysis prompt:
  - *is_living_species* – *where* *is_living_species*ℓ *holds if* ℓ *is a living species* ■

3.4.1. Plants

We start with some examples.

Example 11: Example 11: Plants

- *Although we have not yet come across domains for which the need to model the living species of plants were needed, we give some examples anyway:*
  - *grass,*
  - *tulip,*
  - *rhododendron,*
  - *oak tree.*

Analysis Prompt 13 *is_plant:*

- *The domain analyser analyses "things" (ℓ) into a plant.*
- *The method can thus be said to provide the domain analysis prompt:*
  - *is_plant – where* `is_plant`(ℓ) *holds if ℓ is a plant* ■
- The predicate `is_living_species`(ℓ) is a prerequisite for `is_plant`(ℓ).

3.4.2. Animals

Definition 10 Animal: We refer to the initial definition of *living species* above – while ephasizing the following traits:

- (i) *form animals can be developed to reach*;
- (ii) *causally determined to maintain.*
- (iii) *development and maintenance* in an *exchange of matter with an environment*, and
- (iv) *ability to purposeful movement* ■

Analysis Prompt 14 *is_animal:*

- *The domain analyser analyses "things" (ℓ) into an animal.*
- *The method can thus be said to provide the domain analysis prompt:*
  - *is_animal – where is_animal(ℓ) holds if ℓ is an animal* ■

- The predicate `is_living_species(`ℓ`)` is a prerequisite for `is_animal(`ℓ`)`.

Example 12: Example 12: Animals

- *Although we have not yet come across domains for which the need to model the living species of animals, in general, were needed, we give some examples anyway:*
  - *dolphin,*
  - *goose*
  - *cow*
  - *dog,*
  - *lion,*
  - *fly.*

- We have not decided, for these lectures,
  - whether to model animals singly
  - or as sets of such.

3.4.3. Humans

Definition 11 Human:

- A *human* (a *person*) is an *animal*,
  cf. Definition 10, with the additional properties of having
  - *language*,
  - being *conscious* of *having knowledge* (of its own situation), and
  - *responsibility* ■

Analysis Prompt 15 *is_human:*

- *The domain analyser analyses "things" (ℓ) into a human.*
- *The method can thus be said to provide the domain analysis prompt:*
  - *is_human – where is_human(ℓ) holds if ℓ is a human* ■

- The predicate `is_animal(`ℓ`)` is a prerequisite for `is_human(`ℓ`)`.
- We refer to [**?**, Sects. 10.4–10.5]
  - for a specific treatment of living species, animals and humans,
  - and to [**?**] in general
    for the philosophy background for rationalising
    the treatment of living species, animals and humans.

- We have not, in our many experimental domain modelling efforts
  - had occasion to model humans;
  - or rather:
    - we have modelled, for example, automobiles
      - as possessing human qualities,
      - i.e., "subsuming humans".

- We have found, in these experimental domain modelling efforts
  - that we often confer anthropomorphic qualities on artifacts[25],
  - that is, that these artifacts have human characeristics.
- You, the listener are reminded
  - that when some programmers try to explain their programs
  - they do so using such phrases as
  - *and here the program does ...* so-and-so !

[25] Cf. Sect. below.

3.5. Components

Definition 12 Component:

- By a **component** we shall understand
  - a discrete endurant
  - which we, the domain analyser cum describer
  - chooses to **not** endow with **mereology** ■

- Components are discrete endurants.
  - Usually they come in sets.
  - That is, sets of sets of components of different sorts (cf. Sect. on Slide 161).
  - A discrete endurant can (itself) "be" a set of components.
  - But physical parts may contain (`has_components`) components:
    - natural parts may contain natural components,
    - artifacts may contain natural and artifactual components.
  - We leave it to the listener to provide analysis predicates for natural and artifactual "componentry".

Example 13: Example 13: Components

- *A natural part, say a land area may contain*
  - *gravel pits of sand,*
  - *clay pits*
  - *tar pits and*
  - *other "pits".*
- *An artifact, say a postal letter box may contain*
  - *letters,*
  - *small parcels,*
  - *newspapers and*
  - *advertisement brochures.*

Analysis Prompt 16 `has_components:`

- *The domain analyser analyses discrete endurants e into component entities as prompted by the domain analysis prompt:*
  - `has_components`(*p*) *holds if part p potentially may contain components* ■

- We refer to Sect. on Slide 161 for further treatment of the concept of *components*.

3.6. Continuous Endurants ≡ Materials

Definition 13 Material:

- By a **material** we shall understand a continuous endurant ■

- Materials are continuous endurants.
  - Usually they come in sets.
  - That is, sets of of materials of different sorts (cf. Sect. on Slide 166).
  - So an endurant can (itself) "be" a set of materials.
  - But physical parts may contain (`has_materials`) materials:
    - natural parts may contain natural materials,
    - artifacts may contain natural and artifactual materials.
  - We leave it to the listener to provide
    analysis predicates for natural and artifactual "materials".

Example 14: Example 14: Natural and Man-made Materials

- *A* **natural part**, *say a land area, may contain*
  - *lakes,*
  - *rivers,*
  - *irrigation dams and*
  - *border seas.*
- *An* **artifact**, *say an automobile, usually contains*
  - *gasoline,*
  - *lubrication oil,*
  - *engine cooler liquid and*
  - *window screen washer water.*

Analysis Prompt 17 *has_materials:*

- *The domain analysis prompt:*
  - *has_materials(p) yields* **true** *if part p:P potentially may contain materials otherwise false* ■

- We refer to Sect. on Slide 166 for further treatment of the concept of *materials*.
- We shall soon define the terms unique identification, mereology and attributes.

3.7. Artifacts

Definition 14 Artifacts:

- By artifacts we shall understand
  - a man-made physical part or a man-made material ■

Example 15: Example 15: More Artifacts *From the shipping industry:*

- *ship,*
- *container vessels,*
- *container,*
- *container stack,*
- *container terminal port,*
- *harbour.*

Analysis Prompt 18 *is_artifact*:

- *The domain analyser analyses "things" (p) into artifacts.*
- *The method can thus be said to provide the* domain analysis prompt:
  - *is_artifact* – *where* *is_artifact(p)* *holds if p is an artifact* ■

3.8. States

Definition 15 State:

- By a *state* we shall understand any number of
  - physical parts and/or
  - materials
- each possessing
  - as we shall later introduce them
  - at least one dynamic attribute.
- There is no need to introduce time at this point ■

Example 16: Example 16: Artifactual States

- *The following endurants are examples of states (including being elements of state compounds):*
  - *pipe units (pipes, valves, pumps, etc.) of pipe-lines;*
  - *hubs and links of road nets (i.e., street intersections and street segments);*
  - *automobiles (of transport systems).*
- The notion of *state* becomes relevant in Sect. .
- We shall there exemplify states further: example *Constants and States [Indexed States]* Page 286.

4. Endurants: The Description Calculus

4.1. Parts: Natural or Man-made

- The observer functions of this section apply to
  - both natural parts
  - and man-made parts (i.e., artifacts).

4.1.1. On Discovering Endurant Sorts

- Our aim now
  - is to present the basic principles that let
  - the domain analyser decide on *part sort*s.

- We observe parts one-by-one.

- *(α) Our analysis of parts concludes when we have*
  - *"lifted" our examination of a particular part instance*
  - *to the conclusion that it is of a given sort*[26]*,*
  - *that is, reflects a formal concept.*

[26] We use the term 'sort' for abstract types, i.e., for the type of values whose concrete form we are not describing. The term 'sort' is commonly used in algebraic semantics [**?**].

- Thus there is, in this analysis, a "eureka",
  - a step where we shift focus
  - from the concrete to the abstract,
  - from observing specific part instances
  - to postulating a sort: from one to the many.
- If p is a part of sort P, then we express that as: p:P.

Analysis Prompt 19 *observe_endurant_sorts:*

- *The domain analysis prompt:*
  - *observe_endurant_sorts*
- *directs the domain analyser to observe the sub-endurants of an endurant e and to suggest their sorts.*
- *Let* observe_endurant_sorts$(e) = \{e_1{:}E_1, e_2{:}E_2, \ldots, e_m{:}E_m\}$ ■

- *(β) The analyser analyses, for each of these endurants, e_i,*
  - *which formal concept, i.e., sort, it belongs to;*
  - *let us say that it is of sort E_k;*
  - *thus the sub-parts of p are of sorts $\{E_1, E_2, \ldots, E_m\}$.*
- *Some E_k*
  - *may be natural parts,*
  - *other artifacts (man-made parts)*
  - *or structures,*
  - *and yet others may be components*
  - *or materials.*
- *And parts may be either atomic or composite.*

- The domain analyser continues to examine a finite number of other composite parts: $\{p_j, p_\ell, \ldots, p_n\}$.
  - It is then "discovered", that is, decided, that they all consists of the same number of sub-parts
  - $\{e_{i_1}, e_{i_2}, \ldots, e_{i_m}\}$,
  - $\{e_{j_1}, e_{j_2}, \ldots, e_{j_m}\}$,
  - $\{e_{\ell_1}, e_{\ell_2}, \ldots, e_{\ell_m}\}$,
  - ...,
  - $\{e_{n_1}, e_{n_2}, \ldots, e_{n_m}\}$, of the same, respective, endurant sorts.

- *(γ) It is therefore concluded, that is, decided, that $\{e_i, e_j, e_\ell, \ldots, e_n\}$ are all of the same endurant sort P with observable part sub-sorts $\{E_1, E_2, \ldots, E_m\}$.*

- Above we have *type-font-highlighted* three sentences: (α, β, γ).
- When you analyse what they "prescribe" you will see that they entail a "depth-first search" for part sorts.
  - The β sentence says it rather directly:
  - *"The analyser analyses, for each of these parts, p_k, which formal concept, i.e., part sort it belongs to."*
  - To do this analysis in a proper way, the analyser must ("recursively") analyse
    - structures into sub-structures, parts, components and materials, and
    - parts "down" to their atomicity.
    - Components and materials are considered "atomic", i.e., to not contain further analysable endurants.

- For the structures, parts (whether natural or man-made), components and materials
  of the structure the analyser cum describer decides on their sort,
- and work ("recurse") their way "back",
- through possibly intermediate endurants,
- to the p_ks.

- Of course, when the analyser starts by examining atomic parts, components and materials,
  - then their endurant structure and part analysis "recursion" is not necessary.

4.1.2. Endurant Sort Observer Functions:

- The above analysis amounts to the analyser
  - first "applying" the *domain analysis* prompt
  - `is_composite`(e) to a discrete endurant, e,
  - where we now assume that the obtained truth value is **true**.
  - Let us assume that endurants e:E consist of sub-endurants of sorts
    $\{E_1,E_2,\ldots,E_m\}$.
  - Since we cannot automatically guarantee that our domain descriptions secure that
    - E and each E_i ($1 \leq i \leq$ m)
    - denotes disjoint sets of entities

    we must prove it.

Domain Description Prompt 1 observe_endurant_sorts:

- *If is_composite(p) holds, then the analyser "applies" the domain description prompt*
  - *observe_endurant_sorts(p)*

  resulting in the analyser writing down the endurant sorts and endurant sort observers domain description text according to the following schema:

1. `observe_endurant_sorts` **Observer Schema**

Narration:

[s] ... narrative text on sorts ...
[o] ... narrative text on sort observers ...
[p] ... narrative text on proof obligations ...

Formalisation:

type
[s] E,
[s] E_i i:[1..m] **comment:** E_i i:[1..m] abbreviates E_1, E_2, ..., E_m
value
[o] **obs**_E_i: E → E_i i:[1..m]
proof obligation [Disjointness of endurant sorts]
[p] $\mathscr{PO}$: ∀ e:(E_1|E_2|...|E_m) · ⋀ {**is**_E_i(e) ≡ ⋀ {∼**is**_E_j(e)|j:[1..m] \ {i}}|i:[1

- `is_composite` is a **prerequisite prompt** of `observe_endurant_sorts`.
- That is, the composite may satisfy `is_natural` or `is_artifact` ■

Note: The above schema as well as the following schemes introduce, i.e., define in terms of a function signature, a number of functions whose names begin with bold-faced **obs_...**, **uid_...**, **mereo_...**, **attr_...** et cetera. These observer functions are one of the bases of domain descriptions.

Example 17: Example 17: Composite Endurant Sorts

1 There is the universe of discourse, UoD.

It is structured into

2 a road net, RN, and *3 a fleet of vehicles, FV.*

Both are structures.

type

1 UoD **axiom** $\forall$ uod:UoD · is_structure(uod).

2 RN **axiom** $\forall$ rn:RN · is_structure(rn).

3 FV **axiom** $\forall$ fv:FV · is_structure(fv).

value

2 obs_RN: UoD $\rightarrow$ RN

3 obs_FV: UoD $\rightarrow$ FV

■

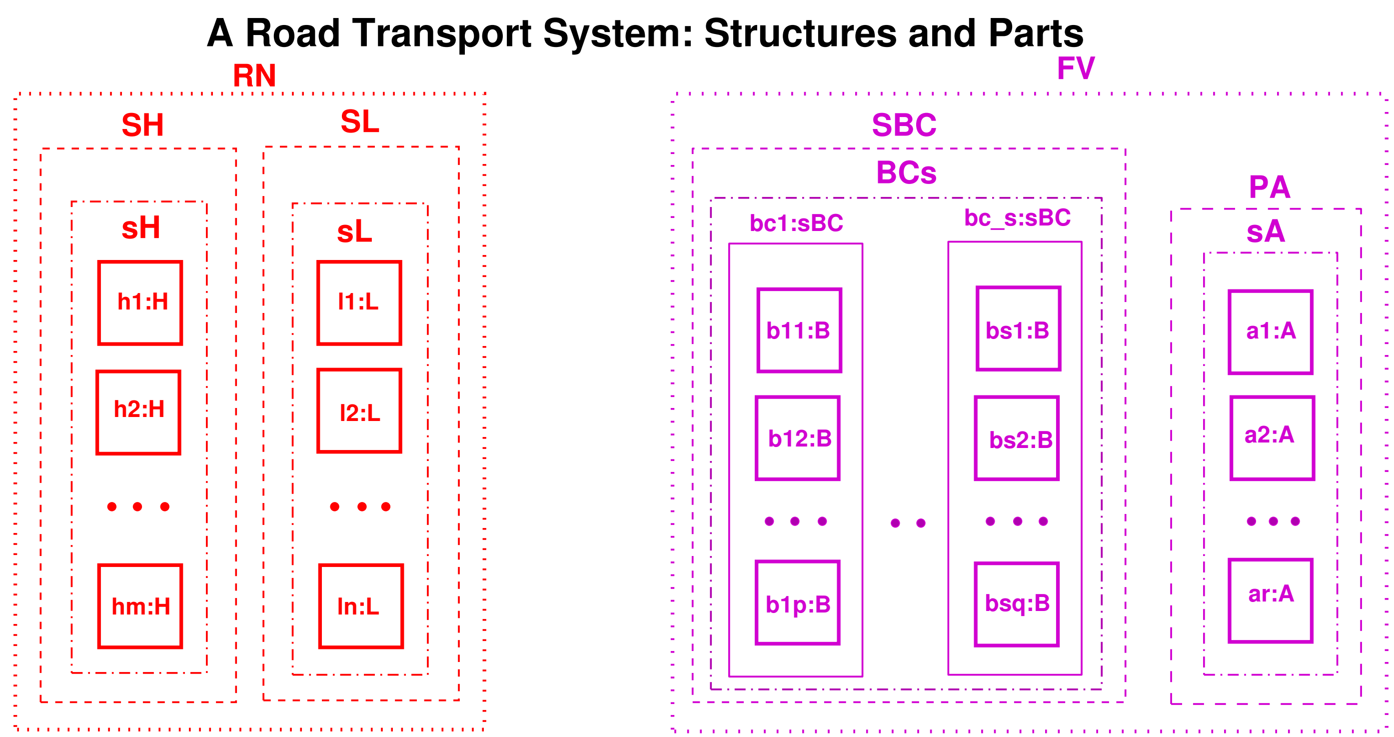


Figure 5: A Road Transport System

Example 18: Example 18: Structures

4 *The road net consists of*

a. *a structure, SH, of hubs and*

b. *a structure, SL, of links.*

5 *The fleet of vehicles consists of*

a. *a structure, SBC, of bus companies, and*

b. *a structure, PA, a pool of automobiles.*

type
4a. SH **axiom** ∀ sh:SH · is_structure(sh)
4b. SL **axiom** ∀ sl:SL · is_structure(sl)
5a. SBC **axiom** ∀ sbc:SBC · is_structure(bc)
5b. PA **axiom** ∀ pa:PA · is_structure(pa)
value
4a. obs_SH: RN → SH
4b. obs_SL: RN → SL
5a. obs_BC: FV → BC
5b. obs_PA: FV → PA

■

4.2. Concrete Part Types

Sometimes it is expedient to ascribe concrete types to sorts.

Analysis Prompt 20 *has_concrete_type:*

- *The domain analyser*
  - *may decide that it is expedient, i.e., pragmatically sound,*
  - *to render a part sort, P, whether atomic or composite, as a concrete type, T.*
  - *That decision is prompted by the holding of the domain analysis prompt:*
    - *has_concrete_type*.
  - *is_discrete is a prerequisite prompt of has_concrete_type* ■

- The reader is reminded that
  - the decision as to whether an abstract type is (also) to be described concretely
  - is entirely at the discretion of the domain engineer.

Domain Description Prompt 2 *observe_part_type*:

- *Then the domain analyser applies the domain description prompt:*
  - *observe_part_type*(*p*)[27]
- *to parts p:P which then yield the part type and part type observers domain description text according to the following schema:*

[27] has_concrete_type is a *prerequisite prompt* of observe_part_type.

2. `observe_part_type` Observer Schema

Narration:

[t_1] ... narrative text on sorts and types S_i ...
[t_2] ... narrative text on types T ...
[o] ... narrative text on type observers ...

Formalisation:

type
[t_1] S_1, S_2, ..., S_m, ..., S_n,
[t_2] $T = \mathscr{E}(S_1,S_2,...,S_n)$
value
[o] **obs**_T: P $\rightarrow$ T ■

- Usually it is wise to restrict the part type definitions, $\mathsf{T}_i = \mathscr{E}_i(\mathsf{Q,R,...,S})$, to simple type expressions.[28]

28

- T=A-**set** or
- T=A* or
- T=ID $\overrightarrow{m}$A or
- T=$\mathsf{A}_t|\mathsf{B}_t|...|\mathsf{C}_t$

where

- ID is a sort of unique identifiers,
- T=$\mathsf{A}_t|\mathsf{B}_t|...|\mathsf{C}_t$ defines the disjoint types
  - A_t==mkA_t(s:A_s),
  - B_t==mkB_t(s:B_s), ...,
  - C_t==mkC_t(s:C_s),

  and where
- A, A_s, B_s, ..., C_s are sorts.
- Instead of A_t==mkA_t(a:A_s), etc., we may write A_t::A_s etc.

- The type name,
  - T, of the concrete type,
  - as well as those of the auxiliary types, S_1,S_2,...,S_m,
  - are chosen by the domain describer:
    - they may have already been chosen
    - for other sort–to–type descriptions,
    - or they may be new.

Example 19: Example 19: Concrete Part Types

6 *The structure of hubs is a set, sH, of atomic hubs, H.*

7 *The structure of links is a set, sL, of atomic links, L.*

8 *The structure of buses is a set, sBC, of composite bus companies, BC.*

9 *The composite bus companies, BC, are sets of buses, sB.*

10 *The structure of private automobiles is a set, sA, of atomic automobiles, A.*

6 H, sH = H-**set** **axiom** ∀ h:H · is_atomic(h)
7 L, sL = L-**set** **axiom** ∀ l:L · is_atomic(l)
8 BC, BCs = BC-**set** **axiom** ∀ bc:BC · is_composite(bc)
9 B, Bs = B-**set** **axiom** ∀ b:B · is_atomic(b)
10 A, sA = A-**set** **axiom** ∀ a:A · is_atomic(a)
value
6 obs_sH: SH → sH
7 obs_sL: SL → sL
8 obs_sBC: SBC → BCs
9 obs_Bs: BCs → Bs
10 obs_sA: SA → sA

■

4.3. On Endurant Sorts

4.3.1. Derivation Chains

- Let E be a composite sort.
- Let E_1, E_2, ..., E_m be the part sorts "discovered" by means of `observe_endurant_sorts(e)` where e:E.
- We say that E_1, E_2, ..., E_m are (immediately) **derived** from E.
- If E_k is derived from E_j and E_j is derived from E_i, then, by transitivity, E_k is **derived** from E_i.

4.3.2. No Recursive Derivations:

- We “mandate” that
  - if E_k is derived from E_j
  - then there
    - E_j is different from E_k and there
    - can be no E_k derived from E_j,
    - that is, E_k cannot be derived from E_k.
- That is, we do not “provide for” recursive domain sorts.
- It is not a question, actually of allowing recursive domain sorts.
  - It is, we claim to have observed,
  - in very many *analysis & description* experiments,
  - that there are no recursive domain sorts ![29]

[29] Some readers may object, but we insist ! If *trees* are brought forward as an example of a recursively definable domain, then we argue: Yes, trees can be recursively defined, but it is not recursive. Trees can, as well, be defined as a variant of graphs, and you wouldn’t claim, would you, that graphs are recursive ?

