DOMAIN SCIENCE & ENGINEERING

The TU Wien Lectures, Fall 2022

—

Dines Bjørner

Technical University of Denmark

The Triptych Dogma

In order to *specify* **software**,
we must understand its requirements.

In order to *prescribe* **requirements**
we must understand the **domain**.

So we must **study, analyse** and **describe** domains.

Preface

- These lectures expound a method:
- By a **method** we shall understand
  - a set of **principles** and **procedures**
  - for selecting and applying a set of
  - **techniques** and **tools**
  - in order to achieve an orderly construction
  - of a **solution** to a **problem**.
- By **methodology** we shall understand
  - the *study & application* of one or more methods.

- By a **formal method** we shall understand a method
  - whose decisive *principles* include that of considering its artefacts as *mathematical* quantities;
  - whose decisive *procedures* include those of
  - whose decisive *techniques* include those of
  - whose decisive *tools* include those of one or more **formal languages**

- By a **language** we shall here understand
  a set of strings of characters, i.e., sentences,
  - sentences which are structured according to
    some **syntax**, i.e., **grammar**,
  - are given meaning by some **semantics**,
  - and are used according to some **pragmatics**.
- By a **formal language** we shall here understand a languages
  - whose *syntax* and *semantics* can
    both be expressed **mathematically**
  - and for whose sentences one can
    rationally reason (*argue, prove*) **properties**.

• • •

- In these lectures we shall especially enunciate these:
  - principles,
  - procedures,
  - techniques, and
  - tools.

Lecture 1: Domains

- In this lecture, i.e., the next 45 mins.,
  - I shall survey "all" of the most important
  - aspect of **Domain Analysis & Description**.
- These will all be further explained
  and more aspects will be introduced
  in all the subsequent lectures.

CHAPTER 3. Domains

3.1 Domain Definition

Definition 1 . ***Domain:*** By a *domain* we shall understand

- a *rationally describable* segment of
- a *discrete dynamics* segment of
- a *human assisted reality*, i.e., of the world;
- its *solid or fluid entities*:
  - *natural* ["God-given"] and
  - *artefactual* ["man-made"],
- and its *living species entities*:
- *plants* and *animals* – including, notably, *humans* ■

Example 1 . **Domains:** A few, more-or-less self-explanatory examples:

- **Rivers –** with their natural sources, deltas, tributaries, waterfalls, etc., and their man-made dams, harbours, locks, etc. [19]
- **Road nets –** with street segments and intersections, traffic lights, and automobiles.
- **Pipelines –** with their wells, pipes, valves, pumps, forks, joins and wells [8].
- **Container terminals –** with their container vessels, containers, cranes, trucks, etc. [14] ■

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

3.2 Phenomena and Entities

- **Definition 2 .** *Phenomena:* By a *phenomenon* we shall understand a fact that is observed to exist or happen ■
  - Some phenomena are rationally describable – to a large or full degree – others are not.
- **Definition 3 .** *Entities:* By an *entity* we shall understand a more-or-less rationally describable phenomenon ■
- **Example 2 .** **Phenomena and Entities:** Some, but not necessarily all aspects of a river can be rationally described, hence can be still be considered entities. Similarly, many aspects of a road net can be rationally described, hence will be considered entities ■

3.3 Endurants and Perdurants

3.3.1 Endurants

- **Definition 4 .** *Endurants:* those quantities of domains that we can observe (see and touch), in *space*, as "complete" entities at no matter which point in *time* – "material" entities that persists, endures ■

  Example 3 . Endurants: a street segment [link], a street intersection [hub], an automobile ■

- Domain endurants, when eventually modelled in software, typically become data. Hence the careful analysis of domain endurants is a prerequisite for subsequent careful conception and analyses of data structures for software, including data bases.

3.3.2 Perdurants

- **Definition 5 .** *Perdurants* those quantities of domains for which only a fragment exists, in *space*, if we look at or touch them at any given snapshot in *time* ■

  Example 4 . Perdurant: a moving automobile ■

- Domain perdurants, when eventually modelled in software, typically become processes. Hence the careful analysis of domain perdurants is a prerequisite for subsequent careful conception and analyses of functions (procedures).

3.4 External and Internal Endurant Qualities

3.4.1 External Qualities

Definition 6 . *External qualities:* of endurants of a manifest domain

- are, in a simplifying sense, those we can
  - sea,
  - touch and
  - have spatial extent.
- They, so to speak, take form.

Example 5 . External Qualities:

- The Cartesian[1]
  - of sets of solid atomic street intersections, and
  - of sets of solid atomic street segments, and
  - of sets of solid automobiles

  of a road transport system
- where the
  - Cartesian,
  - sets,
  - atomic, and
  - solid

  reflect external qualities ■

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

3.4.1.1 Discrete or Solid Endurants

Definition 7 . ***Discrete or Solid Endurants:*** By a *solid* [or *discrete*] endurant we shall understand an endurant

- which is separate, individual or distinct in form or concept,
- or, rephrasing: have 'body' [or magnitude] of three-dimensions: length, breadth and depth [32, Vol. II, pg. 2046] ■

Example 6 . Solid Endurants:

- The

  – wells, – valves, – forks, – sinks
  – pipes, – pumps, – joins and

  of pipelines are solids.
- [These units may, however, and usually will, contain fluids, e.g., oil, gas or water] ■

- We shall mostly be analysing and describing solid endurants.
- As we shall see, in the next section,
  - we analyse and describe solid endurants as
  - either parts
  - or living species: animals and humans.
- We shall mostly be concerned with parts.
  - That is, we shall just, as: “in passing”,
  - for sake of completeness,
  - mention living species !

3.4.1.2 Fluids

- **Definition 8 .** *Fluid Endurants:*
  - By a *fluid endurant* we shall understand an endurant which is
    - ⋆ prolonged, without interruption, in an unbroken series or pattern;
    - ⋆ or, rephrasing: a substance (liquid, gas or plasma) having the property of flowing, consisting of particles that move among themselves [32, Vol. I, pg. 774] ■

Example 7 . Fluid Endurants:

- water,
- oil,
- gas,
- compressed air,
- smoke ■

- Fluids are otherwise
  - liquid, or
  - gaseous, or
  - plasmatic, or
  - granular[2], or
  - plant products, i.e., chopped sugar cane, threshed, or otherwise[3],
  - et cetera.
- Fluid endurants will be analysed and described in relation to solid endurants, viz. their "containers".

[2] This is a purely pragmatic decision. "Of course" sand, gravel, soil, etc., are not fluids, but for our modelling purposes it is convenient to "compartmentalise" them as fluids !
[3] See footnote 2.

3.4.1.3 Parts

- **Definition 9 .** *Parts:*
  - The non-living species solids are what we shall call parts ■
- Parts are the “work-horses” of man-made domains.
- That is, we shall mostly be concerned
  with the analysis and description of endurants into parts.

Example 8 . **Parts:** The previous example of solids was also an example of parts ■

- We distinguish between atomic and compound parts.

3.4.1.3.1 Atomic Parts

Definition 10 . *Atomic Part, I:*

- By an *atomic part* we shall understand a part
  - which the domain analyser considers to be indivisible
  - in the sense of not meaningfully,
  - for the purposes of the domain under consideration,
  - that is, to not meaningfully consist of sub-parts ■

3.4.1.3.2 Compound Parts

- We, pragmatically, distinguish between
  - Cartesian-product-, and
  - set-

  oriented parts.
- If Cartesian-oriented, to consist of two or more distinctly sort-named endurants (solids or fluids),
- If set-oriented, to consist of an indefinite number of zero, one or more parts.

Definition 11 . *Compound Part, I:*

- *Compound part*s are those which are
  - either Cartesian-product-
  - or are set-
- oriented parts ■

Example 9 . Compound Parts: A road net consisting of

- a set of hubs, i.e., street intersections or "end-of-streets", and
- a set of links, i.e., street segments (with no contained hubs),

is a Cartesian compound;

- and the sets of hubs and the sets of links

are part set compounds ■

3.4.2 An Aside: An Upper Ontology

- We have been reasonably careful
  - to just introduce and state informal definitions
  - of phenomena and some classes thereof.
- In the next chapter we shall, in a sense, “repeat” coverage of these phenomena.
  - But now in a more analytic manner.
  - Figure 3.1 on the next slide is intended to indicate this.

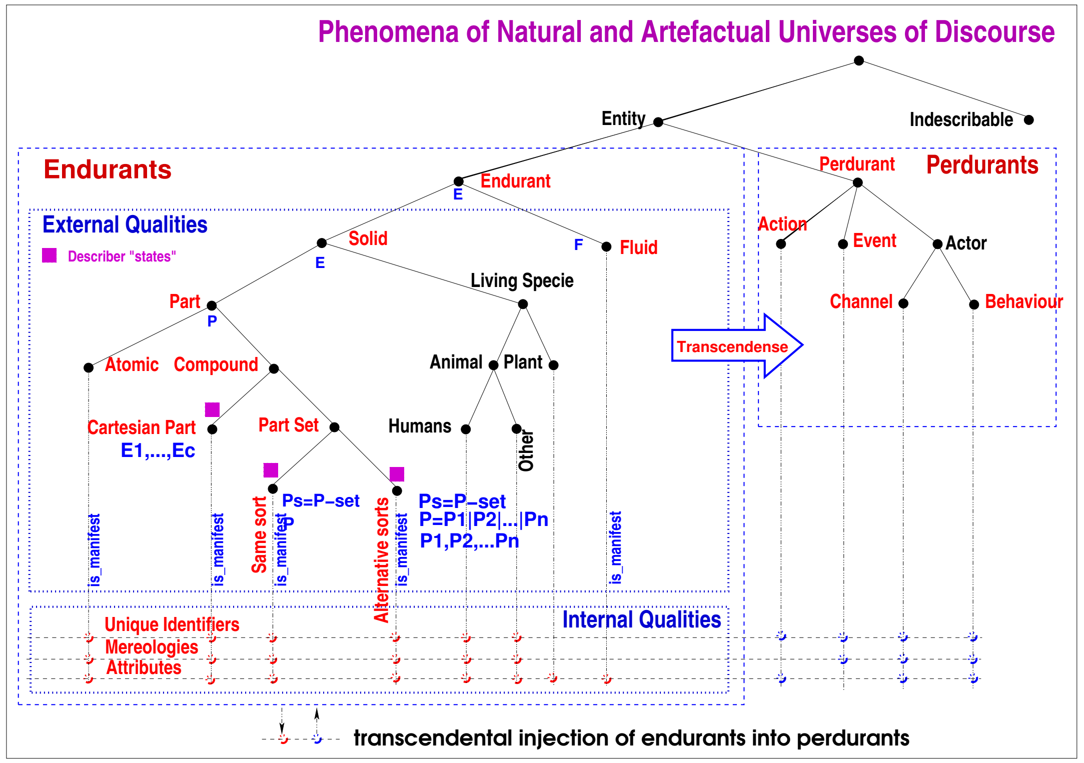

Figure 3.1: Upper Ontology

- So far we have only touched upon
  - the ‘External Qualities’ labeled, dotted-dashed box
  - of the ‘Endurants’ label-led dashed box of Fig. 3.1.
- In Chapter 4 we shall treat external qualities in more depth —
- more systematically: analytically and descriptionally.

3.4.3 Internal quality

Definition 12 . ***Internal qualities:***

- those properties [of endurants]
- that do not occupy *space*
- but can be measured or spoken about ■

Example 10 . **Internal qualities:**

- the unique identity of a part,
- the relation of part to other parts, and
- the endurant attributes such as temperature, length, colour ■

3.4.3.1 Unique identity

- **Definition 13 .** *Unique identity:*
  - an immaterial property
  - that distinguishes two *spatially* distinct solids ■

Example 11 . Unique identities:

- Each hub in a road net is unique identified,
- so is each link
- and automobile ■

3.4.3.2 Mereology

- **Definition 14 .** ***Mereology:*** a theory of [endurant] part-hood relations:
  - of the relations of an [endurant] parts to a whole
  - and the relations of [endurant] parts to [endurant] parts within that whole ■

Example 12 . Mereology:

- that a link is topologically *connected* to exactly two specific hubs,
- that hubs are *connected* to zero, one or more specific links,
- and that links and hubs are *open* to specific subsets of automobiles ■

3.4.3.3 Attribute

Definition 15 . ***Attributes:*** Properties of endurants

- that are not *spatially* observable,
- but can be either physically
- (electronically, chemically, or otherwise)
- measured or can be objectively spoken about ■

Example 13 . Attribute: Links have

- lengths, and,
- at any one time,
- zero, one or more automobiles are occupying the links ■

3.5 Prompts

3.5.1 Analysis Prompts

- **Definition 16 .** *Analysis prompt:*
  - a predicate or a function
  - that may be posed by humans to facets of a domain.
  - Observing the domain the analyser may then
  - act upon the combination of the particular prompt
  - (whether a predicate or a function,
    and then what particular one of these it is)
  - thus “applying” it to a domain phenomena,
  - and yielding, in the minds of the humans,
  - either a truth value or some other form of value ■

3.5.1.1 Analysis Predicate

- **Definition 17 .** *Analysis predicates:*
  - an analysis prompt
  - which yields a truth value ■

Example 14 . Analysis Predicates: General examples are

- can an observable phenomena be rationally described, i.e., an entity,
- is an entity a solid or a fluid.
- is a solid endurant a part or a living species ■

3.5.1.2 Analysis Function

Definition 18 . ***Analysis function:***

- an analysis prompt which yields some RSL-**Text** ■

Example 15 . Analysis Functions: Two examples:

- one yields the endurants of a Cartesian part
  and their respective sort names,
- another yields the set of a parts of a part set
  and their common type ■

3.5.2 Description Prompt

- **Definition 19 .** *Description prompt:*
  - a function that may be posed by humans
  - who may then act upon it:
    - ⋆ "applying" it to a domain phenomena, and
    - ⋆ "yielding" narrative and formal RSL-**Text**s describing what is being observed ■

Example 16 . Description Prompts:

- result in RSL-**Text**s describing for example a
  - (i) Cartesian endurant, or
  - (ii) its unique identifier,
  - (iii) or its mereology, or
  - (iv) its attributes,
  - (iv) or other ■

3.6 Perdurant Concepts

3.6.1 “Morphing” Parts into Behaviours

- As already indicated we shall
  - transcendentally deduce
  - (perdurant) behaviours from
  - those (endurant) parts
    - ⋆ which we, as domain analysers cum describers,
    - ⋆ have endowed with all three kinds of internal qualities:
    - ⋆ unique identifiers, mereologies and attributes.
- Chapter 6, will show how.

3.6.2 State

Definition 20 . ***State:***

- A state is any set of the parts of a domain ■

Example 17 . A Road System State: The domain analyser cum describer may,
In brief, decide that a road system state consists of

- the road net aggregate (of hubs and links)[4],
- all the hubs, and all the links, and
- the automobile aggregate (of all the automobiles)[5], and
- all the individual automobiles ■

[4]The road net aggregate, in its perdurant form, may "model" the *Department of Roads* of some country, province, or town.
[5]The automobile aggregate aggregate, in its perdurant form, may "model" the *Department of Vehicles* of some country, province, or town.

3.6.3 Actors

Definition 21 . ***Actors:***

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

3.6.3.1 Action

Definition 22 . ***Actions:***

- An action is a function that can purposefully change a state ■

Example 18 . Road Net Actions: These are some road net actions:

- The insertion of a new or removal of an existing hub; or
- the insertion of a new, or removal of an existing link;

3.6.3.2 Event

Definition 23 . ***Events:***

- An event is a function that surreptitiously changes a state ■

Example 19 . Road Net Events: These are some road net events:

- The blocking of a link due to a mud slide;
- the failing of a hub traffic signal due to power outage;
- the blocking of a link due to an automobile accident.

3.6.3.3 Behaviour

Definition 24 . ***Behaviours***

- a behaviour is a set of sequences of actions, events and behaviours

  ■

Example 20 . Road Net Traffic:

- Road net traffic can be seen as a behaviour
  - of all the behaviours of automobiles,
    - ⋆ where each automobile behaviour is seen as sequence of start, stop, turn right, turn left, etc., actions;

- of all the behaviours of links
  - where each link behaviour is seen
    as a set of sequences (i.e., behaviours) of “following” the
    - link entering,
    - link leaving, and
    - movement

    of automobiles on the link;
- of all the behaviours of hubs (etc.);
- of the behaviour of the aggregate of roads,
  viz. *The Department of Roads*, and
- of the behaviour of the aggregate of automobiles,
  viz, *The Department of Vehicles*.

3.6.4 Channel

- **Definition 25 .** *Channel:*
  - A channel is anything
  - that allows synchronisation and communication
  - of values
  - between two behaviours ■

- We shall use Tony Hoare's CSP concept [30]
  - to express synchronisation and communication of values between behaviours.
  - Hence the behaviour *i* statement
    ch[index] ! value
    to state that behaviour *i*
    offers, "outputs": !, *value*
    to behaviours indicated by *index*.
  - And behaviour j expresses
    ch[index] ?
    that it is willing to accept
    "input from & synchronise with" behaviour *i*, ?,
    any *value*.

3.7 Domain Analysis & Description

3.7.1 Domain Analysis

Definition 26 . *Domain Analysis*

- is the act of studying a domain
- as well as the result of that study
- in the form of informal statements ■

3.7.2 Domain Description

Definition 27 . *Domain Description*

- is the act of describing a domain
- as well as the result of that act
- in the form of narratives and formal RSL-Text ■

3.8 Closing

- This lecture has introduced
  the main concepts of domains such as we shall treat
  (analyse and describe) domains.[6]
- The next lectures shall now systematically treat
  the analysis and description of domains.
  - That treatment takes concept by concept and
    - ⋆ provides proper definitions and
    - ⋆ introduces appropriate analysis and description prompts;
    - ⋆ one-by-one, in an almost pedantic,
    - ⋆ hence perhaps "slow" progression !

[6]We have omitted treatment of *living species: plants* and *animals* – the latter including *humans*. They will be treated in the next chapter !

- The student may be excused
  - if they, now-and-then, loose sight of "their way".
- Hence the present chapter.
  - To show "the way":
  - that, for example,
    when we treat external endurant qualities,
  - there is still the internal endurant qualities,
  - and that the whole thing leads of to perdurants:
    - ⋆ actors,
    - ⋆ actions,
    - ⋆ events and
    - ⋆ behaviours.

THANKS

- The next 45 minute lecture shall present fragments of a *road transport system* example.

Lecture 2: External Qualities, Analysis $\frac{1}{2}$

- This is the first, properly systematic treatment lecture of some of the
  - principles,
  - procedures,
  - techniques and
  - tools

  of the *Domain Engineering* method.
- In this lecture we cover those of
  - analysing endurants.

CHAPTER 4. Endurants: External Domain Qualities

- This, the present chapter
  - is based on Chapter 4 of [18].
  - You may wish to study that chapter for more detail.

4.1 Universe of Discourse

Definition 28 . *Universe of Discourse, UoD:*

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

4.1.1 Identification

- The **first task** of a domain analyser cum describer
  - is to settle upon the domain to be analysed and described.
  - That domain has first to be given a *name*.

4.1.2 Naming

- A **first decision** is to give a name to the overall domain sort,
  * that is, the type of the domain seen as an endurant,
  * with that sort, or type, name
    being freely chosen by the analyser cum describer –
  * with no such sort names having been chosen so far !

4.1.3 Examples

- Examples of UoDs

- *railways* [2, 22, 4],
- *"The Market"* [3],
- *container shipping* [5],
- *Web systems* [6],
- *stock exchange* [7],
- *oil pipelines* [8],
- *credit card systems* [10],
- *weather information* [11],
- *swarms of drones* [12],
- *document systems* [13],
- *container terminals* [14],
- *retail systems* [16],
- *assembly plants* [17],
- *waterway systems* [19],
- *shipping* [20],
- *urban planning* [23].

4.1.4 Sketching

- The **second task** of a domain analyser cum describer
  is to develop a *rough sketch narrative* of the domain.
- The rough-sketching of [what] a domain [is,] is not a trivial matter.
  - It is not done by a committee !
  - It usually requires repeated “trial sketches”.
  - To carry it out, i.e., the sketching,
    normally requires a combination of
    - ⋆ physical visits to domain examples, if possible;
    - ⋆ talking with domain professionals, at all levels; and
    - ⋆ reading relevant literature.
    - ⋆ It also includes searching the Internet for information.
- We shall show an example next.

Example 21 . Sketch of a Road Transport System UoD:

- The road transport system that we have in mind consists of
  - a road net and
  - a set of automobiles (private, trucks, buses, etc.)
  - such that the road net serves to convey automobiles.
- We consider the road net to consist of
  - hubs, i.e., street intersections, including street segment connection points, and
  - links, i.e., street segments between adjacent hubs[1] ■

[1] This "rough" narrative fails to narrate ...

4.1.5 Universe of Discourse Description

The general universe of discourse, i.e., domain, description prompt can be expressed as follows:

Domain Description Prompt 1 . *calc_Universe_of_Discourse:*

calc_Universe_of_Discourse ***describer***

❝ **Naming:**
  type UoD
Rough Sketch:
  Text ❞

The above ❝ RSL-**Text** ❞ expresses that the calc_Universe_of_Discourse() domain describer generates RSL-**Text**.

Here is another example rough sketch:

Example 22 . A Rough Sketch Domain Description:

- The example is that of the production of rum, say of a **Rum Production** domain.
- From

1. the sowing, watering, and tending to of sugar cane plants;
2. via the "burning" of these prior to harvest;
3. the harvest;
4. the collection of harvest from sugar cane fields to
5. the chopping, crushing, (and sometimes repeated) boiling, cooling and centrifuging of sugar cane when making sugar and molasses (into A, B, and low grade batches);
6. the fermentation, with water and yeast, producing a 'wash';

7. the (pot still or column still) distilling of the wash into rum;
8. the aging of rum in oak barrels;
9. the charcoal filtration of rum;
10. the blending of rum;
11. the bottling of rum;
12. the preparation of cases of rum for sales/export; and
13. the transportation away from the rum distiller of the rum ■

Some comments on this example:

- Each of the enumerated items above is phrased in terms of perdurants.
  - Behind each such perdurant lies some endurant.
  - That is, in English, *"every noun can be verbed", and vice-versa.*
  - So we anticipate the transcendence, from endurants to perdurants.

● ● ●

Method Principle 1 . ***From the "Overall" to The Details:***

- *Our first principle, as the first task in any new domain modelling project, is*
  - *to "focus" on the "overall",*
  - *that is, on the "entire",*
  - *though specific domain* ■

4.2 Entities

A core concept of domain modelling is that of an *entity*.

Definition 29 . *Entity:*

- By an ***entity*** we shall understand a ***phenomenon***, i.e., something
  - that can be ***observe***d, i.e., be
    - ⋆ seen or touched by humans,
    - ⋆ *or* that can be ***conceive***d
    - ⋆ 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.*
- If a phenomenon cannot be so observed and described then it is not en entity ■

Analysis Predicate Prompt 1 . `is_entity`:

- *The domain analyser analyses "things" (θ) into either entities or non-entities.*
- *The method provides the* ***domain analysis prompt****:*
  - `is_entity` – *where* `is_entity`(θ) *holds if* θ *is an entity* ■ [2]

- `is_entity` is said to be
  - a *prerequisite prompt*
  - for all other prompts.
- `is_entity` is a method tool.

[2] ■ marks the end of an analysis prompt definition.

On Analysis Prompts: The `is_entity` predicate function represents the first of a number of analysis prompts.

- They are “applied” by the domain analyser to phenomena of domains.
- They yield truth values, true or false, “left” in the mind of the domain analyser ■

• • •

- We have just shown how the `is_entity` predicate prompt can be applied to a universe of discourse.
- From now on we shall see prompts being applicable to successively more analysed entities.
- Figure 4.1 [Page 63][3] diagrams
  a domain description ontology of entities.
- That ontology indicates the sub-classes of endurants
  for which we shall motivate and for which we shall introduce
  - prompts,
  - predicates and
  - functions.

[3]This ontology was first shown, as Fig. 3.1 [Page 25]

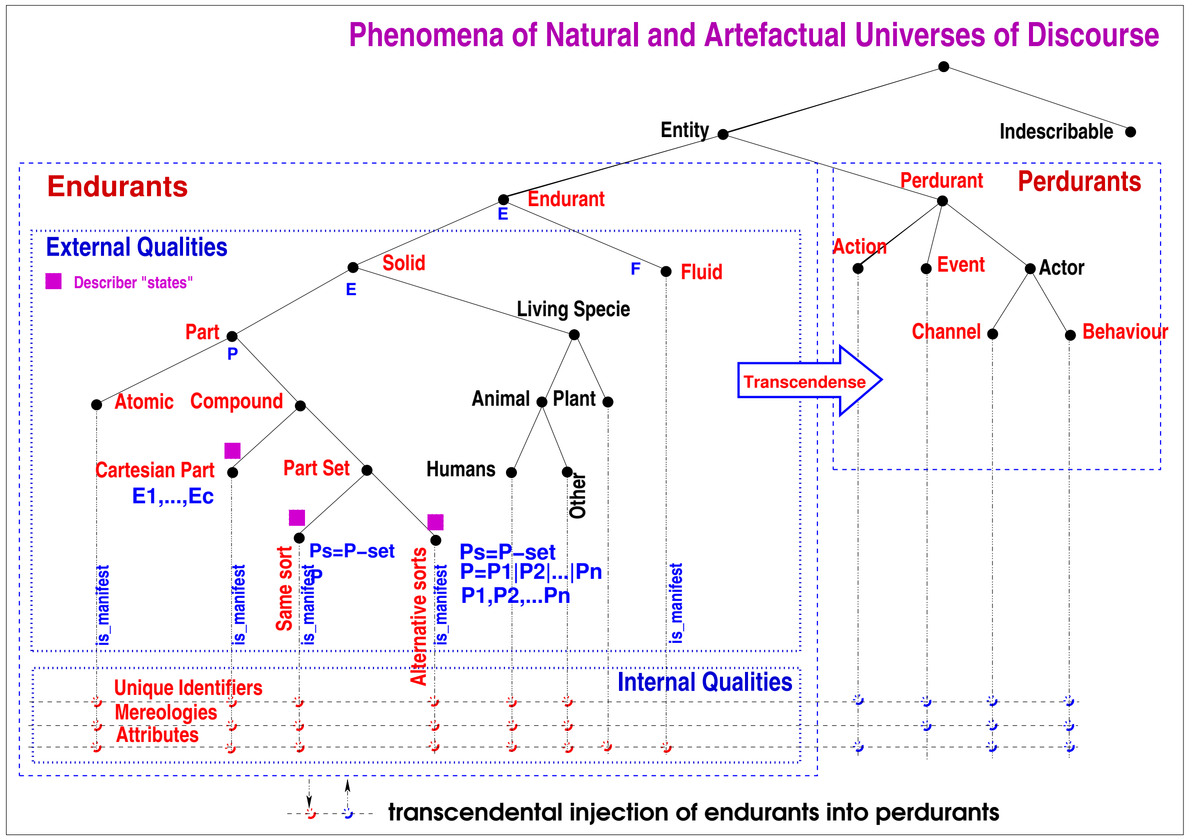

Figure 4.1: The Upper Ontology

- The present chapter shall focus only
  - on the external qualities,
  - that is, on the “contents” of the leftmost dotted box.

• • •

Method Principle 2 . *Justifying Analysis along Philosophical Lines:*

- *The concept of* entities *as a main focal point*
  - *is justified in* Kai Sørlander*'s philosophy.*
  - *Entities are in that philosophy referred to as* primary objects.
  - *They are the ones about which we express predicates* ■

4.3 Endurants and Perdurants

Method Principle 3 . ***Separation of Endurants and Perdurants:***

- *As we shall see in this primer, the domain analysis & description method calls for the separation of*
  - *first considering*
    - ⋆ *the careful analysis & description*
    - ⋆ *of endurants,*
  - *then considering*
    - ⋆ *perdurants.*
- *This principle is based on*
  - *the transcendental deduction*
  - *of the latter from the former* ■

4.3.1 Endurants

Definition 30 . *Endurant:*

- By an *endurant*, to repeat, 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 23 . Natural and Artefactual Endurants:

Geography Endurants:

- fields,
- meadows,
- lakes,
- rivers,
- forests,
- hills,
- mountains,
- et cetera.

Railway Track Endurants:

- a railway track,
- its net,
- its individual tracks,
- switch points,
- trains,
- their individual locomotives,
- signals,
- et cetera.

Road Transport System Endurants:

- the transport system,
- its road net aggregate and the aggregate of automobiles,
- the set of links (road segments) and hubs (road intersections) of the road net aggregate,
- these links and hubs, and
- the automobiles.

Analysis Predicate 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.
- is_endurant is a method tool.

4.3.2 Perdurants

Definition 31 . *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 ■

Example 24 . Perdurants:

Geography Perdurants:

- *the continuous changing of the weather (meteorology);*
- *the erosion of coastlines;*
- *the rising of some land area and the “sinking” of other land area;*
- *volcanic eruptions;*
- *earthquakes;*
- *et cetera.*

Railway System Perdurants:

- *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 Predicate 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` ■
- `is_perdurant` is a method tool.

● ● ●

- We repeat method principle 3 on Slide 66:

Method Principle 4 . *Separation of Endurants and Perdurants:*

- *First domain analyse & describe endurants;*
- *then domain analyse & describe perdurants* ■

4.4 Solids and Fluids

- For *pragmatic* reasons we distinguish between
  - solids and
  - fluids.