4.3.3. Names of Part Sorts and Types:

- The domain analysis & description text prompts
  - observe_endurant_sorts,
  - as well as the below-defined
  - observe_part_type,
  - observe_component_sorts and
  - observe_material_sorts,

  – as well as the further below defined

  - attribute_names,
  - observe_material_sorts,
  - observe_unique_identifier,
  - observe_mereology and
  - observe_attributes

  prompts introduced below – "yield" type names.

- That is, it is as if there is
  - a reservoir of an indefinite-size set of such names
  - from which these names are "pulled",
  - and once obtained are never "pulled" again.

- There may be domains for which two distinct part sorts may be composed from identical part sorts.
- *In this case the domain analyser indicates so by prescribing a part sort already introduced.*

4.4. Components

- We refer to Sect. on Slide 121
  for our initial treatment of 'components'.

Domain Description Prompt 3 *observe_component_sorts*:

- *The domain description prompt:*
  - *observe_component_sorts(p)*
  - *yields the component sorts and component sort observer domain description text according to the following schema –*
  - *whether or not the actual part p contains any components:*

3. observe_component_sorts Observer Schema

Narration:

[s] ... narrative text on component sorts ...

[o] ... narrative text on component observers ...

[p] ... narrative text on component sort proof obligations ...

Formalisation:

type

[s] K1, K2, ..., Kn

[s] K = K1| K2 | ... | Kn

[s] KS = K-**set**

value

[o] **obs_components**_P: P → KS

Proof Obligation: [Disjointness of Component Sorts]

[p] $\mathscr{PO}$: $\forall\ k_i$:(K$_1$|K$_2$|...|K$_n$) · $\bigwedge$ **is**_K$_i$(k_i) ≡ $\bigwedge${∼**is**_K$_j$(k_j)|j:[1..n] \ {i}} i:[1..n] ■

- The **is**_K$_j$(e) is defined by Ki, i:[1..n].

Example 20: Example 20: Components

- To illustrate the concept of components
  - we describe timber yards, waste disposal areas, road material storage yards, automobile scrap yards, end the like
  - as special "cul de sac" hubs with components.
  - Here we describe road material storage yards.

11 Hubs may contain components, but only if the hub is connected to exactly one link.

12 These "cul-de-sac" hub components may be such things as Sand, Gravel, Cobble Stones, Asphalt, Cement or other.

value

11 has_components: H → **Bool**

type

12 Sand, Gravel, Stones, Asphalt, Cement, ...

12 KS = (Sand|Gravel|Stones|Asphalt|Cement|...)-**set**

value

11 obs_components_H: H → KS

11 **pre**: obs_components_H(h) ≡ **card** mereo(h) = 1

■

- We have presented one way of tackling the issue of describing components.
  - There are other ways.
  - We leave those ‘other ways’ to the reader.
- We are not going to suggest techniques and tools for analysing, let alone ascribing qualities to components.
  - We suggest that conventional abstract modelling techniques and tools be applied.

4.5. Materials

- We refer to Sect. on Slide 125
  for our initial treatment of 'materials'.
- Continuous endurants (i.e., **materials**) are entities, *m*, which satisfy:
  - is_material(e) $\equiv$ is_continuous(e)
- If is_material(e) holds
  - then we can apply the *domain description prompt*:
  - observe_material_sorts(e).

Domain Description Prompt 4 *observe_material_sorts*:

- *The domain description prompt:*
  - *observe_material_sorts(e)*

  yields the material sorts and material sort observers'
  domain description text
  according to the following schema
  whether or not part p actually contains materials:

4. observe_material_sorts Observer Schema

Narration:

[s] ... narrative text on material sorts ...
[o] ... narrative text on material sort observers ...
[p] ... narrative text on material sort proof obligations ...

Formalisation:

type
[s] M1, M2, ..., Mn
[s] M = M1 | M2 | ... | Mn
[s] MS = M-**set**
value
[o] **obs**_M$_i$: P → M, [i:1..n]
proof obligation [Disjointness of Material Sorts]
[p] $\mathscr{PO}$: ∀ m_i:M · ⋀ {**is**_M$_i$(m_i) ≡ ⋀{∼**is**_M$_j$(m_j)|j ∈ {1..m} \ {i}}|i:[1..n]}

- The **is**_M$_j$(e) is defined by Mi, i:[1..n].

- *Let us assume that parts p:P embody materials of sorts* $\{M_1, M_2, \ldots, M_n\}$.
- *Since we cannot automatically guarantee that our domain descriptions secure that*
  - *each* M_i *(*$[\,1 \leq i \leq n\,]$*)*
  - *denotes disjoint sets of entities*

  we must prove it ■

Example 21: Example 21: Materials

- *To illustrate the concept of materials*
  - *we describe waterways (river, canals, lakes, the open sea) along links*
  - *as links with material of type water.*

13 Links may contain material.

14 That material is water, W.

type
14 W
value
13 obs_material: L → W
13 **pre**: obs_material(l) ≡ has_material(h)

■

5. Endurants: Analysis & Description of Internal Qualities

- We remind the listener that internal qualities cover
  - *unique Identifiers* (Sect.),
  - *mereology* (Sect.) and
  - *attributes* (Sect.).

5.1. Unique Identifiers

- We introduce a notion of unique identification of parts and components.
- We assume
  - (i) that all parts and components, p, of any domain P, have *unique identifier*s,
  - (ii) that *unique identifier*s (of parts and components p:P) are *abstract values* (of the *unique identifier* sort PI of parts p:P),
  - (iii) such that distinct part or component sorts, P_i and P_j, have distinctly named *unique identifier* sorts, say PI_i and PI_j,
  - (iv) that all π_i:PI_i and π_j:PI_j are distinct, and
  - (v) that the observer function **uid**_P applied to p yields the unique identifier, π:PI, of p.

- The description language function **type_name**
  - applies to unique identifiers, p_ui:P_UI, and
  - yield the name of the type, P, of the parts
  - having unique identifiers of type P_UI.

Representation of Unique Identifiers:

- Unique identifiers are abstractions.
  - When we endow two parts (say of the same sort) with distinct unique identifiers
  - then we are simply saying that these two parts are distinct.
  - We are not assuming anything about how these identifiers otherwise come about.

Domain Description Prompt 5 *observe_unique_identifier*:

- *We can therefore apply the domain description prompt:*
  - *observe_unique_identifier*
- *to parts p:P*
  - *resulting in the analyser writing down*
  - *the unique identifier type and observer domain description text according to the following schema:*

5. observe_unique_identifier Observer Schema

Narration:

[s] ... narrative text on unique identifier sort PI ...

[u] ... narrative text on unique identifier observer **uid**_P ...

[a] ... axiom on uniqueness of unique identifiers ...

Formalisation:

type

[s] PI

value

[u] **uid**_P: P $\rightarrow$ PI

axiom [Disjointness of Domain Identifier Types]

[a] $\mathscr{A}$: $\mathscr{U}$(PI,PI_i,PI_j,...,PI_k)

Example 22: Example 22: Unique Identifiers

15 *We assign unique identifiers to all parts.*

16 *By a road identifier we shall mean a link or a hub identifier.*

17 *By a vehicle identifier we shall mean a bus or an automobile identifier.*

18 *Unique identifiers uniquely identify all parts.*

a. *All hubs have distinct [unique] identifiers.*

b. *All links have distinct identifiers.*

c. *All bus companies have distinct identifiers.*

d. *All buses of all bus companies have distinct identifiers.*

e. *All automobiles have distinct identifiers.*

f. *All parts have distinct identifiers.*

type
15 H_UI, L_UI, BC_UI, B_UI, A_UI
16 R_UI = H_UI | L_UI
17 V_UI = B_UI | A_UI
value
18a. uid_H: H → H_UI
18b. uid_L: H → L_UI
18c. uid_BC: H → BC_UI
18d. uid_B: H → B_UI
18e. uid_A: H → A_UI

- *Section on Slide 438 presents some auxiliary functions related to unique identifiers*

■

- We ascribe, in principle, unique identifiers
  - to all parts
    - whether natural
    - or artifactual,

    and
  - to all components.
- We find, from our many experiments, cf. the *Universes of Discourse* example, Page 50,
  - that we really focus on those domain entities which are
    - artifactual endurants and
    - their behavioural "counterparts".

5.2. Mereology

- Mereology is the study and knowledge of parts and part relations.
  - Mereology, as a logical/philosophical discipline, can perhaps best be attributed to the Polish mathematician/logician Stanisław Leśniewski [?, ?].

5.2.1. Part Relations:

- Which are the relations that can be relevant for part-hood ?
- We give some examples.
  - (i) Two otherwise distinct parts may *"share"* values.
    - By *'sharing'* values we shall, as a generic example, mean that two parts of different sorts has the same attributes
    - but that one *'defines'* the attribute, like, for example *'programming'* its values, cf. Defn.8 Page209,
    - whereas the other *'uses'* these values, like, for example considering them *'inert'*, cf. Defn.3 Page207.
  - (ii) Two otherwise distinct parts may be said to, for example, be topologically "adjacent" or one "embedded" within the other.

- These examples are in no way indicative of the “space” of part relations that may be relevant for part-hood.
- The domain analyser is expected to do a bit of experimental research in order to discover necessary, sufficient and pleasing “mereology-hoods” !

5.2.2. Part Mereology: Types and Functions

Analysis Prompt 21 *has_mereology:*

- *To discover necessary, sufficient and pleasing "mereology-hoods" the analyser can be said to endow a truth value,* **true**, *to the domain analysis prompt:*
  - *has_mereology*
- When the domain analyser decides that
  - some parts are related in a specifically enunciated mereology,
  - the analyser has to decide on suitable
    - *mereology types* and
    - *mereology observers* (i.e., part relations).

19 We may, to illustration, define a **mereology type** of a part p:P as a triplet type expression over set of unique [part] identifiers.

20 There is the identification of all those part types $P_{i_1}, P_{i_2}, ..., P_{i_m}$ where at least one of whose properties "`is_of_interest`" to parts p:P.

21 There is the identification of all those part types $P_{io_1}, P_{io_2}, ..., P_{io_n}$ where at least one of whose properties "`is_of_interest`" to parts p:P and vice-versa.

22 There is the identification of all those part types $P_{o_1}, P_{o_2}, ..., P_{o_o}$ for whom properties of p:P "`is_of_interest`" to parts of types $P_{o_1}, P_{o_2}, ..., P_{o_o}$.

23 The the mereology triplet sets of unique identifiers are disjoint and are all unique identifiers of the universe of discourse.

- The three part mereology is just a suggestion.
  - As it is formulated here
    we mean the three ‘sets’ to be disjoint.
  - Other forms of expressing a mereology should be considered
  - for the particular domain and for the particular parts of that domain.
- We leave out further characterisation of
  - the seemingly vague notion `"is_of_interest"`.

type

20 iPI = iPI1 | iPI2 | ... | iPIm

21 ioPI = ioPI1 | ioPI2 | ... | ioPIn

22 oPI = oPI1 | oPI2 | ... | oPIo

19 MT = iPI-**set** × ioPI-**set** × oPI-**set**

axiom

23 ∀ (iset,ioset,oset):MT ·

23 **card** iset + **card** ioset + **card** oset = **card** ∪{iset,ioset,oset}

23 ∪{iset,ioset,oset} ⊆ unique_identifiers(uod)

value

23 unique_identifiers: P → UI-**set**

23 unique_identifiers(p) ≡ ...

Domain Description Prompt 6 observe_mereology:

- *If* has_mereology(*p*) *holds for parts p of type* P,
  - *then the analyser can apply the* **domain description prompt***:*
    - observe_mereology
  - *to parts of that type*
  - *and write down the* mereology types *and* observer *domain description text*
    according to the following schema:

6. `observe_mereology` Observer Schema

Narration:

[t] ... narrative text on mereology type ...
[m] ... narrative text on mereology observer ...
[a] ... narrative text on mereology type constraints ...

Formalisation:

type
[t] MT[30]
value
[m] **mereo**_P: P → MT
axiom [Well−formedness of Domain Mereologies]
[a] $\mathscr{A}$: $\mathscr{A}$(MT)

[30] The mereology descriptor, MT will be referred to in the sequel.

- *$\mathscr{A}(MT)$ is a predicate over possibly all unique identifier types of the domain description.*
- *To write down the concrete type definition for MT requires a bit of analysis and thinking.*
- `has_mereology` *is a* **prerequisite prompt** *for* `observe_mereology` ■

Example 23: Example 23: Mereology

24 *The mereology of hubs is a pair: (i) the set of all bus and automobile identifiers*[31]*, and (ii) the set of unique identifiers of the links that it is connected to and the set of all unique identifiers of all vehicle (buses and private automobiles).*[32]*.*

25 *The mereology of links is a pair: (i) the set of all bus and automobile identifiers, and (ii) the set of the two distinct hubs they are connected to.*

26 *The mereology of of a bus company is a set the unique identifiers of the buses operated by that company.*

27 *The mereology of a bus is a pair: (i) the set of the one single unique identifier of the bus company it is operating for, and (ii) the unique identifiers of all links and hubs*[33]*.*

28 *The mereology of an automobile is the set of the unique identifiers of all links and hubs*[34]*.*

[31]This is just another way of saying that the meaning of hub mereologies involves the unique identifiers of all the vehicles that might pass through the hub `is_of_interest` to it

[32]... its link identifiers designate the links, zero, one or more, that a hub is connected to `is_of_interest` to both the hub and that these links is `interested` in the hub.

[33]that the bus might pass through

[34]that the automobile might pass through

type

24 H_Mer = V_UI-**set**×L_UI-**set**

24 **axiom** ∀ (vuis,luis):H_Mer · luis⊆$l_{ui}s$ ∧ vuis=$v_{ui}s$

25 L_Mer = V_UI-**set**×H_UI-**set**

25 **axiom** ∀ (vuis,huis):L_Mer ·

25 vuis=$v_{ui}s$ ∧ huis⊆$h_{ui}s$ ∧ **card**huis=2

26 BC_Mer = B_UI-**set**

26 **axiom** ∀ buis:H_Mer · buis = $b_{ui}s$

27 B_Mer = BC_UI×R_UI-**set**

27 **axiom** ∀ (bc_ui,ruis):H_Mer·bc_ui∈$bc_{ui}s$∧ruis=$r_{ui}s$

28 A_Mer = R_UI-**set**

28 **axiom** ∀ ruis:A_Mer · ruis=$r_{ui}s$

value

24 mereo_H: H → H_Mer

25 mereo_L: L → L_Mer

26 mereo_BC: BC → BC_Mer

27 mereo_B: B → B_Mer

28 mereo_A: A → A_Mer

- We can express some additional axioms,
- in this case for relations between hubs and links:

29 If hub, h, and link, l, are in the same road net,

30 and if hub h connects to link l then link l connects to hub h.

```
axiom
29  ∀ h:H,l:L • h ∈ hs ∧ l ∈ ls ⇒
    let (_,luis)=mereo_H(h), (_,huis)=mereo_L(l)
30      in uid_L(l)∈luis⇒uid_H(h)∈huis end
```

- More mereology axioms need be expressed –
- but we leave, to the student,
- to narrate and formalise those

■

5.2.3. Formulation of Mereologies:

- The `observe_mereology` domain descriptor, Slide 187,
  - may give the impression that the mereo type MT can be described
  - "at the point of issue" of the `observe_mereology` prompt.
  - Since the MT type expression may depend on any part sort
  - the mereo type MT can, for some domains,
  - "first" be described when all part sorts have been dealt with.

5.2.4. Some Modelling Observations:

- It is, in principle, possible to find examples of mereologies of natural parts:
  - rivers: their confluence, lakes and oceans; and
  - geography: mountain ranges, flat lands, etc.
- But in our experimental case studies, cf. Example on Page 50, we have found no really interesting such cases.
- All our experimental case studies appears to focus on the mereology of artifacts.

- And, finally, in modelling humans,
  - we find that their mereology encompass
    - all other humans
    - and all artifacts !
  - Humans cannot be tamed to refrain from interacting with everyone and everything.
- Some domain models may emphasize *physical mereologies* based on spatial relations,
- others may emphasize *conceptual mereologies* based on logical “connections”.

5.3. Attributes

- To recall: there are three sets of **internal qualities**:
  - unique part identifiers,
  - part mereology and
  - attributes.
- Unique part identifiers and part mereology
  are rather definite kinds of internal endurant qualities.
- Part attributes form more "free-wheeling" sets of **internal qualities**.

5.3.1. Technical Issues:

- We divide Sect. into two subsections:
  - *technical issues*, the present one, and
  - *modelling issues*, Sect. .

5.3.1.1 Inseparability of Attributes from Parts and Materials:

- Parts and materials are
  - typically recognised because of their spatial form
  - and are otherwise characterised by their intangible, but measurable attributes.
- We equate all endurants which, besides possible type of unique identifiers (i.e., excepting materials) and possible type of mereologies (i.e.,, excepting components and materials), have the same types of attributes, with one sort.
- Thus removing a quality from an endurant makes no sense:
  - the endurant of that type
  - either becomes an endurant of another type
  - or ceases to exist (i.e., becomes a non-entity) !

Attribute Quality and Attribute Value:

- We distinguish between
  - an attribute (as a logical proposition, of a name, i.e.) type, and
  - an attribute value, as a value in some value space.

Analysis Prompt 22 *attribute types:*

- *One can calculate the set of attribute types of parts and materials with the following domain analysis prompt:*
  - *attribute_types*
- *Thus for a part p we may have* attribute_types$(p) = \{A_1, A_2, ..., A_m\}$.

- Whether by `attribute_types`(p)
  - we mean the names of the types $\{A_1, A_2, ..., A_m\}$
    - for example $\{\eta A_1, \eta A_2, ..., \eta A_m\}$
    - where η is some meta-function which applies to a type and yields its name, or
  - or we mean the [full] types themselves,
    - i.e., some possibly infinite, suitably structured set
    - of values (of that type),
  - we shall here leave open !

5.3.1.2 Attribute Types and Functions:

- Let us recall that attributes cover qualities other than unique identifiers and mereology.
- Let us then consider that parts and materials have one or more attributes.
  - These attributes are qualities
  - which help characterise "what it means" to be a part or a material.
- Note that we expect every part and material to have at least one attribute.
- The question is now, in general, how many and, particularly, which.

Domain Description Prompt 7 `observe_attributes`:

- *The domain analyser experiments, thinks and reflects about part attributes.*
- *That process is initiated by the* domain description prompt*:*
  - `observe_attributes`.
- *The result of that* domain description prompt *is that the domain analyser cum describer writes down the* attribute (sorts or) types and observers *domain description text according to the following schema:*

7. observe_attributes Observer Schema

Narration:

[t] ... narrative text on attribute sorts ...

[o] ... narrative text on attribute sort observers ...

[p] ... narrative text on attribute sort proof obligations ...

Formalisation:

type

[t] A_i $[1 \leq i \leq n]$

value

[o] **attr**_A_i: P→A_i i:[1..n]

proof obligation [Disjointness of Attribute Types]

[p] $\mathscr{PO}$: **let** P be any part sort **in** [the domain description]

[p] **let** a:(A_1|A_2|...|A_n) **in** **is**_A_i(a) ≠ **is**_A_j(a) **end end** [i≠i, i,j:[1..n]]

- The **is**_A_j(e) is defined by Ai, i:[1..n].

- The **type** (or rather sort) definitions: A_1, A_2, ..., A_n, inform us that the domain analyser has decided to focus on the distinctly named A_1, A_2, ..., A_n attributes.
- And the **value** clauses
  - **attr**_A_1:P→A_1,
  - **attr**_A_2:P→A_2,
  - ...,
  - **attr**_A_n:P→A_n

  are then "automatically" given:
  - if a part, p:P, has an attribute A_i
  - then there is postulated, "by definition" [eureka] an attribute observer function **attr**_A_i:P→A_i etcetera ■

- We cannot automatically, that is, syntactically, guarantee that our domain descriptions secure that
  - the various attribute types
  - for a part sort
  - denote disjoint sets of values.

  Therefore we must prove it.

Attribute Categories:

- Michael A. Jackson [?] has suggested a hierarchy of attribute categories:
  - static or
  - dynamic values – and within the dynamic value category:
    - inert values or
    - reactive values or
    - active values – and within the dynamic active value category:
      - autonomous values or
      - biddable values or
      - programmable values.
- We now review these attribute value types.
  The review is based on [?, M.A. Jackson].

- *Part attributes* are either constant or varying, i.e., **static** or **dynamic** attributes.

Attribute Category: 1 • By a **static attribute**, a:A, `is_static_attribute`(a), we shall understand an attribute whose values

- are constants,
- i.e., cannot change.

Attribute Category: 2 • By a **dynamic attribute**, a:A, `is_dynamic_attribute`(a), we shall understand an attribute whose values

- are variable,
- i.e., can change.

Dynamic attributes are either *inert*, *reactive* or *active* attributes.

Attribute Category: 3 • By an **inert attribute**, a:A,
is_inert_attribute(a),
we shall understand a dynamic attribute whose values

- only change as the result of external stimuli where
- these stimuli prescribe new values.

Attribute Category: 4 • By a **reactive attribute**, a:A,
is_reactive_attribute(a),
we shall understand a dynamic attribute whose values,

- if they vary, change in response to external stimuli,
- where these stimuli come from outside the domain of interest.

Attribute Category: 5 • By an **active attribute**, a:A, is_active_attribute(a), we shall understand a dynamic attribute whose values

- change (also) of its own volition.

Active attributes are either *autonomous, biddable* or *programmable* attributes.

Attribute Category: 6 • By an **autonomous attribute**, a:A, is_autonomous_attribute(a), we shall understand a dynamic active attribute

- whose values change only "on their own volition".[35]

[35] The values of an autonomous attributes are a "law onto themselves and their surroundings".

Attribute Category: 7 • By a **biddable attribute**, a:A, `is_biddable_attribute`(a) we shall understand a dynamic active attribute whose values

– *are prescribed*

– *but may fail to be observed as such.*

Attribute Category: 8 • By a **programmable attribute**, a:A, `is_programmable_attribute`(a), we shall understand a dynamic active attribute whose values

– can be prescribed.

- Figure 6 captures an attribute value ontology.

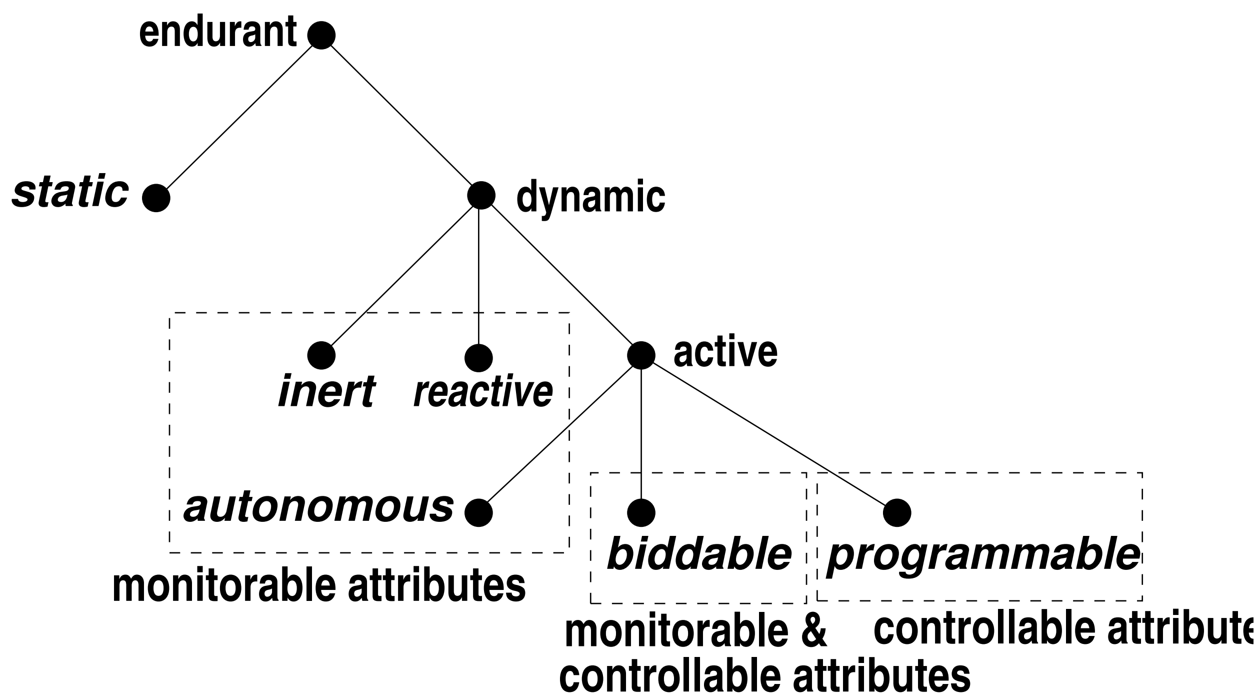


Figure 6: Attribute Value Ontology

Example 24: Example 24: Attributes

- *We treat part attributes, sort by sort.*

Hubs: *We show just a few attributes:*

31 *There is a hub state. It is a set of pairs, (l_f,l_t) of link identifiers, where these link identifiers are in the mereology of the hub. The meaning of the hub state, in which, e.g., (l_f,l_t) is an element, is that the hub is open,* **"green"** *, for traffic $f$rom link l_f to link l_t. If a hub state is empty then the hub is closed, i.e.,* **"red"** *for traffic from any connected links to any other connected links.*

32 *There is a hub state space. It is a set of hub states. The meaning of the hub state space is that its states are all those the hub can attain. The current hub state must be in its state space.*

33 Since we can think rationally about it, it can be described, hence it can model, as an attribute of hubs a history of its traffic: the recording, per unique bus and automobile identifier, of the time ordered presence in the hub of these vehicles.

34 The link identifiers of hub states must be in the set, $l_{ui}s$, of the road net's link identifiers.

type
31 HΣ = (L_UI×L_UI)-**set**
axiom
31 ∀ h:H · obs_HΣ(h) ∈ obs_HΩ(h)
type
32 HΩ = HΣ-**set**
33 H_Traffic
33 H_Traffic = (A_UI|B_UI) $\overrightarrow{m}$ ($\mathscr{T}$ × VPos)*
axiom
33 ∀ ht:H_Traffic,ui:(A_UI|B_UI)·ui ∈ **dom** ht
33 ⇒ time_ordered(ht(ui))
value
31 attr_HΣ: H → HΣ
32 attr_HΩ: H → HΩ
33 attr_H_Traffic: : → H_Traffic
axiom
34 ∀ h:H · h ∈ hs ⇒
34 **let** hσ = attr_HΣ(h) **in**
34 ∀ (l$_{ui}i$,li$_{ui}i'$):(L_UI×L_UI) · (l$_{ui}i$,l$_{ui}i'$) ∈ hσ
34 ⇒ {l$_{ui_i}$,l$'_{ui_i}$} ⊆ l$_{ui}s$ **end**
value
33 time_ordered: $\mathscr{T}^*$ → **Bool**
33 time_ordered(tvpl) ≡ ...

■

Calculating Attributes:

35 Given a part *p* we can *meta-linguistically*[36] calculate names for its static attributes.

36 Given a part *p* we can *meta-linguistically* calculate name for its monitorable attributes attributes.

37 Given a part *p* we can *meta-linguistically* calculate name for its monitorable and controllable attributes.

38 Given a part *p* we can *meta-linguistically* calculate names for its controllable attributes.

39 These three sets make up all the attributes of part *p*.

The type names nSA, nMA nMCA, nCA designate sets of names.

[36]By using the term *meta-linguistically* here we shall indicate that we go outside what is computable – and thus appeal to the reader's forbearance.

value

35 stat_attr_typs: P → nSA-**set**

36 mon_attr_typs: P → nMA-**set**

37 mon_ctrl_attr_typs: P → nMCA-**set**

38 ctrl_attr_typs: P → nCA-**set**

axiom

39 ∀ p:P ·

35 **let** stat_nms = stat_attr_typs(p),

36 mon_nms = mon_attr_typs(p),

37 mon_ctrl_nms = mon_ctrl_attr_typs(p),

38 ctrl_nms = mon_ctrl_typs(p) **in**

39 **card** stat_nms + **card** mon_nms + **card** mon_ctrl_nms + **card** ctrl_nms

39 = **card**(stat_nms ∪ mon_nms ∪ mon_ctrl_nms ∪ ctrl_nms) **end**

The above formulas are indicative, like mathematical formulas, they are not computable.

40 Given a part *p* we can *meta-linguistically* calculate its static attribute values.

41 Given a part *p* we can *meta-linguistically* calculate its controllable, i.e., programmable attribute values.

Et cetera for monitorable and monitorable & controllable attribute values.
The type names sa1, ..., cac refer to the types denoted by the corresponding types name nsa1, ..., ncac.

value

40 stat_attr_vals: P → SA1×SA2×...×SAs
40 stat_attr_vals(p) ≡ **let** {nsa1,nsa2,...,nsas}
40 = stat_attr_typs(p) **in** (attr_sa1(p),attr_sa2(p),...,attr_sas(p)) **end**

41 ctrl_attr_vals: P → CA1×CA2×...×CAc
41 ctrl_attr_vals(p) ≡ **let** {nca1,nca2,...,ncac}
41 = ctrl_attr_typs(p) **in** (attr_ca1(p),attr_ca2(p),...,attr_cac(p)) **end**

- The “ordering” of type values,
  - (attr_sa1(p),...,attr_sas(p)), respectively
  - (attr_ca1(p),...,attr_cac(p)),
  - is arbitrary.

5.3.2. Basic Principles for Ascribing Attributes:

- Section dealt with technical issues of expressing attributes.
- This section will indicate some modelling principles.

Natural Parts:

- are subject to laws of physics.
- So basic attributes focus on physical (including chemical) properties.
- These attributes cover the full spectrum of attribute categories outlined in Sect. .

Materials:

- are subject to laws of physics.
- So basic attributes focus on physical, especially chemical properties.
- These attributes cover the full spectrum of attribute categories outlined in Sect. .
- The next paragraphs, **living species**, **animate entities** and **humans**, reflect Sørlander's Philosophy [**?**, pp 14–182].

• • •

Causality of Purpose:

- If there is to be *the possibility of language and meaning*
  - then there must exist primary entities
  - which are *not entirely encapsulated within the physical conditions*;
  - that they are stable and
  - can influence one another.
- This is only possible if such primary entities are
  - subject to a *supplementary causality*
  - *directed at the future*:
  - a *causality of purpose*.

Living Species:

- These primary entities are here called *living species*.
- What can be deduced about them ? They are

- characterised by *causality of purpose*:
- they *have some form they can be developed to reach;*
- and which *they must be causally determined to maintain;*
- this development and maintenance must further in *an exchange of matter with an environment*.
- It must be possible that living species occur in one of two forms:
  - one form which is characterised by *development*, *form* and *exchange*,
  - and another form which, additionally, can be characterised by the ability to *purposeful movement*s.
- The first we call *plant*s, the second we call *animal*s.

Animate Entities:

- For an animal to purposefully move around
  - there must be "additional conditions" for such self-movements to be in accordance with the principle of causality:
    - they must have *sensory organ*s sensing among others the immediate purpose of its movement;
    - they must have *means of motion* so that it can move; and
    - they must have *instinct*s, *incentive*s and *feeling*s as causal conditions that what it senses can drive it to movements.
  - And all of this in accordance with the laws of physics.

Animals: To possess these three kinds of "additional conditions",

- must be built from special units which have an inner relation to their function as a whole;
- Their *purposefulness* must be built into their physical building units,
- that is, as we can now say, their *genome*s.
- That is, animals are built from genomes which give them the *inner determination* to such building blocks for *instinct*s, *incentive*s and *feeling*s.
- Similar kinds of deduction can be carried out with respect to plants.
- Transcendentally one can deduce basic principles of evolution but not its details.

Humans: Consciousness and Learning:

- The existence of animals is a necessary condition for there being language and meaning in any world.
  - That there can be *language* means that animals are capable of *developing language*.
  - And this must presuppose that animals can *learn from their experience.*
  - To learn implies that animals
    - can *feel* pleasure and distaste
    - and can *learn*.
  - One can therefore deduce that animals must possess such building blocks whose inner determination is a basis for learning and consciousness.

Language:

- Animals with higher social interaction
  - uses *signs*, eventually developing a *language*.
  - These languages adhere to the same system of defined concepts
  - which are a prerequisite for any description of any world:
    - namely the system that philosophy lays bare from a basis
    - of transcendental deductions and
    - the *principle of contradiction* and
    - its *implicit meaning theory*.
- A *human* is an animal which has a *language*.

Knowledge:

- Humans must be *conscious*
  - of having *knowledge* of its concrete situation,
  - and as such that human can have knowledge about what he feels
  - and eventually that human can know whether what he feels is true or false.
  - Consequently *a human can describe his situation correctly*.

Responsibility:

- In this way one can deduce that humans
  - can thus have *memory*
  - and hence can have *responsibility*,
  - be *responsible*.
  - Further deductions lead us into *ethics*.
- We shall not develop the theme of
  - *living species: plants and animals*,
  - thus excluding, most notably *humans*,
  - much further in this paper.

- We claim that the present paper,
  - due to its foundation in Kai Sørlander's Philosophy,
  - provides a firm foundation
  - withing which we, or others, can further develop
  - this theme: *analysis & description of living species.*

Intentionality:

- *Intentionality is*
  - *a philosophical concept*
  - *and is defined by the*
    Stanford Encyclopedia of Philosophy[37] *as*
    - *"the power of minds to be about, to represent, or to stand for,*
    - *things, properties and states of affairs."*

[37] Jacob, P. (Aug 31, 2010). *Intentionality*. Stanford Encyclopedia of Philosophy (https://seop.illc.uva.nl/entries/intentionality/) October 15, 2014, retrieved April 3, 2018.

Definition 16 Intentional Pull:

- Two or more artifactual parts
  - of different sorts, but with overlapping sets of intents
  - 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 intent*, ι.
  - *Manifestations* of these, their common intent
  - must somehow be *subject to constraints*,
  - and these must be *expressed predicatively*.

Example 25: Intentional Pull

- Vehicles are meant to drive on roads: hubs and links.
- Hubs and links are meant to accept vehicles driving on them.
- These two facts are recorded separately in the
  - histories of automobiles and buses, and in the
  - histories of roads: hubs and links.
- These two histories must be commensurate.
- That is the in-avoidable intentional "pull" !

The paper version of this example is very detailed on this.
Also formally so !

Artifacts:

- Humans create artifacts –
  for a reason, to serve a purpose, that is, with **intent**.
  - Artifacts are like parts.
  - They satisfy the laws of physics –
  - and serve a *purpose*, fulfill an *intent*.

Assignment of Attributes:

- So what can we deduce from the above, a little more than two pages ?
- The attributes of **natural parts** and **natural materials**
  - are generally of such concrete types –
  - expressible as some **real** with a dimension[38] of
  - the International System of Units:
  - `https://physics.nist.gov/cuu/Units/units.html`.
- Attribute values usually enter *differential equations* and *integrals*,
- that is, classical calculus.

[38]Basic units are *m*eter, *k*ilogram, *s*econd, *A*mpere, *K*elvin, *mol*e, and *cand*ela. Some derived units are: *N*ewton: $kg \times m \times s^{-2}$, *W*eber: $kg \times m^2 \times s^{-2} \times A^{-1}$, etc.

- The attributes of **humans**, besides those of parts,
  - significantly includes one of a usually non-empty set of *intents*.
    - In directing the creation of artifacts
    - humans create these with an intent.

Example 26: Intentional Pull

- *These are examples of human intents:*
  - *they create roads and automobiles with the intent of transport,*
  - *they create houses with the intents of living, offices, production, etc., and*
  - *they create pipelines with the intent of oil or gas transport* ■

- Human attribute values usually enter into *modal logic* expressions.

Artifacts, including Man-made Materials:

- Artifacts, besides those of parts,
  - significantly includes a usually singleton set of *intents*.

Example 27: Intents

- *Roads and automobiles possess the intent of transport;*
- *houses possess either one of the intents of living, offices, production; and*
- *pipelines possess the intent of oil or gas transport.*

- Artifact attribute values usually enter into *mathematical logic* expressions.
- We leave it to the listener to formulate attribute assignment principles for plants and non-human animals.

5.4. Some Axioms and Proof Obligations

- To remind you, an **axiom** – in the *context* of domain analysis & description –
  means
  - a logical expression, usually a predicate,
  - that constrains the types and values, including
  - unique identifiers and mereologies
  - of domain models.
- Axioms,
  - together with the sort, including type definitions, and the
  - unique identifier, mereology and attribute observer functions,
  - define the domain value spaces.

- We refer to axioms in Item [a] of domain description prompts of
  - *unique identifiers:* 5 on Slide 175 and of
  - *mereologies:* 6 on Slide 187.

- Another reminder: a **proof obligation** – in the *context* of domain analysis & description –
  means
  - a logical expression
  - that predicates relations between
  - the types and values, including
  - unique identifiers, mereologies and attributes
  - of domain models,
  - where these predicates must be shown, i.e., proved, to hold.

- Proof obligations supplement axioms.
- We refer to proof obligations in
  - Item [p] of domain description prompts about
  - *endurant sorts:* 1 on Slide 144, about
  - *components sorts:* 3 on Slide 162, about
  - *materials sorts:* 4 on Slide 168, and about
  - *attribute types:* 7 on Slide 202.
- The difference between expressing axioms and expressing proof obligations is this:

- **We use axioms**
  - when our formula cannot otherwise express it simply,
  - but when physical or other *properties of the domain*[39]
  - dictates property consistency.
- **We use proof obligations**
  - where necssary constraints
  - are not necessarily physically impossible.
- **Proof obligations** finally arise
  - in the transition from endurants to perdurants
  - where endurant axioms
  - become properties that must be proved to hold.

[39]– examples of such properties are: (i) topologies of the domain makes certain compositions of parts physically impossible, and (ii) conservation laws of the domain usually dictates that endurants cannot suddenly arise out of nothing.

- When considering *endurants* we interpret these as stable, i.e.,
  - that although they may have, for example, programmable attributes,
  - when we observe them, we observe them at any one moment,
  - but *we do not consider them over a time*.
  - That is what we turn to next: *perdurants.*

- When considering a part with, for example, a programmable attribute, at two different instances of time
  - we expect the particular programmable attribute
  - to enjoy any expressed well-formedness properties.
- We shall, as from Slide 282,
  - see how these programmable attributes
  - re-occur as explicit behaviour parameters,
  - "programmed" to possibly new values
  - passed on to recursive invocations of the same behaviour.

- If well-formedness axioms were expressed
  - for the part on which the behaviour is based,
  - then a *proof obligation* arises,
  - one that must show that new values of the programmed attribute
  - satisfies the part attribute axiom.
- This is, but one relation between *axioms* and *proof obligations*.
- We refer to remarks made in the bullet (•) named **Biddable Access** Slide 337.

5.5. Discussion of Endurants

- Domain descriptions are, as we have already shown, formulated,
  - both informally
  - and formally,

  by means of abstract types,
  - that is, by sorts
  - for which no concrete models are usually given.
- Sorts are made to denote
  - possibly empty,
  - possibly infinite,
  - rarely singleton,
  - sets of entities on the basis of the qualities defined for these sorts, whether external or internal.

- By **junk** we shall understand
  - that the domain description
  - unintentionally denotes undesired entities.
- By **confusion** we shall understand
  - that the domain description
  - unintentionally have two or more identifications
  - of the same entity or type.
- The question is
  - *can we formulate a [formal] domain description*
  - *such that it does not denote junk or confusion* ?
- The short answer to this is no !

- So, since one naturally wishes "no junk, no confusion" what does one do ?
- The answer to that is
  - *one proceeds with great care !*

6. A Transcendental Deduction

6.1. An Explanation

- It should be clear to the reader that in domain analysis & description
  - we are reflecting on a number of philosophical issues.
  - First and foremost on those of *epistemology*, especially *ontology*.
  - In this section on a sub-field of epistemology, namely that of a number of issues of *transcendental* nature, we refer to
    [**?**, Oxford Companion to Philosophy, pp 878–880]
    [**?**, The Cambridge Dictionary of Philosophy, pp 807–810]
    [**?**, The Blackwell Dictionary of Philosophy, pp 54–55 (1998)].

Definition 17 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.

Definition 18 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: Example 27: Some Transcendental Deductions

- *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 – that tool is one of several forms of transcendental deductions;*

◦ *The software tools, an automatic theorem prover*[40] *and a model checker, for example* `SPIN` *[**?**], 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.*

[40] `ACL2` [**?**], `Coq` [**?**], `Isabelle/HOL` [**?**], `STeP` [**?**], `PVS` [**?**] and `Z3` [**?**]

- *Yes, indeed, any abstract interpretation [?, ?] reflects a transcendental deduction:*
  - *First these examples show that there are many transcendental deductions.*
  - *Secondly they show that there is no single-most preferred transcendental deduction.*

- A transcendental deduction, crudely speaking,
  - is just any abstraction
  - that can be “linked” to another,
  - not by logical necessity,
  - but by logical (and philosophical) possibility !

Definition 19 Transcendentality:
By **transcendentality** we shall here mean the philosophical notion:
the state or condition of being transcendental ■

Example 28: Example 28: Transcendentality

- *We can speak of a bus in at least three senses:*

  (i) The bus as it is being `"maintained, serviced, refueled"`*;*
  (ii) the bus as it `"speeds"` *down its route; and*
  (iii) the bus as it `"appears"` *(listed) in a bus time table.*

- *The three senses are:*

  (i) as an **endurant** *(here a part),*
  (ii) as a **perdurant** *(as we shall see a behaviour), and*
  (iii) as an **attribute**[41]

■

[41]– in this case rather: as a fragment of a bus time table *attribute*

- The above example, we claim, reflects *transcendentality* as follows:

(i) We have knowledge of an endurant (i.e., a part) being an endurant.

(ii) We are then to assume that the perdurant referred to in (ii) is an aspect of the endurant mentioned in (i) – where perdurants are to be assumed to represent a different kind of knowledge.