Method Principle 5 . ***Abstraction, I:***

- *The principle of abstraction is now brought into "full play":*
  - *In analysing & describing entities the domain analyser cum describer*
  - *is "free" to not consider all facets of entities,*
  - *that is, to abstract.*

4.4.1 Solids

Definition 32 . *Solid Endurant::*

- By a ***solid endurant*** we shall understand an endurant
- which is
  - separate,
  - individual or
  - distinct in form or concept,
- or, rephrasing:
  - a body
  - or magnitude

  of three-dimensions,
  having length, breadth and thickness [32, Vol. II, pg. 2046] ■

Analysis Predicate Prompt 4 . `is_solid`:

- *The domain analyser analyses endurants, e,*
  into solid entities
  as prompted by the **domain analysis prompt**:
  - `is_solid` – *e is solid if* `is_solid(e)` *holds* ■
- To simplify matters we shall allow
  separate elements of a solid endurant to be fluid !
- That is, a solid endurant, i.e., a part,
  may be conjoined with a fluid endurant, a fluid.
- `is_solid` is a method tool.

Example 25 . Artefactual Solid Endurants:

- The individual endurants of the above example of **railway system** endurants, Example 23 on Slide 68, were all solid.
- Here are examples of solid endurants of **pipeline systems**.
  - A pipeline and
  - its individual units:
    - ⋆ wells,
    - ⋆ pipes,
    - ⋆ valves,
    - ⋆ pumps,
    - ⋆ forks,
    - ⋆ joins,
    - ⋆ regulator, and
    - ⋆ sinks.

4.4.2 Fluids

Definition 33 . ***Fluid Endurant:***

- By a ***fluid endurant*** we shall understand
  an endurant which is
  - prolonged, without interruption,
    in an unbroken series or pattern;

  or, rephrasing:
  - a substance (liquid, gas or plasma)
    having the property of flowing,
    consisting of particles
    that move among themselves [32, Vol. I, pg. 774] ■

Analysis Predicate Prompt 5 . `is_fluid:`

- *The domain analyser analyses endurants e into fluid entities as prompted by the* ***domain analysis prompt****:*
  - `is_fluid` – *e is fluid if* `is_fluid(e)` *holds* ■
- `is_fluid` is a method tool.

- Fluids are otherwise
  - liquid, or
  - gaseous, or
  - plasmatic, or
  - granular[4], or
  - plant products[5],
  - et cetera.

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

[5]i.e., chopped sugar cane, threshed, or otherwise. See footnote 4.

Example 26 . Fluids:

- Specific examples of fluids are:
  - water, oil, gas, compressed air, etc.
- A container, which we consider a solid endurant,
  - may be *conjoined* with another, a fluid,
  - like a gas pipeline unit may "contain" gas ■

4.5 Parts and Living Species

- We analyse endurants into either of two kinds:
  - *parts* and
  - *living species*.
- The distinction between *parts* and *living species*
  is motivated in Kai Sørlander's Philosphy [44, 45, 46, 47, 48].

4.5.1 Parts

Definition 34 . ***Parts:***

- By a *part* we shall understand
  - a solid endurant existing in time and
  - subject to laws of physics,
  - including the *causality principle* and
  - *gravitational pull*[6] ■

[6]This characterisation is the result of our study of relations between philosophy and computing science, notably influenced by Kai Sørlander's Philospphy [44, 45, 46, 47, 48]

Analysis Predicate Prompt 6 . `is_part:`

- *The domain analyser analyses “things” (e) into part.*
- *The method can thus be said to provide the* ***domain analysis prompt****:*
  - `is_part` – *where* `is_part(e)` *holds if e is a part* ■

`is_part` is a method tool.

- *Parts* are
  - either *natural* parts, or are
  - *artefactual* parts, i.e. man-made.
- Natural and man-made parts are either
  - *atomic* or
  - *compound*.

4.5.1.1 Atomic Parts

- The term ‘atomic’ is, perhaps, misleading.
  - It is not used in order to refer to nuclear physics.
  - It is, however, chosen in relation to the notion of *atomism:*
    - ⋆ *a doctrine that the physical or physical and mental universe*
    - ⋆ *is composed of simple indivisible minute particles* [Merriam Webster].

Definition 35 . ***Atomic Part, II:***

- By an *atomic part* we shall understand a part
  - which the domain analyser considers to be indivisible
  - in the sense of not meaningfully,
  - for the purposes of the domain under consideration,
  - that is, to not meaningfully consist of sub-parts ■

Example 27 . Atomic Parts:

- We refer to Example 25 on Slide 78: pipeline systems. The
  - wells,
  - pumps,
  - valves,
  - pipes,
  - forks,
  - joins and
  - sinks

  can be considered atomic ■

Analysis Predicate Prompt 7 . `is_atomic:`

- *The domain analyser analyses "things" (e) into atomic part.*
- *The method can thus be said to provide the* ***domain analysis prompt****:*
  - `is_atomic` – *where* `is_atomic(e)` *holds if e is an atomic part* ■

`is_atomic` is a method tool.

4.5.1.2 Compound Parts, II

- We, pragmatically, distinguish between
  - Cartesian-product-, and
  - set-

  oriented parts.
- That is, if Cartesian-product-oriented,
  to consist of two or more
  distinctly sort-named endurants (solids or fluids),
- or, if set-oriented,
  to consist of an indefinite number of zero, one or more
  identically sort-named parts.

Definition 36 . ***Compound Part:***

- *Compound part*s are those which are
  - either Cartesian-product-
  - or are set-
- oriented parts ■

Analysis Predicate Prompt 8 . `is_compound:`

- *The domain analyser analyses “things” (e) into compound part.*
- *The method can thus be said to provide the* **domain analysis prompt***:*
  - `is_compound` – *where* `is_compound(e)` *holds if e is a compound part* ■

`is_compound` is a method tool.

4.5.1.2.1 Cartesian Parts

Definition 37 . ***Cartesian Part:***

- *Cartesian part*s are those (compound parts)
  - which consists of an “indefinite number”
  - of two or more parts
  - of distinctly named sorts ■

Example 28 . Cartesian Automobiles:

- We refer to Example 23 on Slide 69, the **transport system** sub-example.
- We there viewed (hubs, links and) automobiles as atomic parts.
- From another point of view we shall here understand automobiles as Cartesian parts:
  - the engine train,
  - the chassis,
  - the car body,
  - four doors (left front, left rear, right front, right rear), and
  - the wheels.
- These may again be considered Cartesian parts.

Analysis Predicate Prompt 9 . `is_Cartesian:`

- *The domain analyser analyses “things” (e) into Cartesian part.*
- *The method can thus be said to provide the* ***domain analysis prompt****:*
  - `is_Cartesian` – *where* `is_Cartesian(e)` *holds if e is a Cartesian part* ■

`is_Cartesian` is a method tool.

4.5.1.2.2 Calculating Cartesian Part Sorts

- The above analysis amounts to the analyser
  - first "applying" the *domain analysis* prompt
  - is_compound(e) to a solid 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 so!**

● ● ●

On Determination Functions:

- Determination functions
  - apply to compound parts
  - and yield their sub-parts and the sorts of these.
- *That is,*
  - *we observe the domain*
  - *and our observation results*
  - *in a focus on a subset of that domain*
  - *and sort information about that subset.*

An RSL Extension:

- The determine_$\cdots$ functions below are expressed as follows:

  value determine_$\cdots$(e) **as** (parts,sorts)

  - where we focus here on the sorts clause.
  - Typically that clause is of the form ηA,ηB,...,ηC.[7]
  - That is, a "pattern" of sort names: A,B,...,C.
- These sort names are provided by the domain analyser cum describer.
- They are chosen as "full names", or as mnemonics,
  to capture an essence of the (to be) described sort.
- Repeated invocations, by the domain analyser cum describer,
  of these (...,sorts) analysis functions normally
  lead to new sort names
  distinct from previously chosen such names.

4.5.1.2.2.1 Cartesian Part Determination

Observer Function Prompt 1 . *determine_Cartesian_parts:*

- *The domain analyser analyses a part into a Cartesian part.*
- *The method provides the **domain observer prompt**:*
  - determine_Cartesian_parts — *it directs the domain analyser to determine the definite number of values and corresponding distinct sorts of the part.*

    value

    determine_Cartesian_parts: E → (E1×E2×...×En) × (ηE1×ηE2×...×ηEn)[8]

    determine_Cartesian_parts(e) **as** ((e1,...,en),(ηE1,...,ηEn))

    where by E, Ei we mean endurants, i.e., part values, and by ηEi we mean the names of the corresponding types.

determine_Cartesian_parts is a method tool.

[7]ηA,ηB,...,ηC are **names** of types. $\eta\theta$ is the type of all type names !
[8]The ordering, ((e1,...,en),(ηE1,...,ηEn)), is pairwise arbitrary.

On Calculate Prompts:

- Calculation prompts
  - apply to compound parts: Cartesians and sets,
  - and yield an RSL-**Text** description.

Domain Description Prompt 2 . calc_Cartesian_parts:

- *If* is_Cartesian(*e*) *holds, then the analyser "applies" the* **domain description prompt**
  - calc_Cartesian_parts(*e*)

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

calc_Cartesian_parts *describer*

let $(\_^{9},(\eta\mathsf{E}_1,\ldots,\eta\mathsf{E}_m))$ = determine_Cartesian_parts_sorts(e)[10] **in**
❝ **Narration:**
[s] ... narrative text on sorts ...
[o] ... narrative text on sort observers ...
[p] ... narrative text on proof obligations ...
Formalisation:
type
[s] E_1, ❞...❝ , E_m
value
[o] obs_E_1: $\mathsf{E} \rightarrow \mathsf{E}_1$, ❞...❝ , obs_$\mathsf{E}_m$: $\mathsf{E} \rightarrow \mathsf{E}_m$
proof obligation
[p] [Disjointness of endurant sorts] ❞
end

`calc_Cartesian_parts` is a method tool.

[9] The use of the underscore, _ , shall inform the reader that there is no need, here, for naming a value.

[10] For `determine_composite_parts` see Sect. 4.5.1.2.2.1 on Slide 99

Elaboration 1 *Type, Values and Type Names:*

- *Note the use of quotes above.*
- *Please observe that when we write* obs_E *then* obs_E *is the name of a function.*
- *The* E, *when juxtaposed to* obs_ *is now a name* ■

Observer Function Prompt 2 . ***type_name, type_of :***

*The definition of **type_name, type_of***
implies the informal definition of

$\mathsf{obs\_E}_i(\mathsf{e})=\mathsf{e}_i \equiv \mathsf{type\_name}(\mathsf{e}_i)=$“ E_i ”$\wedge$
$\mathsf{type\_of}(\mathsf{e}_i) \equiv \mathsf{E}_i \wedge$
$\mathsf{is\_E}_i(\mathsf{e}_i)$

Example 29 . A Road Transport System Domain: Cartesians:

14. There is the *universe of discourse*, RTS.

 It is composed from

15. a *road net*, RN, and
16. an *aggregate of automobiles*, AA.

```
type
14 RTS
15 RN
16 AA
value
15 obs_RN: RTS → RN
16 obs_AA: RTS → AA ■
```

- We continue the analysis & description of “our” road transport system:

17. The road net consists of
    (a) an aggregate, AH, of hubs and
    (b) an aggregate, AL, of links.

type
17a AH
17b AL
value
17a obs_AH: RN $\rightarrow$ AH
17b obs_AL: RN $\rightarrow$ AL

4.5.1.2.3 Part Sets

Definition 38 . ***Part Sets:***

- ***Part sets*** are those which,
  - in a given context,
  - are deemed to *meaningfully* consist of separately observable
    - a [“root”] part and
    - an indefinite number of proper [“sibling”] *'sub-parts* ■

- For pragmatic reasons we distinguish between parts sets all of whose parts are
  - of the same, single, further un-analysed sort, and
  - of two or more distinct atomic sorts.

Definition 39 . *Single Sort Part Sets:*

- *Single sort part set*s are those which,
  - in a given context,
  - are deemed to *meaningfully* consist of separately observable
    - ⋆ a [“root”] part and
    - ⋆ an indefinite number of proper [“sibling”] $^\iota$*sub-part*s of the same, i.e., single sort ■

Analysis Predicate Prompt 10 . `is_single_sort_set:`

- *The domain analyser analyses a solid endurant, i.e., a part p into a set endurant:*
  - `is_single_sort_set`: *p is a composite endurant if* `is_single_sort_set(`*p*`)` *holds* ■

`is_single_sort_set` is a method tool.

- The `is_single_sort_set` predicate is informal.
- So are all the domain analysis predicates (and functions).
- That is,
  - Their values are “calculated” by a human, the domain analyser.
  - That person observes parts in the “real world”.
  - The determination of the predicate values, hence, are subjective.

Definition 40 . *Alternative Atomic Part Sets:*

- *Alternative sorts part sets* are those which,
  - in a given context,
  - are deemed to *meaningfully* consist of separately observable
    - a ["root"] part and
    - an indefinite number of proper ["sibling"] *ι sub-parts* of two or more atomic parts of distinct sorts ■

Analysis Predicate Prompt 11 . `is_alternative_sorts_set:`

- *The domain analyser analyses a solid endurant, i.e., a part p into a set endurant:*
  - `is_alternative_sorts_set`*: p is a composite endurant if* `is_alternative_sorts_set(p)` *holds* ■

`is_alternative_sorts_set` is a method tool.

4.5.1.2.3.1 Determine Same Sort Part Sets

Observer Function Prompt 3 . *determine_same_sort_parts_set:*

- *The domain analyser observes parts into same sorts part sets.*
- *The method provides the* ***domain observer prompt****:*
  - *determine_alternative_sorts_part_set* *directs the domain analyser to determine the values and corresponding sorts of the part.*

    value

    determine_same_sort_part_set: E $\rightarrow$ (P-**set**$\times\theta$P)
    determine_same_sort_part_set(e) **as** (ps,ηPn)

determine_same_sort_part_set is a method tool.

4.5.1.2.3.2 Determine Alternative Sorts Part Sets

Observer Function Prompt 4 .

determine_alternative_sorts_part_set:

- *The domain analyser observes parts into alternative sorts part sets.*
- *The method provides the* ***domain observer prompt****:*
  - determine_alternative_sorts_part_set
    directs the domain analyser to determine the values and corresponding sorts of the part.

    value

    determine_alternative_sorts_part_set: E → ((P1×θP1)×...×(Pn,θPn))
    determine_alternative_sorts_part_set(e) **as** ((p1,ηp1),...,(pn,ηPn))

  - *The set of parts, of different sorts, may have more than one element, p, p′, ..., p″ being of the same sort Ei.*

determine_alternative_sorts_part_set is a method tool.

4.5.1.2.3.3 Calculating Single Sort Part Sets

Domain Description Prompt 3 . ***`calc_single_sort_parts_sort`*:**

- *If `is_single_set_sort_parts`(e) holds, then the analyser "applies" the **domain description prompt***
  - *`calc_single_sort_parts_sort`(e)*

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

`calculate_single_sort_parts_sort(e)` *Describer*

let (_,ηP) = determine_single_sort_part(e)[11] **in**

❝ **Narration:**

[s] ... narrative text on sort ...
[o] ... narrative text on sort observer ...
[p] ... narrative text on proof obligation ...

Formalisation:

type
[s] P
[s] Ps = P-**set**
value
[o] obs_Ps: E $\rightarrow$ Ps ❞
end

`calculate_single_sort_parts_sort` is a method tool.

[11] For `determine_single_sort_part` see Defn. 39 on Slide 109.

Elaboration 2 *Type, Values and Type Names:*

- *Note the use of quotes above.*
- *Please observe that when we write* obs_Ps *then* obs_Ps *is the name of a function.*
- *The* Ps, *when juxtaposed to* obs_ *is now a name* ■

Example 30 . Road Transport System: Sets of Hubs, Links and Automobiles: We refer to Example 29 on Slide 106.

18. The road net aggregate of road net hubs consists of a set of [atomic] hubs,
19. The road net aggregate of road net links consists of a set of [atomic] links,
20. The road net aggregate of automobiles consists of a set of [atomic] automobiles.

type
18. Hs = H-**set**, H
18. Ls – L-**set**, L
18. As = A-**set**, A

value
18. obs_Hs: AH $\rightarrow$ Hs
18. obs_Ls: AL $\rightarrow$ Ls
18. obs_As: AA $\rightarrow$ As ■

4.5.1.2.3.4 Calculating Alternative Sort Part Sets

- We leave it to the reader to decipher the `calculate_alternative_sort_part_sorts` prompt.

Domain Description Prompt 4 .
calculate_alternative_sort_part_sorts:

- *If* `is_alternative_sort_parts_sorts`*(e) holds, then the analyser "applies" the* **domain description prompt**
  - `calculate_alternative_sort_part_sorts`*(e)*

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

calculate_alternative_sort_part_sorts(e) *Describer*

let ((p1,ηE_1),...,(pn,ηE_n)) = determine_alternative_sorts_part_set_sorts(e)[12] **in**

❝ **Narration:**

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

Formalisation:

```
type
[ s ]  Ea = E_1 | ... | E_n
[ s ]  E_1 :: End_1, ..., E_n :: End_n
value
[ o ]  obs_Ea: E → Ea
axiom
[ p ]  [ disjointness of alternative sorts ] E_1, ..., E_n ❞
end
```

- The set of parts, of different sorts, may have more than one element, say $p, p', ..., p''$ being of the same sort E_i.
  - Since parts are not mentioned in the sort description above, cf.,
    _,
  - only the distinct alternative sort observers appear in that description.

calculate_alternative_sort_part_sorts is a method tool.

[12]For determine_alternative_sort_part_sorts see Defn. 40 on Slide 112.

Example 31 . Alternative Rail Units:

21. The example is that of a railway system.
22. We focus on railway nets. They can be observed from the railway system.
23. The railway net embodies a set of [railway] net units.
24. A net unit is either a
    - straight or curved **linear** unit, or a
    - simple switch, i.e., a **turnout**, unit[13] or
    - a simple cross-over, i.e., a **rigid** crossing unit, or a
    - single switched cross-over, i.e., a **single** slip unit, or a
    - double switched cross-over, i.e., a **double** slip unit, or a
    - **terminal** unit.
25. As a formal specification language technicality disjointness of the respective rail unit types is afforded by RSL's :: type definition construct.

- We refer to Figure 4.2 on the next slide.

```
type
21. RS
22. RN
value
22. obs_RN: RS → RN
type
23. NUs = NU-set
24. NU = LU|PU|RU|SU|DU|TU
25. LU :: LinU
25. PU :: PntU
25. SU :: SwiU
25. DU :: DblU
25. TU :: TerU
value
23. obs_NUs: RN → NUs
```

[13] https://en.wikipedia.org/wiki/Railroad_switch

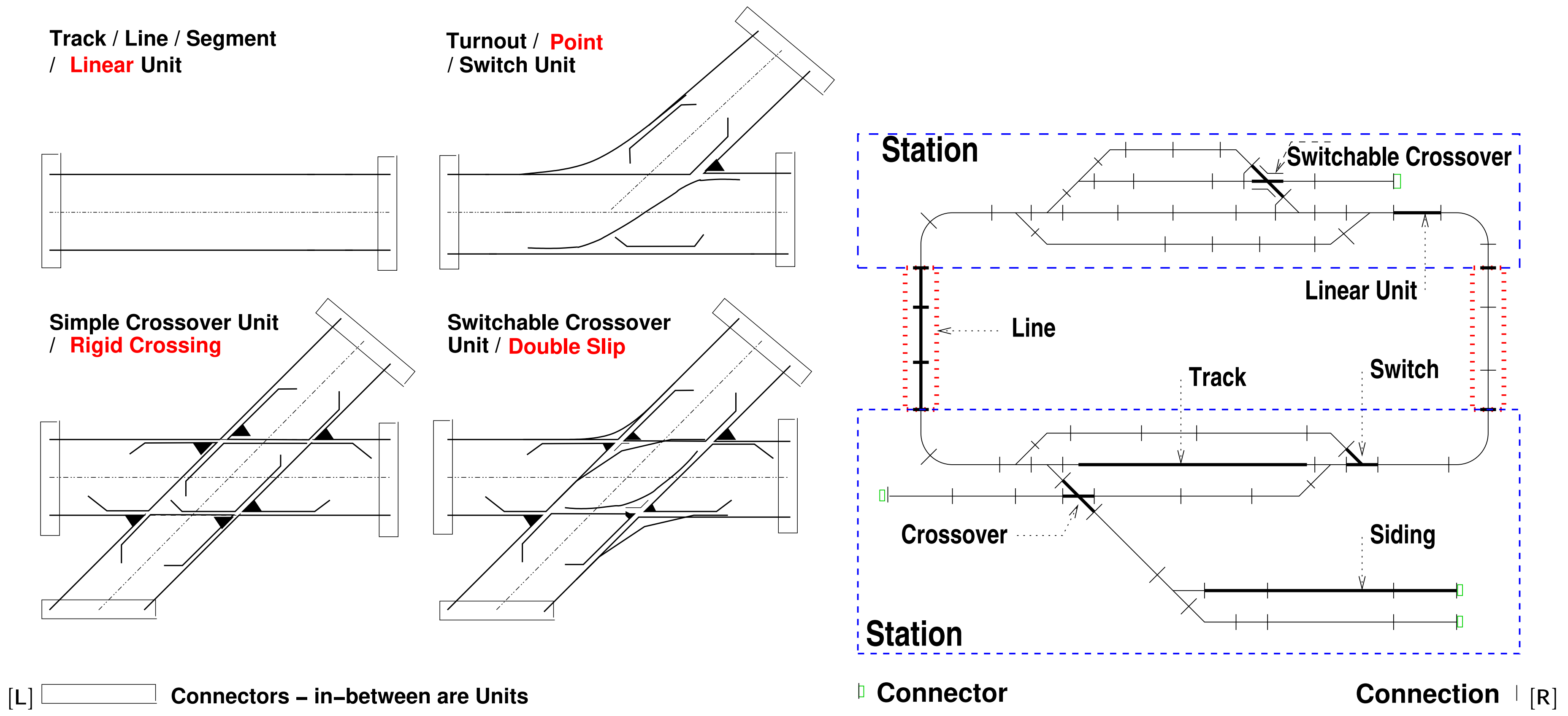
Track / Line / Segment / Linear Unit
Turnout / Point / Switch Unit
Simple Crossover Unit / Rigid Crossing
Switchable Crossover Unit / Double Slip
[L]
Connectors – in-between are Units
Station
Switchable Crossover
Linear Unit
Line
Track
Switch
Crossover
Siding
Station
Connector
Connection
[R]

Figure 4.2: Left: Four net units (LU, PU, SU, DU); Right: A railway net

• • •

Method Principle 6 . *Pedantic Steps of Development:*

- *This section, i.e., Sect. 4.5.1, has illustrated a principle of "small, pedantic" analysis & description steps.*
  - *You could also call it a principle of separation of concerns* ■

4.5.1.3 Ontology and Taxonomy

- We can speak of two kinds of ontologies:
  - the general ontologies of domain analysis & description, cf. Fig. 4.1 on Slide 63, and
  - a specific domain's possible endurant ontologies.
  - We shall here focus on a [“restricted”] concept of taxonomies[14]

[14]By taxonomy (or taxonomical classification) we shall here understand a scheme of classification, especially a hierarchical classification, in which things are organized into groups [Wikipedia].

Definition 41 . ***Domain Taxonomy:*** By a domain taxonomy we shall understand

- a hierarchical structure,
  usually depicted as a(n "upside-down") tree,
- whose "root" designates a compound part
- and whose "siblings" (proper sub-trees)
  designate parts or fluids ■

- The 'restriction' amounts to considering only endurants.
- That is, not considering perdurants.
- `Taxonomy` is a method technique.

Example 32 . The Road Transport System Taxonomy:

- Figure 4.3 shows a schematised, i.e., the . . . , taxonomy for the *Road Transport System* domain of Example 4.1 on Slide 63.

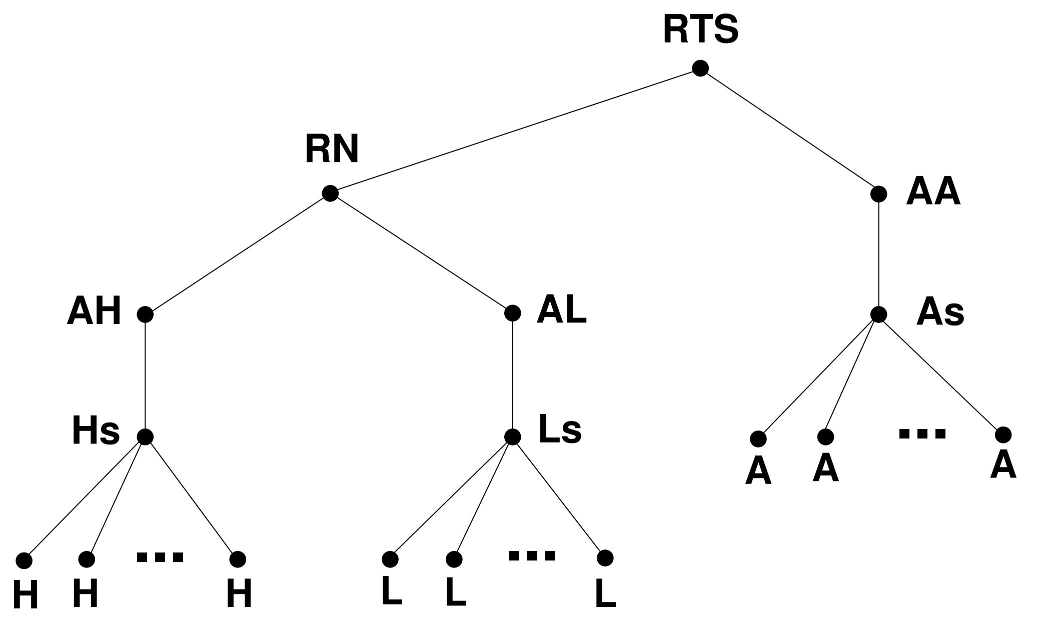

Figure 4.3: A Road Transport System Ontology ■

4.5.1.4 "Root" and "Sibling" Parts

- For compound parts, cf. Definition 36 on Slide 91,
  - we introduce the specific domain taxonomy concepts of "root" and "sibling" parts.
  - (We also refer to Fig. 4.3 on the preceding slide.)
- When observing, as a human, a compound part one may ask the question
  - *"a tree consisting of a specific domain taxonomy node labelled, e.g., X*
  - *and the sub-trees labelled, e.g., $Y_1, Y_2, \ldots, Y_n$*
  - *does that tree designate one "indivisible" part*
  - *or does it designate $n+1$ parts ?"*
  - We shall, in general, consider the answer to be the latter: $n+1$!

- We shall, in general, consider compound parts to consist of
  - a “root” parts
  - and *n* “sibling parts and fluids”.
- What the analyser cum describer observes
  - appears as one part, “the whole”,
  - with *n* “embedded” sub-parts.
- What the analyser cum describer is asked to model is
  - 1, the root part, and
  - *n*, the sibling, parts and fluids.

- The fact that the root part is separately modelled from the sibling parts,
- may seem to disappear in this separate modelling —
- but, as You shall see, in the next chapter,
  - their relation: the siblings to “the whole”, i.e., the root,
  - will be modelled, specifically through their mereologies,
  - as will be covered in Sect. 5.3,
  - but also through their respective attributes, Sect. 5.4.
- We shall see this non-embbedness of root and sibling parts
  - further accentuated in the modelling of their transcendentally deduced
  - respective (perdurant) behaviours as distinct concurrent behaviours
  - in Chapter 6.

4.5.2 Living Species

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

Definition 42 . *Living Species:*

- By a *living species* we shall understand
  - a solid endurant,
  - subject to laws of physics, and
  - additionally subject to *causality of purpose*.
- Living species
  - must have some *form they can be developed to reach*;
  - a form they must be *causally determined to maintain*.
  - This *development and maintenance* must further engage in *exchanges of matter with an environment*.
- It must be possible that living species occur in two forms:
  - **plants**, respectively **animals**,
  - forms which are characterised by *development, form and exchange*,
  - which, additionally, can be characterised by the *ability of purposeful movement* ■

Analysis Predicate Prompt 12 . `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* ■

`is_living_species` is a method tool.

- It is appropriate here to mention **Carl Linnaeus** (1707–1778).
  - He was a Swedish botanist, zoologist, and physician
  - who formalised, in the form of a binomial nomenclature,
  - the modern system of naming organisms.
  - He is known as the “father of modern taxonomy”.
  - We refer to his ‘Species Plantarum’ `gutenberg.org/files/20771/20771-h/20771-h.htm`.

4.5.2.1 Plants

Example 33 . 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 Predicate 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* ■
- `is_plant` is a method tool.
- The predicate `is_living_species(`ℓ`)` is a prerequisite for `is_plant(`ℓ`)`.

4.5.2.2 Animals

Definition 43 . *Animal:* We refer to the initial definition of *living species* above – while emphasizing the following traits:

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

Analysis Predicate 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* ■
- `is_animal` is a method tool.
- The predicate `is_living_species(`ℓ`)` is a prerequisite for `is_animal(`ℓ`)`.
- We distinguish, motivated by [47], between
  - humans and
  - other.

4.5.2.2.1 Humans

Definition 44 . *Human:*

- A *human* (a *person*) is an *animal* ,
  cf. Definition 43 on Slide 139,
  with the additional properties of having
  - *language* ,
  - being *conscious* of *having knowledge* (of its own situation), and
  - *responsibility* ■

Analysis Predicate 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* ■
- `is_human` is a method tool.
- The predicate `is_animal(`ℓ`)` is a prerequisite for `is_human(`ℓ`)`.

- 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 artefacts,
  - that is, that these artefacts have human characteristics.
- You, the listeners, 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 !

4.5.2.2.2 Other

- We shall skip any treatment of other than human animals !