(iii) And, finally, we are to further assume that the attribute mentioned in (iii) is somehow related to both (i) and (ii) – where at least this attribute is to be assumed to represent yet a different kind of knowledge.

- In other words:
  - two (i–ii) kinds of different knowledge;
  - that they relate *must indeed* be based on *a priori knowledge*.
  - Someone claims that they relate !
- The two statements (i–ii) are claimed to relate transcendentally.[42]

[42]– the attribute statement was "thrown" in "for good measure", i.e., to highlight the issue !

6.2. Classical Transcendental Deductions

- We present a few of the transcendental deductions of [?, Kai Sørlander: *Introduction to The Philosophy*, 2016]

6.2.1. Space:

- [?, pp 154] *The two relations asymmetric and symmetric, by a transcendental deduction, can be given an interpretation:*
  - The relation (spatial) *direction* is asymmetric; and
  - the relation (spatial) *distance* is symmetric.
  - Direction and distance are spatial relations.
  - From these relations are derived the relation *in-between*.
- Hence we must conclude that *primary entities exist in space.*
- *Space* is therefore an unavoidable characteristic of any possible world

6.2.2. Time:

- [?, pp 159] *Two different states must necessarily be ascribed different incompatible predicates.*
  - *But how can we ensure so ?*
  - *Only if states stand in an asymmetric relation to one another.*
  - *This state relation is also transitive.*
  - *So that is an indispensable property of any world.*
  - *By a transcendental deduction we say that primary entities exist in time.*
- *So every possible world must exist in time*

6.3. Some Special Notation

- The *transcendentality* that we are referring to is one in which we **"translate"** endurant descriptions of
  - *parts* and their
  - *unique identifiers, mereologies* and *attributes*
- into descriptions of perdurants, i.e., transcendental interpretations of parts
  - as *behaviours*,
  - part mereologies as *channels*, and
  - part attributes as *attribute value accesses*.
- The *translations* referred to above,
  - *compile* endurant descriptions into $\mathtt{RSL}^{+}$Text.
- We shall therefore first explain some aspects of this translation.

- Where in the function definition bodies
  - we enclose some RSL$^+$Text,
  - e.g., rsl$^+$_text, in ⟪⟫s,
  - i.e., ⟪rsl$^+$_text ⟫
  - we mean that text.
- Where in the function definition bodies
  - we write ⟪rsl$^+$_text ⟫ function_expression
  - we mean that rsl$^+$_text
  - concatenated to the RSL$^+$Text emanating from function_expression.

- Where in the function definition bodies
  - we write $\not\lll\!\not\ggg$ function_expression
  - we mean just rsl$^+$_text emanating from function_expression.
  - That is: $\not\lll\!\not\ggg$ function_expression $\equiv$ function_expression and $\not\lll\!\not\ggg$ $\not\lll\!\not\ggg \equiv \not\lll\!\not\ggg$.
- Where in the function definition bodies
  - we write $\{ \not\lll\ f(x)\ \not\ggg |$ x:RSL$^+$Text$\}$
  - we mean the "expansion" of the RSL$^+$Text $f(x)$, in arbitrary, linear text order, for appropriate RSL$^+$Texts x.

7. Space and Time

- This section is a necessary prelude to our treatment of perdurants.
- Following Kai Sørlander's Philosophy
  we must accept that space and time
  are rationally potentially mandated in any domain description.
  - It is, however not always necessary to model space and time.
  - We can talk about space and time;
  - **and** when we do, we must model them.

7.1. Space

7.1.1. General:

- Mathematicians and physicists model space
  - in, for example, the form of Hausdorf (or topological) space[43];
  - or a metric space
    - which is a set for which distances between all members of the set are defined;
    - Those distances, taken together, are called a metric on the set;
    - a metric on a space induces topological properties like open and closed sets, which lead to the study of more abstract topological spaces;
  - or Euclidean space, due to *Euclid of Alexandria*.

[43] Armstrong, M. A. (1983) [1979]. Basic Topology. Undergraduate Texts in Mathematics. Springer. ISBN 0-387-90839-0.

7.1.2. Space Motivated Philosophically

Characterisation 9 **Indefinite Space:**

- We motivate the concept of indefinite space as follows:
- [?, pp 154] *The two relations asymmetric and symmetric, by a transcendental deduction, can be given an interpretation:*
  - The relation (spatial) *direction* is asymmetric; and
  - the relation (spatial) *distance* is symmetric.
  - Direction and distance are spatial relations.
  - From these relations are derived the relation *in-between*.
- Hence we must conclude that *primary entities exist in space.*
- *Space* is therefore an unavoidable characteristic of any possible world ■

***Characterisation 10* Definite Space:**

- By a **definite space** we shall understand
  - a space with a definite metric ■
- *There is but just one space.*
  - *It is all around us, from the inner earth to the farthest galaxy.*
  - *It is not manifest.*
  - *We can not observe it as we observe a road or a human.*

7.1.3. Space Types

The Spatial Value:

42 There is an abstract notion of (definite) $\mathbb{SPACE}$(s) of further unanalysable points; and

43 there is a notion of $\mathbb{POINT}$ in $\mathbb{SPACE}$.

type
42 $\mathbb{SPACE}$
43 $\mathbb{POINT}$

- Space is not an attribute of endurants.
- Space is just there.
- So we do not define an observer, `observe_space`.

- For us, bound to model mostly artifactual worlds on this earth there is but one space.
  - Although $\mathbb{SPACE}$, as a type, could be thought of as defining more than one space we shall consider these isomorphic !

7.1.4. Spatial Observers

44 A point observer, observe_$\mathbb{POINT}$, is a function

- which applies to physical endurants, e,
- and yield a point, $\ell : \mathbb{POINT}$.

value

44 observe_$\mathbb{POINT}$: E $\rightarrow$ $\mathbb{POINT}$

7.2. Time

- Concepts of time[44] continue to fascinate thinkers [?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?].

[44] **Time:** *(i) a moving image of eternity; (ii) the number of the movement in respect of the before and the after; (iii) the life of the soul in movement as it passes from one stage of act or experience to another; (iv) a present of things past: memory, a present of things present: sight, and a present of things future: expectations.[?, (i) Plato, (ii) Aristotle, (iii) Plotinus, (iv) Augustine].*

7.2.1. Time Motivated Philosophically

Characterisation 11 Indefinite Time:

- *We motivate the abstract notion of time as follows.*
  - [?, pp 159] *Two different states must necessarily be ascribed different incompatible predicates.*
    - *But how can we ensure so ?*
    - *Only if states stand in an asymmetric relation to one another.*
    - *This state relation is also transitive.*
    - *So that is an indispensable property of any world.*
    - *By a transcendental deduction we say that primary entities exist in time.*
  - *So every possible world must exist in time* ■

Characterisation 12 **Definite Time:**

- By a **definite time** we shall understand
  - an abstract representation of time
  - such as for example year, month, day, hour, minute, second, et cetera ■

Example 29: Example 29: Temporal Notions of Endurants

- *By temporal notions of endurants we mean*
  - *time properties of endurants,*
  - *usually modelled as attributes.*
- *Examples are:*
  - *(i) the time stamped link traffic, cf. Item 111 on Slide 443 and*
  - *(ii) the time stamped hub traffic, cf. Item 33 on Slide 212.*

7.2.2. Time Values

- We shall not be concerned with any representation of time.
- That is, we leave it to the domain analyser cum describer to choose an own representation [**?**].
- Similarly we shall not be concerned with any representation of time intervals.[45]

45 So there is an abstract type $\mathbb{T}ime$,

46 and an abstract type $\mathbb{TI}$: $\mathbb{T}ime\mathbb{I}nterval$.

47 There is no $\mathbb{T}ime$ origin, but there is a “zero” $\mathbb{TI}$me interval.

48 One can add (subtract) a time interval to (from) a time and obtain a time.

[45]– but point out, that although a definite time interval may be referred to by number of years, number of days (less than 365), number of hours (less than 24), number of minutes (less than 60)number of seconds (less than 60), et cetera, this is not a time, but a time interval.

49 One can add and subtract two time intervals and obtain a time interval – with subtraction respecting that the subtrahend is smaller than or equal to the minuend.

50 One can subtract a time from another time obtaining a time interval respecting that the subtrahend is smaller than or equal to the minuend.

51 One can multiply a time interval with a real and obtain a time interval.

52 One can compare two times and two time intervals.

type

45 $\mathbb{T}$

46 $\mathbb{TI}$

value

47 $\mathbf{0}$:$\mathbb{TI}$

48 $+,-$: $\mathbb{T} \times \mathbb{TI} \to \mathbb{T}$

49 $+,-$: $\mathbb{TI} \times \mathbb{TI} \stackrel{\sim}{\to} \mathbb{TI}$

50 $-$: $\mathbb{T} \times \mathbb{T} \to \mathbb{TI}$

51 $*$: $\mathbb{TI} \times$ **Real** $\to \mathbb{TI}$

52 $<,\leq,=,\neq,\geq,>$: $\mathbb{T} \times \mathbb{T} \to$ **Bool**

52 $<,\leq,=,\neq,\geq,>$: $\mathbb{TI} \times \mathbb{TI} \to$ **Bool**

axiom

48 $\forall$ t:$\mathbb{T}$ $\cdot$ t+$\mathbf{0}$ = t

7.2.3. Temporal Observers

53 We define the signature of the meta-physical time observer.

type
53 $\mathbb{T}$
value
53 **record**_$\mathbb{TIME}$(): **Unit** $\rightarrow$ $\mathbb{T}$

- The time recorder applies to nothing and yields a time.
- **record**_$\mathbb{TIME}$() can only occur in action, event and behavioural descriptions.

7.2.4. Models of Time:

- Modern models of time, by mathematicians and physicists
  - evolve around spacetime[46]
  - We shall not be concerned with this notion of time.
- Models of time related to computing
  differs from those of mathematicians and physicists in focusing on
  - divergence and convergence, zero (Zenon) time and
  - interleaving time [**?**] are relevant
  - in studies of real-time, typically distributed computing systems.
  - We shall also not be concerned with this notion of time.

[46]The concept of **Spacetime** was first "announced" by Hermann Minkowski, 1907–08 – based on work by Henri Poincaré, 1905–06, `https://en.wikisource.org/wiki/Translation: The_Fundamental_Equations_for_Electromagnetic_Processes_in_Moving_Bodies`

7.2.5. Spatial and Temporal Modelling:

- It is not always that we are compelled to endow our domain descriptions with those of spatial and/or temporal properties.
  - In our experimental domain descriptions, for example,
  - [**?**, **?**, **?**, **?**, **?**, **?**, **?**, **?**], we have
  - either found no need to model space and/or time,
  - or we model them explicitly,
  - using slightly different types and observers
  - than presented above.

7.3. Whither Attributes ?

- Are space and time attributes of endurants ?
  - Of course not !
  - Space and time surround us.
  - Every endurant is in the one-and-only space we know of.
  - Every endurant is "somewhere" in that space.
  - We represent that 'somewhere' by a point in space.
  - Every endurant point can be recorded.
  - And every such recording can be time-stamped.

7.4. Whither Entities ?

- Are space and time entities ?
- Of course not !
- They are simply abstract concepts
- that apply to any entity.

8. Perdurants

- The main transcendental deduction of this paper
  - is that of associating
  - with each part
  - a behaviour.
- This section shows the details of this association.
- A main conjecture of this paper is this:

- Perdurants are understood in terms of
  - a notion of *time* and
  - a notion of *state*.
- We covered the notion of
  - state in Sect. on Slide 132 and
  - time in Sect. on Slide 270.

8.1. States, Actors, Actions, Events and Behaviours: A Preview

Example 30: Example 30: Constants and States

Constants: . . . .

54 Let there be given a universe of discourse, rts. It is an example of a state.

From that state we can calculate other states.

55 The set of all hubs, hs.

56 The set of all links, ls.

57 The set of all hubs and links, hls.

58 The set of all bus companies, bcs.

59 The set of all buses, bs.

60 The map from the unique bus company identifiers, see Item 18c. Slide 176, to the set of all the identifies bus company's buses, $bc_{ui}bs$.

61 The set of all private automobiles, as.

62 The set of all parts, ps.

value

54 rts:UoD [54]

55 hs:H-**set** ≡:H-**set** ≡ obs_sH(obs_SH(obs_RN(rts)))

56 ls:L-**set** ≡:L-**set** ≡ obs_sL(obs_SL(obs_RN(rts)))

57 hls:(H|L)-**set** ≡ $hs \cup ls$

58 bcs:BC-**set** ≡ obs_BCs(obs_SBC(obs_FV(obs_RN(rts))))

59 bs:B-**set** ≡ ∪{obs_Bs(bc)|bc:BC•bc ∈ bcs}

60 as:A-**set** ≡ obs_BCs(obs_SBC(obs_FV(obs_RN(rts))))

Indexed States: ....

- We shall

63 index bus companies,

64 index buses, and

65 index automobiles

using the unique identifiers of these parts.

type

63 BC_{ui}

64 B_{ui}

65 A_{ui}

value

63 $ibcs$:BC_{ui}-**set** $\equiv$

63 $\{\ bc_{ui} \mid bc{:}BC, bc{:}BC_{ui}{:}BC_{ui} \cdot bc \in bcs \wedge ui = uid\_BC(bc)\ \}$

64 ibs:B_{ui}-**set** $\equiv$

64 $\{\ b_{ui} \mid b{:}B, b{:}B_{ui}{:}B_{ui} \cdot b \in bs \wedge ui = uid\_B(b)\ \}$

65 ias:A_{ui}-**set** $\equiv$

65 $\{\ a_{ui} \mid a{:}A, a{:}A_{ui}{:}A_{ui} \cdot a \in as \wedge ui = uid\_A(a)\ \}$

■

8.1.1. Actors, Actions, Events, Behaviours and Channels

- To us perdurants are further, pragmatically, analysed into
  - *action*s,
  - *event*s, and
  - *behaviour*s.
- We shall define these terms below.
- Common to all of them is that they potentially change a state.
- Actions and events are here considered atomic perdurants.
- For behaviours we distinguish between
  - discrete and
  - continuous

  behaviours.

8.1.2. Time Considerations

- We shall, without loss of generality, assume
  - that actions and events are atomic
  - and that behaviours are composite.
- Atomic perdurants may "occur" during some time interval,
  - but we omit consideration of and concern for what actually goes on during such an interval.
- Composite perdurants can be analysed into "constituent"
  - actions,
  - events and
  - "sub-behaviours".
- We shall also omit consideration of temporal properties of behaviours.

8.1.3. Actors

Definition 20 Actor: By an **actor** we shall understand

- something that is capable of initiating and/or **carrying out**
  - actions,
  - events or
  - behaviours ■

- We shall, in principle, associate an actor with each part[47].
  - These actors will be described as behaviours.
  - These behaviours evolve around a state.
  - The state is
    - the set of qualities,
      in particular the dynamic attributes,
      of the associated parts
    - and/or any possible components or materials of the parts.

[47] This is an example of a *transcendental deduction*.

8.1.4. Discrete Actions

Definition 21 Discrete Action: By a **discrete action** we shall understand

- a foreseeable thing
- which deliberately and
- potentially changes a well-formed state, in one step,
- usually into another, still well-formed state,
- for which an actor can be made responsible ■

- An action is what happens when a function invocation changes, or potentially changes a state.

8.1.5. Discrete Events

Definition 22 Event: By an **event** we shall understand

- some unforeseen thing,
- that is, some 'not-planned-for' "action", one
- which surreptitiously, non-deterministically changes a well-formed state
- into another, but usually not a well-formed state,
- and for which no particular domain actor can be made responsible

■

- Events can be characterised by
  - a pair of (before and after) states,
  - a predicate over these
  - and, optionally, a *time* or *time interval*.

8.1.6. Discrete Behaviours

Definition 23 Discrete Behaviour: By a **discrete behaviour** we shall understand

- a set of sequences of potentially interacting sets of discrete
  - actions,
  - events and
  - behaviours ■

- Discrete behaviours now become the *focal point* of our investigation.
  - To every part we associate, by transcendental deduction, a behaviour.
  - We shall express these behaviours as CSP *processes* [**?**].
    - For those behaviours we must therefore establish their means of *communication* via *channels*;
    - their *signatures*; and
    - their *definitions* – as *translated* from endurant parts.

Example 31: Example 31: Behaviours

- *In the figure of the Channels example of Page 306*
- *we "symbolically", i.e., the "...", show the following parts:*
  - *each individual hub,*
  - *each individual link,*
  - *each individual bus company,*
  - *each individual bus, and*
  - *each individual automobile*
  - *and all of these.*

- *The idea is that those are the parts for which we shall define behaviours.*
- *That figure, however, and in contrast to Fig. 5 on Slide 147,*
  - *shows the composite parts as not containing their atomic parts,*
  - *but as if they were "free-standing, atomic" parts.*
- *That shall visualise the transcendental interpretation*
  - *as atomic part behaviours*
  - *not being somehow embedded in composite behaviours,*
  - *but operating concurrently, in parallel*

■

8.2. Channels and Communication

- We choose to exploit the CSP [?] subset of RSL
- since CSP is a suitable vehicle for expressing
- suitably abstract synchronisation and communication between behaviours.
- The mereology of domain parts induces channel declarations.
- CSP channels are loss-free.
  - That is: two CSP processes, of which one offers and the other offers to accept a message
  - do so synchronously and without forgetting that message.

- If you model actual, so-called “real-life” communication
  - via queues or allowing “channels” to forget,
  - then you must model that explicitly in CSP.
  - We refer to [**?**, **?**, **?**].

8.2.1. The CSP Story:

- CSP processes (models of domain behaviours), $P_i, P_j, \ldots, P_k$ can proceed in parallel:

    P_i || P_j || ... || P_k

- Behaviours
  - sometimes synchronise
  - and usually communicate.

- Synchronisation and communication is abstracted as
  - the sending (ch ! m) and
  - receipt (ch ?)
  - of messages, m:M,
  - over channels, ch.

type M
channel ch:M

- Communication between (unique identifier) indexed behaviours have their channels modeled as similarly indexed channels:

  out: ch[idx]!m
  in: ch[idx]?
  channel {ch[ide]:M|ide:IDE}

  where IDE typically is some type expression over unique identifier types.

- The expression

  P_i ⊓ P_j ⊓ ... ⊓ P_k

  – can be understood as a choice:
    - either P_i,
    - or P_j,
    - or ...
    - or P_k

    is *non-deterministically* **internally** chosen

  – with no stipulation as to why !

- The expression

  P_i □ P_j □ ... □ P_k

  – can be understood as a choice:

    - either P_i,
    - or P_j,
    - or ...
    - or P_k

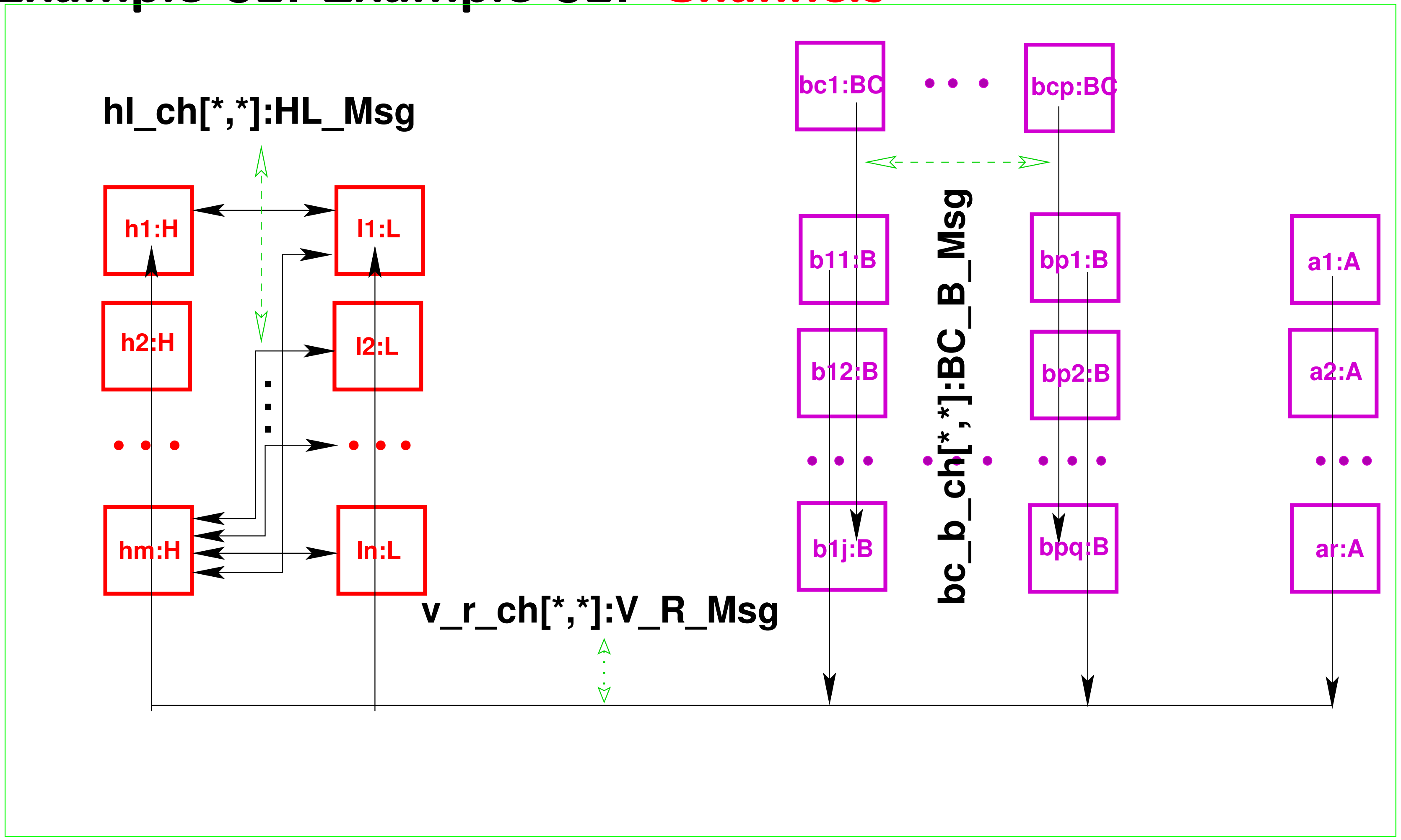
    is *deterministically* **externally** chosen

  – on the basis that the one chosen offers to participate in either an input, ch ?, or an output, ch ! msg, event.

  – If more than one P_i offers a communication then one is arbitrarily chosen.

  – If no P_i offers a communication the behaviour halts till some P_j offers a communication.

Example 32: Example 32: Channels



- *We shall argue for hub-to-link channels based on the mereologies of those parts.*
  - *Hub parts may be topologically connected to any number, 0 or more, link parts.*
  - *Only instantiated road nets knows which.*
  - *Hence there must be channels between any hub behaviour and any link behaviour.*
  - *Vice versa: link parts will be connected to exactly two hub parts.*
  - *Hence there must be channels from any link behaviour to two hub behaviours.*
- *See the figure above.*

Channel Message Types: ....

- *We ascribe types to the messages offered on channels.*

66 Hubs and links communicate, both ways, with one another, over channels, `hl_ch`, whose indexes are determined by their mereologies.

67 Hubs send one kind of messages, links another.

68 Bus companies offer timed bus time tables to buses, one way.

69 Buses and automobiles offer their current, timed positions to the road element, hub or link they are on, one way.

type

67 H_L_Msg, L_H_Msg

66 HL_Msg = H_L_Msg | L_F_Msg

68 BC_B_Msg = T × BusTimTbl

69 V_R_Msg = T × (BPos|APos)

Channel Declarations: . . . .

70 This justifies the channel declaration which is calculated to be:

channel

70 { hl_ch[h_ui,l_ui]:H_L_Msg

70 | h_ui:H_UI,l_ui:L_UI•i ∈ $h_{ui}s$∧j ∈ $lh_{ui}m$(h_ui) }

70 ∪

70 { hl_ch[h_ui,l_ui]:L_H_Msg

70 | h_ui:H_UI,l_ui:L_UI•l_ui ∈ $l_{ui}s$∧i ∈ $lh_{ui}m$(l_ui) }

- *We shall argue for bus company-to-bus channels based on the mereologies of those parts.*
  - *Bus companies need communicate to all its buses, but not the buses of other bus companies.*
  - *Buses of a bus company need communicate to their bus company, but not to other bus companies.*

71 *This justifies the channel declaration which is calculated to be:*

channel
71 { bc_b_ch[bc_ui,b_ui] | bc_ui:BC_UI, b_ui:B_UI
71 · bc_ui $\in$ $bc_{ui}s$ $\wedge$ b_ui $\in$ $b_{ui}s$ }: BC_B_Msg
71 { bc_b_ch[bc_ui,b_ui] | bc_ui:BC_UI,b_ui:B_UI
71 · bc_ui $\in$ $bc_{ui}s$$\wedge$j $\in$ $b_{ui}s$ }: BC_B_MSG
71 { bc_b_ch[bc_ui,b_ui] | bc_ui:BC_UI,b_ui:B_UI
71 · bc_ui $\in$ $bc_{ui}s$$\wedge$j $\in$ $b_{ui}s$ }: BC_B_MSG

- *We shall argue for vehicle to road element channels based on the mereologies of those parts.*
  - *Buses and automobiles need communicate to*
    - *all hubs and*
    - *all links.*

72 This justifies the channel declaration which is calculated to be:

channel

72 { v_r_ch[v_ui,r_ui] | v_ui:V_UI,r_ui:R_UI

72 · v_ui∈ $v_{ui}s$∧r_ui∈ $r_{ui}s$ }: V_R_Msg

- *The channel calculations are described on Slides 338–345*

■

8.2.2. From Mereologies to Channel Declarations:

- The fact
  - that a part, p of sort P with unique identifier p_i,
  - has a mereology, for example the set of unique identifiers $\{q_a, q_b, ..., q_d\}$
  - identifying parts $\{qa, qb, ..., qd\}$ of sort Q, may mean
  - that parts p and $\{qa, qb, ..., qd\}$
  - may wish to exchange – for example, attribute – values,
  - one way (from p to the qs)
    or the other (vice versa)
    or in both directions.

- Figure 7 shows two dotted rectangle box diagrams.

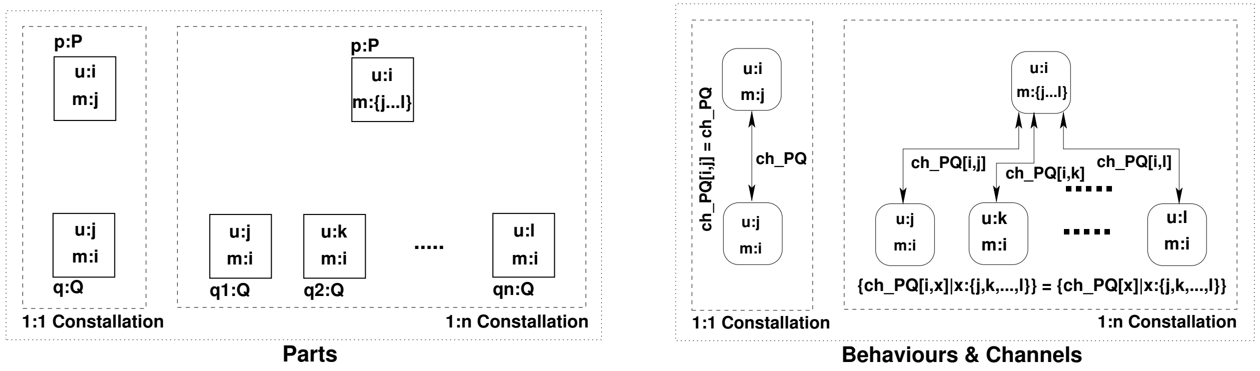


Figure 7: Two Part and Channel Constallations.
u:p *u*nique id. *p*; *m:p* *m*ereology *p*

- The left fragment of the figure intends to show a `1:1 Constallation` of a single p:P box and a single q:Q part, respectively, indicating, within these parts, their unique identifiers and mereologies.
- The right fragment of the figure intends to show a `1:n Constallation` of a single p:P box and a set of q:Q parts, now with arrowed lines connecting the p part with the q parts.
- These lines are intended to show channels.
- We show them with two way arrows.
- We could instead have chosen one way arrows, in one or the other direction.
- The directions are intended to show a direction of value transfer.
- We have given the same channel names to all examples, ch_PQ.
- We have ascribed channel message types MPQ to all channels.[48]

[48]Of course, these names and types would have to be distinct for any one domain description.

- Figure 8 shows an arrangement similar to that of Fig. 7 on Slide 314, but for an `m:n Constallation`.

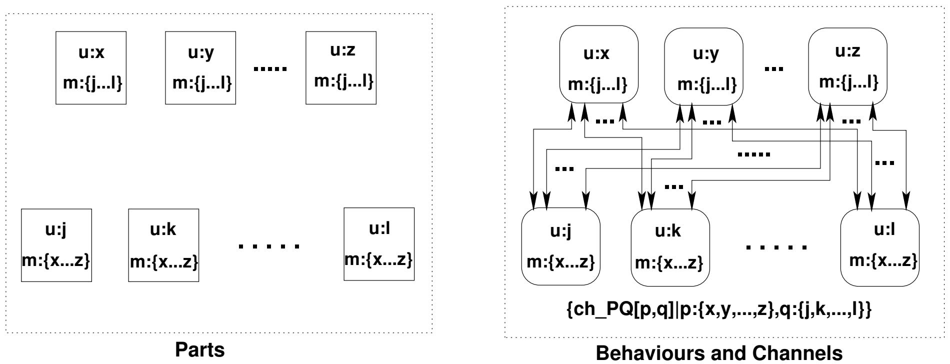


Figure 8: Multiple Part and Channel Arrangements:
*u:p u*nique id. *p*; *m:p m*ereology *p*

- The channel declarations corresponding to Figs. 7 and 8 are:

```
   channel
[1]       ch_PQ[i,j]:MPQ
[2]       { ch_PQ[i,x]:MPQ | x:{j,k,...,l} }
[3]       { ch_PQ[p,q]:MPQ | p:{x,y,...,z}, q:{j,k,...,l} }
```

- Since there is only one index i and j for channel [1], its declaration can be reduced.
- Similarly there is only one i for declaration [2]:

```
    channel
[1]     ch_PQ:MPQ
[2]     { ch_PQ[x]:MPQ | x:{j,k,...,l} }
```

73 The following description identities holds:

```
73  { ch_PQ[x]:MPQ | x:{j,k,...,l} } ≡ ch_PQ[j],ch_PQ[k],...,ch_PQ[l],

73  { ch_PQ[p,q]:MPQ | p:{x,y,...,z}, q:{j,k,...,l} } ≡
73      ch_PQ[x,j],ch_PQ[x,k],...,ch_PQ[x,l],
73      ch_PQ[y,j],ch_PQ[y,k],...,ch_PQ[y,l],
73      ...,
73      ch_PQ[z,j],ch_PQ[z,k],...,ch_PQ[z,l]
```

- We can sketch a diagram
- similar to Figs. 7 on Slide 314 and 8 on Slide 316
- for the case of composite parts.

8.2.3. Continuous Behaviours:

- By a **continuous behaviour** we shall understand
  - a *continuous time*
  - sequence of *state changes*.
- We shall not go into what may cause these *state changes*.
- And we shall not go into continuous behaviours in these lectures.

8.3. Perdurant Signatures

- We shall treat perdurants as function invocations.
- In our cursory overview of perdurants
  - we shall focus on one perdurant quality:
  - function signatures.

Definition 24 Function Signature: By a **function signature** we shall understand

- a *function name* and
- a *function type expression* ■

Definition 25 **Function Type Expression:** By a **function type expression** we shall understand

- a pair of *type expressions*.
- separated by a *function type constructor*
  - either $\rightarrow$ (for **total function**)
  - or $\stackrel{\sim}{\rightarrow}$ (for **partial function**) ■

- The *type expressions*
  - are part sort or type, or material sort or type, or component sort or type, or attribute type names,
  - but may, occasionally be expressions over respective type names involving **-set**, $\times$, *, $\overrightarrow{m}$and $|$ type constructors.

8.3.1. Action Signatures and Definitions:

- Actors usually provide their initiated actions with arguments, say of type VAL.
  - Hence the schematic function (action) signature and schematic definition:

$$\mathsf{action}\colon\ \mathsf{VAL} \to \Sigma \xrightarrow{\sim} \Sigma$$

$$\mathsf{action(v)}(\sigma)\ \textbf{as}\ \sigma'$$

$$\quad \textbf{pre}\colon\ \ \mathscr{P}(\mathsf{v},\sigma)$$

$$\quad \textbf{post}\colon\ \mathscr{Q}(\mathsf{v},\sigma,\sigma')$$

  - expresses that a selection of the domain,
  - as provided by the Σ type expression,
  - is acted upon and possibly changed.

- The partial function type operator $\stackrel{\sim}{\rightarrow}$
  - shall indicate that action(v)(σ)
  - may not be defined for the argument, i.e., initial state σ
  - and/or the argument v:VAL,
  - hence the precondition $\mathscr{P}$(v,σ).
- The post condition $\mathscr{Q}$(v,σ,σ')
  characterises the "after" state, σ':Σ,
  with respect to the "before" state, σ:Σ, and possible arguments (v:VAL).

- Which could be the argument values, v:VAL, of actions ?
  - Well, there can basically be only the following kinds of argument values:
    - parts, components and materials, respectively
    - unique part identifiers, mereologies and attribute values.

- **Perdurant (action) analysis thus proceeds as follows:**
  - identifying relevant actions,
  - assigning names to these,
  - delineating the “smallest” relevant state[49],
  - ascribing signatures to action functions, and
  - determining
    - action pre-conditions and
    - action post-conditions.

[49] By “smallest” we mean: containing the fewest number of parts. Experience shows that the domain analyser cum describer should strive for identifying the smallest state.

- Of these, ascribing signatures is the most crucial:
  - In the process of determining the action signature
  - one oftentimes discovers
  - that part or component or material attributes have been left ("so far") "undiscovered".

8.3.2. Event Signatures and Definitions:

- Events are usually characterised by
  - the absence of known actors and
  - the absence of explicit "external" arguments.
- Hence the schematic function (event) signature:

value
  event: $\Sigma \times \Sigma \stackrel{\sim}{\rightarrow}$ **Bool**
  event(σ,σ') **as** tf
    pre: $P(\sigma)$
    post: tf $= Q(\sigma,\sigma')$

- The event signature expresses
  - that a selection of the domain
  - as provided by the Σ type expression
  - is "acted" upon, by unknown actors, and possibly changed.
- The partial function type operator $\stackrel{\sim}{\to}$
  - shall indicate that event(σ, σ')
  - may not be defined for some states σ.
- The resulting state may, or may not, satisfy axioms and well-formedness conditions over Σ – as expressed by the post condition $Q(\sigma, \sigma')$.

- Events may thus cause well-formedness of states to fail.
- Subsequent actions,
  - once actors discover such “disturbing events”,
  - are therefore expected to remedy that situation, that is,
  - to restore well-formedness.
- We shall not illustrate this point.

8.3.3. Discrete Behaviour Signatures

Signatures:

- We shall only cover behaviour signatures when expressed in `RSL/CSP`.
- The behaviour functions are now called processes.
- That a behaviour function is a never-ending function, i.e., a process, is "revealed" by the "trailing" **Unit**:

  behaviour: ... $\rightarrow$ ... **Unit**

- That a process takes no argument
  is ”revealed” by a “leading” Unit:

  behaviour: **Unit** $\rightarrow$ ...

- That a process accepts channel, viz.: ch, inputs,
  is “revealed” as follows:

  behaviour: ... $\rightarrow$ **in** ch ...

- That a process offers channel, viz.: ch, outputs is "revealed" as follows:

  behaviour: ... → **out** ch ...

- That a process accepts other arguments is "revealed" as follows:

  behaviour: ARG → ...

- where ARG can be any type expression:

  T, T→T, T→T→T, etcetera

  where T is any type expression.

8.3.4. Attribute Access, An Interpretation:

- We shall only be concerned with part attributes.
- And we shall here consider them in the context of part behaviours.
  - Part behaviour definitions embody part attributes.
  - In this section we shall suggest how behaviours embody part attributes.

- **Static attributes** designate constants, cf. Defn. 1 Slide 206.
  As such they can be "compiled" into behaviour definitions.
  We choose, instead to list them,
  in behaviour signatures, as arguments.

- **Inert attributes** designate values provided by external stimuli,
  cf. Defn. 3 Slide 207,
  that is, must be obtained by channel input: attr_Inert_A_ch ?, i.e.,
  are considered monitorable.

- **Reactive attributes** are functions of other attribute values,
  cf. Defn. 4 Slide 207.

- **Autonomous attributes** must be input,
  cf. Defn. 6 Slide 208,
  like inert attributes: attr_Autonomous_A_ch ?, i.e., are considered

monitorable.

- **Programmable attribute** values are calculated by their behaviours,
  cf. Defn. 8 Slide 209.
  We list them as behaviour arguments.
  The behaviour definitions may then specify new values. These are provided in the position of the programmable attribute arguments in *tail recursive* invocations of these behaviours.
- **Biddable attributes** are now considered programmable attributes, but when provided, in possibly tail recursive invocations of their behaviour, the calculated biddable attribute value is *modified*, usually by some *perturbation*[50] of the calculated value – to reflect that although they *are prescribed* they *may fail to be observed as such*, cf. Defn. 7 Slide 209.

[50]– in the sense of https://en.wikipedia.org/wiki/Perturbation_function

8.3.5. Calculating In/Output Channel Signatures:

- Given a part p we can calculate the RSL$^+$Text that designates the input channels on which part p behaviour obtains monitorable attribute values.
- For each monitorable attribute, A, the text ⋘ attr_A_ch⋙ is to be "generated".
- One or more such channel declaration contributions is to be preceded by the text ⋘**in** ⋙.
- If there are no monitorable attributes then no text is t be yielded.

74 The function calc_i_o_chn_refs apply to parts and yield RSL^{+}Text.

a. From p we calculate its unique identifier value, its mereology value, and its monitorable attribute values.

b. If there the mereology is not void and/or the are monitorable values then a (Currying[51]) right pointing arrow, $\rightarrow$, is inserted.[52]

c. If there is an input mereology and/or there are monitorable values then the keyword **in** is inserted
in front of the monitorable attribute values and input mereology.

d. Similarly for the input/output mereology;

e. and for the output mereology.

[51] https://en.wikipedia.org/wiki/Currying

[52] We refer to the three parts of the mereology value as the input, the input/output and the output mereology (values).

value

74 calc_i_o_chn_refs: P → RSL$^+$Text

74 calc_i_o_chn_refs(p) ≡ ;

74a. **let** ui = **uid**_P(p), (ics,iocs,ocs) = **mereo**_(p), atrvs = **obs_attrib_values**_P(p) **in**

74b. **if** ics ∪ iocs ∪ ocs ∪ atrvs ≠ {} **then** ≪ → ≫ **end** ;

74c. **if** ics ∪ atrvs ≠{} **then** ≪**in**≫ calc_attr_chn_refs(ui,atrvs), calc_chn_refs(ui,ichs) **end** ;

74d. **if** iocs≠{} **then** ≪**in,out**≫ calc_chn_refs(ui,iochs) **end** ;

74e. **if** ocs≠{} **then** ≪**out**≫ calc_chn_refs(ui,ochs) **end end**

75 The function calc_attr_chn_refs

a. apply to a set, mas, of monitorable attribute types and yield $\mathtt{RSL}^+$Text.

b. If achs is empty no text is generated.
Otherwise a channel declaration attr_A_ch
is generated for each attribute type whose name, A,
which is obtained by applying η
to an observed attribute value, ηa.

75a. calc_attr_chn_refs: UI $\times$ A-**set** $\rightarrow$ $\mathtt{RSL}^+$Text
75b. calc_attr_chn_refs(ui,mas) $\equiv$ { $\lll$ attr_ηa_ch[ui] $\ggg$ | a:A·a $\in$ mas }

76 The function calc_chn_refs

a. apply to a pair, (ui,uis) of a unique part identifier and a set of unique part identifiers and yield $\mathtt{RSL}^+$Text.

b. If uis is empty no text is generated. Otherwise an array channel declaration is generated.

76a. calc_chn_refs: P_UI × Q_UI-**set** → $\mathtt{RSL}^+$Text
76b. calc_chn_refs(pui,quis) ≡ { ⋘ η(pui,qui)_ch[pui,qui] ⋙ | qui:Q_UI•qui ∈ quis }

77 The function calc_all_chn_dcls

a. apply to a pair, (pui,quis) of a unique part identifier and a set of unique part identifiers and yield $\mathtt{RSL}^+$Text.

b. If quis is empty no text is generated. Otherwise an array channel declaration

- $\{ \lll \eta$(pui,qui)_ch[pui,qui]:η(pui,qui)M $\ggg$ | qui:Q_UI$\cdot$qui $\in$ quis }

is generated.

77a. calc_all_chn_dcls: P_UI $\times$ Q_UI-**set** $\rightarrow$ $\mathtt{RSL}^+$Text
77a. calc_all_chn_dcls(pui,quis) $\equiv$ { $\lll\eta$(pui,qui)_ch[pui,qui]:η(pui,qui)M $\ggg$ | qui:Q_U

- The η(pui,qui) invocation serves to prefix-name both
  - the channel, η(pui,qui)_ch[pui,qui], and
  - the channel message type, η(pui,qui)M.

78 The overloaded η operator[53] is here applied to a pair of unique identifiers.

78 η: (UI $\rightarrow$ RSL$^+$Text)|((X_UI$\times$Y_UI) $\rightarrow$ RSL$^+$Text)
78 η(x_ui,y_ui) $\equiv$ ($\lll$(η x_ui η y_ui$\ggg$))

[53]The η operator applies to a type and yields the ηame of the type.