4.6 Some Observations

- Two observations must be made.
  - (i) The domain analyser cum describer procedures
    - ⋆ illustrated by the analysis functions
    - ⋆ `determine_Cartesian_parts`,
    - ⋆ `determine_same_sort_part_set` and
    - ⋆ `determine_alternative_sorts_part_set`
    - ⋆ yield names of endurant sorts.
    - ⋆ Some of these names may have already been encountered, i.e., discovered.
    - ⋆ That is, the domain analyser cum describer must carefully consider such possibilities.

- (ii) Endurants are <u>**not**</u> **recursively definable** !
  - ⋆ This appears to come as a surprise
    to many computer scientists.
  - ⋆ Immediately many suggest that “tree-like” endurants
    like a river,
  - ⋆ or, indeed, a tree,
  - ⋆ should be defined recursively.
  - ⋆ But we posit that that is not the case.
  - ⋆ A river, for example, has a delta, its “root” so-to-speak,
  - ⋆ but the sub-trees of a recursively defined river endurant
  - ⋆ has no such “deltas” !
  - ⋆ Instead we define such “tree-like” endurants
    as graphs with appropriate mereologies.

4.7 States

- In our continued modelling
  - we shall make good use of a concept of states.

Definition 45 . ***State:*** By a *state* we shall understand

- any collection of one or more parts ■

- In Chapter 5 Sect. 5.4
  we introduce the notion of *attributes*.
  - Among attributes there are the *dynamic attributes*.
  - They model that internal part quality values
    may change dynamically.
  - So we may wish, on occasion, to ‘refine’ our notion of state
    to be just those parts which have dynamic attributes.

4.7.1 State Calculation

- Given any universe of discourse, uod:UoD, we can recursively calculate its "full" state, calc_parts({uod}).

26. Let e be any endurant. Let arg_parts be the parts to be calculated. Let res_parts be the parts calculated. Initialise the calculator with arg_parts={e} and res_parts={}. Calculation stops with arg_parts empty and res_parts the result.
27. If is_Cartesian(e)
28. then we obtain its immediate parts, determine_composite_part(e)
29. add them, as a set, to arg_parts, e removed from arg_parts and added to res_parts calculating the parts from that.
30. If is_single_sort_part_set(e)
31. then the parts, ps, of the single sort set are determined,
32. added to arg_parts and e removed from arg_parts and added to res_parts calculating the parts from that.
33. If is_alternative_sorts_part_set(e) then the parts, ((p1,_),(p2,_),...,(pn,_)), of the alternative sorts set are determined, added to arg_parts and e removed from arg_parts and added to res_parts calculating the parts from that.

```
value
26.  calc_parts: E-set → E-set → E-set
26.  calc_parts(arg_parts)(res_parts) ≡
26.    if arg_parts = {} then res_parts else
26.    let e · e ∈ arg_parts in
27.    is_Cartesian(e) →
28.       let ((e1,e2,...,en),_) = observe_Cartesian_part(e) in
29.       calc_parts(arg_parts\{e} ∪ {e1,e2,...,en})(res_parts ∪ {e}) end
30.    is_single_sort_part_set(e) →
31.       let ps = observe_single_sort_part_set(e) in
32.       calc_parts(arg_parts\{e}∪ ps)(res_parts ∪ {e}) end
33.    is_alternative_sort_part_set(e) →
33.       let ((p1,_),(p2,_),...,(pn,_)) = observe_alternative_sorts_part_set(e) in
33.       calc_parts(arg_parts\{e}∪{p1,p2,...,pn})(res_parts ∪ {e}) end
26.    end end
```

`calc_parts` is a method tool.

Method Principle 7 . *Domain State:*

- *We have found, once all the state components, i.e., the endurant parts, have had their external qualities analysed, that*
  - *it is then expedient to define the domain state.*
  - *It can then be the basis for several concepts*
  - *of internal qualities.*

Example 34 . Constants and States:

34. Let there be given a universe of discourse, rts.
    The set $\{rts\}$ is an example of a state.

From that state we can calculate other states.

35. The set of all hubs, hs.
36. The set of all links, ls.
37. The set of all hubs and links, hls.
38. The set of all automobiles, as.
39. The set of all parts, ps.

value

34 rts:UoD [34]

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

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

37 hls:(H|L)-**set** ≡ $hs \cup ls$

38 as:A-**set** ≡ obs_As(obs_AA(obs_RN(rts)))

39 ps:(UoB|H|L|A)-**set** ≡ $rts \cup hls \cup as$

4.7.2 Update-able States

- We shall, in Sect. 5.4, introduce the notion of parts,
  - having dynamic attributes,
  - that is, having internal qualities that may change.
- To cope with the modelling,
- in particular of so-called *monitor-able* attributes,
- we present the *state* as a global variable:

variable σ := calc_parts({uod})

4.8 An External Analysis and Description Procedure

- We have covered
  - the individual analysis and description steps
  - of our approach to the external qualities modelling
  - of domain endurants.
- We now suggest
  - a ‘formal’ description of the process
  - of linking all these analysis and description steps.

4.8.1 An Analysis & Description State

- Common to all the discovery processes is an idea of a *notice board*.
- A notice board, at any time in the development of a domain description, is a repository of the analysis and description process.
- We suggest to model the notice board in terms of three global variables.
  - The **new** variable holds the parts yet to be described,
  - The **ans** variable holds the sort name of parts that have so far been described,
  - the **gen** variable holds the parts that have so far been described, and
  - the **txt** variable holds the RSL-Text so far generated.
    - ⋆ We model the **txt** variable as a map
    - ⋆ from endurant identifier names to RSL-Text.

A Domain Discovery Notice Board

```
variable
  new := {uod} ,
  asn := { “UoD ”}
  gen := {} ,
  txt:RSL-Text := [ uid_UoD(uod) ↦ ⟨“type UoD”⟩ ]
```

4.8.2 A Domain Discovery Procedure, I

- The discover_sorts pseudo program
  - suggests a systematic way of proceeding
  - through analysis, manifested by the is_$\cdots$ predicates,
  - to ($\rightarrow$) description.
- Some comments are in order.
  - The e-set$_a$$\uplus$e-set$_b$ expression
  - yields a set of endurants that are either in e-set$_a$, or in e-set$_a$, or in both,
  - but such that two endurants, e$_x$ and e$_y$
  - which are of the same endurants type, say E,
  - and are in respective sets is only represented once in the result;
  - that is, if they are type-wise the same, but value-wise different
  - they will only be included once in the result.
- As this is the first time RSL-**Text** is put on the notice board
  we express this as:
  - txt := txt $\cup$ [type_name(v) $\mapsto$ $\langle$RSL-**Text**$\rangle$]
- Subsequent insertion of RSL-**Text**
  for internal quality descriptions and perdurants
  is then concatenated to the end of previously uploaded RSL-**Text**.

An External Qualities Domain Analysis and Description Process

```
value
discover_sorts: Unit → Unit
discover_sorts() ≡ while new ≠ {} do
    let v • v ∈ new in (new := new \ {v} || gen := gen ∪ {v} || ans := ans \ {type_of(v)}) ;
    is_atomic(v) → skip ,
    is_compound(v) →
      is_Cartesian(v) →
       let ((e1,...,en),(ηE1,...,ηEn))=analyse_composite_parts(v) in
       (ans := ans ∪ {ηE1,...,ηEn} || new := new ⊎ {e1,...,en}
         || txt := txt ∪ [type_name(v) ↦ ⟨calculate_composite_part_sorts(v)⟩]) end,
      is_part_set(v) →
       (is_single_sort_set(v) →
          let ({p1,...,pn},ηP)=analyse_single_sort_parts_set(v) in
          (ans := ans ∪ {ηP} || new := new ⊎ {p1,...,pn} ||
            txt := txt ∪ [type_name(v) ↦ calculate_single_sort_part_sort(v)]) end,
         is_alternative_sorts_set(v) →
          let ((p1,ηE1),...,(pn,ηEn))= observe_alternative_sorts_part_set(v) in
          (ans := ans ∪ {ηE1,...,En} || new := new ⊎ {p1,...,pn} ||
            txt := txt ∪ [type_name(v) ↦ calculate_alternative_sorts_part_sort(v)]) end)
  end end
```

4.9 Summary

- We briefly summarise the main findings of this chapter.
- These are the main
  - analysis predicates and functions and
  - the main description functions.
- These, to remind the student, are
  - the *analysis*, the **is_···**, *predicates*,
  - the *analysis*, the **determine_···**, *functions*,
  - the *state calculation* function,
  - the *description* functions, and
  - the *domain discovery* procedure.

- They are summarised in this table:

#	Name	Introduced
	Analysis Predicates	
1	is_entity	page 60
2	is_endurant	page 70
3	is_perdurant	page 73
4	is_solid	page 77
5	is_fluid	page 80
6	is_part	page 85
7	is_atomic	page 89
8	is_compound	page 92
9	is_Cartesian	page 95
10	is_single_sort_set	page 110
11	is_alternative_sorts_set	page 113
12	is_living_species	page 135
13	is_plant	page 138
14	is_animal	page 140
15	is_human	page 142
	Analysis Functions	
1	determine_Cartesian_parts	page 99
3	determine_same_sort_part_set	page 114
4	determine_alternative_sorts_part_set	page 115
	State Calculation	
	calc_parts	page 150
	Description Functions	
1	calc_Universe_of_Discourse	page 55
2	calc_Cartesian_parts	page 101
3	calc_single_sort_parts_sort	page 116
4	calc_alternative_sort_part_sorts	page 117
	Domain Discovery	
	discover_sorts	page 160

● ● ●

- Please consider Fig. 4.1 on Slide 63.
  - This chapter has covered the tree-like structure to the left in Fig. 4.1.
  - The next chapter covers the horisontal and vertical lines, also to the left in Fig. 4.1.

Lecture 3: Unique Identifiers and Mereology

- We now present a properly systematic treatment of some of the
  - principles,
  - procedures,
  - techniques and
  - tools

  of the *Domain Engineering* method, namely for
  - the **internal qualities** of endurants:
    - ⋆ unique identification,
    - ⋆ mereology,
    - ⋆ attributes and
    - ⋆ intentional pull.

CHAPTER 5. Endurants: Internal and Universal Domain Qualities

- Please consider Fig. 4.1 on Slide 63.
  - The previous chapter covered the tree-like structure to the left in Fig. 4.1.
  - This chapter covers the horisontal and vertical lines, also to the left in Fig. 4.1.

• • •

- In this chapter we introduce
  - the concepts of internal qualities of endurants and universal qualities of domains,
  - and cover, first, the analysis and description of internal qualities:
    - ⋆ **unique identifiers** (Sect. 5.2 on Slide 179),

- ⋆ **mereologies** (Sect. 5.3 on Slide 203) and
- ⋆ **attributes** (Sect. 5.4 on Slide 231),

- There is, additionally, three universal qualities:
  - ⋆ space, time (Sect. 5.5 on Slide 294) and
  - ⋆ intentionality (Sect. 5.6 on Slide 326), where
    - · *intentionality* is “something” that expresses
    - · intention, design idea, purpose of artefacts –
    - · well, some would say, also of natural endurants.

- As it turns out,
  - to analyse and describe mereology
  - we need to first analyse and describe unique identifiers;

  and
  - to analyse and describe attributes
  - we need to first analyse and describe mereologies.

- Hence:

Method Procedure 1 .
Sequential Analysis & Description of Internal Qualities:

- *We advise that the domain analyser & describer*
  - **first** *analyse & describe* **unique identification** *of all endurant sorts;*
  - **then** *analyse & describe* **mereologies** *of all endurant sorts;*
  - **finally** *analyse & describe* **attributes** *of all endurant sorts.*

5.1 Internal Qualities

- We shall investigate the, as we shall call them, internal qualities of domains.
- That is the properties of the entities to which we ascribe internal qualities.
- The outcome of this chapter is that the student
  - will be able to model the internal qualities of domains.
  - Not just for a particular domain instance,
  - but a possibly infinite set of domain instances[2].

[2]By this we mean: You are not just analysing a specific domain, say the one manifested around the corner from where you are, but any instance, anywhere in the world, which satisfies what you have described.

5.1.1 General Characterisation

- External qualities of endurants of a manifest domain
  - are, in a simplifying sense, those we can
    - see and
    - touch.
  - They, so to speak, take form.

- Internal qualities of endurants of a manifest domain
  - are, in a less simplifying sense, those which
    - ⋆ we may not be able to see or "feel"
      when touching an endurant,
    - ⋆ but they can, as we now 'mandate' them,
      - · be reasoned about,
        as for unique identifiers
        and mereologies,

      or
    - ⋆ be measured by some **physical/chemical** means,
    - ⋆ or be "spoken of" by **intentional deduction**, and
      - · be reasoned about,
    - ⋆ as we do when we attribute properties to endurants.

5.1.2 Manifest Parts versus Structures

- In [18] we covered a notion of *´structures´*.
  - In this primer we shall treat
    the concept of ‘structures’ differently
  - We do so by distinguishing between
    - ⋆ manifest parts
    - ⋆ and structures.

5.1.2.1 Definitions

Definition 46 . ***Manifest Part:*** By a manifest part we shall understand

- a part which 'manifests' itself
  - either in a physical, visible manner, "occupying" an AREA or a VOLUME and a POSITION in SPACE,
  - or in a conceptual manner forms an organisation in Your mind ! ■
- As we have already revealed,
- endurant parts can be transcendentally deduced into perdurant behaviours
- – with manifest parts indeed being so.

Definition 47 . ***Structure:*** By a structure we shall understand

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

5.1.2.2 Analysis Predicates

Analysis Predicate Prompt 16 . `is_manifest:`

- *The method provides the* ***domain analysis prompt****:*
  - `is_manifest` – *where* `is_manifest(p)` *holds if p is to be considered manifest* ■

Analysis Predicate Prompt 17 . `is_structure:`

- *The method provides the* ***domain analysis prompt****:*
  - `is_structure` – *where* `is_structure(p)` *holds if p is to be considered a structure* ■

- The obvious holds: `is_manifest(p)` ≡ ¬ `is_structure(p)`.

5.1.2.3 Examples

Example 35 . Manifest Parts and Structures:
We refer to Example 29 on Slide 106: the Road Transport System.

- We shall consider all atomic parts: hubs, links and automobiles as being manifest. (They are physical, visible and in $\mathbb{SPACE}$.)
- We shall consider road nets and aggregates of automobiles as being manifest.
  - Road nets are physical, visible and in $\mathbb{SPACE}$.
  - Aggregates of automobiles are here considered conceptual.
  - The road net manifest part,
    - ⋆ apart from it aggregates of hubs and links,
    - ⋆ can be thought of as "representing" a *Department of Roads*[3].
  - The automobile aggregate
    - ⋆ apart from its automobiles,
    - ⋆ can be thought of as "representing" a *Department of Vehicles*[4].
  - We shall consider hub and link aggregates and hub and link set as structures.

[3]– of some country, state, province, city or other.
[4]See above footnote.

5.1.2.4 Modelling Consequence

- In this chapter we introduce internal endurant qualities.
  - If a part is considered manifest
    then we shall endow that part
    with all three kinds of internal qualities.
  - If a part is considered a structure
    then we shall **not** endow that part
    with any of three kinds of internal qualities.

5.2 Unique Identification

- The concept of parts having unique identifiability,
  - that is, that two parts,
  - if they are the same,
  - have the same unique identifier,
  - and if they are not the same,
  - then they have distinct identifiers,
  - that concept is fundamental to our being able
    to analyse and describe internal qualities of endurants.
- So we are left with the issue of 'identity' !

5.2.1 On Uniqueness of Endurants

- We therefore introduce the notion of unique identification of part endurants.
- We assume
  - (i) that all part endurants, e, of any domain E, have ***unique identifier***s,
  - (ii) that ***unique identifier***s (of part endurants e:E) are ***abstract value***s (of the ***unique identifier*** sort UI of part endurants e:E),
  - (iii) that such that distinct part endurant sorts, E_i and E_j, have distinctly named ***unique identifier*** sorts, say UI_i and UI_j[5], and
  - (iv) that all ui_i:UI_i and ui_j:UI_j are distinct.

[5]This restriction is not necessary, but, for the time, we can assume that it is.

- The names of unique identifier sorts, say UI,
  is entirely at the discretion of
  the *domain analyser cum describer*.
- If, for example, the sort name of a part is P,
  then it might be expedient to
  name the sort of the unique identifiers of its parts PI.

Representation of Unique Identifiers:

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

Identifiability of Endurants:

- From a philosophical point of view,
  - and with basis in Kai Sørlander's Philospphy,
  - one can rationally argue that there are many endurants,
  - and that they are unique, and hence uniquely identifiable.
- From an empirical point of view,
  - and since one may eventually
    have a software development in mind,
  - we may wonder how unique identifiablity can be
    accommodated.

- Unique identifiability for solid endurants,
  - even though they may be mobile,
  - is straightforward:
    - ⋆ one can think of many ways
    - ⋆ of ascribing a unique identifier to any part;
    - ⋆ solid endurants do not "morph"[6].
- Hence one can think of many such unique identification schemas.

[6] That is, our domain modelling method
is not thought of as being applied
to the physics situations of
endurants going,
for example, from states of being solid,
via states of melting, to states of fluid.

- Unique identifiability for fluids may seem a bit more tricky.
  - For this *primer* we shall not suggest
  - to endow fluids with unique identification.
  - We have simply not experimented with such part-fluids and fluid-parts domains
    – not enough – to suggest so.

5.2.2 Uniqueness Modelling Tools

- The analysis method offers an observer function uid_E
  which when applied to part endurants, e,
  yields the unique identifier, ui:UI, of e.

Domain Description Prompt 5 . *describe_unique_identifier(e):*

- *We can therefore apply the* **domain description prompt**:
  - *describe_unique_identifier(e)*
- *to endurants* e:E
  - *resulting in the analyser writing down*
  - *the unique identifier type and observer domain description text according to the following schema:*

4. describe_unique_identifier(e) Observer

❝ **Narration:**

[s] ... narrative text on unique identifier sort UI ...[7]
[u] ... narrative text on unique identifier observer uid_E ...
[a] ... axiom on uniqueness of unique identifiers ...

Formalisation:

type
[s] UI
value
[u] uid_E: E $\rightarrow$ UI ❞

- is_part(e) is a prerequisite for describe_unique_identifier(e).

[7]The name, UI, of the unique identifier sort is determined, *"pulled out of a hat"*, by the domain analyser cum describer(s), i.e., the person(s) who "apply" the describe_unique_identifier(e) prompt.

- The unique identifier type name, UI above,
  - chosen, of course, by the *domain analyser cum describer*,
  - usually properly embodies the type name, E,
  - of the endurant being analysed and mereology-described.
  - Thus a part of type-name E
    might be given the mereology type name EI.
  - Generally we shall refer to these names by UI.

Observer Function Prompt 5 . `type_name, type_of, is_:`

- *Given description schema 5*
  - *we have, so-to-speak "in-reverse", that*

$$\forall\ \mathsf{e{:}E} \cdot \mathsf{uid\_E(e){=}ui} \Rightarrow$$
$$\mathsf{type\_of(ui)}{=}\eta\mathsf{UI} \wedge \mathsf{type\_name(ui){=}UI} \wedge \mathsf{is\_UI(ui)}$$

- ηUI is a variable of type $\eta\mathbb{T}$.
- $\eta\mathbb{T}$ is the type of all domain endurant, unique identifier, mereology and attribute type names.
- By the subsequent UI we refer to the unique identifier type name value of ηUI.

Example 36 . Unique Identifiers:

40. We assign unique identifiers to all parts.
41. By a road identifier we shall mean a link or a hub identifier.
42. Unique identifiers uniquely identify all parts.
    (a) All hubs have distinct [unique] identifiers.
    (b) All links have distinct identifiers.
    (c) All automobiles have distinct identifiers.
    (d) All parts have distinct identifiers.

type

```
40  H_UI, L_UI, A_UI
41  R_UI = H_UI | L_UI
```

value

```
42a  uid_H: H → H_UI
42b  uid_L: H → L_UI
42c  uid_A: H → A_UI
```

5.2.3 The Unique Identifier State

- Given a universe of discourse we can calculate the set of the unique identifiers of all its parts.

```
value
    calculate_all_unique_identifiers: UoD → UI-set
    calculate_all_unique_identifiers(uod) ≡
        let parts = calc_parts({uod})({}) in
        { uid_E(e) | e:E · e ∈ parts } end
```

5.2.4 The Unique Identifier State

- We can speak of a unique identifier state:

```
variable
    uod := ...
    uid_σ := discover_uids()
value
    discover_uids: UoD → Unit
    discover_uids(uod) ≡ calculate_all_unique_identifiers(uod)
```

Example 37 . Unique Road Transport System Identifiers:
We can calculate:

43. the *set*, $h_{ui}s$, of *u*nique *h*ub *i*dentifiers;
44. the *set*, $l_{ui}s$, of *u*nique *l*ink *i*dentifiers;
45. the *set*, $r_{ui}s$, of all *u*nique hub and link, i.e., *r*oad *i*dentifiers;
46. the *map*, $hl_{ui}m$, from *u*nique *h*ub *i*dentifiers
    to the *set* of *u*nique *l*ink *i*dentifiers of the links connected
    to the zero, one or more identified hubs,
47. the *map*, $lh_{ui}m$, from *u*nique *l*ink *i*dentifiers
    to the *set* of *u*nique *h*ub *i*identifiers of the two hubs connected
    to the identified link;
48. the *set*, $a_{ui}s$, of *u*nique *a*utomobile *i*dentifiers;

value

43 $h_{ui}s$:H_UI-**set** ≡ {uid_H(h)|h:H·h ∈ hs}

44 $l_{ui}s$:L_UI-**set** ≡ {uid_L(l)|l:L·l ∈ ls}

45 $r_{ui}s$:R_UI-**set** ≡ $h_{ui}s \cup l_{ui}s$

46 $hl_{ui}m$:(H_UI $\underset{m}{\rightarrow}$L_UI-**set**) ≡

46   [h_ui↦luis|h_ui:H_UI,luis:L_UI-**set**·h_ui∈$h_{ui}s$∧(_,luis,_)=mereo_H(η(h_ui))]

47 $lh_{ui}m$:(L+UI $\underset{m}{\rightarrow}$H_UI-**set**) ≡

47   [l_ui↦huis | h_ui:L_UI,huis:H_UI-**set** · l_ui∈$l_{ui}s$ ∧ (_,huis,_)=mereo_L(η(l_ui))]

48 $a_{ui}s$:A_UI-**set** ≡ {uid_A(a)|a:A·a ∈ as}

5.2.5 A Domain Law: Uniqueness of Endurant Identifiers

- We postulate that the unique identifier observer functions
  - are about the uniqueness of the postulated endurant identifiers,
  - but how is that guaranteed ?
  - We know, as *"an indisputable law of domains"*,
  - that they are distinct,
  - but our formulas do not guarantee that !
  - So we must formalise their uniqueness.

All Domain Parts have Unique Identifiers

A Domain Law: 1 ***All Domain Parts have Unique Identifiers:***

49. All parts of a described domain have unique identifiers.

```
axiom
49  card calc_parts({uod}) = card all_uniq_ids()
```

Example 38 . Uniqueness of Road Net Identifiers:

- We must express the following axioms:

50. All hub identifiers are distinct.
51. All link identifiers are distinct.
52. All automobile identifiers are distinct.
53. All part identifiers are distinct.

axiom

50 $\textbf{card}\, hs = \textbf{card}\, h_{ui}s$

51 $\textbf{card}\, ls = \textbf{card}\, l_{ui}s$

52 $\textbf{card}\, as = \textbf{card}\, a_{ui}s$

53 $\textbf{card}\, \{h_{ui}s \cup l_{ui}s \cup bc_{ui}s \cup b_{ui}s \cup a_{ui}s\}$

53 $\quad = \textbf{card}\, h_{ui}s + \textbf{card}\, l_{ui}s + \textbf{card}\, bc_{ui}s + \textbf{card}\, b_{ui}s + \textbf{card}\, a_{ui}s$ ■

- We ascribe, in principle, unique identifiers
  - to all endurants
    - ⋆ whether natural
    - ⋆ or artefactual.
- We find, from our many experiments,
  cf. the *Universes of Discourse* example, Page 52,
  - that we really focus on those domain entities which are
    - ⋆ artefactual endurants and
    - ⋆ their behavioural “counterparts”.

Example 39 . Rail Net Unique Identifiers:

54. With every rail net unit we associate a unique identifier.
55. That is, no two rail net units have the same unique identifier.
56. Trains have unique identifiers.
57. We let *tris* denote the set of all train identifiers.
58. No two distinct trains have the same unique identifier.
59. Train identifiers are distinct from rail net unit identifiers.

type
54. UI
value
54. uid_NU: NU $\rightarrow$ UI
axiom
55. $\forall$ ui_i,ui_j:UI $\cdot$ ui_i = ui_j $\equiv$ uid_NU(ui_i)=uid_NU(ui_j)

5.2.5.1 Part Retrieval

- Given the unique identifier, pi, of a part p,
- but not the part itself,
- and given the universe-of-discourse (uod) state σ,
- we can *retrieve* part, p, as follows:

```
value
    pi:PI, uod:UoD, $\sigma$
    retr_part: UI $\rightarrow$ P
    retr_part(ui) $\equiv$ let p:P $\cdot$ p $\in \sigma \wedge$ uid_P(p)=ui in p end
      pre: $\exists$ p:P $\cdot$ p $\in \sigma \wedge$ uid_P(p)=ui
```

5.2.5.2 Unique Identification of Compounds

- For structures we do not model their unique identification.
  - But their components,
    - whether the structures are *"Cartesian"*
    - or *"sets"*,
  - may very well be non-structures, hence be uniquely identifiable.

5.3 Mereology

Definition 48 . ***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 [25, 9].

5.3.1 Endurant Relations

- Which are the relations
  that can be relevant for "endurant-hood"?
- There are basically two relations:
  - (i) physical ones, and
  - (ii) conceptual ones.
- (i) Physically two or more endurants may be topologically
  - either adjacent to one another, like rails of a line,
  - or within an endurant, like links and hubs of a road net,
  - or an atomic part is conjoined to one or more fluids,
  - or a fluid is conjoined to one or more parts.

- The latter two could also be considered conceptual “adjacencies”.
- (ii) Conceptually some parts, like automobiles,
  - “belong” to an embedding endurant,
    - ⋆ like to an automobile club, or
    - ⋆ are registered in the local department of vehicles,
  - or are ‘intended’ to drive on roads.

5.3.2 Mereology Modelling Tools

- When the domain analyser decides that
  - some endurants are related in
    a specifically enunciated mereology,
  - the analyser has to decide on suitable
    - ⋆ ***mereology types*** and
    - ⋆ ***mereology observers*** (i.e., endurant relations).

60. We may, to illustration, define a ***mereology type*** of an endurant $e{:}E$ as a triplet type expression over set of unique [endurant] identifiers.
61. There is the identification of all those endurant sorts $E_{i_1}, E_{i_2}, ..., E_{i_m}$ where at least one of whose properties "`is_of_interest`" to parts $e{:}E$.
62. There is the identification of all those sorts $E_{io_1}, E_{io_2}, ..., E_{io_n}$ where at least one of whose properties "`is_of_interest`" to endurants $e{:}E$ and vice-versa.
63. There is the identification of all those endurant sorts $E_{o_1}, E_{o_2}, ..., E_{o_o}$ for whom properties of $e{:}E$ "`is_of_interest`" to endurants of sorts $E_{o_1}, E_{o_2}, ..., E_{o_o}$.
64. The mereology triplet sets of unique identifiers are disjoint and are all unique identifiers of the universe of discourse.

- The triplet 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 endurants of that domain.
- We leave out further characterisation of
  - the seemingly vague notion "is_of_interest".

```
type
61  iEI  = iEI1 | iEI2 | ... | iEIm
62  ioEI = ioEI1 | ioEI2 | ... | ioEIn
63  oEI  = oEI1 | oEI2 | ... | oEIo
60  MT = iEI-set × ioEI-set × oEI-set
axiom
64  ∀ (iset,ioset,oset):MT ·
64     card iset + card ioset + card oset = card ∪{iset,ioset,oset}
64     ∪{iset,ioset,oset} ⊆ calc_all_unique_identifiers(uod)
```

Domain Description Prompt 6 . `describe_mereology(e)`:

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

5. describe_mereology(e) *Observer*

❝ **Narration:**

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

Formalisation:

type
[t] MT = $\mathcal{M}$(UI$_i$,UI$_j$,...,UI$_k$)
value
[m] mereo_P: P $\rightarrow$ MT
axiom [Well–formedness of Domain Mereologies]
[a] $\mathcal{A}$: $\mathcal{A}$(MT) ❞

- The mereology type name, MT,
  chosen of course, by the *domain analyser cum describer*,
  usually properly embodies the type name, E,
  of the endurant being analysed and mereology-described.
- The mereology type expression $\mathcal{M}$(UI$_i$,UI$_j$,...,UI$_k$)
  is a type expression over unique identifiers.
  - Thus a part of type-name P
    might be given the mereology type name MP.
- $\mathcal{A}$(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 ■

Example 40 . Mereology of a Road Net:

65. The mereology of hubs is a pair:
    (i) the set of all automobile identifiers[8], and
    (ii) the set of unique identifiers of the links that it is connected to and the set of all unique identifiers of all automobiles.[9]
66. 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.
67. The mereology of an automobile is
    the set of the unique identifiers of all links and hubs[10].