- Repeating these channel calculations over distinct parts p_1,p_2,...,p_n of the same part type P will yield "similar" behaviour signature channel references:

$$\{\mathsf{PQ\_ch}[\mathsf{p1}_{ui},\mathsf{qui}]|\mathsf{p1}_{ui}:\mathsf{P\_UI},\mathsf{qui}:\mathsf{Q\_UI}\cdot\mathsf{qui} \in \mathsf{quis}\}$$
$$\{\mathsf{PQ\_ch}[\mathsf{p2}_{ui},\mathsf{qui}]|\mathsf{p2}_{ui}:\mathsf{P\_UI},\mathsf{qui}:\mathsf{Q\_UI}\cdot\mathsf{qui} \in \mathsf{quis}\}$$
$$\ldots$$
$$\{\mathsf{PQ\_ch}[\mathsf{p}n_{ui},\mathsf{qui}]|\mathsf{p}n_{ui}:\mathsf{P\_UI},\mathsf{qui}:\mathsf{Q\_UI}\cdot\mathsf{qui} \in \mathsf{quis}\}$$

- These distinct single channel references can be assembled into one:

$$\{\ \mathsf{PQ\_ch}[\mathsf{pui},\mathsf{qui}]\ |\ \mathsf{pui}:\mathsf{P\_UI},\mathsf{qui}:\mathsf{Q\_UI} : -\mathsf{pui} \in \mathsf{puis},\mathsf{qui} \in \mathsf{quis}\ \}$$
$$\textbf{where}\ \mathsf{puis} = \{\ \mathsf{p1}_{ui},\mathsf{p2}_{ui},\ldots,\mathsf{p}n_{ui}\}$$

- As an example we have already calculated the array channels for Fig. 8 Slide 316– cf. the left, the **Parts**, of that figure – cf. Items [1–3] Pages 317–319.
- The identities Item 73 Slide 319 apply.

8.4. Discrete Behaviour Definitions

- We associate with each part, p:P, a behaviour name $\mathscr{M}_P$.
- Behaviours have as first argument their unique part identifier: **uid**_P(p).
- Behaviours evolves around a state, or, rather, a set of values:
  - its possibly changing mereology, mt:MT and
  - the attributes of the part.[54]

[54] We leave out consideration of possible components and materials of the part.

- A behaviour signature is therefore:

  $\mathcal{M}_P$: ui:UI×me:MT×stat_attr_typs(p) → ctrl_attr_typs(p) → calc_i_o_chn_refs(p) **Unit**

  where

  – (i) ui:UI is the unique identifier value and type of part p;
  – (ii) me:MT is the value and type mereology of part p;
  – (iii) stat_attr_typs(p): static attribute types of part *p*:*P*;
  – (iv) ctrl_attr_typs(p): controllable attribute types of part *p*:*P*;
  – (v) calc_i_o_chn_refs(p) calculates references to the **in**put, the **in**put/**out**put and the **out**put channels serving the attributes shared between part *p* and the parts designated in its mereology me.

- Let P be a composite sort defined in terms of endurant[55] sub-sorts E_1, E_2, ..., E_n.
  - The behaviour description *translated* from p:P, is composed from
    - a behaviour description, $\mathscr{M}_P$, relying on and handling the unique identifier, mereology and attributes of part p
    - to be *translated* with behaviour descriptions $\beta_1, \beta_2, \ldots, \beta_n$ where
      - β_1 is *translated* from e_1:E_1,
      - β_2 is *translated* from e_2:E_2,
      - ..., and
      - β_n is *translated* from e_n:E_n.

[55]– structures or composite

- The domain description *translation* schematic below "formalises" the above.

Abstract `is_composite(p)` Behaviour Schema

```
value
  Translate_P: P → RSL⁺Text
  Translate_P(p) ≡
    let ui = uid_P(p), me = mereo_P(p),
      sa = stat_attr_vals(p), ca = ctrl_attr_vals(p),
      MT = mereo_type(p), ST = stat_attr_typs(p), CT = ctrl_attr_typs(p),
      IOR = calc_i_o_chn_refs(p), IOD = calc_all_ch_dcls(p) in
    ⋘ channel
        IOD
      value
        ℳ_P: P_UI × MT × ST  CT  IOR  Unit
        ℳ_P(ui,me,sta)(pa) ≡ ℬ_P(ui,me,sta)ca
         ,⋙ Translate_P1(obs_endurant_sorts_E1(p))
        ⋘⋙ Translate_P2(obs_endurant_sorts_E2(p))
        ⋘⋙ ...
        ⋘⋙ Translate_Pn(obs_endurant_sorts_En(p))
    end
```

- Expression $\mathscr{B}_P$(ui,me,sta,pa) stands for the *behaviour definition body* in which the names ui, me, sta, pa are bound to the *behaviour definition head*, i.e., the left hand side of the $\equiv$.
- Endurant sorts E_1, E_2, ..., E_n are obtained from the `observe_endurant_sorts` prompt, Slide 144.
- We informally explain the **Translate**$_{\mathsf{P}_i}$ function.
  - It takes endurants and produces RSL$^+$Text.
  - Resulting texts are bracketed: ⋘rsl_text⋙.

Example 33: Example 33: Signatures

- *We first decide on names of behaviours.*
  - *In Sect. , Pages 346–365,*
  - *we gave schematic names to behaviours of the form $\mathscr{M}_P$.*
  - *We now assign mnemonic names: from part names to names of transcendentally interpreted behaviours*
  - *and then we assign signatures to these behaviours.*

. . . .

79 $hub_{h_{ui}}$:

a. there is the usual "triplet" of arguments: unique identifier, mereology and static attributes;

b. then there are the programmable attributes;

c. and finally there are the input/output channel references: first those allowing communication between hub and link behaviours,

d. and then those allowing communication between hub and vehicle (bus and automobile) behaviours.

```
value
79    hub_{h_ui}:
79a.    h_ui:H_UI×(vuis,luis,__):H_Mer×HΩ
79b.      → (HΣ×H_Traffic)
79c.      → in,out { h_l_ch[h_ui,l_ui] | l_ui:L_UI•l_ui ∈ luis }
79d.            { ba_r_ch[h_ui,v_ui] | v_ui:V_UI•v_ui∈vuis } Unit
79a.       pre: vuis = v_ui s ∧ luis = l_ui s
```

. . . .

80 $link_{l_{ui}}$:

a. there is the usual “triplet” of arguments: unique identifier, mereology and static attributes;

b. then there are the programmable attributes;

c. and finally there are the input/output channel references: first those allowing communication between hub and link behaviours,

d. and then those allowing communication between link and vehicle (bus and automobile) behaviours.

value

80 link$_{l_{ui}}$:
80a. l_ui:L_UI×(vuis,huis,_):L_Mer×LΩ
80b. → (LΣ×L_Traffic)
80c. → **in**,**out** { h_l_ch[h_ui,l_ui] | h_ui:H_UI:h_ui ∈ huis }
80d. { ba_r_ch[l_ui,v_ui] | v_ui:(B_UI|A_UI)·v_ui∈vuis } **Unit**
80a. **pre**: vuis = $v_{ui}s$ ∧ huis = $h_{ui}s$

. . . .

81 $bus\_company_{bc_{ui}}$:

a. there is here just a "doublet" of arguments: unique identifier and mereology;

b. then there is the one programmable attribute;

c. and finally there are the input/output channel references allowing communication between the bus company and buses.

value

81 bus_company$_{bc_{ui}}$:
81a. bc_ui:BC_UI×(_,_,buis):BC_Mer
81b. → BusTimTbl
81c. **in**,**out** {bc_b_ch[bc_ui,b_ui]|b_ui:B_UI·b_ui∈buis} **Unit**
81a. **pre**: buis = $b_{ui}s$ ∧ huis = $h_{ui}s$

. . . .

82 $bus_{b_{ui}}$:

a. there is here just a "doublet" of arguments: unique identifier and mereology;

b. then there are the programmable attributes;

c. and finally there are the input/output channel references: first the input/output allowing communication between the bus company and buses,

d. and the input/output allowing communication between the bus and the hub and link behaviours.

value

82 bus$_{b_{ui}}$:

82a. b_ui:B_UI×(bc_ui,__,ruis):B_Mer

82b. → (LN × BTT × BPOS)

82c. → **out** bc_b_ch[bc_ui,b_ui],

82d. {ba_r_ch[r_ui,b_ui]|r_ui:(H_UI|L_UI)·ui∈$v_{ui}s$} **Unit**

82a. **pre**: ruis = $r_{ui}s$ ∧ bc_ui ∈ $bc_{ui}s$

. . . .

83 automobile$_{a_{ui}}$:

a. there is the usual "triplet" of arguments: unique identifier, mereology and static attributes;

b. then there is the one programmable attribute;

c. and finally there are the input/output channel references allowing communication between the automobile and the hub and link behaviours.

value

83 automobile$_{a_{ui}}$:

83a. a_ui:A_UI×(_,_,ruis):A_Mer×rn:RegNo

83b. → apos:APos

83c. **in**,**out** {ba_r_ch[a_ui,r_ui]|r_ui:(H_UI|L_UI)·r_ui∈ruis} **Unit**

83a. **pre**: ruis = $r_{ui}s$ ∧ a_ui ∈ $a_{ui}s$

■

- For the case that an endurant is a structure
  - there is only its elements to compile;
  - otherwise Schema 2 is as Schema 1.

Abstract is_structure(e) Behaviour Schema

value

$\mathbf{Translate}_{E}(\mathsf{e}) \equiv$

$\quad\quad \mathbf{Translate}_{E_1}(\mathbf{obs\_endurant\_sorts\_}\mathsf{E}_1(\mathsf{e}))$

$\quad \langle\!\langle\!\!\!\not\,\rangle\!\rangle \; \mathbf{Translate}_{E_2}(\mathbf{obs\_endurant\_sorts\_}\mathsf{E}_2(\mathsf{e}))$

$\quad \langle\!\langle\!\!\!\not\,\rangle\!\rangle \; \ldots$

$\quad \langle\!\langle\!\!\!\not\,\rangle\!\rangle \; \mathbf{Translate}_{E_n}(\mathbf{obs\_endurant\_sorts\_}\mathsf{E}_n(\mathsf{e}))$

- Let P be a composite sort defined in terms of the concrete type Q-**set**.
  - The process definition compiled from p:P, is composed from
    - a process, $\mathscr{M}_P$, relying on and handling the unique identifier, mereology and attributes of process *p* as defined by P
    - operating in parallel with processes *q*:**obs**_Qs(p).
- The domain description "compilation" schematic below "formalises" the above.

Concrete `is_composite(p)` Behaviour Schema

type
  Qs = Q-**set**
value
  qs:Q-**set** = **obs**_Qs(p)
  Translate$_P$(p) ≡
    let ui = **uid**_P(p), me = **mereo**_P(p),
        sa = stat_attr_vals(p), ca = ctrl_attr_vals(p)
        ST = stat_attr_typs(p), CT = ctrl_attr_typs(p),
        IOR = calc_i_o_chn_refs(p), IOD = calc_all_ch_dcls(p) **in**
    ≪ **channel**
        IOD
      value
        $\mathscr{M}_P$: P_UI×MT×ST CT IOR **Unit**
        $\mathscr{M}_P$(ui,me,sa)ca ≡ $\mathscr{B}_P$(ui,me,sa)ca ≫
        { ≪,≫ **Translate**$_Q$(q)|q:Q•q ∈ qs }
    end

Atomic `is_atomic(p)` Behaviour Schema

value
  Translate$_P$(p) ≡
    let ui = **uid**_P(p), me = **mereo**_P(p),
      sa = stat_attr_vals(p), ca = ctrl_attr_vals(p),
      ST = stat_attr_typs(p), CT = ctrl_attr_typs(p),
      IOR = calc_i_o_chn_refs(p), IOD = calc_all_chs(p) **in**
    ⋘ **channel**
        IOD
      value
        $\mathscr{M}_P$: P_UI×MT×ST PT IOR **Unit**
        $\mathscr{M}_P$(ui,me,sa)ca ≡ $\mathscr{B}_P$(ui,me,sa)ca ⋙
    end

- The core processes can be understood as never ending, "tail recursively defined" processes:

Core Behaviour Schema

$\mathscr{B}_P$: uid:P_UI×me:MT×sa:SA → ct:CT → **in** in_chns(p) **in**,**out** in_out_chns(me) **Unit**
$\mathscr{B}_P$(p)(ui,me,sa)(ca) ≡ **let** (me′,ca′) = $\mathscr{F}_P$(ui,me,sa)ca **in** $\mathscr{M}_P$(ui,me′,sa)ca′ **end**
$\mathscr{F}_P$: P_UI×MT×ST → CT→ in_out_chns(me) → MT×CT

- We refer to [**?**, Process Schema V: Core Process (II), Page 40] for possible forms of $\mathscr{F}_P$.

Example 35: Automobile Behaviour (at a hub)

- *We define the behaviours in a different order than the treatment of their signatures.*
- *We "split" definition of the* automobile *behaviour*
  - *into the behaviour of* automobiles *when positioned at a hub, and*
  - *into the behaviour* automobiles *when positioned at on a link.*
  - *In both cases the behaviours include the "idling" of the automobile, i.e., its "not moving", standing still.*

Example 35: Automobile Behaviour (at a hub), Contd.

84 We abstract automobile behaviour at a Hub (hui).

85 The vehicle remains at that hub, “idling”,

86 informing the hub behaviour,

87 or, internally non-deterministically,

a. moves onto a link, tli, whose “next” hub, identified by th_ui, is obtained from the mereology of the link identified by tl_ui;

b. informs the hub it is leaving and the link it is entering of its initial link position,

c. whereupon the vehicle resumes the vehicle behaviour positioned at the very beginning (0) of that link,

88 or, again internally non-deterministically,

89 the vehicle “disappears — off the radar” !

Example 35: Automobile Behaviour (at a hub), Contd.

```
84   automobile_{a_ui}(a_ui,({},(ruis,vuis),{}),rn)
84             (apos:atH(fl_ui,h_ui,tl_ui)) ≡
85       (ba_r_ch[a_ui,h_ui] ! (record_TIME(),atH(fl_ui,h_ui,tl_ui));
86        automobile_{a_ui}(a_ui,({},(ruis,vuis),{}),rn)(apos))
87       ⌈⌉
87a.      (let ({fh_ui,th_ui},ruis′)=mereo_L(℘(tl_ui)) in
87a.            assert: fh_ui=h_ui ∧ ruis=ruis′
84        let onl = (tl_ui,h_ui,0,th_ui) in
87b.       (ba_r_ch[a_ui,h_ui] ! (record_TIME(),onL(onl)) ‖
87b.        ba_r_ch[a_ui,tl_ui] ! (record_TIME(),onL(onl))) ;
87c.       automobile_{a_ui}(a_ui,({},(ruis,vuis),{}),rn)
87c.              (onL(onl)) end end)
88       ⌈⌉
89         stop
```

- Section presents the definition of the remaining automobile, hub, link, bus company and bus behaviours.

8.5. Running Systems

- It is one thing
  - to define the behaviours corresponding to all parts,
  - whether composite or atomic.
- It is another thing to
  - specify an initial configuration of behaviours,
  - that is, those behaviours
  - which "start" the overall system behaviour.

- The choice
  - as to which parts, i.e., behaviours,
  - are to represent an initial, i.e., a start system behaviour,
  - cannot be “formalised”,
  - it really depends on the “deeper purpose”
  - of the system.
- In other words:
  - requires careful analysis and is
  - beyond the scope of the present lectures.

Example 36: Initial System, I/VIII

Initial States:

- We recall the *hub, link, bus company, bus* and the *automobile states*
- first mentioned in Sect. Page 132.

value

55 hs:H-**set** $\equiv$ $\equiv$ obs_sH(obs_SH(obs_RN(rts)))

56 ls:L-**set** $\equiv$ $\equiv$ obs_sL(obs_SL(obs_RN(rts)))

58 bcs:BC-**set** $\equiv$ obs_BCs(obs_SBC(obs_FV(obs_RN(rts))))

59 bs:B-**set** $\equiv$ $\cup$\{obs_Bs(bc)|bc:BC·bc $\in$ bcs\}

61 as:A-**set** $\equiv$ obs_BCs(obs_SBC(obs_FV(obs_RN(rts))))

Example 36: Initial System, Starting Initial Behaviours: II/VIII

Starting Initial Behaviours:

- We are reaching the end of this domain modelling example.
  - Behind us there are narratives and formalisations 1 Slide 146 – 133 Slide 462.
  - Based on these we now express the signature and the body of the definition
  - of a *“system build and execute”* function.

Example 36: Initial System, Starting Initial Behaviours: III/VIII

90 The system to be initialised is

a. the parallel composition (‖) of
b. the distributed parallel composition (‖{...|...}) of
c. all the hub behaviours,
d. all the link behaviours,
e. all the bus company behaviours,
f. all the bus behaviours, and
g. all the automobile behaviours.

Example 36: Initial System, Starting Initial Behaviours: IV/VIII

value

90 initial_system: **Unit** → **Unit**
90 initial_system() ≡
90c. $\|$ { $\text{hub}_{h_{ui}}$(h_ui,me,hω)(htrf,hσ)
90c. | h:H·h ∈ hs,
90c. h_ui:H_UI·h_ui=uid_H(h),
90c. me:HMetL·me=mereo_H(h),
90c. hω:HΩ·hω=attr_HΩ(h),
90c. htrf:H_Traffic·htrf=attr_H_Traffic_H(h),
90c. hσ:HΣ·hσ=attr_HΣ(h)$\wedge$hσ ∈ hω
90c. }

Example 36: Initial System, V/VIII

90a. $\|$
90d. $\|$ { link$_{l_{ui}}$(l_ui,me,lω)(ltrf,lσ)
90d. l:L·l $\in$ ls,
90d. l_ui:L_UI·l_ui=uid_L(l),
90d. me:LMet·me=mereo_L(l),
90d. lω:LΩ·lω=attr_LΩ(l),
90d. ltrf:L_Traffic·ltrf=attr_L_Traffic_H(l),
90d. lσ:LΣ·lσ=attr_LΣ(l)$\wedge$lσ $\in$ lω
90d. }

Example 36: Initial System, VI/VIII

```
90a.      ‖
90e.      ‖ { bus_company_{bc_ui}(bcui,me)(btt)
90e.            bc:BC·bc ∈ bcs,
90e.            bc_ui:BC_UI·bc_ui=uid_BC(bc),
90e.            me:BCMet·me=mereo_BC(bc),
90e.            btt:BusTimTbl·btt=attr_BusTimTbl(bc)
90e.        }
```

Example 36: Initial System, VII/VIII

```
90a.      ‖
90f.      ‖ { bus_{b_ui}(b_ui,me)(ln,btt,bpos)
90f.          b:B·b ∈ bs,
90f.          b_ui:B_UI·b_ui=uid_B(b),
90f.          me:BMet·me=mereo_B(b),
90f.          ln:LN:pln=attr_LN(b),
90f.          btt:BusTimTbl·btt=attr_BusTimTbl(b),
90f.          bpos:BPos·bpos=attr_BPos(b)
90f.        }
```

Example 36: Initial System, VIII/VIII

```
90a.     ‖
90g.     ‖ { automobile_a_ui(a_ui,me,rn)(apos)
90g.         a:A•a ∈ as,
90g.         a_ui:A_UI•a_ui=uid_A(a),
90g.         me:AMet•me=mereo_A(a),
90g.         rn:RegNo•rno=attr_RegNo(a),
90g.         apos:APos•apos=attr_APos(a)
90g.       }
```

8.6. Concurrency: Communication and Synchronisation

- Process Schemas I, II, III and V (Slides 349, 362, 364 and 366), reveal
  - that two or more parts, which temporally coexist (i.e., at the same time),
  - imply a notion of *concurrency*.
- Process Schema IV, Page 365,
  - through the `RSL/CSP` language expressions ch ! v and ch ?,
  - indicates the notions of *communication* and *synchronisation*.
- Other than this we shall not cover these crucial notion related to *parallelism*.

8.7. Summary and Discussion of Perdurants

- The most significant contribution of this section has been to show that
  - for every domain description
  - there exists a normal form behaviour —
  - here expressed in terms of a CSP process expression.

8.7.1. Summary

- We have proposed to analyse perdurant entities into actions, events and behaviours – all based on notions of state and time.
- We have suggested modelling and abstracting these notions in terms of functions with signatures and pre-/post-conditions.
- We have shown how to model behaviours in terms of CSP (communicating sequential processes).
- It is in modelling function signatures and behaviours that we justify the endurant entity notions of parts, unique identifiers, mereology and shared attributes.

8.7.2. Discussion

- The analysis of perdurants into actions, events and behaviours represents a choice.
- We suggest skeptical readers to come forward with other choices.

9. Closing

- Domain models abstract some reality.
- They do not pretend to capture all of it.

9.1. What Have We Achieved ?

- A step-wise *method*,
  - its *principles*,
  - *techniques*, and
  - a series of *languages*
- for the rigorous development of domain models has been presented.

- A seemingly large number of domain concepts has been established:
  - *entities*,
  - *endurants* and *perdurants*,
  - *discrete* and *continuous* endurants,
  - *structure, part, component* and *material* endurants,
  - *living species, plants, animals, humans* and *artifacts*,
  - *unique identifiers, mereology* and *attributes*.

It is shown

- how CSP *channels*
  can be calculated from endurant mereologies, and
- how the form of *behaviour arguments*
  can be calculated from respective attribute categorisations.

The domain concepts outlined above

- form a *domain ontology*
- that applies to a wide variety of domains.

The Transcendental Deduction:

- A concept of *transcendental deduction* has been introduced.
  - It is used to justify the interpretation
  - of *endurant parts*
  - as *perdurant behaviours* – à la CSP.
- The interpretation of *endurant parts* as *perdurant behaviours*
  - represents a *transcendental deduction* –
  - and must, somehow, be rationally justified.
  - the justification is here seen as exactly that:
  - a *transcendental deduction*.

- We claim that when, as an example, programmers, in thinking about or in explaining their code, anthropomorphically[56], say that *"the program does so and so"* they 'perform' and transcendental deduction.
- We refer to the forthcoming [**?**, Philosophical Issues in Domain Modeling].
- • This concept should be studied further: *Transcendental Deduction in Computing Science*.

[56] Anthropomorphism is the attribution of human traits, emotions, or intentions to non-human entities.

Living Species:

- The concept of *living species* has been introduced,
  - but it has not been “sufficiently” studied,
  - that is, we have, in Sect. on Slide 218, hinted at a number of ‘living species’ notions:
  - *causality of purpose* et cetera,
  - but no hints has been given as to the kind of attributes
  - that *living species*, especially *humans* give rise to.
- • This concept should be studied further: *Attributes of Living Species in Computing Science*.

Intentional "Pull":

- A new concept of *intentional "pull"* has been introduced.
  - It applies, in the form of attributes, to humans and artifacts.
  - It "corresponds", in a way, to *gravitational pull*;
  - that concept invites further study.
- The pair of gravitational pull and intentional "pull"
  - appears to lie behind the determination
  - of the mereologies of parts;
  - that possibility invites further study.
- • This concept should be studied further: *Intentional "Pull" in Computing Science*.

What Can Be Described ?

- When you read the texts that explain when
  - phenomena can be considered entities,
  - entities can be considered endurants or perdurants,
  - endurants can be considered discrete or continuous,
  - discrete endurants can be considered structures, parts or components, et cetera,

- then you probably,
  - expecting to read a technical/scientific paper,
  - realise that those explanations are not precise in the sense
  - of such papers.
- Many of our definitions are taken
  - from [**?**, The Oxford Shorter English Dictionary] and
  - from the Internet based
    [**?**, The Stanford Encyclopedia of Philosophy].

- In technical/scientific papers definitions are expected
  - to be precise,
  - but can be that only if the definer has set up, beforehand,
  - or the reported work is based on
  - a precise, in our case mathematical framework.
  - That can not be done here.
  - There is no, a priori given, model
    of the domains we are interested in.

- This raises the more general question, such as we see it:
  - *"which are the absolutely necessary and unavoidable bases for describing the world ?"*
  - This is a question of philosophy.
  - We shall not develop the reasoning here.

- Some other issues are to be further studied.
  - (i) When to use *physical mereologies* and when to apply *conceptual mereologies*, cf. final paragraph of Sect. on Slide 193.
  - (ii) How do we know that the categorisation into unique identification, mereology and attributes embodies all internal qualities; could there be a fourth, etc. ?
  - (iii) Is *intent* an attribute, or does it "belong" to a fourth internal quality category, or a fifth ?
  - (iv) It seems that most of what we first thought off as natural parts really are materials: geographic land masses, etc. – subject, still, to the laws of physics: geo-physics.

- • We refer to the forthcoming study [?, Philosophical Issues in Domain Modeling] based on [?, ?, ?, ?].

A Conjecture:

- It could be interesting to study
  - *under what circumstances,*
  - *including for which kind of behaviours,*
  - *we can postulate the following:*

Conjecture: Parts $\cong$ Behaviours

- To every part there is a behaviour,
- and
- to every suitably expressed behaviour there is a part.

We shall leave this study to the reader !

The Contribution:

- In summary we have shown that the domain analysis & description calculi
  - form a sound, consistent and complete approach to domain modelling, and

— that this approach takes its “resting point” in Kai Sørlander’s Philosophy.

9.2. The Four Languages of Domain Analysis & Description

- Usually mathematics, in many of its shades and forms
  - are deployed in *describing* properties of nature,
  - as when pursuing physics,
- Usually the formal specification languages of *computer & computing science*
  - have a precise semantics and a consistent proof system.
  - To have these properties those languages must deal with *computable objects*.
  - *Domains are not computable.*

- So we revert, in a sense, to mathematics as our specification language.
  - Instead of the usual, i.e., the classical style of mathematics,
  - we "couch" the mathematics in a style close to RSL [**?**, **?**].
  - We shall refer to this language as $\mathtt{RSL}^+$.
- Main features of $\mathtt{RSL}^+$ evolves in this paper, mainly in Sect. .

- Here we shall make it clear that we need three languages:
  - (i) an **analysis language**,
  - (ii) a **description language**, i.e., $\mathtt{RSL}^{+}$, and
  - (iii) the language of explaining domain analysis & description,
- (iv) in modelling "the fourth" language,
  - the domain,
  - its syntax
  - and some abstract semantics.

9.2.1. The Analysis Language:

- Use of the *analysis language* is not written down.
- It consists of a number of single, usually `is_` or `has_`, prefixed *domain analysis prompt* and *domain description prompt* names.
- The **domain analysis prompts** are:

The Analysis Prompts

a. `is_ entity`, 7
b. `is_ endurant`, 8
c. `is_ perdurant`, 8
d. `is_ discrete`, 8
e. `is_ continuous`, 9
f. `is_ physical_ part`, 9
g. `is_ living_ species`, 9
h. `is_ structure`, 10
i. `is_ part`, 11
j. `is_ atomic`, 11
k. `is_ composite`, 11
l. `is_ living_ species`, 12
m. `is_ plant`, 12
n. `is_ animal`, 12
o. `is_ human`, 13
p. `has_ components`, 13
q. `has_ materials`, 13
r. `is_ artifact`, 14
s. `observe_ endurant_ sorts`, 14
t. `has_ concrete_ type`, 17
u. `has_ mereology`, 21
v. `attribute_ types`, 24

- *They apply to phenomena in the domain,* that is, to *"the world out there"!*
  - Except for `observe_endurants` and `attribute types` these queries result in truth values;
  - `observe_endurants` results in the *domain scientist cum engineer* noting down, in memory or in typed form, suggestive names [of endurant sorts]; and
  - `attribute_types` results in suggestive names [of attribute types].

- The truth-valued queries directs, as we shall see, the *domain scientist cum engineer* to either further analysis or to "issue" some *domain description prompt*s.
- The 'name'-valued queries help the human analyser to formulate the result of **domain description prompts:**

The Description Prompts

[1] `observe_ endurant_ sorts`, 15
[2] `observe_ part_ type`, 17
[3] `observe_ component_ sorts`, 18
[4] `observe_ material_ sorts`, 19
[5] `observe_ unique_ identifier`, 20
[6] `observe_ mereology`, 22
[7] `observe_ attributes`, 24

- Again *they apply to phenomena in the domain,* that is, to *"the world out there"* !
- In this case they result in RSL$^+$Text !

9.2.2. The Description Language:

- The **description language** is $\mathtt{RSL}^+$.
- It is a basically applicative subset of RSL [?, ?],
  - that is: no assignable variables.
  - Also we omit RSL's elaborate *scheme, class, object* notions.

The Description Language Primitives

- *Structures, Parts, Components and Materials:*
  - **obs**_E, dfn. 1, [o] pg. 144
  - **obs**_T: P, dfn. 2, [t_2] pg. 152
- *Part and Component Unique Identifiers:*
  - **uid**_P, dfn. 5, [u] pg. 175
- *Part Mereologies:*
  - **mereo**_P, dfn. 6, [m] pg. 187
- *Part and Material Attributes:*
  - **attr**_A_i, dfn. 7, [a] pg. 202

- We refer, generally, to all these functions as observer functions.
- They are defined by the analyser cum describer when "applying" description prompts.
- That is, they should be considered user-defined.
- In our examples we use the non-bold-faced observer function names.

9.2.3. The Language of Explaining Domain Analysis & Description:

- In explaining the *analysis & description prompts* we use a natural language which contains terms and phrases typical of
  - the technical language of *computer & computing science*, and
  - the language of *philosophy*, more specifically *epistemology* and *ontology*.
- The reason for the former should be obvious.

- The reason for the latter is given as follows:
  - We are, on one hand, dealing with real, actual segments of domains characterised by their basis in nature, in economics, in technologies, etc., that is, in informal "worlds", and,
  - on the other hand, we aim at a formal understanding of those "worlds".
- There is, in other words, the task of
  - explaining how we observe those "worlds",
  - and that is what brings us close to some issues well-discussed in *philosophy*.

9.2.4. The Language of Domains:

- We consider a domain through the *semiotic looking glass* of
  - its *syntax* and
  - its *semantics*;
  - we shall not consider here its possible *pragmatics*.
- By *"its syntax"* we shall mean
  - the form and "contents",
    - i.e., the *external* and
    - *internal qualities*
  - of the *endurant*s of the domain,
  - i.e., those *entities* that endure.

- By *"its semantics"* we shall, by a *transcendental deduction*, mean the *perdurants*:
  - the *actions*,
  - the *events*, and
  - the *behaviours*
- that center on the the endurants and
- that otherwise characterise the domain.

9.2.5. An Analysis & Description Process:

Program Schema: A Domain Analysis & Description Process, Part I/II

```
type
   V = Part_VAL | Komp_VAL | Mat_VAL
variable
   new:V-set := {uod:UoD} ,
   gen:V-set := {} ,
   txt:Text := {}
value
   discover_sorts: Unit → Unit
   discover_sorts() ≡
       while new ≠ {} do
          let v:V • v ∈ new in
          new := new \ {v} ‖ gen := gen ∪ {v} ;
          is_part(v) →
              ( is_atomic(v) → skip ,
                is_composite(v) →
                   let {e1:E1,e:E2,...,en:En} = observe_endurants(v) in
                   new := new ∪ {e1,e,...,en} ; txt := txt ∪ observe_endurant_sorts(e) end ,
                has_concrete_type(v) →
                   let {s1,s2,...,sm} = new_sort_values(v) in
                   new := new ∪ {s1,s2,...,sm} ; txt := txt ∪ observe_part_type(v) end ) ,
          has_components(v) → let {k1:K1,k2:K2,...,kn:Kn} = observe_components(v) in
                   new := new ∪ {k1,k2,...,kn} ; txt := txt ∪ observe_component_sorts(v) end ,
          has_materials(v) → txt := txt ∪ observe_material_sorts(v) ,
          is_structure(v) → ... EXERCISE FOR THE READER !
          end
       end
```

Program Schema: A Domain Analysis & Description Process, Part II/II

```
discover_uids: Unit → Unit
discover_uids() ≡
    for ∀ v:(PVAL|KVAL) • v ∈ gen
    do txt := txt ∪ observe_unique_identifier(v) end
discover_mereologies: Unit → Unit
discover_mereologies() ≡
    for ∀ v:PVAL • v ∈ gen
    do txt := txt ∪ observe_mereology(v) end
discover_attributes: Unit → Unit
discover_attributes() ≡
    for ∀ v:(PVAL|MVAL) • v ∈ gen
    do txt := txt ∪ observe_attributes(v) end

analysis+description: Unit → Unit
analysis+description() ≡
    discover_sorts(); discover_uids(); discover_mereologies(); discover_attributes()
```

9.3. Relation to Other Formal Specification Languages

- In this contribution we have based the analysis and description calculi and the specification texts emanating as domain descriptions on RSL [?].
- There are other formal specification languages:
  - **Alloy** [?],
  - **B** (etc.) [?],
  - **CafeObj** [?],
  - **CASL** [?],
  - **VDM** [?, ?, ?],
  - **Z** [?],

  to mention a few.

- Two conditions appear to apply for any of these other formal specification languages to become a basis for analysis and description calculi similar to the ones put forward in the current paper:
  - (i) it must be possible, as in RSL, to define and express sorts, i.e., *further undefined types*, and
  - (ii) it must be possible, as with RSL's "built-in" **CSP** [?], in some form or another, to define and express concurrency.
- Insofar as these and other formal languages can satisfy these two conditions, they can certainly also be the basis for domain analysis & description.

- We do not consider **Coq** [?, ?, ?][57], **CSP** [?], **The Duration Calculus** [?] nor **TLA+** [?] as candidates for expressing full-fledged domain descriptions.
- Some of these formal specification languages, like **Coq**, are very specifically oriented towards proofs (of properties of specifications).
- Some, like **The Duration Calculus** and **CSP**, go very well in hand with other formal specification languages like **VDM, RAISE**[58] and **Z**.

[57] http://doi.org/10.5281/zenodo.1028037
[58] A variant of **CSP** is thus "embedded" in **RSL**

- It seems, common to these languages, that, taken in isolation, they can be successfully used for the development and proofs of properties of algorithms and code for, for example safety-critical and embedded systems.
- But our choice (of not considering) is not a "hard nailed" one !
- Also less formal, usually computable, languages, like **Scala** [https://www.scala-lang.org/] or **Python** [https:/www.python.org/], can, if they satisfy criteria (i-ii), serve similarly.

- We refer, for a more general discussion – of issues related to the choice of other formal language being the basis for domain analysis & description – to [?, 40 Years of Formal Methods — 10 Obstacles and 3 Possibilities] for a general discussion that touches upon the issue of formal, or near-formal, specification languages.

9.4. Two Frequently Asked Questions

- *How much of a* DOMAIN *must or should we* ANALYSE & DESCRIBE *?*
  - When this question is raised, after a talk of mine over the subject, and by a colleague researcher & scientist I usually reply:
  - *As large a domain as possible !*
  - This reply is often met by this *comment* (from the audience) *Oh ! No, that is not reasonable !*

- To me that comment shows either or both of:
  - the questioner was not asking as a researcher/scientist, but as an engineer. Yes, an engineer needs only analyse & describe up to and slightly beyond the "border" of the domain-of-interest for a current software development – but
  - a researcher cum scientist is, of course, interested not only in a possible requirements engineering phase beyond domain engineering, but is also curious about the larger context of the domain, in possibly establishing a proper domain theory, etc.

- *How, then, should a domain engineer pursue* DOMAIN MODELLING ?
- My answer assumes a "state-of-affairs" of domain science & engineering
  - in which domain modelling is an established subject, i.e.,
    - where the domain analysis & description topic, i.e., its methodology, is taught,
    - where there are "text-book" examples from relevant fields –
    - that the domain engineers can rely on,
    - and in whose terminology they can communicate with one another;
    - that is, there is an acknowledged *body of knowledge*.

- My answer is therefore:
  - the domain engineer, referring to the relevant *body of knowledge*,
  - develops a domain model
    that covers the domain
    and the context on which the software is to function,
  - just, perhaps covering a little bit more of the context,
  - than possibly necessary — just to be sure.

- Until such a “state-of-affairs” is reached
  - the domain model developer has to act both as a
    - domain scientist and as a
    - domain engineer,
  - researching and developing models
  - for rather larger domains
  - than perhaps necessary
  - while contributing also to
    the **domain science & engineering body of knowledge**.

9.5. On How to Pursue Domain Science & Engineering

- We set up a dogma and discuss a ramification.
  - One thing is the doctrine, the method for domain analysis & description outlined in this paper.
  - Another thing is its practice.
  - I find myself, when experimentally pursuing the modelling of domains, as, for example, reported in [?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?], **that I am often not following the doctrine !**

- That is:
  - (i) in not first, carefully, exploring parts, components and materials, the external properties,
  - (ii) in not then, again carefully settling issues of unique identifiers,
  - (iii) then, carefully, the issues of mereology,
  - (iv) followed by careful consideration of attributes,

  then the transcendental deduction of behaviours from parts;

- (v) carefully establishing channels:
  - (v.i) their message types, and
  - (v.ii) declarations,
- (vi) followed by the careful consideration of behaviour signatures, systematically, one for each transcendentally deduced part,
- (vii) then the careful definition of each of all the deduced behaviours, and, finally,
- (iix) the definition of the overall system initialisation.

- No, instead I faulter,
  get diverted into exploring *"this & that"* in the domain exploration.
  - And I get stuck.
  - When despairing I realise that
    I must *"slavically"* follow the doctrine.
  - When reverting to the strict adherence of the doctrine,
    I find that I, very quickly, find my way,
    and the domain modelling get's *unstuck* !

- I remarked this situation to a dear friend and colleague.
  His remark stressed what was going on:
  - the **creative** engineer **took possession**,
  - the **exploring**, **sceptic** scientist **entered the picture,**
  - the well-trained engineer **lost ground in the realm of imagination**.
  - But perhaps, in the interest of **innovation etc.**
    it is necessary to be **creative** and **sceptic**
    and **loose ground** – for a while !
- I knew that, but had sort-of-forgotten it !
- The lesson is: *waver between adhering to the method and being innovative, curious – a dreamer* !

9.6. Related Work

- The present lectures is but one in
  a series on the topic of *domain science & engineering*.
  - With these lectures the author expects to have laid a foundation.
  - With the many experimental case studies,
    referenced in Example *Universes of Discourse* Page 50, the author seriously think
    that reasonably convincing arguments are given
    for this *domain science & engineering*.

- We comment on some previous publications:
  - [**?**, **?**] explores additional views on analysing & describing domains, in terms of *domain facets:*
    - *intrinsics*,
    - *support technologies*,
    - *rules & regulations*,
    - *scripts*,
    - *management & organisation,*
    - and *human behaviour*.
- [**?**, **?**] explores relations between Stanisław Leśhnieiski's mereology and ours.
- [**?**, **?**] shows how to rigorously transform domain descriptions into software system requirements prescriptions.

- [**?**] explores relations between the present domain analysis & description approach and issues of *safety critical software design*.
- [**?**] discusses various interpretations of domain models: as bases for
  - demos,
  - simulators,
  - real system monitors and
  - real system monitor & controllers.
- [**?**] is a compendium of reports around the management and engineering of software development based in domain analysis & description. These reports were the result of a year at JAIST: Japan Institute of Science & Technology, Ishikawa, Japan.

9.7. Tony Hoare's Summary on 'Domain Modelling'

- In a 2006 e-mail, in response, undoubtedly to my steadfast – perhaps conceived as stubborn – insistence, on domain engineering,
- Tony Hoare summed up his reaction to domain engineering as follows, and I quote[59]:

[59] E-Mail to Dines Bjørner, July 19, 2006

"There are many unique contributions that can be made by domain modelling.

1 *The models describe all aspects of the real world that are relevant for any good software design in the area.*
They describe possible places to define the system boundary for any particular project.

2 *They make explicit the preconditions about the real world that have to be made in any embedded software design,*
especially one that is going to be formally proved.

3 *They describe the whole range of possible designs for the software,*
and the whole range of technologies available for its realisation.

4 *They provide a framework for a full analysis of requirements,*
which is wholly independent of the technology of implementation.

5 *They enumerate and analyse the decisions that must be taken earlier or later in any design project,*
and identify those that are independent and those that conflict.
Late discovery of feature interactions can be avoided."

- All of these issues were covered in [**?**, Part IV].

9.8. Acknowledgements

- I thank the three reviewers for their thorough reviews, for their many fine observations and suggestions, and for pointing out confusing texts.
- I also thank colleagues in Austria, China, Germany, France, Norway, Singapore, Sweden and the United States:
  - Yamine Ait Ameur,
  - Dominique Méry,
  - Andreas Harmfeldt,
  - Magne Haveraaen,
  - Klaus Havelund,
  - Otthein Herzog,
  - Steve McKeever
  - Jens Knoop,
  - Hans Langmaack,
  - Chin Wei Ngan, and
  - Min Zhang.

- I appreciate very much their comments on recent papers, their inviting me, over recent years, to lecture in their departments where their students have acted as sounding boards also for the case studies, leading to a number of
  - clarifications,
  - simplifications and
  - solidifications

  of the *domain analysis & description* method of [**?**]
  now reported in the present paper.

- I thank Wang ShuLin
  for incisive questions – answers to which are found,
  in particular, in Sect. of this paper.
- And I thank Ole N. Oest for some remarks
  that lead to my remarks on Slide 429.

10. The Example Concluded

Example 37: Unique Identifiers, Attributes and Behaviours

10.1. Unique Identifier Concepts

- We define a few concepts related to unique identification.

Extract Parts from Their Unique Identifiers: . . . .

91 *From the unique identifier of a part we can retrieve, ℘, the part having that identifier.*

type
91 P = H | L | BC | B | A
value
91 ℘: H_UI→H | L_UI→L | BC_UI→BC | B_UI→B | A_UI→A
91 ℘(ui) ≡ **let** p:(H|L|BC|B|A)•p∈ps∧uid_P(p)=ui **in** p **end**

Unique Identifier Constants . . . .