- We presently omit treatment of road net and automobile aggregate mereologies.
- For road net mereology we refer to Example 69, Item 153 on Slide 416.

type

65 H_Mer = V_UI-**set**×L_UI-**set**

66 L_Mer = V_UI-**set**×H_UI-**set**

67 A_Mer = R_UI-**set**

value

65 mereo_H: H → H_Mer

66 mereo_L: L → L_Mer

67 mereo_A: A → A_Mer

[7] 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.

[8] The 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.

[9] — that the automobile might pass through

5.3.2.1 Invariance of Mereologies

- For mereologies one can usually express some invariants.
  - Such invariants express *"law-like properties"*,
  - facts which are indisputable.

Example 41 . Invariance of Road Nets:

- The observed mereologies
  must express identifiers of the state of such for road nets:

axiom

```
65  ∀ (auis,luis):H_Mer · luis⊆$l_{ui}s$ ∧ auis=$a_{ui}s$
66  ∀ (auis,huis):L_Mer · auis=$a_{ui}s$ ∧ huis⊆$h_{ui}s$ ∧ **card** huis = 2
67  ∀ ruis:A_Mer · ruis=$r_{ui}s$
```

68. For all hubs, *h*, and links, *l*, in the same road net,
69. if the hub *h* connects to link *l*
    then link *l* connects to hub *h*.

axiom

```
68  ∀ h:H,l:L · h ∈ $hs$ ∧ l ∈ $ls$ ⇒
68      let (_,luis)=mereo_H(h), (_,huis)=mereo_L(l)
69      in uid_L(l)∈luis ≡ uid_H(h)∈huis end
```

70. For all links, l, and hubs, h_a, h_b, in the same road net,
71. if the l connects to hubs h_a and h_b,
    then h_a and h_b both connects to link l.

```
axiom
70  ∀ h_a,h_b:H,l:L · {h_a,h_b} ⊆ hs ∧ l ∈ ls ⇒
70      let (_,luis)=mereo_H(h), (_,huis)=mereo_L(l)
71      in uid_L(l)∈luis ≡ uid_H(h)∈huis end
```

5.3.2.2 Deductions made from Mereologies

- Once we have settled basic properties of the mereologies of a domain
  - we can, like for unique identifiers, cf. Example 36 on Slide 190,
  - *"play around"* with that concept:
    'the mereology of a domain'.

Example 42 . Consequences of a Road Net Mereology:

72. are there [isolated] units from which one can not "reach" other units ?
73. does the net consist of two or more "disjoint" nets ?
74. et cetera.

- We leave it to the reader to narrate and formalise the above properly.

5.3.3 Formulation of Mereologies

- The `observe_mereology` domain descriptor, Slide 211,
  - 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 had their unique identifiers defined.

5.3.4 Fixed and Varying Mereologies

- The mereology of parts is not necessarily fixed.

Definition 49 . *Fixed Mereology:*

- By a **fixed mereology** we shall understand
  - a mereology of a part
  - which remains fixed
  - over time.

Definition 50 . *Varying Mereology:*

- By a **varying mereology** we shall understand
  - a mereology of a part
  - which may vary
  - over time.

Example 43 . Fixed and Varying Mereology:

- Let us consider a road net[10].
  - If hubs and links never change “affiliation”, that is:
    - ⋆ hubs are in fixed relation to zero one or more links, and
    - ⋆ links are in a fixed relation to exactly two hubs
    - ⋆ then the mereology of Example 40 on Slide 213 is a *fixed mereology*.

[10] cf. Examples 21 on Slide 54,
29 on Slide 106,
30 on Slide 119,
32 on Slide 129,
35 on Slide 177,
36 on Slide 190,
38 on Slide 198,
39 on Slide 200,
40 on Slide 213 and
41 on Slide 216

- If, on the other hand
  - ⋆ hubs may be inserted into or removed from the net, and/or
  - ⋆ links may be removed from or inserted between any two existing hubs,
  - ⋆ then the mereology of Example 40 on Slide 213 is a *varying mereology*.

5.3.5 No Fluids Mereology

- We comment on our decision, for this *primer*,
  to not endow fluids with mereologies.
  - A first reason is that
    we “restrict” the concept of mereology
    to part endurants,
    that is, to solid endurants –
    those with “more-or-less” *fixed extents*.
  - Fluids can be said to normally not have fixed extents,
    that is, they can “morph” from small, fixed
    into spatially extended forms.

  - For domains of part-fluid conjoins this is particularly true.
  - The fluids in such domains flow through and between parts.
  - Some parts, at some times, embodying large, at other times small amounts of fluid.
  - Some proper, but partial amount of fluid flowing from one part to a next.
  - Et cetera.
  - It is for the same reason that we do not endow fluids with identity.
- So, for this *primer* we decide to not suggest the modelling of fluid mereologies.

5.3.6 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 52, we have found no really interesting such cases.
- All our experimental case studies appears to focus on the mereology of artefacts.

- And, finally, in modelling humans,
  - we find that their mereology encompass
    - all other humans
    - and all artefacts !
  - 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”.
- Some domain models may emphasize *physical mereologies* based on spatial relations,
- others may emphasize *conceptual mereologies* based on logical “connections”.

Example 44 . Rail Net Mereology:

75. A linear rail unit is connected to exactly
two distinct other rail net units of any given rail net.
76. A point unit is connected to exactly
three distinct other rail net units of any given rail net.
77. A rigid crossing unit is connected to exactly
four distinct other rail net units of any given rail net.
78. A single and a double slip unit is connected to exactly
four distinct other rail net units of any given rail net.
79. A terminal unit is connected to exactly
one distinct other rail net unit of any given rail net.
80. So we model the mereology of a railway net unit
as a pair of sets of rail net unit unique identifiers
distinct from that of the rail net unit.

```
value
80.  mereo_NU: NU → (UI-set×UI-set)
axiom
80.  ∀ nu:NU ·
80.     let (uis_i,uis_o)=mereo_NU(nu) in
80.     case (card uis_i,card usi_o) =
75.         (is_LU(nu) → (1,1),
76.          is_PU(nu) → (1,2) ∨ (2,1),
77.          is_RU(nu)  → (2,2),
78.          is_SU(nu) → (2,2), is_DU(nu) → (2,2),
79.          is_TU(nu)  → (1,0) ∨ (0,1),
80.          _ → chaos) end
80.    ∧ uis_i∩uis_o={}
80.    ∧ uid_NU(nu) ∉ (uis_i ∪ uis_o)
80.     end
```

- Figure 5.1
  - illustrates the mereology of four rail units.

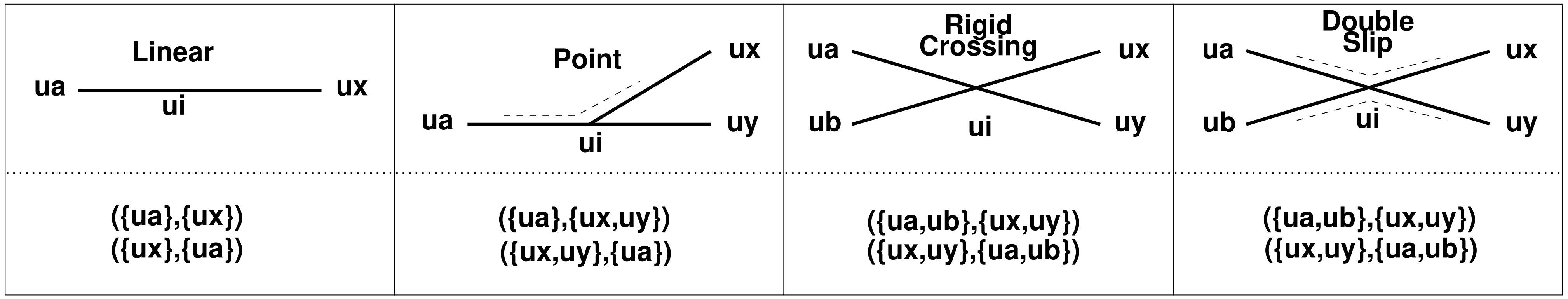

Figure 5.1: Four Symmetric Rail Unit Mereologies

Lecture 4: Attributes and Summary

5.4 Attributes

- To recall: there are three sets of *internal qualities*:
  - unique identifiers,
  - mereologies and
  - attributes.
- Unique identifiers and mereologies
  are rather definite kinds of internal endurant qualities;
- attributes form more "free-wheeling" sets of *internal qualities*.
- Whereas, for this *primer*, we suggest to not endow
  fluids with unique identification and mereologies
  all endurants, i.e., including fluids, are endowed with attributes.

5.4.1 Inseparability of Attributes from Parts and Fluids

- Parts and fluids are
  - typically recognised because of their spatial form
  - and are otherwise characterised by their intangible, but measurable attributes.
- We equate all endurants — which have

  the same type of unique identifiers,

  the same type of mereologies,

  and the same types of attributes

  — with one sort.
- Thus removing an internal 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) !

- We can roughly distinguish between two kinds of attributes:
  - those which can be motivated
    by **physical** (incl. chemical) **concerns**, and
  - those,
    - which, although they embody some form of ‘physics measures’,
    - appear to reflect on **event histories**:
      - *“if ‘something’, ϕ, has ‘happened’ to an endurant, e_a,*
      - *then some ‘commensurate thing’, ψ, has ‘happened’ to another (one or more) endurants, e_b.”*
    - where the *‘something’* and *‘commensurate thing’*
    - usually involve some ‘interaction’ between the two (or more) endurants.
  - It can take some reflection and analysis to properly identify
    - endurants e_a and e_b and
    - commensurate events ϕ and ψ.

5.4.2 Attribute Modelling Tools

5.4.2.1 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.

5.4.2.2 Concrete Attribute Types

- By a *concrete type* shall understand a sort (i.e., a type) which is defined in terms of some type expression: $\mathsf{T} = \mathcal{T}(...)$.
- This is referred to below as [=...].

5.4.2.3 Attribute Types and Functions

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

Domain Description Prompt 7 . `describe_attributes:`

- *The domain analyser experiments, thinks and reflects about endurant,* e, *attributes.*
- *That process is initiated by the* domain description prompt*:*
  - `describe_attributes(e)`.
- *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:*

6. describe_attributes Observer

let $\{\eta A_1, ..., \eta A_m\}$ = analyse_attribute_type_names(e) **in**

❝ **Narration:**

```
    [t]  ... narrative text on attribute sorts ...
           some Ais may be concretely defined: [Ai=...]
    [o]  ... narrative text on attribute sort observers ...
    [p]  ... narrative text on attribute sort proof obligations ...
```

Formalisation:

```
    type
    [t]  $A_1$[=...] , ..., $A_m$[=...]
    value
    [o]  attr_$A_1$: E→$A_1$, ..., attr_$A_m$: E→$A_m$
    proof obligation [Disjointness of Attribute Types]
    [p]  $\mathcal{PO}$: let P be any part sort in [the domain description]
    [p]        let a:($A_1$|$A_2$|...|$A_m$) in is_$A_i$(a) ≠ is_$A_j$(a) [i≠i, i,j:[1..m]] end end ❞
end
```

- Let $\mathsf{A}_1, ..., \mathsf{A}_n$ be the set of all conceivable attributes of endurants $e{:}E$.
  - (Usually n is a rather large natural number, say in the order of a hundred conceivable such.)
  - In any one domain model the domain analyser cum describer selects a modest subset, $\mathsf{A}_1, ..., \mathsf{A}_m$, i.e., $m < n$.
  - Across many domain models
    for *"more-or-less the same"* domain m varies
    and the attributes, $\mathsf{A}_1, ..., \mathsf{A}_m$,
    selected for one model may differ
    from those, $\mathsf{A}'_1, ..., \mathsf{A}'_{m'}$, chosen for another model.

- The **type** definitions: A_1, ..., A_m, inform us that the domain analyser has decided to focus on the distinctly named A_1, ..., A_m attributes.
- The **value** clauses
  - $\mathsf{attr\_A}_1\mathsf{:P}\rightarrow\mathsf{A}_1$,
  - ...,
  - $\mathsf{attr\_A}_n\mathsf{:P}\rightarrow\mathsf{A}_n$

  are then "automatically" given:
  - if an endurant, e:E, has an attribute A_i
  - then there is postulated, "by definition" [eureka] an attribute observer function $\mathsf{attr\_A}_i\mathsf{:E}\rightarrow\mathsf{A}_i$ et cetera ■

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

  Therefore we must prove it.

5.4.2.4 Attribute Categories

- Michael A. Jackson [31] has suggested a hierarchy of attribute categories:
  - from static
  - to dynamic values – and within the dynamic value category:
    - ⋆ inert values,
    - ⋆ reactive values,
    - ⋆ active values – and within the dynamic active value category:
      - · autonomous values,
      - · biddable values and
      - · programmable values.
- We now review these attribute value types.
  The review is based on [31, M.A.Jackson].

- *Endurant attributes* are
  - either constant, i.e., static,
  - or varying, i.e., 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 ■

Example 45 . Static Attributes:

- Let us exemplify road net attributes
  in this and the next examples.
- And let us assume the following attributes:
  - year of first link construction and
  - link length at that time.
- We may consider both to be static attributes:
  - The year first established,
    seems an obvious static attribute and
  - the length is fixed at the time the road was first built.

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 ■

Example 46 . Inert Attribute:

- And let us now further assume the following link attribute:
  - link name.
- We may consider it to be an inert attribute:
  - the name is not “assigned” to the link by the link itself,
  - but probably by some road net authority
  - which we are not modelling.

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
    - ⋆ either come from outside the domain of interest
    - ⋆ or from other endurants ■

Example 47 . Reactive Attributes:

- Let us further assume the following two link attributes:
  - "wear and tear", respectively
  - "icy and slippery".
- We will consider those attributes to be reactive in that
  - automobiles (another part) traveling the link, an external "force",
    typically causes the "wear and tear", respectively
  - the weather (outside our domain)
    causes the "icy and slippery" property.

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*,
  - or *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".
  - The values of an autonomous attributes are a "law onto themselves and their surroundings" ■

Example 48 . Autonomous Attributes:

- We enlarge scope of our examples of attribute categories to now also include automobiles (on the road net).
- In this example we assume that an automobile is driven by a human [behaviour].
- These are some automobile attributes:
  - velocity,
  - acceleration, and
  - moving straight, or turning left, or turning right.
- We shall consider these three attributes to be autonomous.
  - It is the driver, not the automobile, who decides
  - whether the automobile should drive at constant velocity, including 0, or accelerate or decelerate, including stopping.
  - And it is the driver who decides when to turn left or right, or not turn at all.

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* ■

Example 49 . Biddable Attributes: In the context of automobiles these are some biddable attributes:

- turning the wheel, to drive right at a hub
  – with the automobile failing to turn right;
- pressing the accelerator, to obtain a higher speed
  – with the automobile failing to really gaining speed;
- pressing the brake, to stop
  – with the automobile failing to halt ■

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 ■

Example 50 . Programmable Attribute:

- We continue with the automobile on the road net examples.
- In this example we assume that an automobile includes, as one inseparable entity, “the driver”.
- These are some automobile attributes:
  - position on a link,
  - velocity, acceleration (incl. deceleration), and
  - direction: straight, turning left, turning right.
- We shall now consider these three attributes to be programmable.

- Figure 5.2 captures an attribute value ontology.

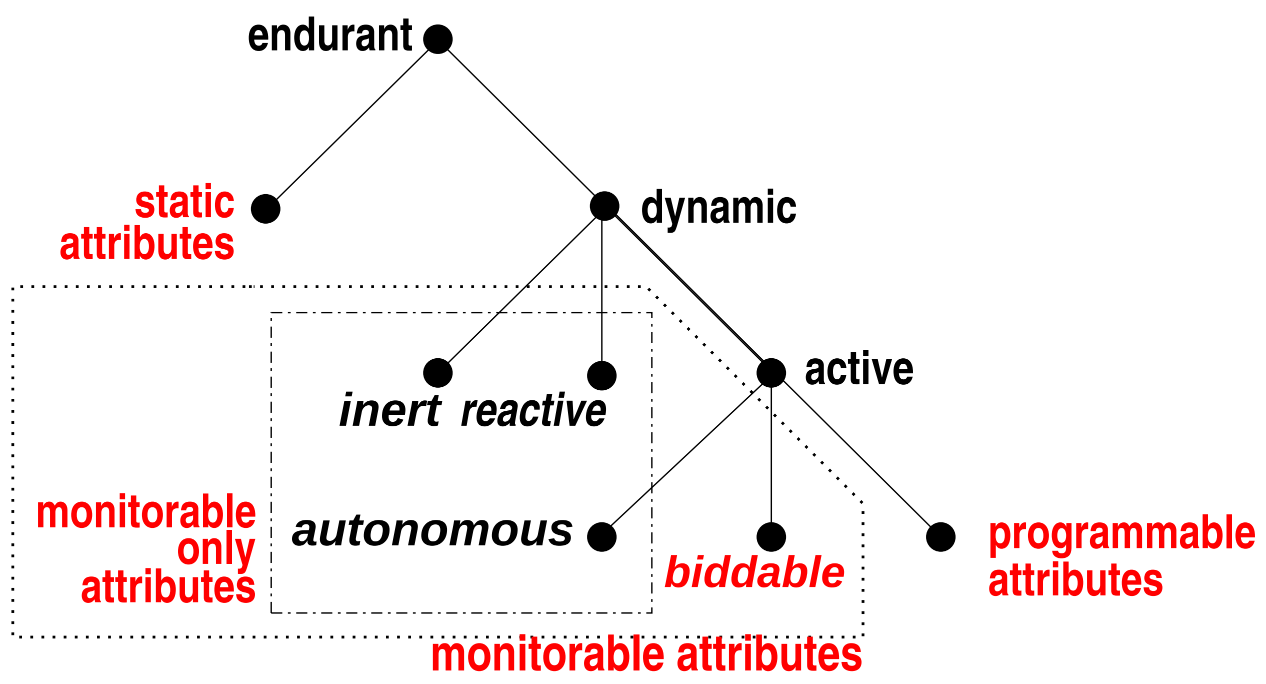

Figure 5.2: Attribute Value Ontology

- Figure 5.2 hints at three categories of dynamic attributes:
  - monitorable only,
  - biddable and
  - programmable

  attributes.

Attribute Category: 9 .

- By a *monitorable only attribute*, a:A,
  `is_monitorable_only_attribute`(a),
  we shall understand a dynamic active attribute which is either
  - *inert* or
  - *reactive* or
  - *autonomous*.

That is:

value
  is_monitorable_only: E $\rightarrow$ **Bool**
  is_monitorable_only(e) $\equiv$ is_inert(e) $\vee$ is_reactive(e) $\vee$ is_autonomous(e)

Example 51 . Road Net Attributes:

- We treat some attributes of the hubs of a road net.

81. There is a hub state.
    - It is a set of pairs, $(\mathsf{l}_f,\mathsf{l}_t)$, of link identifiers,
      - where these link identifiers are in the mereology of the hub.
    - The meaning of the hub state
      - in which, e.g., $(\mathsf{l}_f,\mathsf{l}_t)$ is an element,
      - is that the hub is open, “green”,
      - for traffic *f*rom link l_f *t*o 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.

82. There is a hub state space.
    - It is a set of hub states.
    - The current hub state must be in its state space.
    - The meaning of the hub state space is
      - that its states are all those the hub can attain.
83. Since we can think rationally about it,
    - it can be described, hence we 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.
    - Hub history is an *event history*.

type
81 $\mathsf{H\Sigma = (L\_UI{\times}L\_UI)}$**-set**
82 $\mathsf{H\Omega = H\Sigma}$**-set**
83 $\mathsf{H\_Traffic = (A\_UI|B\_UI) \xrightarrow[m]{} (\mathbb{TIME} \times VPos)^*}$
axiom
81 $\mathsf{\forall\ h{:}H \cdot obs\_H\Sigma(h) \in obs\_H\Omega(h)}$
83 $\mathsf{\forall\ ht{:}H\_Traffic,ui{:}(A\_UI|B\_UI) \cdot\ ui \in}$ **dom** $\mathsf{ht \Rightarrow time\_ordered(ht(ui))}$
value
81 $\mathsf{attr\_H\Sigma{:}\ H \rightarrow H\Sigma}$
82 $\mathsf{attr\_H\Omega{:}\ H \rightarrow H\Omega}$
83 $\mathsf{attr\_H\_Traffic{:}\ H \rightarrow H\_Traffic}$
83 $\mathsf{time\_ordered{:}\ (\mathbb{TIME} \times VPos)^* \rightarrow}$ **Bool**
83 $\mathsf{time\_ordered(tvpl) \equiv \ldots}$

- In Item 83 we model the time-ordered sequence of traffic as a discrete sampling, i.e., $\xrightarrow[m]{}$, rather than as a continuous function, $\rightarrow$.

Example 52 . Invariance of Road Net Traffic States:

- We continue Example 51 on Slide 259.

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

axiom

```
84  ∀ h:H · h ∈ $hs$ ⇒
84      **let** h$\sigma$ = attr_H$\Sigma$(h) **in**
84      ∀ (l$_{ui}i$,li$_{ui}i'$):(L_UI×L_UI) · (l$_{ui}i$,l$_{ui}i'$) ∈ h$\sigma$ ⇒ {l$_{ui_i}$,l$'_{ui_i}$} ⊆ l$_{ui}s$ **end**
```

- You may skip Example 53 in a first reading.

Example 53 . Road Transport: Further Attributes:

Links:

We show just a few attributes.

85. 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 *t*o hub h_t. Link states can have either 0, 1 or 2 elements.
86. 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".

87. 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.
88. The hub identifiers of link states must be in the set, $h_{ui}s$, of the road net's hub identifiers.

```
type
85  LΣ = H_UI-set
86  LΩ = LΣ-set
87  L_Traffic
87  L_Traffic = (A_UI|B_UI) $\overrightarrow{m}$ ($\mathbb{T}$×(H_UI×Frac×H_UI))$^*$
87  Frac = Real, axiom frac:Fract · 0<frac<1
value
85  attr_LΣ: L → LΣ
86  attr_LΩ: L → LΩ
87  attr_L_Traffic: : → L_Traffic
axiom
85  ∀ l$\sigma$:LΣ·card l$\sigma$=2
85  ∀ l:L · obs_LΣ(l) ∈ obs_LΩ(l)
87  ∀ lt:L_Traffic,ui:(A_UI|B_UI)·ui ∈ dom ht ⇒ time_ordered(ht(ui))
88  ∀ l:L · l ∈ $ls$ ⇒  let l$\sigma$ = attr_LΣ(l) in ∀ (h$_{ui}i$,h$_{ui}i'$):(H_UI×K_UI) ·
88                       (h$_{ui}i$,h$_{ui}i'$) ∈ l$\sigma$ ⇒ {h$_{ui_i}$,h$'_{ui_i}$} ⊆ h$_{ui}s$ end
```

Automobiles:

- We illustrate but a few attributes:

89. Automobiles have static number plate registration numbers.
90. Automobiles have dynamic positions on the road net:
    (a) either *at a hub* identified by some h_ui,
    (b) 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.
    (c) Fraction is a real properly between 0 and 1.

```
type
89   RegNo
90   APos == atHub | onLink
90a  atHub :: h_ui:H_UI
90b  onLink :: fh_ui:H_UI × l_ui:L_UI × frac:Fract × th_ui:H_UI
90c  Fract = Real
axiom
90c  frac:Fract · 0<frac<1
value
89   attr_RegNo: A → RegNo
90   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.

- In Items Sli. 260 and Sli. 264, 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[11] ■

[11] In this day and age of road cameras and satellite surveillance these traffic recordings may not appear so strange: We now know, at least in principle, of technologies that can record approximations to the hub and link traffic attributes.

5.4.2.5 Calculating Attribute Category Type Names

- One can calculate sets of all attribute type names, of static, so-called monitorable and programmable attribute types of parts and fluids with the following ***domain analysis prompt***s:
  - `analyse_attribute_type_names`,
  - `sta_attr_types`,
  - `mon_attr_types`, and
  - `pro_attr_types`.

  - `analyse_attribute_type_names` applies to parts and yields a set of all attribute names of that part.
  - `mon_attr_types` applies to parts and yields a set of attribute names of *monitorable* attributes of that part.[12]

[12] $\eta\mathbb{A}$ is the type of all attribute types.

Observer Function Prompt 6 . *analyse_attribute_types:*

value
    analyse_attribute_type_names: P $\rightarrow$ ηA-**set**
    analyse_attribute_type_names(p) **as** {ηA1,ηA,...,ηAm }

Observer Function Prompt 7 . *sta_attr_types:*

```
value
    sta_attr_types: P $\rightarrow \eta\mathbb{A}\times\eta\mathbb{A}\times...\times\eta\mathbb{A}$
    sta_attr_types(p) as ($\eta$A1,$\eta$A2,...,$\eta$An)
        where: {$\eta$A1,$\eta$A2,...,$\eta$An} $\subseteq$ analyse_attribute_type_names(p)
            $\wedge$ let anms = analyse_attribute_type_names(p)
             $\forall$ anm:$\eta\mathbb{A}$ $\cdot$ anm $\in$ anms \ {$\eta$A1,$\eta$A2,...,$\eta$An}
              $\Rightarrow$ $\sim$ is_static_attribute{anm}
            $\wedge$ $\forall$ anm:$\eta\mathbb{A}$ $\cdot$ anm $\in$ {$\eta$A1,$\eta$A2,...,$\eta$An}
              $\Rightarrow$ is_static_attribute{anm} end
```

Observer Function Prompt 8 . *mon_attr_types:*

```
value
    mon_attr_types: P → $\eta\mathbb{A}\times\eta\mathbb{A}\times\ldots\times\eta\mathbb{A}$
    mon_attr_types(p) as ($\eta$A1,$\eta$A2,...,$\eta$An)
        where:  {$\eta$A1,$\eta$A2,...,$\eta$An} ⊆ analyse_attribute_type_names(p)
             ∧ let anms = analyse_attribute_type_names(p)
              ∀ anm:$\eta\mathbb{A}$ · anm ∈ anms \ {$\eta$A1,$\eta$A2,...,$\eta$An}
                ⇒ ∼ is_monitorable_attribute{anm}
             ∧ ∀ anm:$\eta\mathbb{A}$ · anm ∈ {$\eta$A1,$\eta$A2,...,$\eta$An}
                ⇒ is_monitorable_attribute{anm} end
```

Observer Function Prompt 9 . *pro_attr_types:*

```
value
   pro_attr_types: P → $\eta\mathbb{A}\times\eta\mathbb{A}\times\ldots\times\eta\mathbb{A}$
   pro_attr_types(p) as ($\eta$A1,$\eta$A2,...,$\eta$An)
      where:  {$\eta$A1,$\eta$A2,...,$\eta$An} ⊆ analyse_attribute_type_names(p)
          ∧ let anms = analyse_attribute_type_names(p)
            ∀ anm:$\eta\mathbb{A}$ · anm ∈ anms \ {$\eta$A1,$\eta$A2,...,$\eta$An}
             ⇒ ∼ is_monitorable_attribute{anm}
          ∧ ∀ anm:$\eta\mathbb{A}$ · anm ∈ {$\eta$A1,$\eta$A2,...,$\eta$An}
             ⇒ is_monitorable_attribute{anm} end
```

- Some comments are in order.
  - The analyse_attribute_type_names function is, as throughout, meta-linguistic, that is, informal, not-computable, but decidable by the domain analyser cum describer. Applying it to a part or fluid yields, at the discretion of the domain analyser cum describer, a set of attribute type names "freely" chosen by the domain analyser cum describer.
  - The sta_attr_type_names, the mon_attr_type_names, and the pro_attr_type_names functions are likewise meta-linguistic; their definition here relies on the likewise meta-linguistic is_static, is_monitorable and is_programmable analysis predicates.

5.4.2.6 Calculating Attribute Values

- Let (ηA1, ηA2, ... , ηAn) be a grouping of attribute types for part p (or fluid f).
- Then (attr_A1(p), attr_A2(p), ... ,attr_An(p))
- (respectively f)
- yields (a1, a2, ... , an), the grouping of values for these attribute types.
- We can "formalise" this conversion:

value

types_to_values: $\eta\mathbb{A}_1 \times \eta\mathbb{A}_2 \times ... \times \eta\mathbb{A}_n \rightarrow \mathsf{A}_1 \times \mathsf{A}_2 \times ... \times \mathsf{A}_n$

5.4.2.7 Calculating Attribute Names

- The meta-linguistic, i.e., “outside” RSL proper, name for attribute type names is introduced here as $\eta\mathbb{A}$.

91. Given endurant e we can *meta-linguistically*[13] calculate names for its *static* attributes.
92. Given endurant e we can *meta-linguistically* calculate name for its *monitorable* attributes attributes.
93. Given endurant e we can *meta-linguistically* calculate names for its *programmable* attributes.
94. These four sets make up all the attributes of endurant e.

[13] 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.

The type names ST, MA, PT designate mutually disjoint

- sets, ST, of names of static attributes,
- sets, MA, of names of monitoriable, i.e., monitorable-only and biddable, attributes,
- sets, PT, of names of programmable, i.e., fully controllable attributes.

type

91 ST = $\eta\mathbb{A}$-**set**

92 MA = $\eta\mathbb{A}$-**set**

93 PT = $\eta\mathbb{A}$-**set**

value

91 stat_attr_types: E $\rightarrow$ ST

92 moni_attr_types: E $\rightarrow$ MA

93 prgr_attr_types: E $\rightarrow$ PT

```
axiom
94  ∀ e:E ·
91  let stat_nms = stat_attr_types(e),
92      moni_nms = moni_attr_types(e),
93      prgr_nms = prgr_types(e) in
94  card stat_nms + card moni_nms + card prgr_nms
94  = card(stat_nms ∪ mon_nms ∪ prgr_nms) end
```

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

95. Given endurant *e* we can *meta-linguistically* calculate its static attribute values, stat_attr_vals;
96. given endurant *e* we can *meta-linguistically* calculate its monitorable-only attribute values, moni_attr_vals; and
97. given endurant *e* we can *meta-linguistically* calculate its programmable attribute values, prgr_attr_vals.

The type names sa1, ..., pap refer to the types denoted by the corresponding types name nsa1, ..., npap.

```
value
95 stat_attr_vals: E → SA1×SA2×...×SAs
95 stat_attr_vals(e) ≡
95    let {nsa1,nsa2,...,nsas} = stat_attr_types(e) in
95    (attr_sa1(e),attr_sa2(e),...,attr_sas(e)) end

96 moni_attr_vals: E → MA1×MA2×...×MAm
96 moni_attr_vals(e) ≡
96    let {nma1,nma2,...,nmam} = moni_attr_types(e) in
96    (attr_ma1(e),attr_ma2(e),...,attr_mam(e)) end
```

```
97 prgr_attr_vals: E → PA1×PA2×...×PAp
97 prgr_attr_vals(e) ≡
97     let {npa1,npa2,...,npap} = prgr_attr_types(e) in
97     (attr_pa1(e),attr_pa2(e),...,attr_pap(e)) end
```

- The “ordering” of type values,
  - (attr_sa1(e),...,attr_sas(e)),
  - (attr_ma1(e),...,attr_mam(e)), et cetera,
  - is arbitrary.

5.4.3 Operations on Monitorable Attributes of Parts

- We remind the student of the notions of
  - states in general, Sect. 4.7 and
  - updateable states, Sect. 4.7.2 on Slide 155.
    - ⋆ For every domain description there possibly is an updateable state.
    - ⋆ The is such a state if there is at least one part with at least one monitorable attribute.
  - Below we refer to the updateable states as σ.

- Given a part, p, with attribute A,
  - the simple operation attr_A(p)
  - thus yields the value of attribute A
  - for that part.
- But what if, what we have is just
  - the global state σ, of the set of all monitorable parts of a given universe-of-discourse, uod,
  - the unique identifier, uid_P(p), of a part of σ, and
  - the name, ηA, of an attribute of p ?
    - ⋆ Then how do we
      - · ascertain the attribute value for A of p,
      - · and, for *biddable* attributes A,
      - · "update" p, in σ, to some A value ?
    - ⋆ Here is how we express these two issues.

5.4.3.1 Evaluation of Monitorable Attributes

98. Let pi:PI be the unique identifier of any part, p, with monitorable attributes, let A be a monitorable attribute of p, and let ηA be the name of attribute A.
99. Evaluation of the [current] attribute A value of p is defined by function read_A_from_P – retr_part(pi) is defined in Sect. 5.2.5.1 on Slide 201.

value

98. pi:PI, a:A, ηA:$\eta\mathbb{T}$

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

5.4.3.2 Update of Biddable Attributes

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

value

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

100.  update_P_with_A: PI $\times$ A $\times$ $\eta\mathbb{T}$ $\rightarrow$ **write** $\sigma$
100.  update_P_with_A(pi,a,$\eta$A) $\equiv$
101.      **let** p = retr_part(pi) **in**
102.      **let** p′:P $\cdot$
102a.         uid_P(p′)=pi
102b.         $\wedge$ mereo_P(p)=mereo_P(p′)
102c.         $\wedge$ $\forall$ $\eta$A′ **in** analyse_attribute_type_names(p) \ {$\eta$A}
102c.                 $\Rightarrow$ attr_A(p)=attr_A(p′)
102d.         $\wedge$ attr_A(p′)=a **in**
103.      $\sigma$ := $\sigma$ \ {p} $\cup$ {p′}
100.      **end end**
```

5.4.3.3 Stationary and Mobile Attributes

- Endurants are either **stationary** or **mobile**.

Definition 51 . ***Stationary:*** An endurant is said to be stationary if it never moves ■

- Being stationary is a static attribute.

Analysis Predicate Prompt 18 . `is_stationary:`

- *The method provides the* ***domain analysis prompt****:*
  - `is_stationary` – *where* `is_stationary(e)` *holds if e is to be considered stationary* ■

Example 54 . Stationary Endurants:

- Examples of stationary endurants could be:
  - road hubs and links;
  - container terminal stacks;
  - pipeline units; and
  - sea, lake and river beds ■

Definition 52 . ***Mobile:*** An endurant is said to be mobile if it is capable of being moved – whether by its own, or otherwise ■

- Being mobile is a static attribute.

Analysis Predicate Prompt 19 . *`is_mobile:`*

- *The method provides the* ***domain analysis prompt****:*
  - *`is_mobile`* – *where* `is_mobile`*(e)* *holds if e is to be considered mobile* ■

Example 55 . Mobile Endurants:

- Examples of mobile endurants are:
  - automobiles;
  - container terminal vessels, containers, cranes and trucks;
  - pipeline oil (or gas, or water, ...);
  - sea, lake and river water ■

- Being stationary or mobile is an attribute of any manifest endurant.
  - Foe every manifest endurant, *e*, it is the case that
  - `is_stationary(e)≡∼is_mobile(e).`

● ● ●

- Being stationary or, vice-versa, being mobile
  is often **tacitly assumed.**
  - Having external or internal qualities of a certain kind
    is often also tacitly assumed.
  - A major point of the domain analysis & description approach,
    - ⋆ of these lectures,
    - ⋆ is to help the domain analyser cum describer –
    - ⋆ the domain engineer cum researcher –
    - ⋆ to unveil as many, if not all, these qualities.
  - Tacit understanding would not be a common problem
    was it not for us to practice it "excessively" !

5.5 SPACE and TIME

- The two concepts: space and time are not attributes of entities.
- In fact, they are not internal qualities of endurants.
- They are universal qualities of any world.
  - As argued in Sect. **??** on Slide ??, SPACE and TIME are unavoidable concepts of any world.
  - But we can ascribe spatial attributes to any concrete, manifest endurant.
  - And we can ascribe attributes to endurants that record temporal concepts.

5.5.1 $\mathbb{SPACE}$

- Space is just there.
  - So we do not define an observer, observe_space.
  - For us – bound to model mostly artefactual 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 to be isomorphic !
- $\mathbb{SPACE}$ is considered to consist of (an infinite number of) $\mathbb{POINT}$s.

104. We can assume a point observer, observe_$\mathbb{POINT}$, is a function which applies to endurants, e, and yield a point, $pt : \mathbb{POINT}$

104. observe_$\mathbb{POINT}$: E $\to$ $\mathbb{POINT}$

- At which "point" of an endurant, e, observe_$\mathbb{POINT}(e)$, is applied, or
- which of the (infinitely) many points of an endurant E, observe_$\mathbb{POINT}(e)$, yields
  we leave up to the domain analyser cum describer to decide !

- We suggest, besides POINTs, the following spatial attribute possibilities:

105. EXTENT as a dense set of POINTs;
106. Volume, of concrete type, for example, m^3, as the “volume” of an EXTENT such that
107. SURFACEs as dense sets of POINTs have no volume, but an
108. Area, of concrete type, for example, m^2, as the “area” of a dense set of POINTs;
109. LINE as dense set of POINTs with no volume and no area, but
110. Length, of concrete type, for example, m.

- For these we have that

111. the *intersection*, $\bigcap$, of two $\mathbb{EXTENT}$s is an $\mathbb{EXTENT}$ of possibly nil Volume,
112. the intersection, $\bigcap$, of two $\mathbb{SURFACE}$s may be either a possibly nil $\mathbb{SURFACE}$ or a possibly nil $\mathbb{LINE}$, or a combination of these.
113. the intersection, $\bigcap$, of two $\mathbb{LINE}$s may be either a possibly nil $\mathbb{LINE}$ or a $\mathbb{POINT}$.

- Similarly we can define

114. the *union*, $\bigcup$, of two not-disjoint $\mathbb{EXTENT}$s,
115. the *union*, $\bigcup$, of two not-disjoint $\mathbb{SURFACE}$s,
116. the *union*, $\bigcup$, and of two not-disjoint $\mathbb{LINE}$s.

- and:

117. the *[in]equality*, $\neq, =$, of
    pairs of $\mathbb{EXTENT}$,
    pairs of $\mathbb{SURFACE}$s, and
    pairs of $\mathbb{LINE}$s.

- We invite the reader to first
  - first express the signatures for these operations,
  - then their pre-conditions,
  - and finally, being courageous, appropriate fragments of axiom systems.

- We leave it up to the reader to introduce, and hence define, functions that
  - add, subtract, compare, etc.,
  - $\mathbb{EXTENT}$s, $\mathbb{SURFACE}$s, $\mathbb{LINE}$s, etc.

5.5.2 Mathematical Models of Space

- Figure 5.3 on Slide 305 diagrams some mathematical models of space.
- We shall hint[14] at just one of these spaces.

[14] Figure 5.3 on Slide 305 is taken from `https://en.wikipedia.org/wiki/Space_(m`

5.5.2.1 Metric Spaces

Metric Space

Axiom System 1 .

- *A metric space is an ordered pair* (M,d) *where* M *is a set and* d *is a metric on* M*, i.e., a function:*

$$d : M \times M \to \textbf{\textit{Real}}$$

- *such that for any* $x,y,z \in M$*, the following holds:*

$$d(x,y) = 0 \equiv \ x = y \quad \textit{identity of indiscernibles} \tag{5.1}$$

$$d(x,y) = d(y,x) \quad \textit{symmetry} \tag{5.2}$$

$$d(x,z) \leq d(x,y) + d(y,z) \quad \textit{sub-additivity or triangle inequality} \tag{5.3}$$

- *Given the above three axioms, we also have that* $d(x,y) \geq \ 0$ *for any*

$x, y \in M$.

- *This is deduced as follows:*

$$d(x,y) + d(y,x) \geq d(x,x) \quad \textit{triangle inequality} \tag{5.4}$$

$$d(x,y) + d(y,x) \geq d(x,x) \quad \textit{by symmetry} \tag{5.5}$$

$$2d(x,y) \geq 0 \quad \textit{identity of indiscernibles} \tag{5.6}$$

$$d(x,y) \geq 0 \quad \textit{non-negativity} \tag{5.7}$$

- *The function d is also called distance function or simply distance.*
- *Often, d is omitted and one just writes M for a metric space if it is clear from the context what metric is used.*

•

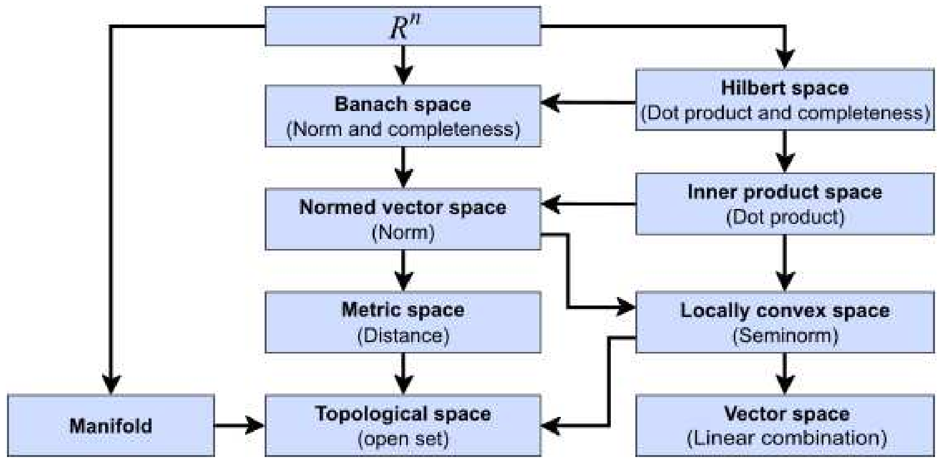

Figure 5.3: Variety of Abstract Spaces. An arrow from space A to space B implies that A is also a kind of B.

5.5.3 TIME

a moving image of eternity;
the number of the movement in respect of the before and the after;
the life of the soul in movement as it passes
from one stage of act or experience to another;
a present of things past: memory,
a present of things present: sight,
and a present of things future: expectations[15]

This thing all things devours:
Birds, beasts, trees, flowers;
Gnaws iron, bites steel,
Grinds hard stones to meal;
Slays king, ruins town,
And beats high mountain down.[16]

[15]Quoted from [1, Cambridge Dictionary of Philosophy]
[16]J.R.R. Tolkien, The Hobbit

- Concepts of time continue to fascinate philosophers and scientists [50, 28, 33, 34, 37, 38, 39, 40, 41, 42, 43] and [29].
- J.M.E. McTaggart (1908, [33, 28, 43]) discussed theories of time around the notions of
  - **"A-series":** with concepts like "past", "present" and "future", and
  - **"B-series":** has terms like "precede", "simultaneous" and "follow".
- Johan van Benthem [50] and
  Wayne D. Blizard [24, 1980]
  relates abstracted entities to spatial points and time.
- A recent computer programming-oriented treatment is given in [29, Mandrioli et al., 2013].

5.5.3.1 Time Motivated Philosophically

Definition 53 . *Indefinite Time::*

- *We motivate the abstract notion of time as follows.*
  - *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* ■

Definition 54 . ***Definite Time::***

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

■

Example 56 . 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 87 on Slide 264 and*
  - *(ii) the time stamped hub traffic, cf. Item 83 on Slide 260* ■

5.5.3.2 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 [29].
- Similarly we shall not be concerned with any representation of time intervals.[17]

118. So there is an abstract type $\mathbb{T}ime$,
119. and an abstract type $\mathbb{TI}$: $\mathbb{T}ime\mathbb{I}nterval$.
120. There is no $\mathbb{T}ime$ origin, but there is a "zero" $\mathbb{TI}$me interval.
121. One can add (subtract) a time interval to (from) a time and obtain a time.

[17]– 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.

122. 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.
123. One can subtract a time from another time obtaining a time interval respecting that the subtrahend is smaller than or equal to the minuend.
124. One can multiply a time interval with a real and obtain a time interval.
125. One can compare two times and two time intervals.

type

118 $\mathbb{T}$

119 $\mathbb{TI}$

value

120 $\mathbf{0}$:$\mathbb{TI}$

121 $+,-$: $\mathbb{T} \times \mathbb{TI} \to \mathbb{T}$

122 $+,-$: $\mathbb{TI} \times \mathbb{TI} \overset{\sim}{\to} \mathbb{TI}$

123 $-$: $\mathbb{T} \times \mathbb{T} \to \mathbb{TI}$

124 $*$: $\mathbb{TI} \times$ **Real** $\to \mathbb{TI}$

125 $<,\leq,=,\neq,\geq,>$: $\mathbb{T} \times \mathbb{T} \to$ **Bool**

125 $<,\leq,=,\neq,\geq,>$: $\mathbb{TI} \times \mathbb{TI} \to$ **Bool**

axiom

121 $\forall$ t:$\mathbb{T}$ $\cdot$ t+$\mathbf{0}$ = t

5.5.3.3 Temporal Observers

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

```
type
126  𝕋
value
126  record_𝕋𝕀𝕄𝔼(): Unit → 𝕋
```

- The time recorder applies to nothing and yields a time.
- **record**_𝕋𝕀𝕄𝔼() can only occur in action, event and behavioural descriptions.

5.5.3.4 "Soft" and "Hard" Real-time

- We loosely identify a spectrum of from "soft" to "hard" temporalities — through some informally worded texts.
- On that background we can introduce the term 'real-time'.
- And hence distinguish between 'soft' and 'hard' real-time issues.
- From an example of trying to formalise these in RSL,
- we then set the course for these next lectures.

5.5.3.4.1 Soft Temporalities

- You have often wished, we assume, that
  "your salary never goes down, say between your ages of 25 to 65".
- How to express that?
- Taking into account other factors, you may additionally wish that
  "your salary goes up."
- How do we express that?
- Taking also into account that your job is a seasonal one, we may need to refine the above into
  "between un-employments your salary does not go down".
- How now to express that?

5.5.3.4.2 Hard Temporalities

- The above quoted ("...") statements may not have convinced you about the importance of speaking precisely about time, whether narrating or formalising.
- So let's try some other examples:
  - *"The alarm clock must sound exactly at 6 am unless someone has turned it off sometime between 5am and 6 am the same morning."*
  - *"The gas valve must be open for exactly 20 seconds every 60 seconds."*

- *"The sum total of time periods — during which the gas valve is open and there is no flame consuming the gas — must not exceed one twentieth of the time the gas valve is open."*
- *"The time between pressing an elevator call button on any floor and the arrival of the cage and the opening of the cage door at that floor must not exceed a given time* t_{arrival}*".*

• The next lecture items will hint at ways and means of speaking of time.

5.5.3.4.3 Soft and Hard Real-time

- The informally worded temporalities of “soft real-time”
  - can be said to involve time in a very “soft” way:
  - No explicit times (eg., 15:45:00), deadlines
    (eg., *“27’th February 2004”*),
    or time intervals
    (eg., *“within 2 hours”*),
    were expressed.
- The informally worded temporalities of “hard real-time”, in contrast,
  - can be said to involve time in a “hard” way:
  - Explicit times were mentioned.

- For pragmatic reasons, we refer to the former examples, the former “invocations” of ‘temporality’, as being representative of soft real-time,
- whereas we say that the latter invocations are typical of hard real-time.
- Please do not confuse the issue of soft versus hard real-time:
  - It is as much hard real-time if we say that something must happen two light years and five seconds from tomorrow at noon!

Example 57 . Soft Real-Time Models Expressed in Ordinary RSL Logic:

- Let us assume a salary data base SDB
- which at any time records your salary.
- In the conventional way of modelling time in RSL we assume that SDB maps time into Salary:

type
  Time, Sal
  SDB = Time $\underset{m}{\rightarrow}$ Sal
value
  hi: (Sal×Sal)|(Time×Time) → **Bool**
  eq: (Sal×Sal)|(Time×Time) → **Bool**
  lo: (Sal×Sal)|(Time×Time) → **Bool**
axiom
  ∀ σ:SDB,t,t′:Time · {t,t′}⊆**dom**σ∧hi(t′,t)⇒∼lo(σ(t′),σ(t))
  ∀ t,t′:Time ·
    (hi(t′,t)≡∼(eq(t′,t)∨lo(t′,t))) ∧
    (lo(t′,t)≡∼(eq(t′,t)∨hi(t′,t))) ∧
    (eq(t′,t)≡∼(lo(t′,t)∨hi(t′,t))) ... /∗ same for Sal ∗/

Example 58 . Hard Real-Time Models Expressed in “Ordinary” RSL Logic:

- To express hard real-time using just RSL we must assume a demon, a process which represents the clock:

type
  $\mathbb{T}$ = **Real**
value
  time: **Unit** $\rightarrow$ $\mathbb{T}$
  time() **as** t
axiom
  time() $\neq$ time()

- The axiom is informal:
  - It states that no two invocations of the time function yields the same value.
  - But this is not enough.
  - We need to express that “immediately consecutive” invocations of the time function yields “adjacent” time points.
- $\mathbb{T}$ provides a linear model of real-time.

```
variable
  t1,t2 : 𝕋
axiom
  □ (t1 := time();
     t2 := time();
     t2 − t1 = /∗ infinitesimally small time interval: 𝕋𝕀∗/ ∧
     t2 > t1 ∧ ∼∃ t:𝕋· t1 < t < t2 )
```

- 𝕋𝕀 provides a linear model of intervals of real-time.[18]
- The □ operator is here the "standard" RSL modal operator over states:
  - Let *P* be a predicate involving globally declared variables.
  - Then □*P* asserts that *P* holds in any state (of these variables).
- But even this is not enough. Much more is needed ■

[18]Of course, we really do not need make a distinction between 𝕋 and 𝕋𝕀, The former tries to model a real-time since time immemorial, i.e., the creation of the universe. If we always work with a time axis from "that started recently", i.e., a relative one, then we can "collapse" 𝕋 and 𝕋𝕀 into just 𝕋.

5.6 Intentional Pull

Left out of the TU Wien lectures

- In this part of the lecture
  we shall encircle the 'intention' concept
  by extensively quoting from Kai Sørlander's Philosphy
  [44, 45, 46, 47].
- *Intentionality*[19] "expresses" conceptual, abstract relations between otherwise, or seemingly unrelated entities.
- Intentional properties of a domain is not an internal quality of any (pair or group of) entities.
- They are potential, universal qualities of any world.

[19] The Oxford English Dictionary [32] characterises intentionality as follows: *"the quality of mental states (e.g. thoughts, beliefs, desires, hopes) which consists in their being directed towards some object or state of affairs"*.

5.6.1 Issues Leading Up to Intentionality

5.6.1.1 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."*

5.6.1.2 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 occur 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 movements.*
  - *The first we call plants, the second we call animals.”*

5.6.1.3 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 organs sensing among others the immediate purpose of its movement;*
    - *they must have means of motion so that it can move; and*
    - *they must have instincts, incentives and feelings as causal conditions that what it senses can drive it to movements.*
  - *And all of this in accordance with the laws of physics."*

5.6.1.4 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 genomes.*
- *That is, animals are built from genomes which give them the inner determination to such building blocks for instincts, incentives and feelings.*
- *Similar kinds of deduction can be carried out with respect to plants.*
- *Transcendentally one can deduce basic principles of evolution but not its details."*

5.6.1.5 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."*

- *“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.”*

5.6.1.6 Knowledge

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

5.6.1.7 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 further develop the theme of
  - *living species: plants and animals*,
  - thus excluding, most notably *humans*,
  - in this chapter.

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

● ● ●

5.6.2 Intentionality

- *Intentionality* as
  - a philosophical concept
  - is defined by the
    `Stanford Encyclopedia of Philosophy`[20] as
    - *"the power of minds*
      to be about,
      to represent, or
      to stand for,
    - *things, properties and states of affairs."*

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

5.6.2.1 Intentional Pull

- Two or more artefactual 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 59 . Double Bookkeeping:

- A classical example of intentional pull
  is found in double bookkeeping
  - which states that every financial transaction
  - has equal and opposite effects in at least two different accounts.
  - It is used to satisfy the accounting equation:
    Assets = *Liabilities* + *Equity*.
  - The intentional pull is then reflected in commensurate postings,
    for example:
    - ⋆ either in both debit and passive entries
    - ⋆ or in both credit and passive entries.

- When a compound artefact
  - is modelled as put together
    with a number of distinct sort endurants
  - then it does have an intentionality and
  - the components' individual intentionalities does,
    i.e., shall relate to that.
    - ⋆ The composite road transport system has intentionality
      of the road serving the automobile part, and
    - ⋆ the automobiles have
      the intent of being served by the roads,
      across "a divide", and vice versa,
      the roads of serving the automobiles.

- Natural endurants, for example,
  - rivers, lakes, seas[21] and oceans become, in a way, artefacts when mankind use them for transport;
  - natural gas becomes an artefact when drilled for, exploited and piped; and
  - harbours make no sense without artefactual boats sailing on the natural water.

[21]Seas are smaller than oceans and are usually located where the land and ocean meet. Typically, seas are partially enclosed by land. The Sargasso Sea is an exception. It is defined only by ocean currents [oceanservice.noaa.gov/facts/oceanorsea.html].

5.6.2.2 The Type Intent

- This, perhaps vague, concept of intentionality
  has yet to be developed into something of a theory.
- Despite that this is yet to be done,
  we shall proceed to define an *intentionality analysis function*.
- First we postulate a set of **intent designators**.
  - An *intent designator* is really a further undefined quantity.
  - But let us, for the moment,
    think of them as simple character strings, that is, literals,
    for example `""transport", "eating", "entertainment"`, etc.

    type Intent

5.6.2.3 Intentionalities

Observer Function Prompt 10 . `analyse_intentionality:`

- *The domain analyser analyses an endurant as to the finite number of intents, zero or more, with which the analyser judges the endurant can be associated.*
- *The method provides the* ***domain analysis prompt****:*
  - `analyse_intentionality` *directs the domain analyser to observe a set of intents.*

    value analyse_intentionality(e) ≡ {i_1,i_2,...,i_n}⊆Intent

Example 60 . Intentional Pull: Road Transport:

- We simplify the link, hub and automobile histories –
- aiming at just showing an essence of the intentional pull concept.

127. With links, hubs and automobiles
we can associate history attributes:
    (a) link history attributes time-stamped records,
    as an ordered list, the presence of automobiles;
    (b) hub history attributes time-stamped records,
    as an ordered list, the presence of automobiles; and
    (c) automobile history attributes time-stamped records,
    as an ordered list, their visits to links and hubs.

type	**value**
127a. LHist = AI $\underset{m}{\rightarrow}$ $\mathbb{TIME}^*$	127a. attr_LHist: L $\rightarrow$ LHist
127b. HHist = AI $\underset{m}{\rightarrow}$ $\mathbb{TIME}^*$	127b. attr_HHist: H $\rightarrow$ HHist
127c. AHist = (LI|HI) $\underset{m}{\rightarrow}$ $\mathbb{TIME}^*$	127c. attr_AHist: A $\rightarrow$ AHist

5.6.2.4 Wellformedness of Event Histories

- Some observations must be made with respect to the above modelling of time-stamped event histories.

128. Each $\tau_\ell : \mathbb{TIME}^*$ is an indefinite list.
We have not expressed any criteria
for the recording of events: *all the time, continuously* ! (?)

129. Each list of times, $\tau_\ell : \mathbb{TIME}^*$, is here to be in decreasing, *continuous* order of times.

130. Time intervals from when an automobile enters a link (a hub)
till it first time leaves that link (hub)
must not overlap with other such time intervals for that automobile.

131. If an automobile leaves a link (a hub), at time τ, then it may enter a hub (resp. a link)
and then that must be at time τ'
where τ' is some infinitesimal, sampling time interval, quantity larger that τ.
Again we refrain here from speculating on the issue of sampling !

132. Altogether, ensembles of link and hub event histories
for any given automobile define routes that automobiles travel across the road net.
Such routes must be in the set of routes defined by the road net.

- As You can see, there is enough of interesting modelling issues to tackle !

5.6.2.5 Formulation of an Intentional Pull

133. An *intentional pull* of any road transport system, rts, is then if:
   (a) for any automobile, a, of rts,
   on a link, ℓ (hub, h),
   at time τ,
   (b) then that link, ℓ, (hub h)
   "records" automobile a
   at that time.
134. and:
   (c) for any link, ℓ (hub, h)
   being visited by an automobile, a,
   at time τ,
   (d) then that automobile, a,
   is visiting that link, ℓ (hub, h),
   at that time.

axiom

```
133a. ∀ a:A · a ∈ as ⇒
133a.    let ahist = attr_AHist(a) in
133a.    ∀ ui:(LI|HI) · ui ∈ dom ahist ⇒
133b.       ∀ τ:TIME · τ ∈ elems ahist(ui) ⇒
133b.          let hist = is_LI(ui) → attr_LHist(retr_L(ui))(σ),
133b.                        _ → attr_HHist(retr_H(ui))(σ) in
133b.          τ ∈ elems hist(uid_A(a)) end end
134.  ∧
134c. ∀ u:(L|H) · u ∈ ls∪hs ⇒
134c.    let uhist = attr(L|H)Hist(u) in
134d.    ∀ ai:AI · ai ∈ dom uhist ⇒
134d.       ∀ τ:TIME · τ ∈ elems uhist(ai) ⇒
134d.          let ahist = attr_AHist(retr_A(ai))(σ) in
134d.          τ ∈ elems uhist(ai) end end
```

- Please note, that *intents* are not [thought of as] attributes.
  - We consider *intents* to be a fourth,
    a comprehensive internal quality of endurants.
  - They, so to speak, govern relations between the three other
    internal quality of endurants:
    the unique identifiers, the mereologies and the attributes.
  - That is, they predicate them, "arrange" their comprehensiveness.
- Much more should be said about intentionality.
- It is a truly, I believe, worthy research topic of its own ■

Example 61 . Aspects of Comprehensiveness of Internal Qualities:

- Let us illustrate the issues “at play” here.
  - Consider a road transport system uod.
    - ⋆ Applying `analyse_intentionality`(uod) may yield the set `{"transport", ...}`.
  - Consider a financial service industry, fss.
    - ⋆ Applying `analyse_intentionality`(fss) may yield the set `{"interest on deposit", ...}`.
  - Consider a health care system, hcs.
    - ⋆ Applying `analyse_intentionality`(hcs) may yield the set `{"cure diseases", ...}`.
- What these analyses of intentionality yields,
  with respect to expressing intentional pull,
  is entirely of the discretion of the *domain analyser & describer* ■

- We bring the above example,
  Example 61 on the preceding slide, to indicate,
  as the name of the example reveals,
  “Aspects of Comprehensiveness of Internal Qualities”.
  - That the various components of artefactual systems
    relate in – further to be explored – ways.
  - In this respect, performing domain analysis & description
    is not only an engineering pursuit, but also one of research.
  - We leave it to the students to pursue this research aspect
    of domain analysis & description.

5.6.3 Artefacts

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

5.6.4 Assignment of Attributes

- So what can we deduce from the above, almost three pages ?
- The attributes of natural parts and natural fluids
  - are generally of such concrete types –
  - expressible as some **real** with a dimension[22] of
  - the International System of Units:
  - `https://physics.nist.gov/cuu/Units/units.html`.
- Attribute values usually enter into *differential equations* and *integrals*,
- that is, classical calculus.

[22]Basic units are *meter*, *kilogram*, *second*, *Ampere*, *Kelvin*, *mole*, and *candela*. Some derived units are: *Newton*: $kg{\times}m{\times}s^{-2}$, *Weber*: $kg \times m^2 \times s^{-2} \times A^{-1}$, 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 artefacts
    - ⋆ humans create these with an intent.

Example 62 . Intentional Pull: General Transport:

- 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.