We can calculate:

92 *the set, $h_{ui}s$, of unique hub identifiers;*

93 *the set, $l_{ui}s$, of unique link identifiers;*

94 *the map, $hl_{ui}m$, from unique hub identifiers to the set of unique link iidentifiers of the links connected to the zero, one or more identified hubs,*

95 *the map, $lh_{ui}m$, from unique link identifiers to the set of unique hub iidentifiers of the two hubs connected to the identified link;*

96 *the set, $r_{ui}s$, of all unique hub and link, i.e., road identifiers;*

97 *the set, $bc_{ui}s$, of unique bus company identifiers;*

98 *the set, $b_{ui}s$, of unique bus identifiers;*

99 *the set, $a_{ui}s$, of unique private automobile identifiers;*

100 *the set, $v_{ui}s$, of unique bus and automobile, i.e., vehicle identifiers;*

101 *the map, $bcb_{ui}m$, from unique bus company identifiers to the set of its unique bus identifiers; and*

102 *the (bijective) map, $bbc_{ui}bm$, from unique bus identifiers to their unique bus company identifiers.*

92 $h_{ui}s$:H_UI-**set** $\equiv$ {uid_H(h)|h:H•h $\in$ hs}

93 $l_{ui}s$:L_UI-**set** ≡ {uid_L(l)|l:L•l ∈ ls}
96 $r_{ui}s$:R_UI-**set** ≡ $h_{ui}s$∪$l_{ui}s$
94 $hl_{ui}m$:(H_UI $\overrightarrow{m}$ L_UI-**set**) ≡
94 [h_ui↦luis|h_ui:H_UI,luis:L_UI-**set**•h_ui∈$h_{ui}s$
94 ∧(_,luis,_)=mereo_H(η(h_ui))] [cf. Item 24]
95 $lh_{ui}m$:(L+UI $\overrightarrow{m}$ H_UI-**set**) ≡
95 [l_ui↦huis [cf. Item 25]
95 | h_ui:L_UI,huis:H_UI-**set** • l_ui∈$l_{ui}s$
95 ∧ (_,huis,_)=mereo_L(η(l_ui))]
97 $bc_{ui}s$:BC_UI-**set** ≡ {uid_BC(bc)|bc:BC•bc ∈ bcs}
98 $b_{ui}s$:B_UI-**set** ≡ ∪{uid_B(b)|b:B•b ∈ bs}
99 $a_{ui}s$:A_UI-**set** ≡ {uid_A(a)|a:A•a ∈ as}
100 $v_{ui}s$:V_UI-**set** ≡ $b_{ui}s$ ∪ $a_{ui}s$
101 $bcb_{ui}m$:(BC_UI $\overrightarrow{m}$ B_UI-**set**) ≡
101 [bc_ui ↦ buis
101 | bc_ui:BC_UI, bc:BC •
101 bc∈bcs ∧ bc_ui=uid_BC(bc)
101 ∧ (_,_,buis)=mereo_BC(bc)]
102 $bbc_{ui}bm$:(B_UI $\overrightarrow{m}$ BC_UI) ≡
102 [b_ui ↦ bc_ui
102 | b_ui:B_UI,bc_ui:BC_ui •
102 bc_ui=**dom**$bcb_{ui}m$∧b_ui∈$bcb_{ui}m$(bc_ui)]

Uniqueness of Part Identifiers: . . . .

- We refer to Sect. Slide 239.
- We must express the following axioms:

103 All hub identifiers are distinct.

104 All link identifiers are distinct.

105 All bus company identifiers are distinct.

106 All bus identifiers are distinct.

107 All private automobile identifiers are distinct.

108 All part identifiers are distinct.

103 $\textbf{card}\, hs = \textbf{card}\, h_{ui}s$

104 $\textbf{card}\, ls = \textbf{card}\, l_{ui}s$

105 $\textbf{card}\, bcs = \textbf{card}\, bc_{ui}s$

106 $\textbf{card}\, bs = \textbf{card}\, b_{ui}s$

107 $\textbf{card}\, as = \textbf{card}\, a_{ui}s$

108 $\textbf{card}\,\{h_{ui}s \cup l_{ui}s \cup bc_{ui}s \cup b_{ui}s \cup a_{ui}s\}$
108 $\quad = \textbf{card}\,h_{ui}s + \textbf{card}\,l_{ui}s + \textbf{card}\,bc_{ui}s + \textbf{card}\,b_{ui}s + \textbf{card}\,a_{ui}s$

10.2. Further Transport System Attributes

Links: We show just a few attributes. ....

109 There is a link state. It is a set of pairs, $(\mathsf{h}_f,\mathsf{h}_t)$, of distinct hub identifiers, where these hub identifiers are in the mereology of the link. The meaning of a link state in which $(\mathsf{h}_f,\mathsf{h}_t)$ is an element is that the link is open, **"green"**, for traffic $f$rom hub h_f to hub h_t. Link states can have either 0, 1 or 2 elements.

110 There is a link state space. It is a set of link states. The meaning of the link state space is that its states are all those the which the link can attain. The current link state must be in its state space. If a link state space is empty then the link is (permanently) closed. If it has one element then it is a one-way link. If a one-way link, l, is

imminent on a hub whose mereology designates that link, then the link is a "trap", i.e., a "blind cul-de-sac".

111 Since we can think rationally about it, it can be described, hence it can model, as an attribute of links a history of its traffic: the recording, per unique bus and automobile identifier, of the time ordered positions along the link (from one hub to the next) of these vehicles.

112 The hub identifiers of link states must be in the set, $h_{ui}s$, of the road net's hub identifiers.

10. The Example Concluded 10.2. Further Transport System Attributes

type
109 LΣ = H_UI-**set** [programmable, Df.8 Pg.209]
axiom
109 ∀ lσ:LΣ·**card** lσ=2
109 ∀ l:L • obs_LΣ(l) ∈ obs_LΩ(l)
type
110 LΩ = LΣ-**set** [static, Df.1 Pg.206]
111 L_Traffic [programmable, Df.8 Pg.209]
111 L_Traffic = (A_UI|B_UI) $\overrightarrow{m}$ (𝕋×(H_UI×Frac×H_UI))*

```
111  Frac = Real, axiom frac:Fract • 0<frac<1
value
109  attr_LΣ: L → LΣ
110  attr_LΩ: L → LΩ
111  attr_L_Traffic: : → L_Traffic
axiom
111  ∀ lt:L_Traffic,ui:(A_UI|B_UI)•ui ∈ dom ht
111        ⇒ time_ordered(ht(ui))

112  ∀ l:L • l ∈ ls ⇒
112     let lσ = attr_LΣ(l) in
112     ∀ (h_ui i,h_ui i′):(H_UI×K_UI) •
112           (h_ui i,h_ui i′) ∈ lσ ⇒ {h_ui_i, h′_ui_i} ⊆ h_ui s end
```


Bus Companies: ....

- *Bus companies operate a number of lines that service passenger transport along routes of the road net. Each line being serviced by a number of buses.*

113 *Bus companies create, maintain, revise and distribute*

[to the public (not modeled here), and to buses]

bus time tables, not further defined.

type

113 BusTimTbl [programmable, Df.8 Pg.209]

value

113 attr_BusTimTbl: BC → BusTimTbl

- *There are two notions of time at play here:*
  - *the indefinite "real" or "actual" time; and*
  - *the definite calendar, hour, minute and second time designation occurring in some textual form in, e.g., time tables.*

Buses: *We show just a few attributes:* . . . .

114 *Buses run routes, according to their line number, ln:LN, in the*

115 *bus time table, btt:BusTimTbl obtained from their bus company, and and keep, as inert attributes, their segment of that time table.*

116 *Buses occupy positions on the road net:*

a. *either at a hub identified by some h_ui,*

b. *or on a link, some fraction, f:Fract, down an identified link, l_ui, from one of its identified connecting hubs, fh_ui, in the direction of the other identified hub, th_ui.*

117 *Et cetera.*

type

114 LN [programmable, Df.8 Pg.209]

115 BusTimTbl [inert, Df.3 Pg.207]

116 BPos == atHub | onLink [programmable, Df.8 Pg.209]

116a. atHub :: h_ui:H_UI

116b. onLink :: fh_ui:H_UI×l_ui:L_UI×frac:Fract×th_ui:H_UI

116b. Fract = **Real**, **axiom** frac:Fract · 0<frac<1
117 ...
value
115 attr_BusTimTbl: B → BusTimTbl
116 attr_BPos: B → BPos

Private Automobiles: *We show just a few attributes:* . . . .

- *We illustrate but a few attributes:*

118 Automobiles have static number plate registration numbers.

119 Automobiles have dynamic positions on the road net:

[116a.] either at a hub identified by some h_ui,

[116b.] or on a link, some fraction, frac:Fract down an identified link, l_ui, from one of its identified connecting hubs, fh_ui, in the direction of the other identified hub, th_ui.

10. The Example Concluded 10.2. Further Transport System Attributes

type
118 RegNo [static, Df.1 Pg.206]
119 APos == atHub | onLink [programmable, Df.8 Pg.209]
116a. atHub :: h_ui:H_UI
116b. onLink :: fh_ui:H_UI × l_ui:L_UI × frac:Fract × th_ui:H_UI
116b. Fract = **Real**, **axiom** frac:Fract · 0<frac<1
value
118 attr_RegNo: A → RegNo
119 attr_APos: A → APos

- *Obvious attributes that are not illustrated are those of*
  - *velocity and acceleration,*
  - *forward or backward movement,*
  - *turning right, left or going straight,*
  - *etc.*

- *The acceleration, deceleration, even velocity,* or *turning right, turning left, moving straight,* or *forward* or *backward are seen as command actions.*
  - *As such they denote actions by the automobile —*
  - *such as pressing the accelerator,* or *lifting accelerator pressure* or *braking,* or *turning the wheel in one direction* or *another, etc.*
  - *As actions they have a kind of counterpart in the velocity, the acceleration, etc. attributes.*

10. The Example Concluded 10.2. Further Transport System Attributes

10.2.0.1 Discussion:

- Observe that bus companies each have their own distinct *bus time table*, and that these are modeled as *programmable*, Item 113 on Slide 444, Page 444.
- Observe then that buses each have their own distinct *bus time table*,

and that these are model-led as *inert*, Item 115 on Slide 445, Page 445.

- In Items 129–130b. Slide 457 we shall see how the buses communicate with their respective bus companies in order for the buses to obtain

the *programmed* bus time tables "in lieu" of their *inert* one !

- In Items 33 Slide 212 and 111 Slide 443, we illustrated an aspect of domain analysis & description that may seem, and at least some decades ago would have seemed, strange: namely that if we can think, hence speak, about it, then we can model it "as a fact" in the domain. The case in point is that we include among hub and link attributes their histories of the timed whereabouts of buses and automobiles.[60]

10.3. Behaviours

Automobile Behaviour (on a link) . . . .

120 We abstract automobile behaviour on a Link.

a. Internally non-deterministically, either
   i the automobile remains, “idling”, i.e., not moving, on the link,
   ii however, first informing the link of its position,

b. or
   i **if** if the automobile’s position on the link *has not yet reached the hub*, **then**
      A then the automobile moves an arbitrary small, positive **Real**-valued *increment* along the link
      B informing the hub of this,

      C while resuming being an automobile ate the new position, or
   ii **else**,
      A while obtaining a “next link” from the mereology of the hub (where that next link could very well be the same as the link

the vehicle is about to leave),

B the vehicle informs both the link and the imminent hub that it is now at that hub, identified by th_ui,

C whereupon the vehicle resumes the vehicle behaviour positioned at that hub;

c. or

d. the vehicle "disappears — off the radar" !


```
120        automobile_aui(a_ui,({},ruis,{}),rno)
120                      (vp:onL(fh_ui,l_ui,f,th_ui)) ≡
120(a.)ii    (ba_r_ch[thui,aui]!atH(lui,thui,nxt_lui) ;
120(a.)i      automobile_aui(a_ui,({},ruis,{}),rno)(vp))
120b.        ⌈⌉
120(b.)i     (if not_yet_at_hub(f)
120(b.)i        then
120(b.)iA         (let incr = increment(f) in
84                 let onl = (tl_ui,h_ui,incr,th_ui) in
120(b.)iB           ba−r_ch[l_ui,a_ui] ! onL(onl) ;
120(b.)iC           automobile_aui(a_ui,({},ruis,{}),rno)
120(b.)iC                          (onL(onl))
```

```
120(b.)i        end end)
120(b.)ii     else
120(b.)iiA      (let nxt_lui:L_UI•nxt_lui ∈ mereo_H(℘(th_ui)) in
120(b.)iiB       ba_r_ch[thui,aui]!atH(l_ui,th_ui,nxt_lui) ;
120(b.)iiC       automobile_aui(a_ui,({},ruis,{}),rno)
120(b.)iiC                (atH(l_ui,th_ui,nxt_lui)) end)
120(b.)i      end)
120c.      ⌈⌉
120d.       stop
120(b.)iA  increment: Fract → Fract
```


Hub Behaviour . . . We model the hub behaviour vis-a-vis vehicles: buses and automobiles.

121 The hub behaviour

a. non-deterministically, externally offers

b. to accept timed vehicle positions —

c. which will be at the hub, from some vehicle, v_ui.

d. The timed vehicle hub position is appended to the front of that vehicle's entry in the hub's traffic table;

e. whereupon the hub proceeds as a hub behaviour with the updated hub traffic table.

f. The hub behaviour offers to accept from any vehicle.

g. A **post** condition expresses what is really a **proof obligation**: that the hub traffic, ht′ satisfies the **axiom** of the endurant hub traffic attribute Item 33 Slide 212.

value

121 hub$_{h_{ui}}$(h_ui,(,(luis,vuis)),hω)(hσ,ht) $\equiv$

121a. $\square$

121b. { **let** m = ba_r_ch[h_ui,v_ui] ? **in**

121c. **assert:** m=(_,atHub(_,h_ui,_))

121d. **let** ht′ = ht † [h_ui ↦ ⟨m⟩⌢ht(h_ui)] **in**

121e. hub$_{h_{ui}}$(h_ui,(,(luis,vuis)),(hω))(hσ,ht′)

```
121f.             | v_ui:V_UI•v_ui∈vuis end end }
121g.      post: ∀ v_ui:V_UI•v_ui ∈ dom ht′⇒time_ordered(ht′(v_ui))
```

Link Behaviour ....

122 The link behaviour non-deterministically, externally offers

123 to accept timed vehicle positions —

124 which will be on the link, from some vehicle, v_ui.

125 The timed vehicle link position is appended to the front of that vehicle's entry in the link's traffic table;

126 whereupon the link proceeds as a link behaviour with the updated link traffic table.

127 The link behaviour offers to accept from any vehicle.

128 A **post** condition expresses what is really a **proof obligation**: that

the link traffic, lt′ satisfies the **axiom** of the endurant link traffic attribute Item 111 Slide 443.

```
122  link_{l_ui}(l_ui,(_,(huis,vuis),_),lω)(lσ,lt) ≡
122     □
123       { let m = ba_r_ch[l_ui,v_ui] ? in
124              assert: m=(_,onLink(_,l_ui,_,_))
125          let lt′ = lt † [l_ui ↦ ⟨m⟩⌢lt(l_ui)] in
126          link_{l_ui}(l_ui,(huis,vuis),hω)(hσ,lt′)
127        | v_ui:V_UI·v_ui∈vuis end end }
128     post: ∀ v_ui:V_UI·v_ui ∈ dom lt′⇒time_ordered(lt′(v_ui))
```

10. The Example Concluded 10.3. Behaviours

Bus Company Behaviour . . . .

- We model bus companies very rudimentary.
  - Bus companies keep a fleet of buses.

- Bus companies create, maintain, distribute bus time tables.
- Bus companies deploy their buses to honor obligations of their bus time tables.
- We shall basically only model the distribution of bus time tables to buses.
- We shall not cover other aspects of bus company management, etc.

129 Bus companies non-deterministically, internally, chooses among

a. updating their bus time tables

b. whereupon they resume being bus companies, albeit with a new bus time table;

130 “interleaved” with

a. offering the current time-stamped bus time table to buses which offer willingness to received them

b. whereupon they resume being bus companies with unchanged bus time table.


```
81      bus_company_bc_ui(bcui,(_,buis,_))(btt) ≡
129a.     (let btt′ = update(btt,...) in
129b.      bus_company_bc_ui(bcui,(_,buis,_))(btt′) end )
130     ⌈⌉
130a.     ( ⌈⌉ {bc_b_ch[bc_ui,b_ui] ! btt | b_ui:B_UI•b_ui∈buis
130b.            bus_company_bc_ui(bcui,(_,buis,_))(record_TIME(),btt) } )
```

- We model the interface between buses and their owning companies —
- as well as the interface between buses and the road net,
- the latter by almost "carbon-copying" all elements of the automo-

bile behaviour(s).

131 The bus behaviour chooses to either

a. accept a (latest) time-stamped buss time table from its bus company –

b. where after it resumes being the bus behaviour now with the updated bus time table.

132 or, non-deterministically, internally,

a. based on the bus position

i if it is at a hub then it behaves as prescribed in the case of automobiles at a hub,

ii else, it is on a link, and then it behaves as prescribed in the case of automobiles on a link.

10. The Example Concluded 10.3. Behaviours

131 $\mathsf{bus}_{b_{ui}}$(b_ui,(_,(bc_ui,ruis),_))(ln,btt,bpos) ≡

131a. (**let** btt′ = b_bc_ch[b_ui,bc_ui] ? **in**
131b. bus$_{b_{ui}}$(b_ui,({},(bc_ui,ruis),{}))(ln,btt′,bpos) **end**)
132 ⊓
132a. (**case** bpos **of**
132(a.)i atH(fl_ui,h_ui,tl_ui) →
132(a.)i atH_bus$_{b_{ui}}$(b_ui,(_,(bc_ui,ruis),_))(ln,btt,bpos),
132(a.)ii aonL(fh_ui,l_ui,f,th_ui) →
132(a.)ii onL_bus$_{b_{ui}}$(b_ui,(_,(bc_ui,ruis),_))(ln,btt,bpos)
132a. **end**)

Bus Behaviour at a Hub ....

- The atH_bus$_{b_{ui}}$ behaviour definition is a simple transcription of the
  - automobile$_{a_{ui}}$ (atH) behaviour definition:
    - mereology expressions being changed from to ,

- ◦ programmed attributes being changed from atH(fl_ui,h_ui,tl_ui) to (ln,btt,atH(fl_ui,h_ui,tl_ui)),
- ◦ channel references a_ui being replaced by b_ui, and
- ◦ behaviour invocations renamed from automobile$_{a_{ui}}$ to bus$_{b_{ui}}$.

- So formula lines 85–120d. below presents "nothing new" !

```
132(a.)i   atH_bus_b_ui(b_ui,(_,(bc_ui,ruis),_))
132(a.)i                   (ln,btt,atH(fl_ui,h_ui,tl_ui)) ≡
85            (ba_r_ch[b_ui,h_ui] ! (record_TIME(),atH(fl_ui,h_ui,tl_ui));
86             bus_b_ui(b_ui,({},(bc_ui,ruis),{}))(ln,btt,bpos))
131a.          ⊓
87a.           (let ({fh_ui,th_ui},ruis′)=mereo_L(℘(tl_ui)) in
87a.                  assert: fh_ui=h_ui ∧ ruis=ruis′
84             let onl = (tl_ui,h_ui,0,th_ui) in
87b.            (ba_r_ch[b_ui,h_ui] ! (record_TIME(),onL(onl)) ‖
```

```
87b.        ba_r_ch[b_ui,tl_ui] ! (record_TIME(),onL(onl))) ;
87c.       bus_b_ui(b_ui,({},(bc_ui,ruis),{}))
87c.                  (ln,btt,onL(onl)) end end )
120c.      ⊓
120d.      stop
```


Bus Behaviour on a Link . . . .

- The onL_bus$_{b_{ui}}$ behaviour definition is a similar simple transcription of the automobile$_{a_{ui}}$ (onL) behaviour definition.
- So formula lines 85–120d. below presents “nothing new” !

133 – this is the “almost last formula line” !

```
132(a.)ii  onL_bus_b_ui(b_ui,(_,(bc_ui,ruis),_))
132(a.)ii               (ln,btt,bpos:onL(fh_ui,l_ui,f,th_ui)) ≡
85         (ba_r_ch[b_ui,h_ui] ! (record_TIME(),bpos);
86          bus_b_ui(b_ui,({},(bc_ui,ruis),{}))(ln,btt,bpos))
131a.        ⊓
```

```
120(b.)i      (if not_yet_at_hub(f)
120(b.)i        then
120(b.)iA          (let incr = increment(f) in
84                  let onl = (tl_ui,h_ui,incr,th_ui) in
120(b.)iB           ba−r_ch[l_ui,b_ui] ! onL(onl) ;
120(b.)iC           bus_{b_ui}(b_ui,({},(bc_ui,ruis),{}))
120(b.)iC                 (ln,btt,onL(onl))
120(b.)i            end end)
120(b.)ii       else
120(b.)iiA         (let nl_ui:L_UI•nxt_lui∈mereo_H(℘(th_ui)) in
120(b.)iiB          ba_r_ch[thui,b_ui]!atH(l_ui,th_ui,nxt_lui) ;
120(b.)iiC          bus_{b_ui}(b_ui,({},(bc_ui,ruis),{}))
120(b.)iiC                (ln,btt,atH(l_ui,h_ui,nxt_lui))
120(b.)iiA          end)end)
120c.         ⌈⌉
120d.          stop
```

10.4. Example Index

Indexes:

Sorts

Types . . . .

Contents