5.6.5 Galois Connections

- Galois Theory was first developed by Évariste Galois [1811-1832] around 1830[23].
- Galois theory emphasizes a notion of **Galois connections**.
- We refer to standard textbooks on Galois Theory, e.g., [49, 2009].

[23] en.wikipedia.org/wiki/Galois_theory

5.6.5.1 Galois Theory: An Ultra-brief Characterisation

- To us, an essence of Galois connections can be illustrated as follows:
  - Let us observe[24] properties of a number of endurants, say in the form of attribute types.
  - Let the function $\mathcal{F}$ map sets of entities to the set of common attributes.
  - Let the function $\mathcal{G}$ map sets of attributes to sets of entities that all have these attributes.
  - $(\mathcal{F},\mathcal{G})$ is a Galois connection
    - if, when including more entities, the common attributes remain the same or fewer, and
    - if when including more attributes, the set of entities remain the same or fewer.
    - $(\mathcal{F},\mathcal{G})$ is monotonously decreasing.

[24]The following is an edited version of an explanation kindly provided by Asger Eir, e-mail, June 5, 2020 [26, 27, 21].

Example 63 . LEGO Blocks:

- We[25] have
  - There is a collection of LEGO™ blocks.
  - From this collection, A, we identify the **red** square blocks, e.
  - That is $\mathcal{F}(A)$ is $B = \{$attr_Color(e) = **red**,attr_Form(e)=**square**$\}$.
  - We now add all the **blue** square blocks.
  - And obtain A'.
  - Now the common properties are their **squareness**:
    $\mathcal{F}(A')$ is $B' = \{$attr_Form(e)=**square**$\}$.
  - More blocks as argument to $\mathcal{F}$ yields fewer or the same number of properties.
  - The more entities we observe, the fewer common attributes they possess ■

[25] The E-mail, June 5, 2020, from Asger Eir

Example 64 . Civil Engineering: Consultants and Contractors:

Less playful, perhaps more seriously, and certainly more relevant to our endeavour, is this next example.

- Let X be the set of civil engineering, i.e., building, consultants, i.e., those who, like architects and structural engineers design buildings – of whatever kind.
- Let Y be the set of building contractors, i.e., those firms who actually implement, i.e., build to, those designs.
- Now a subset, $X_{bridges}$ of X, contain exactly those consultants who specialise in the design of bridges, with a subset, $Y_{bridges}$, of Y capable of building bridges.
- If we change to a subset, $X_{bridges,tunnels}$ of X, allowing the design of both bridges **and** tunnels, then we obtain a corresponding subset, $Y_{bridges,tunnels}$, of Y.

- So when
  - we enlarge the number of properties
    from ‘bridges’ to ‘bridges and tunnels’,
  - we reduce, most likely, the number of contractors able to fulfill such properties,
  - and vice versa,
- then we have a Galois Connection[26] ■

[26]This was, more formally, shown Dr. Asger Eir’s PhD thesis [26].

5.6.5.2 Galois Connections and Intentionality – A Possible Research Topic ?

- We have a hunch[27] !
  - Namely that there are some sort of Galois Connections with respect to intentionality.
- We leave to the interested student to pursue this line of inquiry.

[27]Hunch: a feeling or guess based on intuition rather than fact.

5.6.6 Discovering Intentional Pulls

- The analysis and description of a domain's
  - external qualities and
  - the internal qualities of
    unique identifiers, mereologies and attributes
  - can be pursued systematically –
  - endurant sort by sort.

- Not so with the discovery of
  a domain's possible intentional pulls.
- Basically *"what is going on"* here is
  - that the domain analyser cum describer
  - considers pairs, triples or more
    part "independent"[28] endurants
  - and reflects on whether they stand
    in an *intentional pull*
    relation to one another.
- We refer to Sects. 5.6.2.2 – 5.6.2.3.

[28]By "independent" we shall here mean that these endurants are not 'derived' from one-another !

5.7 A Domain Discovery Procedure, II

- We continue from Sect. 4.8.

5.7.1 The Process

- We shall again emphasize some aspects
  of the *domain analyser & describer* method.
  - A **method procedures** is that of *exhaustively analyse & describe* all internal qualities of the domain under scrutiny.
  - A **method technique** implied here is that sketched below.
  - The **method tools** are here all the analysis and description prompts covered so far.

- Please be reminded of *Discovery Schema 0*'s declaration of *Notice Board* variables (Slide 158).
- In this section of the lecture we collect
  - the *description of unique identifiers* of all parts of the state;
  - the *description of mereologies* of all parts of the state; and
  - the *description of attributes* of all parts of the state.
- We finally gather these into the *discover_internal_endurant_qualities* procedures.

An Endurant Internal Qualities Domain Analysis and Description Process, I

```
value
   discover_uids: Unit → Unit
   discover_uids() ≡
      for ∀ v · v ∈ gen
         do txt := txt † [ type_name(v)↦txt(type_name(v))⁀⟨describe_unique_identifier(v)⟩ ] end
   discover_mereologies: Unit → Unit
   discover_mereologies() ≡
      for ∀ v · v ∈ gen
         do txt := txt † [ type_name(v)↦txt(type_name(v))⁀⟨describe_mereology(v)⟩ ] end
   discover_attributes: Unit → Unit
   discover_attributes() ≡
      for ∀ v · v ∈ gen
         do txt := txt † [ type_name(v)↦txt(type_name(v))⁀⟨describe_attributes(v)⟩ ] end
   discover_intentional_pulls: Unit → Unit
   discover_intentional_pulls() ≡
      for ∀ (v′,v″) · {v′,v″} ⊆ gen
         do txt := txt † [ type_name(v′)↦txt(type_name(v′))⁀⟨describe_intentional_pull()⟩ ]
                † [ type_name(v″)↦txt(type_name(v″))⁀⟨describe_intentional_pull()⟩ ] end
   describe_intentional_pull: Unit → ...
   describe_intentional_pull() ≡ ...
```

An Endurant Internal Qualities Domain Analysis and Description Process, II

```
value
  discover_internal_qualities: Unit → Unit
  discover_internal_qualities() ≡
      discover_uids() ;
          axiom [ all parts have unique identifiers ]
      discover_mereologies() ;
          axiom [ all unique identifiers are mentioned in sum total of ]
              [ all mereologies and no isolated proper sets of parts ]
      discover_attributes() ;
          axiom [ sum total of all attributes span all parts of the state ]
      discover_intentional_pulls()
```

- We shall comment on the axioms in the next section.

5.7.2 A Suggested Analysis & Description Approach, II

- Figure 4.3 on Slide 129 possibly hints at an analysis & description order in which
  - not only the external qualities of endurants are analysed & described,
  - but also their internal qualities of unique identifiers, mereologies and attributes.
- In Sect. 4.8 on Slide 156 we were concerned with the analysis & description order of endurants.

- We now follow up on the issue of (in Sect. 4.5.1.3 on Slide 127) on how compounds are treated: namely as both a “root” parts and as a composite of two or more “sibling” parts and/or fluids.
  - The taxonomy of the road transport system domain, cf. Fig. 4.3 on Slide 129 and Example 29 on Slide 106, thus gives rise to many different analysis & description traversals.
  - Figure 5.4 on the facing slide illustrates one such order.

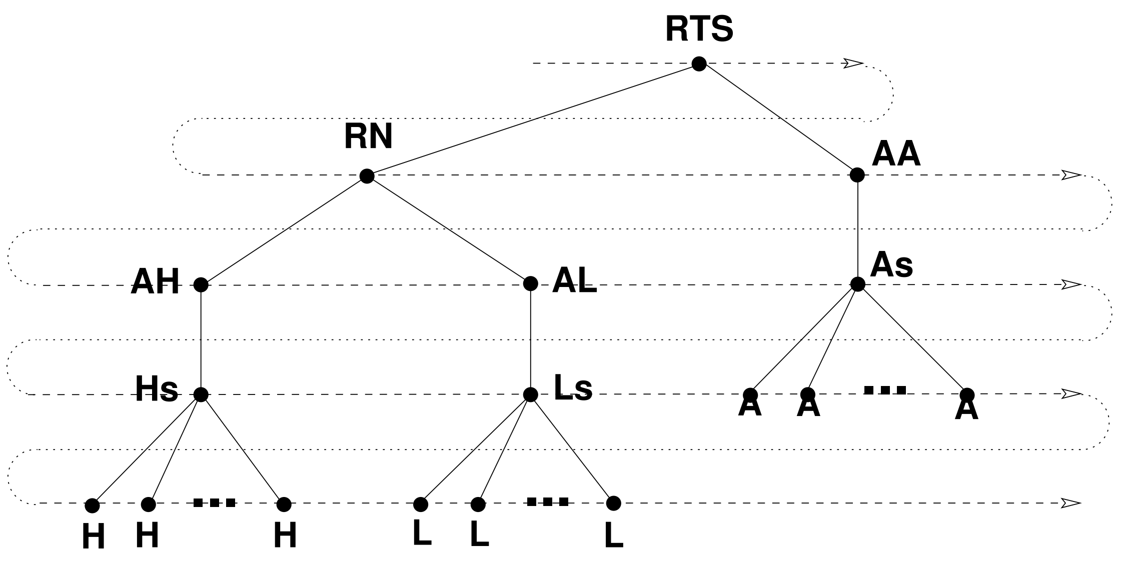

Figure 5.4: A Breadth-First, Top-Down Traversal

- Again, it is up to the domain engineer cum scientist to decide.
  - ⋆ If the domain analyser cum describer decides to not endow a compound “root” with internal qualities,
  - ⋆ then an ‘internal qualities’ traversal will not have to neither analyse nor describe those qualities.

5.8 Summary

#	Name	Introduced
	Analysis Predicates	
16	is_manifest	page 176
17	is_structure	page 176
	Attribute Analysis Predicates	
1	is_static_attribute	page 242
2	is_dynamic_attribute	page 244
3	is_inert_attribute	page 245
4	is_reactive_attribute	page 247
5	is_active_attribute	page 249
6	is_autonomous_attribute	page 250
7	is_biddable_attribute	page 252
8	is_programmable_attribute	page 254
9	is_monitorable_only_attribute	page 258
	Analysis Functions	
	all_uniq_ids	page 193
	calculate_all_unique_identifiers	page 192
6	analyse_attribute_types	page 272
7	sta_attr_types	page 273
8	mon_attr_types	page 274
9	pro_attr_types	page 275
	Retrieval, Read and Write Functions	
	retr_part	page 201
99	read_A_from_P	page 286
100	update_P_with_A	page 287
	Description Functions	
5	describe_unique_identifier	page 186
6	describe_mereology	page 210
7	describe_attributes	page 236
	Domain Discovery	
	discover_uids	page 366
	discover_mereologies	page 366
	discover_attributes	page 366
	discover_internal_qualities	page 366

• • •

- Please consider Fig. 4.1 on Slide 63.
  - This chapter has covered the horisontal and vertical lines to the left in Fig. 4.1.

Lecture 6: Perdurants, I

CHAPTER 6. Perdurants

- Please consider Fig. 4.1 on Slide 63.
  - The previous two chapters covered the left of Fig. 4.1.
  - This chapter covers the right of Fig. 4.1.

• • •

- This chapter is a rather "drastic" reformulation and simplification of [18, *Chapter 7, i.e., pages 159–196*].
  - Besides, Sect. 6.5 is new.
- In this chapter we transcendentally "morph" manifest
  - **parts** into **behaviours,** that is:
  - **endurants** into **perdurants**.

- We analyse that notion and its constituent notions of
  - **actors**,
  - **channels** and **communication**,
  - **actions** and
  - **behaviours**.
- We shall investigate the, as we shall call them, perdurants of domains.
- That is state and time-evolving domain phenomena.
- The outcome of this chapter is that the student
  - will be able to model the perdurants of domains.
  - Not just for a particular domain instance,
  - but a possibly infinite set of domain instances[1].

[1] By this we mean: You are not just analysing a specific domain, say the one manifested around the corner from where you are, but any instance, anywhere in the world, which satisfies what you have described.

6.1 Part Behaviours – An Analysis

6.1.1 Behaviour Definition Analysis

- Parts co-exist;
  - they do so endurantly as well as perdurantly:
  - endure and perdure.
- Part perdurants, i.e., behaviours, interact with their surroundings, that is, with other behaviours.
- This is true for both natural and man-made parts.
- The present domain modelling method is mainly focused on man-made parts, that is artefacts.
- So our next analysis will take its clues from artefactual parts.
- We can, roughly, analyse part behaviours into three kinds.

- **Proactive Behaviours:** Behaviour B_i offers to synchronise and communicate values – *internal non-deterministically* with either of a definite number of distinct part sort behaviours B_a, B_b, ..., B_c:

```
B(i)(args) ≡
      (... ch[ {i,a} ] ! a_val ; ... ; B(i)(args′))
  ⊓ (... ch[ {i,b} ] ! b_val ; ... ; B(i)(args″))
  ⊓ ...
  ⊓ (... ch[ {i,c} ] ! c_val ; ... ; B(i)(args‴))
```

The tail-recursive invocation of B_i indicates a possible “update” of behaviour B_i arguments. More on this later.

- **Responsive Behaviours:** Behaviour B_i *external non-deterministically* expresses willingness to synchronisation with and accept values from either of a definite number of distinct part sort behaviours B_a, B_b, ..., B_c:

```
B(i)(args) ≡
      (... let av = ch[{i,a}] ? in ...  B(i)(args′) end)
  ⎕ (... let bv = ch[{i,b}] ? in ... ; B(i)(args″) end)
  ⎕ ...
  ⎕ (... let cv = ch[{i,c}] ? in ... ; B(i)(args‴) end)
```

- **Mixed Behaviours:** Or behaviours, more generally, “are” an internal non-deterministic “mix” of the above:

```
B(i)(args) ≡
     ((... ch[ {i,a} ] ! a_val ; ... ; B(i)(args′))
 ⊓  (... ch[ {i,b} ] ! b_val ; ... ; B(i)(args″))
 ⊓  ...
 ⊓  (... ch[ {i,c} ] ! c_val ; ... ; B(i)(args‴)))
 ⊓ ((... let av = ch[ {i,a} ] ? in ...  B(i)(args′) end)
 □  (... let bv = ch[ {i,b} ] ? in ... ; B(i)(args″) end)
 □  ...
 □  (... let cv = ch[ {i,c} ] ? in ... ; B(i)(args‴) end))
```

- The "bodies" of the B_i behaviour definitions, i.e., "...", may contain interactions with [yet other] behaviours. Schematically for example:

```
ch[{i,x}] ! x_val
{ ch[{i,z}] ! z_val | z:{z1,z2,...,zm} }
let yv = ch[{i,y}] ? in ... end
let zv = □ { ch[{i,z}] ? | z:{z1,z2,...,zm} } in ... end
```

Etcetera. The full force of CSP with RSL is at play !

6.1.2 Channel Analysis

- This is the first of two treatments of the concept of *channels*;
  the present treatment is informal, motivational,
  the second treatment, Sect. 6.2 (right next !), is more formal.
- The CSP concept of *channel*
  is to be our way of expressing the "medium"
  in which behaviours interact.
  - Channels is thus an abstract concept.
  - Please do not think of it as a physical,
    an IT (information technology) device.
  - As an abstract concept it is defined in terms of,
    roughly, the laws, the semantics, of CSP [30].
  - We write 'roughly' since the CSP
    we are speaking of, is "embedded" in RSL.

6.2 Domain Channel Description

- We simplify the general treatment of channel declarations.
  - Basically all we can say, for any domain,
  - is that any two distinct part behaviours
  - may need to communicate.
  - Therefore we declare a vector of channels
  - indexed by sets of two distinct part identifiers.

    value
    discover_channels: **Unit** $\rightarrow$ **Unit**
    discover_channels() $\equiv$
    ❝ **channel** $\{$ ch[$\{$ij,ik$\}$] | ij,ik:UI $\cdot$ $\{$ij,ik$\}\subseteq$ uid$_\sigma$ $\wedge$ ij$\neq$ik $\}$ M ❞
  - Initially we shall leave the type of messages over channels further undefined.
  - As we, laboriously, work through the definition of behaviours, we shall be able to make M precise.

6.3 Behaviour Definition Description

- Behaviours have to be described.
  - Behaviour definitions are in the form of function definitions and are here expressed in RSL
    relying, very much, on its CSP component.
  - Behaviour definitions describe
    the type of the arguments
    the function, i.e., the behaviour, for which it is defined,
    that is, which kind of values it accepts.
  - Behaviour definitions further describe
- Thus there are two elements to a behaviour definition:
  - the behaviour *signature* and
  - the behaviour *body*

  definitions.

6.3.1 Behaviour Signatures

6.3.1.1 General

- Function, F, signatures consists of two textual elements:
  - the function name and
  - the function type:

    value F: A → B, or F: a:A → B

  - where A and B are the types of
    - ⋆ function ("input") arguments, respectively
    - ⋆ function ("output") values for such arguments.
  - The first form F: A → B is what is normally referred to as the form for function signatures.
  - The second form: F: a:A → B "anticipates" the general for for function F invocation: F(a).

6.3.1.2 Domain Behaviour Signatures

- A schematic form of part (p) behaviour signatures is:

```
b: bi:BI→me:Mer→svl:StaV*→mvl:MonV*→prgl:PrgV* channels Unit
```

- We shall motivate the general form of part behaviour, B, signatures, “step-by-step”:

α.	b	the [chosen] name of part p behaviours.
β	U→V→...→W→Z:	The function signature is expressed in the `Schönfinkel/Curr` style – corresponding to the invocation form F(u)(v)...(w)
γ.	bi:BI:	a general value and the type of part p unique identifier
δ.	me:Mer:	a general value and the type of part p mereology
ϵ.	svl:StaV*:	a general (possibly empty) list of values and types of part p's (possibly empty) list of static attributes
ζ.	mvl:MonV*:	a general list of names of types of part p's (possibly empty) list of monitorable attributes
η.	prgl:PrgV*:	a general list of values and types of part p's (possibly empty) list of programmable attributes
θ.	channels:	are usually of the form: {ch[{i,j}]|(i,j)∈$\mathcal{I}$(me)} and express the su of channels over which behaviour Bs interact with other behav
ι.	**Unit**:	designates the single value ()

[2]Moses Schönfinkel (1888–1942) was a Russian logician and mathematician accredited with having invented combinatory logic [https://en.wikipedia.org/wiki/Moses_Schönfinkel]. Haskell B. Curry (1900–1982) was an American mathematician and logician known for his work in combinatory logic [https://en.wikipedia.org/wiki/Haskell_Curry]

In detail:

α. **Behaviour name:** In each domain description there are many sorts, B, of parts. For each sort there is a generic behaviour, whose name, here b. is chosen to suitably reflect B.

β. **Currying** is here used in the pragmatic sense of grouping "same kind of arguments", i.e., separating these from one-another, by means of the $\rightarrow$s.

γ. The **unique identifier** of part sort B is here chosen to be BI. Its value is a constant.

δ. The **mereology** is a usually constant. For same part sorts it may be a variable.

Example 65 . Variable Mereologies:

- For a road transport system where we focus on the transport the mereology is a constant.
- For a road net where we focus on the development of the road net: building new roads: inserting and removing hubs and links, the mereology is a variable.
- Similar remarks apply to canal systems `www.imm.dtu.dk/~dibj/2021/Graphs/Rivers-and-Canals.pdf`, pipeline systems [8], container terminals [14], assembly line systems [15], etc. ■

ϵ. **Static attribute values** are constants. The use of static attribute values in behaviour body definitions is expressed by an identifier of the stvl list of identifiers.

ζ. **Monitorable attribute values** are generally, ascertainable, i.e., readable, cf. Sect. 5.4.3.1 on Slide 286. Some are *biddable*, can be changed by a, or the behaviour, cf. Sect. 5.4.3.2 on Slide 287, but there is no guarantee, as for programmable attributes, that they remain fixed.

- The use of a[ny] monitorable attribute value in behaviour body definitions is expressed by a read_A_from_P(mv,bi) where mv is an identifier of the mvl list of identifiers and bi is the unique part identifier of the behaviour definition in which the read occurs.
- The update of a biddable attribute value in behaviour body definitions is expressed by a update_P_with_A(bi,mv,a).

η. **Programmable attribute values** are just that. They vary as specified, i.e., “programmed”, by the behaviour body definition. Tail-recursive invocations of behaviour B_i “replace” relevant programmable attribute argument list elements with “new” values.

θ. **channels:** I(me) expresses a set of unique part identifiers different from bi, hence of behaviours, with which behaviour b(i) interacts.

ι. The **Unit** of the behaviour signature is a short-hand for the behaviour either **read**ing the value of a monitorable attribute, hence global state σ, or performing a **write**, i.e., an *update*, on σ.

6.3.1.3 Action Signatures

- Actions come in any forms:

135. Some take no arguments, say action_a(), but read the global state component σ, and
136. others also take no arguments, say action_b(), but update the global state component σ.
137. Some take an argument, say, action_c(c), but do not “touch” a global state component,
138. while others both take an argument and deliver a value, say action_d(d) and also do not “touch” a global state component.
139. Et cetera !

type A, B, C, D, ...
value
135. action_a: **Unit** → **read** σ A
136. action_b: **Unit** → **write** σ B
137. action_c: C → **Unit**
138. action_d: D → E **Unit**
139. ...

- An example of 137 are the CSP output: ch[...]!c, and
- an example of 138 are the CSP input: **let** e = ch[...]? **in** ... **end**.

6.3.2 Behaviour Invocation

- The general form of behaviour invocation is shown below.
  - The invocation follows the "Currying" of the behaviour type signature.
  - [Normally one would write all this on one line: b(i)(m)(s)(m)(p) ≡.]

```
behaviour_name
  (unique_identifier)
    (mereology)
      (static_values)
        (monitorable_attribute_names)
          (programmable_variables) ≡
  ... body ...
```

- When first "invoked":

```
value
  discover_ behaviour_signature: P → RSL-Text
  discover_ behaviour_signature(p) ≡
  ❝ behaviour_name:
        UId → Mereo → StaVL → MonVL → ProVL → channels Unit
    behaviour_name
        (uid_B(p))
          (mereo_B(p))
            (types_to_values(static_attribute_types(p)))
              (mon_attribute_types(p))
                (types_to_values(programmable_attribute_types(p))) ≡ ❞
    pre: is_B(p) ∧ is_manifest(p)

  discover_ behaviour_signatures: Unit → RSL-Text
  discover_ behaviour_signatures() ≡
    { discover_ behaviour_signature(p) | p ∈ σ ∧ is_manifest(p) }
```

6.3.3 Behaviour Definition Bodies

- We remind the student of Sect. 6.1.1 on Slide 375.
- The general, “mixed”, form of behaviour definitions was given as:

```
B(i)(args) ≡
        ( ( ... )
  ⌈⌉    ( ... ch[ {i,b} ] ! b_val ; ... ; B(i)(args″) )
  ⌈⌉    ( ... ) )
  ⌈⌉  ( ( ... )
  ⌈⌉    (... let bv = ch[ {i,b} ] ? in ... ; B(i)(args″) end )
  ⌈⌉    ( ... ) )
```

- We can express the same
  - by separating the alternatives
  - into invocations of separately defined behaviuors.

```
B(i)(args) ≡
      ( ...
 ⊓    Bin_j(i)(args)
 ⊓    ... )
 ⊓  ( ...
 ▯    Bxn_k(i)(args)
 ▯    ...  )
```

- where
  - the internal don-deterministically invoked behaviours Bin$_j$(i)(args) and
  - the external don-deterministically invoked behaviours Bin$_k$(i)(args)
- are then separately defined:

Bin$_j$(i)(args) ≡ (... Bin$_j$(i)(args′))
Bxn$_k$(i)(args) ≡ (... Bxn$_k$(i)(args″))

6.3.4 Discover Behaviour Definition Bodies

- In other words,
  - for current lack of a more definitive methodology
  - for "discovering" the bodies of behaviour definitions
  - we resort to "…" !

```
value
  discover_behaviour_definition: P → RSL-Text
  discover_behaviour_definition(p) ≡ ...

  discover_behaviour_definitions: Unit → RSL-Text
  discover_behaviour_definitions() ≡
    { discover_behaviour_definition(p) | p ∈ σ ∧ is_manifest(p) }
```

Example 66 . Automobile Behaviour:

Signatures

140. automobile:
    (a) there is the usual “triplet” of arguments: unique identifier, mereology and static attributes;
    (b) then there are two programmable attributes: the automobile position (cf. Item 90 on Slide 266), and the automobile history (cf. Item 127c on Slide 344);
    (c) and finally there are the input/output channel references allowing communication between the automobile and the hub and link behaviours.
141. Similar for
    (a) link and
    (b) hub behaviours.

- We omit the modelling of monitorable attributes (...).

value

```
140a,140a automobile: ai:AI → ((_,uis):AM) → ...
140b        → (apos:APos × ahist:AHist)
140c          in out {ch[ {ai,ui} ]|ai:AI,ui:(HI|LI) · ai∈ais ∧ ui ∈ uis} Unit
141a      link: li:LI → (his,ais):LM → LΩ → ...
141a        → (LΣ×L_Hist)
141a          in out {ch[ {li,ui} ]|li:LI,ui:(AI|HI)-set · ai∈ais ∧ li ∈lis∪his} Unit
141b      hub: hi:HI → ((_,ais):HM) → HΩ ...
141b        → (HΣ×H_Host)
141b          in out {ch[ {ai,ui} ]|hi:HI,ai:AI · ai∈ais ∧ hi ∈ uis} Unit
```

Definitions: Automobile at a Hub

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

```
142  automobile(ai)(aai,uis)(...)(apos:atH(fli,hi,tli),ahist) ≡
142a     automobile_remains_in_hub(ai)(aai,uis)(...)(apos:atH(fli,hi,tli),ahist)
142b   ⊓
142c     automobile_leaving_hub(ai)(aai,uis)(...)(apos:atH(fli,hi,tli),ahist)
142d   ⊓
142e     automobile_stop(ai)(aai,uis)(...)(apos:atH(fli,hi,tli),ahist)
```

143. [142a] The automobile remains in the hub:

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

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

```
143   automobile_remains_in_hub(ai)(aai,uis)(...)(apos:atH(fli,hi,tli),ahist) ≡
143      let τ = record_𝕋𝕀𝕄𝔼() in
143b      ch[ ai,hi ] ! τ ;
143a      automobile(ai)(aai,uis)(...)(apos,upd_hist(τ,hi)(ahist))
143      end

143a   upd_hist: (𝕋𝕀𝕄𝔼×I) → (AHist|LHist|HHist) → (AHist|LHist|HHist)
143a   upd_hist(τ,i)(hist) ≡ hist † [ i ↦ ⟨τ⟩⁀hist(i) ]
```

144. [142c] The automobile leaves the hub entering a link:

(a) tli, whose “next” hub, identified by thi, is obtained from the mereology of the link identified by tli;

(b) informs the hub it is leaving and the link it is entering,

(c) “whereupon” the vehicle resumes (i.e., “while at the same time” resuming) the vehicle behaviour positioned at the very beginning (0) of that link.

```
144  automobile_leaving_hub(ai)(aai,uis)(...)(apos:atH(fli,hi,tli),ahist) ≡
144a    (let ({fhi,thi},ais) = mereo_L(retr_L(tli)(σ)) in assert: fhi=hi
144b     ( ch[ai,hi] ! τ ∥ ch[ai,tli] ! τ ) ;
144c     automobile(ai)(aai,uis)(...)
144c             (onL(tli,(hi,thi),0),upd_hist(τ,tli)(upd_hist(τ,hi)(ahist))) end)
```

145. [142e] Or the automobile "disappears — off the radar" !

145 automobile_stop(ai)(aai,uis),(...)(apos:atH(fli,hi,tli),ahist) ≡ **stop**

- Similar behaviour definitions can be given for *automobiles on a link*, for *link*s and for *hub*s.
- Together they must reflect, amongst other things:
  - the time continuity of automobile flow,
  - that automobiles follow routes,
  - that automobiles, links and hubs together adhere to the intentional pull expressed earlier,
  - et cetera.
- A specification of these aspects must be proved to adhere to these properties.

6.4 Domain Behaviour Initialisation

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

Example 67 . The Road Transport System Initialisation: We "wrap up" the main example of this *primer*:

- We omit treatment of monitorable attributes.

146. Let us refer to the system initialisation as an action.
147. All links are initialised,
148. all hubs are initialised,
149. all automobiles are initialised,
150. etc.

value

```
146. rts_initialisation: Unit → Unit
146. rts_initialisation() ≡
147.     ∥ { link(uid_L(l))(mereo_L(l))(attr_LEN(l),attr_LΩ(l))(attr_L_Traffic(l),attr_LΣ(
148.     ∥ ∥ { hub(uid_H(l))(mereo_H(l))(attr_HΩ(l))(attr_H_Traffic(l),attr_HΣ(l))| h:H ·
149.     ∥ ∥ { automobile(uid_A(a))(mereo_A(a))(attr_RegNo(a))(attr_APos(a)) | a:A ·
150.     ∥ ...
```

- We have here omitted possible monitorable attributes.
- We refer to
  - *ls*: Item 36 on Slide 153,
  - *hs*: Item 37 on Slide 153, and
  - *as*: Item 38 on Slide 153 ■

6.5 Discrete Dynamic Domains

- Up till now our analysis & description of a domain,
  - has, in a sense, been *static:*
  - in analysing a domain we considered its entities
  - to be of a definite number.
- In this section we shall consider the case where the number of entities change:
  - where new entities are *created*
  - and existing entities are *destroyed*,
  - that is:
    - ⋆ where new parts, and hence behaviours, arise, and
    - ⋆ existing parts, and hence behaviours, cease to exist.

6.5.1 Create and Destroy Entities

- In the domain we can expect that its behaviours create and destroy entities.

Example 68 . Creation and Destruction of Entities:

- In the *road transport* domain
  - new hubs, links and automobiles may be inserted into the road net, and
  - existing links, hubs and automobiles may be removed from the road net.
- In a *container terminal* domain [5, 14]
  - new containers are introduced, old are discarded;
  - new container vessels are introduced, old are discarded;
  - new ship-to-shore cranes are introduced, old are discarded;
  - et cetera.

- In a *retailer* domain [16]
  - new customers are introduced, old are discarded;
  - new retailers are introduced, old are discarded;
  - new merchandise is introduced, old is discarded;
  - et cetera.
- In a *financial system* domain
  - new customers are introduced, old are discarded;
  - new banks are introduced, old are discarded;
  - new brokers are introduced, old are discarded;
  - et cetera ■

- **The issue here is:**
  - When hubs and links are inserted or removed
    - ⋆ the mereologies of “neighbouring” road elements change,
    - ⋆ and so does the mereology of automobiles.
  - When automobiles are inserted or removed
    - ⋆ The mereology of road elements
    - ⋆ have to be changed
    - ⋆ to take account of the insertions and removals,
    - ⋆ and so does the mereology of automobiles.
  - And, some domain laws must be re-expressed:
    - ⋆ The domain part state, σ, must be updated[3],
    - ⋆ and so must the unique identifier state, uid_σ[4].

[3] Cf. Sect. 4.7.2 on Slide 155
[4] Cf. Sect. 5.2.4 on Slide 193

6.5.1.1 Create Entities

- It is taken for granted here that there are behaviours,
  one or more, which take the initiative to and carry out
  the creation of specific entities.
  Let us refer to such a behaviour as the "creator".
- To create an entity implies the following three major steps
  - [A.–C.] the step wise creation of the part
    and initialisation of the transduced behaviour, and
  - [D.] the adjustment of all such part behaviours that might have
    their mereologies and attributes updated to accept such requests
    from creators.

A. To decide on the part sort – in order to create that part – that is

- to obtain a unique identifier – one hitherto not used;
- to obtain a mereology, one
  - ⋆ according to the general mereology for parts of that sort,
  - ⋆ and how the part specifically is to “fit” into its surroundings;
- to obtain an appropriate set of attributes:
  - ⋆ again according to the attribute types for that part sort
  - ⋆ and, more specifically, choosing initial attribute values.
- This part is then “joined” to σ[5] and
- its unique identifier “joined” to uid_σ[6].

[5](the global part state), Cf. Sect. 4.7.2 on Slide 155

[6](the global unique identifier state), Cf. Sect. 5.2.4 on Slide 193

B. Then to transcendentally deduce that part into a behaviour:
   - initialised (according to Sect. 6.3.1) with
     - ⋆ the unique identifier,
     - ⋆ the mereology, and
     - ⋆ the attribute values
   - This behaviour is then invoked and "joined" to the set of current behaviours, cf. Sect. 6.4 on Slide 403 – i.e., just above !

C. Then, finally, to "adjust" the mereologies of topologically or conceptually related parts,
   - that is, for each of these parts to update:
   - their mereology and possibly some
   - state and state space

   arguments of their corresponding behaviours.

D. The update of the mereologies
of already “running” behaviours requires the following:

- that, potentially all, behaviours offers to accept
- mereology update requests from the “creator” behaviour.

- The latter means, practically speaking,
  - that each part/behaviour
  - which may be subject to mereology changes
  - externally non-deterministically
  - expresses an offer to accept such a change.

Example 69 . Road Net Administrator:

- We introduce the road net behaviour – based on the road net composite part, RN.

151. The road net has a programmable attribute: a *road net (development & maintenance) graph*.[7]

- The road net graph consists of a quadruple:
  - a map that for each hub identifier records “all” the information that the road net administrator deems necessary[8] for the maintenance and development of road net hubs;
  - a map that for each link identifier records “all” the information[9] that the road net administrator deems necessary for the maintenance and development of road net links;
  - and a map from the hub identifiers to the set of identifiers of the links it is connected to, and
  - the set of all automobile identifiers.

[7] The presentation of the road net Behaviour, rn, is simplified.
[8] We presently abstract from what this information is.
[9] See footnote 8.

152. This graph is commensurate with the actual topology of the road net.

```
type
151.  $\mathsf{G} = (\mathsf{HI} \xrightarrow[m]{} \mathsf{H\_Info}) \times (\mathsf{LI} \xrightarrow[m]{} \mathsf{L\_Info}) \times (\mathsf{HI} \xrightarrow[m]{} \mathsf{LI}\text{-}\mathbf{set}) \times \mathsf{AI}\text{-}\mathbf{set}$
value
151.  $\mathsf{attr\_G}: \mathsf{RN} \rightarrow \mathsf{G}$
axiom
151.  $\forall\ (\mathsf{hi\_info},\mathsf{li\_info},\mathsf{map},\mathsf{ais}):\mathsf{G} \cdot$
151.     $\mathbf{dom}\ \mathsf{map} = \mathbf{dom}\ \mathsf{hi\_info} = his \wedge \cup\ \mathbf{rng}\ \mathsf{map} = \mathbf{dom}\ \mathsf{li\_info} = lis \wedge$
152.     $\forall\ \mathsf{hi}:\mathsf{HI} \cdot \mathsf{hi} \in \mathbf{dom}\ \mathsf{hi\_info} \Rightarrow$
152.        $\mathbf{let}\ \mathsf{h}:\mathsf{H} \cdot \mathsf{h} \in \sigma \wedge \mathsf{uid\_H(h)}{=}\mathsf{hi}\ \mathbf{in}$
152.        $\mathbf{let}\ (\mathsf{lis'},...) = \mathsf{mereo\_H(h)}\ \mathbf{in}\ \mathsf{lis'} = \mathsf{map(hi)}$
152.        $\mathsf{ais} \subseteq ais \wedge ...$
152.        end end
```

- Please note the fundamental difference between
  - the *road net (development & maintenance) graph* and
  - the road net.
- The latter pretends to be “the real thing”.
- The former is “just” an abstraction thereof !

153. The road net mereology ("bypasses") the hub and link aggregates, and comprises a set of hub identifiers and a set of link identifiers – of the road net[10].

type
153. H_Mer = AI-**set** $\times$ LI-**set**
153. mereo_RN: RN $\rightarrow$ RNMer
axiom
153. $\forall$ rts:RTS $\cdot$ **let** (__,lis) = mereo_H(obs_RN(rts)) **in** lis $\subseteq$ lis **end**
value

[10]This is a repeat of the hub mereology given in Item 65 on Slide 213.

154. The road net [administrator] behaviour,
155. amongst other activities (. . .)
156. internal non-deterministically decides upon
    (a) either a hub insertion,
    (b) or a link insertion,
    (c) or a hub removal,
    (d) or a link removal;

- These four sub-behaviours each resume being the road net behaviour.

value

```
154.  rn: RNI → RNMer → G → in,out{ch[ {i,j} ]|{i,j}⊆$uid_\sigma$}
154.  rn(rni)(rnmer)(g) ≡
155.      ...
156a.     ⨅ insert_hub(g)(rni)(rnmer)
156b.     ⨅ insert_link(g)(rni)(rnmer)
156c.     ⨅ remove_hub(g)(rni)(rnmer)
156d.     ⨅ remove_link(g)(rni)(rnmer)
```

157. These road net sub-behaviours require information about
   (a) a hub to be inserted: its initial state, state space and [empty] traffic history, or
   (b) a link to be inserted: its length, initial state, state space and [empty] traffic history, or
   (c) a hub to be removed: its unique identifier, or
   (d) a link to be removed: its unique identifier.

type
157. $\mathsf{Info} == \mathsf{nHInfo} \mid \mathsf{nLInfo} \mid \mathsf{oHInfo} \mid \mathsf{oLInfo}$
157. $\mathsf{nHInfo} :: \mathsf{H}\Sigma \times \mathsf{H}\Omega \times \mathsf{H\_Traffic}$
157. $\mathsf{nLInfo} :: \mathsf{LEN} \times \mathsf{L}\Sigma \times \mathsf{L}\Omega \times \mathsf{L\_Traffic}$
157. $\mathsf{oHInfo} :: \mathsf{HI}$
157. $\mathsf{oLInfo} :: \mathsf{LI}$ ■

Example 70 . Road Net Development: Hub Insertion:

- Road net development alternates between design,
  - based on the *road net (development & maintenance) graph*, and
- actual, “real life”, construction
  - taking place in the real surroundings of the road net.

158. If a hub insertion then the road net behaviour, based on the hub and link information and the road net layout in the *road net (development & maintenance) graph* selects
    (a) an initial mereology for the hub, h_mer,
    (b) an initial hub state, hσ, and
    (c) an initial hub state space, hω, and
    (d) an initial, i.e., empty hub traffic history;
159. updates its *road net (development & maintenance) graph* with information about the new hub,
160. and results in a suitable grouping of these.

```
value
158.  design_new_hub: G → (nHInfo×G)
158.  design_new_hub(g) ≡
158a.     let h_mer:HMer = $\mathcal{M}_{ih}$(g),
158b.         h$\sigma$:H$\Sigma$ = $\mathcal{S}_{ih}$(g),
158c.         h$\omega$:H$\Omega$ = $\mathcal{O}_{ih}$(g),
158d.         h_traffic = [ ],
159.          g′ = $\mathcal{MSO}_{ih}$(g) in
160.      ((h_mer,h$\sigma$,h$\omega$,h_traffic),g′) end
```

- We leave open, in Items 158a–158c, as to what the initial hub mereology, state and state space should be initialised, i.e., the $\mathcal{M}_{ih}$,$\mathcal{S}_{ih}$,$\mathcal{O}_{ih}$ and $\mathcal{MSO}_{ih}$ functions.

161. To insert a new hub the road net administrator
    (a) first designs the new hub,
    (b) then selects a hub part
    (c) which satisfies the design,
        whereupon it updates the global states
    (d) of parts σ,
    (e) of unique identifiers, and
    (f) of hub identifiers –

  in parallel, and in parallel with

162. initiating a new hub behaviour
163. and resuming being the road net behaviour.

```
161.  insert_hub: G×RNI×RNMer → Unit
161.  insert_hub(g,rni,rnmer) ≡
161a.     let ((h_mer,$h\sigma$,$h\omega$,h_traffic),g′) = design_new_hub(g) in
161b.     let h:H · h$\notin\sigma$ ·
161c.             mereo_H(h)=h_mer ∧ $h\sigma$=attr_H$\Sigma$(h) ∧
161c.             $h\omega$=attr_H$\Omega$(h) ∧ h_traffic=attr_HTraffic(h) in
161d.     $\sigma$ := $\sigma$ ∪ {h}
161e.   ∥ $uid_\sigma$ := $uid_\sigma$ ∪ {uid_H(h)}
161f.   ∥ his := his ∪ {uid_H(h)}
162.    ∥ hub(uid_H(h))(attr_H$\Sigma$(h),attr_H$\Omega$(h),attr_H$\Omega$(h))
163.    ∥ rn(rni)(rnmer)(g′)
161.     end end  ■
```

Example 71 . Road Net Development: Link Insertion:

164. If a link insertion then the road net behaviour
     based on the hub and link information and the road net layout
     in the *road net (development & maintenance) graph* selects
     (a) the mereology for the link, h_mer[11],
     (b) the (static) length (attribute),
     (c) an initial link state, lσ,
     (d) an initial link state space lω, and
     (e) and initial, i.e., empty, link traffic history;
165. updates its *road net (development & maintenance) graph*
     with information about the new link,
166. and results in a suitable grouping of these.

[11]that is, the two existing hub identifiers between whose hubs the new link is to be inserted

```
value
164.  design_new_link: G → (nLInfo×G)
164.  design_new_link(g) ≡
164a.     let l_mer:LMer = $\mathcal{M}_{il}$(g),
164b.         le:LEN = $\mathcal{L}_{il}$(g),
164c.         l$\sigma$:L$\Sigma$ = $\mathcal{S}_{il}$(g),
164d.         l$\omega$:L$\Omega$ = $\mathcal{O}_{il}$(g),
164e.         l_hist:L_Hist = [ ]
165.          g′:G = $\mathcal{MLSO}_{il}$(g) in
166.      ((l_mer,le,l$\sigma$,l$\omega$,l_hist),g′) end
```

- We leave open, in Items 164a–164d, as to what the initial link mereology, state and state space should be initialised.

167. To insert a new link the road net administrator
   (a) first designs the new link,
   (b) then selects a link part
   (c) which satisfies the design,
       whereupon it updates the global states
   (d) of parts, σ,
   (e) of unique part identifiers, and
   (f) of link identifiers –

   in parallel, and in parallel with

168. initiating a new link behaviour and
169. updating the mereologies and possibly the state and the state space attributes of the connected hubs.

```
value
167.  insert_link: G → Unit
167.  insert_link(rni,l) ≡
167a.     let ((l_mer,le,lσ,lω,l_traffic_hist),g′) = design_new_link(g) in
167c.     let l:L · l∉σ · mereo_L(l)=l_mer ∧
167c.                 le=attr_LEN(l) ∧ lσ=attr_LΣ(l) ∧
167c.                 lω=attr_LΩ(l) ∧ l_traffic_hist=attr_HTraffic(l) in
167d.     σ := σ ∪ {l}
167e.  ‖ uid_σ := uid_σ ∪ {uid_L(l)}
167f.  ‖ lis := list ∪ {}
168.   ‖ link(uid_L(l))(l_mer)(le,lω)(lσ,l_traffic)
169.   ‖ ch[ {rni,hi1} ] ! updH(M_il(g),Σ_il(g),Ω_il(g))
169.   ‖ ch[ {rni,hi2} ] !
167.      end end ■
```

- We leave undefined the mereology and the state σ and state space ω update functions.

6.5.1.2 Destroy Entities

- The introduction to Sect. 6.5.1.1 on Slide 409
  on the *creation of entities*
  - outlined a number of creation issues ([A, B, C, D]).
- For the *destruction of entities*
  - description matters are a bit simpler.
- It is, almost, simply a matter
  - of designating, by its unique identifier,
  - the entity: part and behaviour to be destroyed.
- Almost !
  - The mereology of the destroyed entity
  - must be such that the destruction
  - does not leave “dangling” references !

Example 72 . Road Net Development: Hub Removal:

170. If a hub removal then the road net design_remove_hub behaviour, based on the *road net (development & maintenance) graph*, calculates the *unique hub identifier* of the "isolated" hub to be removed – that is, is not connected to any links,
171. updates the *road net (development & maintenance) graph*, and
172. results in a pair of these.

```
value
170.  design_remove_hub: G → (HI×G)
170.  design_remove_hub(g) as (hi,g′)
170.     let hi:HI · hi ∈ $his$ ∧ let (_,lis) = mereo_H(retr_part(hi)) in lis={} end in
171.     let g′ = $\mathcal{M}_{rh}$(hi,g) in
172.     (hi,g′) end end
```

173. To remove a hub the road net administrator
    (a) first designs which old hub is to be removed
    (b) then removes the designated hub,
        whereupon it updates the global states
    (c) of parts σ,
    (d) of unique identifiers, and
    (e) of hub identifiers –

    in parallel, and in parallel with

174. stopping the old hub behaviour
175. and resuming being a road net behaviour.

```
value
173.  remove_hub: G → RNI → RNMer → Unit
173.  remove_hub(g)(rni)(rnmer) ≡
173a.      let (hi,g′) = design_remove_hub(g) in
173b.      let h:H · uid_H(h)=hi ∧ ... in
173c.      σ := σ \ {h}
173d.  ‖ uid_σ := uid_σ \ {hi}
173e.  ‖ his := his \ {hi}
174.     ‖ ch[ {rni,hi} ] ! mkStop()
175.     ‖ rn(rni)(rnmer)(g′)
173.       end end ■
```

6.5.2 Adjustment of Creatable and Destructable Behaviours

- When an entity
  - is created or destroyed
  - its creation, respectively destruction
  - affects the neurologically related parts and their behaviours.
    - ⋆ their mereology
    - ⋆ and possibly their programmable state attributes
    - ⋆ need be adjusted.
  - And when entities are destroyed
    their behaviours are **stopped** !
  - These entities are “informed” so by the creator/destructor entity
    – as was shown in Examples 70–72.
- The next example will illustrate how such ‘affected’ entities
  handle such creator/destructor communication.

Example 73 . Hub Adjustments:

- We have not yet illustrated hub (nor link) behaviours.
- Now we have to !

176. The mereology of a hub is a triple:
 the identification of the set of automobiles that may enter the hub,
 the identification of the set of links that connect to the hub,
 and the identification of the road net.
177. The hub behaviour external non-deterministically (⌷) alternates between
178. doing "own work",
179. or accepting a stop "command" from the road net administrator, or
180. or accepting mereology & state update information,
181. or other.

```
type
176.  HMer = AI-set × LI-set × RNI
value
176.  mereo_H: H → HMer
177.  hub: hi:HI → (auis,lis,rni):HMer → hω:HΩ → (hσ:HΣ×ht:HTraffic) →
177.          {ch[ hi,ui ]|ui:(RNI|AI) · ui=rni∨ui ∈ auis}  Unit
177.  hub(hi)(hm:(auis,lis,rni))(hω)(hσ,ht) ≡
178.         ...
179.      ⌷ let mkStop() = ch[ hi,rni ] ? in stop end
180.      ⌷ let mkUpdH(hm′,hσ′,hσ′) = ch[ {rni,hi} ] ? in
180.         hub(hi)(hm′)(hω′)(hσ′,ht) end
181.         ...
```

- Observe from formula Item 179 that the hub behaviour ends,
- whereas “from” Item 180 it tail recurses ! ■

6.5.3 Summary on Creatable & Destructable Entities

- We have sketched how we may model
  the dynamics of creating and destroying entities.
  - It is, but a sketch.
  - We should wish for a more methodological account.
  - So, that is what we are working on – amongst other issues – at the moment.

6.6 Domain Engineering: Description and Construction

- There are two meanings to the term 'Domain Engineering'.
  - the construction of *descriptions* of domains, and
  - the construction of *domains*.
  - Most sections of Chapters 4–6
    are "devoted" to the former;
  - the previous section, Sect. 6.5 to the latter.

6.7 Domain Laws

TO BE WRITTEN

6.8 A Domain Discovery Procedure, III

The predecessors of this section are Sects. 4.8.2 on Slide 159 and 5.7 on Slide 363.

6.8.1 Review of the Endurant Analysis and Description Process

- The discover_.... functions below were defined in Sects. 4.8.2 on Slide 159 and 5.7 on Slide 363.

```
value
   endurant_analysis_and_description: Unit → Unit
   endurant_analysis_and_description() ≡
      discover_sorts();                              [Page 160]
      discover_internal_endurant_qualities()   [Page 365]
```

- We are now to define a perdurant_analysis_and_description procedure –
- to follow the above endurant_analysis_and_description procedure.

6.8.2 A Domain Discovery Process, III

- We define the perdurant_analysis_and_description procedure
  - in the reverse order of that of Sect. 5.7 on Slide 363,
  - first the full procedure,
  - then its sub-procedures.

A Domain Endurant Analysis and Description Process

```
value
  perdurant_analysis_and_description: Unit → Unit
  perdurant_analysis_and_description() ≡
     discover_state();                    axiom ...  [ Note (a) ]
     discover_channels();                 axiom ...  [ Note (b) ]
     discover_behaviour_signatures(); axiom ...  [ Note (c) ]
     discover_behaviour_definitions(); axiom ...  [ Note (d) ]
     discover_initial_system()            axiom ...  [ Note (e) ]
```

- ***Notes***:
  - (a) The States: σ and ui_σ
    - We refer to Sect. 4.7.2 on Slide 155 and Sect. 5.2.4 on Slide 193.
    - The state calculation, as shown on Page 150, must be replicated, i.e., re-discovered, in any separate domain analysis & description.
    - The purpose of the state, i.e., σ, is to formulate appropriate axiomatic constraints and domain laws.
  - (b) The Channels:
    - We refer to Sects. 6.1.2 on Slide 380 and 6.2 on Slide 381.
    - Thus we indiscriminately declare a channel for each pair of distinct unique part identifiers
      whether the corresponding pair of part behaviours,
      if at all invoked, communicate or not.

- (c) **Behaviour Signatures:**
  - ⋆ We refer to Sect. 6.3.1.2 on Slide 384.
  - ⋆ We find it more productive to first settle on the signatures of all behaviours – careful thinking has to go into that –
  - ⋆ before tackling the far more time-consuming work on defining the behaviours:
- (d) **Behaviour Definitions:**
  - ⋆ We refer to Sect. 6.3.3 on Slide 394.
- (e) **The Running System:**
  - ⋆ We refer to Sect. 6.4 on Slide 403.

6.9 Summary

Perdurants: Analysis & Description

#	Name	Introduced
	Discovery Functions	
	discover_channels	page **381**
	discover_behaviour_signatures	page **393**
	discover_behaviour_definitions	page **396**
	discover_initial_system	page **403**
	perdurant_analysis_and_description	page **440**

• • •

- Please consider Fig. 4.1 on Slide 63.
  - This chapter has covered the right of Fig. 4.1.

- **Traversal of Analysis & Description Ontology Graph**

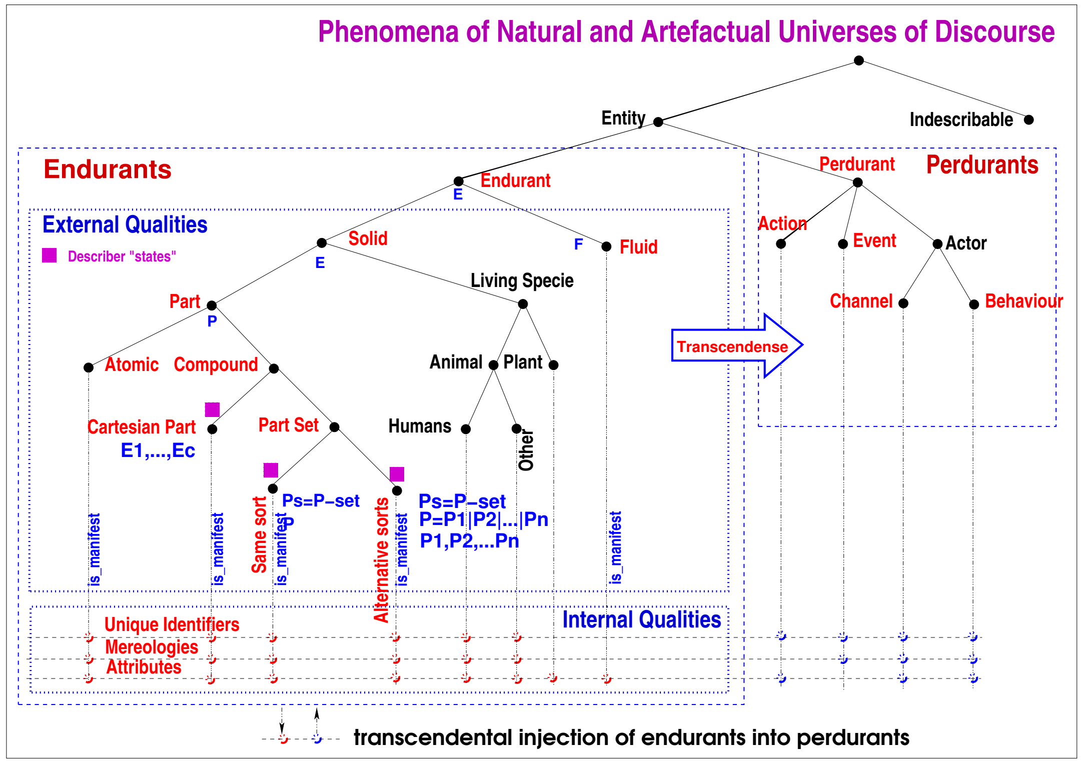

Figure 7.1: Upper Ontology

- From Programming Language Semantics to Domain Models
  - Programming Language Semantics
    - ⋆ The *IBM Vienna Labor* PL/I Definition, 1974.
    - ⋆ The *Dansk Datamatik Center, DDC* `CHILL` and Ada Formal Descriptions, 1978-1985.
  - Domain Models give semantics to
    - ⋆ nouns, endurants, and
    - ⋆ verbs, perdurants,

    of domains.

- **Domain Specific Languages: DSL**
  - A **DSL** is a language whose "primitives"
    directly reflects a specific domain's basic entities.
  - **RSL** is not a 'domain specific language'
  - **Domain Models** form the basis for
    the conception of one or more specific **DSL**s.
  - The **semantics** of these **DSL**s derives, then,
    from the **Domain Model.**
  - It is suggested, therefore, that **DSL**s be conceived on the basis of
    domain models.

- RSL vs. RSL$^+$
  - Informal RSL$^+$ is used
    in explaining the domain analysis & description method.
  - RSL is used, independently,
    as the formal specification language
    in which to describe domains.
  - The two are otherwise unrelated !

- **Algorithms vs. Domain Descriptions**
  - Algorithms are ‘the’ hallmark of Computing !
  - Clever algorithms (and data structures) are needed to efficiently implement requirements prescriptions.
  - Thus algorithmics enter our concern during software design.

- Domain Facets
  - Intrinsics
  - Technology Support
  - Rules & Regulations
  - Scripts, Contracts
  - Management & Organisation
  - Human Behaviour

- Requirements Engineering
  - 'The Machine'
  - Domain Engineering
    - Projection
    - Instantiation
    - Determination
    - Extension
    - Fitting
  - Interface and Derived Requirements
    - Interface Requirements
      - Shared Endurants
      - Shared Perdurants
    - Derived Requirements
      - Shared with Machine
  - Machine Requirements

- Research/PhD Study Topics
  - Intentional Pull
  - Discrete vs Continuous
  - A Calculus of Perdurants
  - Human Interaction
  - Transcendental Deduction
  - Formal Ontology Models
  - Philosophy

THANKS

Appendix A. Road Transport

A.1 The Road Transport Domain

- Our universe of discourse in this chapter is the road transport domain.

A.1.1 Naming

type RTS

A.1.2 Rough Sketch

- The road transport system that we have in mind consists of
  - a road net and
  - a set of vehicles
  - such that the road net serves to convey vehicles.
- We consider the road net to consist of
  - hubs, i.e., street intersections, or just street segment connection points, and
  - links, i.e., street segments between adjacent hubs.

- We consider vehicles to additionally include
  - departments of motor vehicles (DMVs),
  - bus companies, each with zero, one or more buses, and
  - vehicle associations, each with
    - zero, one or more members
    - who are owners of zero, one or more vehicles[1] ■

[1]This “rough” narrative fails to narrate what ...

A.2 External Qualities

A Road Transport System, I – Manifest External Qualities:

- Our intention is that the manifest external qualities of a road transport system are those of its
  - roads,
    - their **hubs** i.e., road (or street) intersections, and
    - their **links**, i.e., the roads (streets) between hubs, and
  - **vehicles**, i.e., automobiles – that ply the roads –
    - the buses, trucks, private cars, bicycles, etc. ■

A.2.1 A Road Transport System, II – Abstract External Qualities

- Examples of what could be considered abstract external qualities of a road transport domain are:
  - the aggregate of all hubs and all links,
  - the aggregate of all buses, say into bus companies,
  - the aggregate of all bus companies into public transport, and
  - the aggregate of all vehicles into a department of vehicles.
- Some of these aggregates may, at first be treated as abstract.
- Subsequently, in our further analysis & description we may decide to consider some of them as concretely manifested in, for example, actual
  - departments of roads.

A.2.2 Transport System Structure

- A transport system is modeled 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 modeled as set of hubs, respectively links.

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

A.2.3 Atomic Road Transport Parts

- From one point of view all of the following can be considered atomic parts:
  - hubs,
  - links[2], and
  - automobiles.

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

A.2.4 Compound Road Transport Parts

A.2.4.1 The Composites

182. There is the *universe of discourse*, UoD.

It is structured into

183. a *road net*, RN, and
184. a *fleet of vehicles*, FV.

Both are structures. . . . .

type
182 UoD **axiom** ∀ uod:UoD · is_structure(uod).
183 RN **axiom** ∀ rn:RN · is_structure(rn).
184 FV **axiom** ∀ fv:FV · is_structure(fv).
value
183 obs_RN: UoD → RN
184 obs_FV: UoD → FV ■

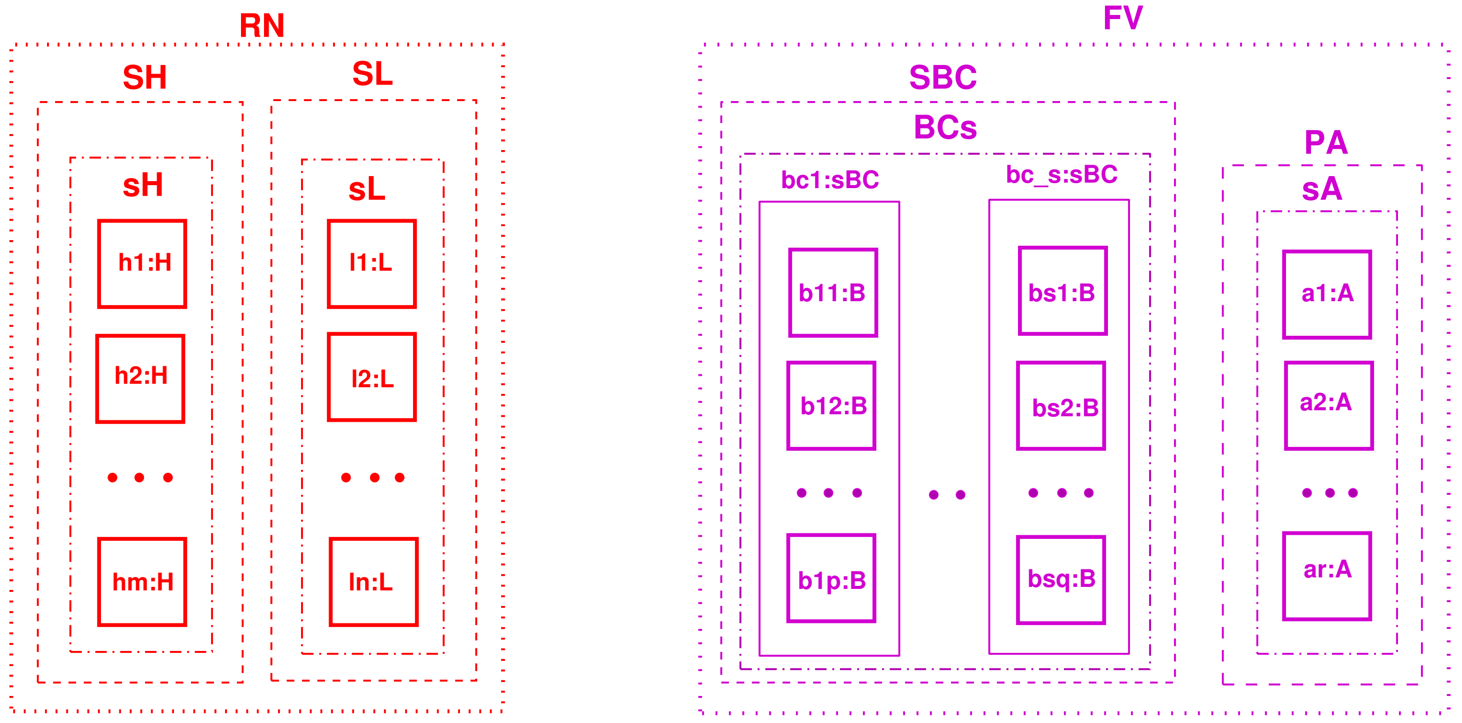

Figure A.1: A Road Transport System Compounds and Structures

A.2.4.2 The Part Parts

185. The structure of hubs is a set, sH, of atomic hubs, H.
186. The structure of links is a set, sL, of atomic links, L.
187. The structure of buses is a set, sBC, of composite bus companies, BC.
188. The composite bus companies, BC, are sets of buses, sB.
189. The structure of private automobiles is a set, sA, of atomic automobiles, A.

```
type
185  H, sH = H-set  axiom ∀ h:H · is_atomic(h)
186  L, sL = L-set  axiom ∀ l:L · is_atomic(l)
187  BC, BCs = BC-set  axiom ∀ bc:BC · is_composite(bc)
188  B, Bs = B-set  axiom ∀ b:B · is_atomic(b)
189  A, sA = A-set  axiom ∀ a:A · is_atomic(a)
value
185  obs_sH: SH → sH
186  obs_sL: SL → sL
187  obs_sBC: SBC → BCs
188  obs_Bs: BCs → Bs
189  obs_sA: SA → sA ■
```

A.2.5 The Transport System State

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

From that state we can calculate other states.

191. The set of all hubs, *hs*.
192. The set of all links, *ls*.
193. The set of all hubs and links, *hls*.
194. The set of all bus companies, *bcs*.
195. The set of all buses, *bs*.
196. The set of all private automobiles, *as*.
197. The set of all parts, *ps*.

value

190 rts:UoD [34]

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

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

193 hls:(H|L)-**set** ≡ hs∪ls

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

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

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

197 ps:(UoB|H|L|BC|B|A)-**set** ≡ rts∪hls∪bcs∪bs∪as

A.3 Internal Qualities

A.3.1 Unique Identifiers

198. We assign unique identifiers to all parts.
199. By a road identifier we shall mean a link or a hub identifier.
200. By a vehicle identifier we shall mean a bus or an automobile identifier.
201. 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
198   H_UI, L_UI, BC_UI, B_UI, A_UI
199   R_UI = H_UI | L_UI
200   V_UI = B_UI | A_UI
value
201a   uid_H: H → H_UI
201b   uid_L: H → L_UI
201c   uid_BC: H → BC_UI
201d   uid_B: H → B_UI
201e   uid_A: H → A_UI
```

A.3.1.1 Extract Parts from Their Unique Identifiers

202. From the unique identifier of a part we can retrieve, $\wp$, the part having that identifier.

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

A.3.1.2 All Unique Identifiers of a Domain

We can calculate:

203. the *set*, $h_{ui}s$, of *u*nique *h*ub *i*dentifiers;
204. the *set*, $l_{ui}s$, of *u*nique *l*ink *i*dentifiers;
205. the *map*, $hl_{ui}m$, from *u*nique *h*ub *i*dentifiers to the *set* of *u*nique *l*ink *i*identifiers of the links connected to the zero, one or more identified hubs,
206. the *map*, $lh_{ui}m$, from *u*nique *l*ink *i*dentifiers to the *set* of *u*nique *h*ub *i*identifiers of the two hubs connected to the identified link;
207. the *set*, $r_{ui}s$, of all *u*nique hub and link, i.e., *r*oad *i*dentifiers;
208. the *set*, $bc_{ui}s$, of *u*nique *b*us *c*ompany *i*dentifiers;

209. the *s*et, $b_{ui}s$, of *u*nique *b*us *i*dentifiers;
210. the *s*et, $a_{ui}s$, of *u*nique private *a*utomobile *i*dentifiers;
211. the *s*et, $v_{ui}s$, of *u*nique bus and automobile, i.e., *v*ehicle *i*dentifiers;
212. the *m*ap, $bcb_{ui}m$, from *u*nique *b*us *c*ompany *i*dentifiers to the *s*et of its *u*nique *b*us *i*dentifiers; and
213. the (*b*ijective) *m*ap, $bbc_{ui}bm$, from *u*nique *b*us *i*dentifiers to their *u*nique *b*us *c*ompany *i*dentifiers.

value

203 $h_{ui}s$:H_UI-**set** ≡ {uid_H(h)|h:H·h ∈ hs}

204 $l_{ui}s$:L_UI-**set** ≡ {uid_L(l)|l:L·l ∈ ls}

207 $r_{ui}s$:R_UI-**set** ≡ $h_{ui}s$∪$l_{ui}s$

205 $hl_{ui}m$:(H_UI $\underset{m}{\rightarrow}$L_UI-**set**) ≡

205 [h_ui↦luis|h_ui:H_UI,luis:L_UI-**set**·h_ui∈$h_{ui}s$∧(_,luis,_)=mereo_H(η(h_ui))]

206 $lh_{ui}m$:(L+UI $\underset{m}{\rightarrow}$H_UI-**set**) ≡

206 [l_ui↦huis | h_ui:L_UI,huis:H_UI-**set** · l_ui∈$l_{ui}s$ ∧ (_,huis,_)=mereo_L(η(l_ui)

208 $bc_{ui}s$:BC_UI-**set** ≡ {uid_BC(bc)|bc:BC·bc ∈ bcs}

209 $b_{ui}s$:B_UI-**set** ≡ ∪{uid_B(b)|b:B·b ∈ bs}

210 $a_{ui}s$:A_UI-**set** ≡ {uid_A(a)|a:A·a ∈ as}

211 $v_{ui}s$:V_UI-**set** ≡ $b_{ui}s$ ∪ $a_{ui}s$

212 $bcb_{ui}m$:(BC_UI $\underset{m}{\rightarrow}$B_UI-**set**) ≡

212 [bc_ui ↦ buis | bc_ui:BC_UI, bc:BC · bc∈bcs ∧ bc_ui=uid_BC(bc) ∧ (_,_,buis

213 $bbc_{ui}bm$:(B_UI $\underset{m}{\rightarrow}$BC_UI) ≡

213 [b_ui ↦ bc_ui | b_ui:B_UI,bc_ui:BC_ui · bc_ui=**dom**$bcb_{ui}m$∧b_ui∈$bcb_{ui}m$(bc_u

A.3.1.3 Uniqueness of Road Net Identifiers

- We must express the following axioms:

214. All hub identifiers are distinct.
215. All link identifiers are distinct.
216. All bus company identifiers are distinct.
217. All bus identifiers are distinct.
218. All private automobile identifiers are distinct.
219. All part identifiers are distinct.

axiom

214 $\mathbf{card}\ hs = \mathbf{card}\ h_{ui}s$

215 $\mathbf{card}\ ls = \mathbf{card}\ l_{ui}s$

216 $\mathbf{card}\ bcs = \mathbf{card}\ bc_{ui}s$

217 $\mathbf{card}\ bs = \mathbf{card}\ b_{ui}s$

218 $\mathbf{card}\ as = \mathbf{card}\ a_{ui}s$

219 $\mathbf{card}\ \{h_{ui}s \cup l_{ui}s \cup bc_{ui}s \cup b_{ui}s \cup a_{ui}s\}$

219 $\quad = \mathbf{card}\ h_{ui}s + \mathbf{card}\ l_{ui}s + \mathbf{card}\ bc_{ui}s + \mathbf{card}\ b_{ui}s + \mathbf{card}\ a_{ui}s$ ■

A.3.2 Mereology

A.3.2.1 Mereology Types and Observers

220. The mereology of hubs is a pair: (i) the set of all bus and automobile identifiers[3], and (ii) the set of unique identifiers of the links that it is connected to and the set of all unique identifiers of all vehicles (buses and private automobiles).[4]
221. 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.
222. The mereology of a bus company is a set the unique identifiers of the buses operated by that company.
223. 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[5].
224. The mereology of an automobile is the set of the unique identifiers of all links and hubs[6].

type
220 H_Mer = V_UI-**set**×L_UI-**set**
221 L_Mer = V_UI-**set**×H_UI-**set**
222 BC_Mer = B_UI-**set**
223 B_Mer = BC_UI×R_UI-**set**
224 A_Mer = R_UI-**set**

value
220 mereo_H: H → H_Mer
221 mereo_L: L → L_Mer
222 mereo_BC: BC → BC_Mer
223 mereo_B: B → B_Mer
224 mereo_A: A → A_Mer

A.3.2.2 Invariance of Mereologies

- For mereologies one can usually express some invariants.
  - Such invariants express *“law-like properties”*,
  - facts which are indisputable.

[3]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.
[4]The 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.
[5]— that the bus might pass through
[6]— that the automobile might pass through

A.3.2.2.1 Invariance of Road Nets

- The observed mereologies must express identifiers of the state of such for road nets:

axiom

220 ∀ (vuis,luis):H_Mer · luis$\subseteq l_{ui}s$ $\wedge$ vuis$=v_{ui}s$

221 ∀ (vuis,huis):L_Mer · vuis$=v_{ui}s$ $\wedge$ huis$\subseteq h_{ui}s$ $\wedge$ **card**huis=2

222 ∀ buis:H_Mer · buis $= b_{ui}s$

223 ∀ (bc_ui,ruis):H_Mer·bc_ui$\in bc_{ui}s$$\wedge$ruis$=r_{ui}s$

224 ∀ ruis:A_Mer · ruis$=r_{ui}s$

225. For all hubs, h, and links, l, in the same road net,
226. if the hub h connects to link l then link l connects to hub h.

```
axiom
225  ∀ h:H,l:L · h ∈ hs ∧ l ∈ ls ⇒
225      let (_,luis)=mereo_H(h), (_,huis)=mereo_L(l)
226      in uid_L(l)∈luis ≡ uid_H(h)∈huis end
```

227. For all links, l, and hubs, h_a, h_b, in the same road net,
228. if the l connects to hubs h_a and h_b, then h_a and h_b both connects to link l.

```
axiom
227  ∀ h_a,h_b:H,l:L · {h_a,h_b} ⊆ hs ∧ l ∈ ls ⇒
227      let (_,luis)=mereo_H(h), (_,huis)=mereo_L(l)
228      in uid_L(l)∈luis ≡ uid_H(h)∈huis end
```

A.3.2.2.2 Possible Consequences of a Road Net Mereology

229. are there [isolated] units from which one can not “reach” other units ?
230. does the net consist of two or more “disjoint” nets ?
231. et cetera.

- We leave it to the reader to narrate and formalise the above properly.

A.3.2.2.3 Fixed and Varying Mereology

- Let us consider a road net.
  - If hubs and links never change "affiliation", that is:
    - ⋆ hubs are in fixed relation to zero one or more links, and
    - ⋆ links are in a fixed relation to exactly two hubs
    - ⋆ then the mereology is a *fixed mereology*.

- If, on the other hand
  - ⋆ hubs may be inserted into or removed from the net, and/or
  - ⋆ links may be removed from or inserted between any two existing hubs,
  - ⋆ then the mereology is a *varying mereology*.

A.3.3 Attributes

A.3.3.1 Hub Attributes

- We treat some attributes of the hubs of a road net.

232. There is a hub state.
    - It is a set of pairs, $(\mathsf{l}_f,\mathsf{l}_t)$, of link identifiers,
        - where these link identifiers are in the mereology of the hub.
    - The meaning of the hub state
        - in which, e.g., $(\mathsf{l}_f,\mathsf{l}_t)$ 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.

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

234. Since we can think rationally about it,
    - it can be described, hence we 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.
    - Hub history is an *event history*.

type
232 $H\Sigma$ = (L_UI×L_UI)-**set**
axiom
232 $\forall$ h:H · obs_HΣ(h) $\in$ obs_HΩ(h)
type
233 HΩ = HΣ-**set**
234 H_Traffic
234 H_Traffic = (A_UI|B_UI) $\xrightarrow[m]{}$ ($\mathbb{TIME}$ × VPos)*
axiom
234 $\forall$ ht:H_Traffic,ui:(A_UI|B_UI) ·
234 ui $\in$ **dom** ht $\Rightarrow$ time_ordered(ht(ui))
value
232 attr_HΣ: H $\rightarrow$ HΣ
233 attr_HΩ: H $\rightarrow$ HΩ
234 attr_H_Traffic: H $\rightarrow$ H_Traffic
value
234 time_ordered: ($\mathbb{TIME}$ × VPos)* $\rightarrow$ **Bool**
234 time_ordered(tvpl) $\equiv$ ...

- In Item 234 on the preceding slide we model the time-ordered sequence of traffic as a discrete sampling, i.e., $\xrightarrow[m]{}$, rather than as a continuous function, $\rightarrow$.

A.3.3.2 Invariance of Traffic States

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

axiom

```
235 ∀ h:H · h ∈ $hs$ ⇒
235     let hσ = attr_HΣ(h) in
235     ∀ ($l_{ui}i$,$li_{ui}i'$):(L_UI×L_UI) · ($l_{ui}i$,$l_{ui}i'$) ∈ hσ ⇒ {$l_{ui_i}$,$l'_{ui_i}$} ⊆ $l_{ui}s$ end
```

A.3.3.3 Link Attributes

We show just a few attributes.

236. There is a link state. It is a set of pairs, (h_f,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 (h_f,h_t) is an element is that the link is open, “green”, for traffic *f*rom hub h_f *t*o hub h_t. Link states can have either 0, 1 or 2 elements.
237. 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”.

238. 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.
239. The hub identifiers of link states must be in the set, $h_{u_i}s$, of the road net's hub identifiers.

```
type
236  LΣ = H_UI-set
axiom
236  ∀ lσ:LΣ·card lσ=2
236  ∀ l:L · obs_LΣ(l) ∈ obs_LΩ(l)
type
237  LΩ = LΣ-set
238  L_Traffic
238  L_Traffic = (A_UI|B_UI) $\xrightarrow[m]{}$ (𝕋×(H_UI×Frac×H_UI))*
238  Frac = Real, axiom frac:Fract · 0<frac<1
value
236  attr_LΣ: L → LΣ
237  attr_LΩ: L → LΩ
238  attr_L_Traffic: : → L_Traffic
axiom
238  ∀ lt:L_Traffic,ui:(A_UI|B_UI)·ui ∈ dom ht ⇒ time_ordered(ht(ui))
239  ∀ l:L · l ∈ $ls$ ⇒
239      let lσ = attr_LΣ(l) in ∀ ($h_{ui}i$,$h_{ui}i'$):(H_UI×K_UI) ·
239         ($h_{ui}i$,$h_{ui}i'$) ∈ lσ ⇒ {$h_{ui_i}$,$h'_{ui_i}$} ⊆ $h_{ui}s$ end
```

A.3.3.4 Bus Company Attributes

- 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.

240. Bus companies create, maintain, revise and distribute
[to the public (not modeled here), and to buses]
bus time tables, not further defined.

type
240 BusTimTbl
value
240 attr_BusTimTbl: BC $\rightarrow$ 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.

A.3.3.5 Bus Attributes

We show just a few attributes.

241. Buses run routes, according to their line number, ln:LN, in the
242. bus time table, btt:BusTimTbl obtained from their bus company, and and keep, as inert attributes, their segment of that time table.
243. 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.
244. Et cetera.

type
241 LN
242 BusTimTbl
243 BPos == atHub | onLink
243a atHub :: h_ui:H_UI
243b onLink :: fh_ui:H_UI×l_ui:L_UI×frac:Fract×th_ui:H_UI
243b Fract = **Real**, **axiom** frac:Fract · 0<frac<1
244 ...
value
242 attr_BusTimTbl: B → BusTimTbl
243 attr_BPos: B → BPos

A.3.3.6 Private Automobile Attributes

- We illustrate but a few attributes:

245. Automobiles have static number plate registration numbers.
246. Automobiles have dynamic positions on the road net:

    [243a] either *at a hub* identified by some h_ui,

    [243b] 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.

type
245 RegNo
246 APos == atHub | onLink
243a atHub :: h_ui:H_UI
243b onLink :: fh_ui:H_UI $\times$ l_ui:L_UI $\times$ frac:Fract $\times$ th_ui:H_UI
243b Fract = **Real**, **axiom** frac:Fract $\cdot$ 0<frac<1
value
245 attr_RegNo: A $\to$ RegNo
246 attr_APos: A $\to$ 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.

- Observe that bus companies each have their own distinct *bus time table*, and that these are modeled as *programmable*, Item 240 on Slide 489, page 489.
- Observe then that buses each have their own distinct *bus time table*, and that these are model-led as *inert*, Item 242 on Slide 490, page 490.

- In Items Sli. 260 and Sli. 264, 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.[7]

[7] In this day and age of road cameras and satellite surveillance these traffic recordings may not appear so strange: We now know, at least in principle, of technologies that can record approximations to the hub and link traffic attributes.

A.3.3.7 Intentionality

247. Seen from the point of view of an automobile there is its own traffic history, A_Hist, which is a (time ordered) sequence of timed automobile's positions;
248. seen from the point of view of a hub there is its own traffic history, H_Traffic Item Sli. 260, which is a (time ordered) sequence of timed maps from automobile identities into automobile positions; and
249. seen from the point of view of a link there is its own traffic history, L_Traffic Item Sli. 264, which is a (time ordered) sequence of timed maps from automobile identities into automobile positions.

- The *intentional "pull"* of these manifestations is this:

250. The union, i.e. proper merge of all automobile traffic histories, AllATH, must now be identical to the same proper merge of all hub, AllHTH, and all link traffic histories, AllLTH.

type

247 A_Hi = $(\mathbb{T} \times \text{APos})^*$

234 H_Trf = A_UI $\underset{m}{\rightarrow}$ $(\mathbb{TIME} \times \text{APos})^*$

238 L_Trf = A_UI $\underset{m}{\rightarrow}$ $(\mathbb{TIME}\times\text{APos})^*$

250 AllATH=$\mathbb{TIME}$ $\underset{m}{\rightarrow}$(AUI $\underset{m}{\rightarrow}$APos)

250 AllHTH=$\mathbb{TIME}$ $\underset{m}{\rightarrow}$(AUI $\underset{m}{\rightarrow}$APos)

250 AllLTH =$\mathbb{TIME}$ $\underset{m}{\rightarrow}$(AUI $\underset{m}{\rightarrow}$APos)

axiom

250 **let** allA=mrg_AllATH({(a,attr_A_Hi(a))|a:A·a ∈ *as*}),

250   allH=mrg_AllHTH({attr_H_Trf(h)|h:H·h ∈ *hs*}),

250   allL =mrg_AllLTH({attr_L_Trf(l)|l:L·h ∈ *ls*}) **in**

250 allA = mrg_HLT(allH,allL) **end**

- We leave the definition of the merge functions to the listener !
  - We endow
    - ⋆ each automobile with its history of timed positions and
    - ⋆ each hub and link with their histories of timed automobile positions.
  - These histories are facts !
  - They are not something that is laboriously recorded, where such recordings may be imprecise or cumbersome[8].
  - The facts are there, so we can (but may not necessarily) talk about these histories as facts.

[8]or thought technologically in-feasible – at least some decades ago!

- It is in that sense that the purpose (`transport')
  - $\ast$ for which man let automobiles, hubs and link be made
  - $\ast$ with their `transport' intent
  - $\ast$ are subject to an *intentional "pull"*.

- *It can be no other way: if automobiles "record" their history, then hubs and links must together "record" identically the same history!.*

Intentional Pull – General Transport:

- 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* ■

A.4 Perdurants

- In this section we transcendentally “morph” **parts** into **behaviours.**
- We analyse that notion and its constituent notions of
  - **actors,**
  - **channels** and **communication,**
  - **actions** and
  - **events.**
- The main transcendental deduction of this chapter
  - is that of associating
  - with each part
  - a behaviour.
- This section shows the details of that association.

- Perdurants are understood in terms of
  - a notion of ***state*** and
  - a notion of ***time***.

State Values versus State Variables:

- Item 197 on Slide 465 expresses the value of all parts of a road transport system:

197. ps:(UoB|H|L|BC|B|A)-**set** $\equiv rts \cup hls \cup bcs \cup bs \cup as$.

251. We now introduce the set of variables, one for each part value of the domain being modeled.

251. { **variable** vp:(UoB|H|L|BC|B|A) | vp:(UoB|H|L|BC|B|A) $\cdot$ $vp \in ps$ }

Buses and Bus Companies

- A bus company is like a “root” for its fleet of “sibling” buses.
- But a bus company may cease to exist without the buses therefore necessarily also ceasing to exist.
- They may continue to operate, probably illegally, without, possibly.
  a valid bus driving certificate.
- Or they may be passed on to either private owners or to other bus companies.
- We use this example as a reason for not endowing a “block structure” concept on behaviours.

A.4.1 Channels and Communication

A.4.1.1 Channel Message Types

- We ascribe types to the messages offered on channels.

252. Hubs and links communicate, both ways, with one another, over channels, `hl_ch`, whose indexes are determined by their mereologies.
253. Hubs send one kind of messages, links another.
254. Bus companies offer timed bus time tables to buses, one way.
255. Buses and automobiles offer their current, timed positions to the road element, hub or link they are on, one way.

```
type
253  H_L_Msg, L_H_Msg
252  HL_Msg = H_L_Msg | L_F_Msg
254  BC_B_Msg = T × BusTimTbl
255  V_R_Msg = T × (BPos|APos)
```

A.4.1.2 Channel Declarations

256. This justifies the channel declaration which is calculated to be:

channel

256 { hl_ch[h_ui,l_ui]:H_L_Msg | h_ui:H_UI,l_ui:L_UI·i $\in h_{ui}s \wedge$ j $\in lh_{ui}m$(h_ui) }
256 $\cup$
256 { hl_ch[h_ui,l_ui]:L_H_Msg | h_ui:H_UI,l_ui:L_UI·l_ui $\in l_{ui}s \wedge$ i $\in lh_{ui}m$(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.

257. This justifies the channel declaration which is calculated to be:

channel
257 { bc_b_ch[bc_ui,b_ui] | bc_ui:BC_UI, b_ui:B_UI · bc_ui ∈ $bc_{ui}s$ ∧ b_ui ∈ $b_{ui}s$ }: 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.

258. This justifies the channel declaration which is calculated to be:

channel

258 { v_r_ch[v_ui,r_ui] | v_ui:V_UI,r_ui:R_UI • v_ui∈ $v_{ui}s$∧r_ui∈ $r_{ui}s$ }: V_R_Msg

A.4.2 Behaviours

A.4.2.1 Road Transport Behaviour Signatures

- We first decide on names of behaviours.
  - In the translation schemas
  - we gave schematic names to behaviours
  - of the form $\mathcal{M}_P$.
- We now assign mnemonic names:
  - from part names to names of transcendentally interpreted behaviours
  - and then we assign signatures to these behaviours.

A.4.2.1.1 Hub Behaviour Signature

259. 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
259   hub$_{h_{ui}}$:
259a  h_ui:H_UI×(vuis,luis,_):H_Mer×HΩ
259b    → (HΣ×H_Traffic)
259c    → in,out { h_l_ch[ h_ui,l_ui ] | l_ui:L_UI·l_ui ∈ luis }
259d          { ba_r_ch[ h_ui,v_ui ] | v_ui:V_UI·v_ui∈vuis } Unit
259a     pre: vuis = $v_{ui}s$ ∧ luis = $l_{ui}s$
```

A.4.2.1.2 Link Behaviour Signature

260. $\mathsf{link}_{l_{ui}}$:
   (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
260   link_{l_ui}:
260a  l_ui:L_UI×(vuis,huis,_):L_Mer×LΩ
260b     → (LΣ×L_Traffic)
260c     → in,out { h_l_ch[ h_ui,l_ui ] | h_ui:H_UI:h_ui ∈ huis }
260d          { ba_r_ch[ l_ui,v_ui ] | v_ui:(B_UI|A_UI)·v_ui∈vuis } Unit
260a     pre: vuis = $v_{ui}s$ ∧ huis = $h_{ui}s$
```

A.4.2.1.3 Bus Company Behaviour Signature

261. 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
261   bus_company$_{bc_{ui}}$:
261a  bc_ui:BC_UI×(_,_,buis):BC_Mer
261b    → BusTimTbl
261c      in,out {bc_b_ch[ bc_ui,b_ui ]|b_ui:B_UI·b_ui∈buis} Unit
261a    pre: buis = $b_{ui}s$ ∧ huis = $h_{ui}s$
```

A.4.2.1.4 Bus Behaviour Signature

262. $\text{bus}_{b_{ui}}$:
    (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
262   bus$_{b_{ui}}$:
262a  b_ui:B_UI×(bc_ui,__,ruis):B_Mer
262b     → (LN × BTT × BPOS)
262c     → out bc_b_ch[ bc_ui,b_ui ],
262d          {ba_r_ch[ r_ui,b_ui ]|r_ui:(H_UI|L_UI)·ui∈$v_{ui}s$} Unit
262a     pre: ruis = $r_{ui}s$ ∧ bc_ui ∈ $bc_{ui}s$
```

A.4.2.1.5 Automobile Behaviour Signature

263. 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
263   automobile$_{a_{ui}}$:
263a  a_ui:A_UI×(_,_,ruis):A_Mer×rn:RegNo
263b    → apos:APos
263c      in,out {ba_r_ch[ a_ui,r_ui ]|r_ui:(H_UI|L_UI)·r_ui∈ruis} Unit
263a    pre: ruis = $r_{ui}s$ ∧ a_ui ∈ $a_{ui}s$ ■
```

A.4.2.2 Behaviour Definitions

- We only illustrate automobile, hub and link behaviours.

A.4.2.2.1 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.

264. We abstract automobile behaviour at a Hub (hui).
265. The vehicle remains at that hub, “idling”,
266. informing the hub behaviour,
267. 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,
268. or, again internally non-deterministically,
269. the vehicle “disappears — off the radar” !

```
264  automobile$_{a_{ui}}$(a_ui,({},(ruis,vuis),{}),rn)
264          (apos:atH(fl_ui,h_ui,tl_ui)) ≡
265     (ba_r_ch[a_ui,h_ui] ! (record_TIME(),atH(fl_ui,h_ui,tl_ui));
266      automobile$_{a_{ui}}$(a_ui,({},(ruis,vuis),{}),rn)(apos))
267    ⊓
267a    (let ({fh_ui,th_ui},ruis′)=mereo_L(℘(tl_ui)) in
267a          assert: fh_ui=h_ui ∧ ruis=ruis′
264      let onl = (tl_ui,h_ui,0,th_ui) in
267b      (ba_r_ch[a_ui,h_ui] ! (record_TIME(),onL(onl)) ||
267b       ba_r_ch[a_ui,tl_ui] ! (record_TIME(),onL(onl))) ;
267c       automobile$_{a_{ui}}$(a_ui,({},(ruis,vuis),{}),rn)
267c             (onL(onl)) end end)
268    ⊓
269      stop
```

A.4.2.2.2 Automobile Behaviour On a Link

270. 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" !

```
270     automobile$_{a_{ui}}$(a_ui,({},ruis,{}),rno)
270                        (vp:onL(fh_ui,l_ui,f,th_ui)) ≡
270(a)ii  (ba_r_ch[ thui,aui ]!atH(lui,thui,nxt_lui) ;
270(a)i    automobile$_{a_{ui}}$(a_ui,({},ruis,{}),rno)(vp))
270b     ⊓
270(b)i  (**if** not_yet_at_hub(f)
270(b)i     **then**
270(b)iA      (**let** incr = increment(f) **in**
264           **let** onl = (tl_ui,h_ui,incr,th_ui) **in**
270(b)iB       ba−r_ch[ l_ui,a_ui ] ! onL(onl) ;
270(b)iC       automobile$_{a_{ui}}$(a_ui,({},ruis,{}),rno)
270(b)iC                      (onL(onl))
270(b)i       **end end**)
270(b)ii     **else**
270(b)iiA     (**let** nxt_lui:L_UI•nxt_lui ∈ mereo_H(℘(th_ui)) **in**
270(b)iiB      ba_r_ch[ thui,aui ]!atH(l_ui,th_ui,nxt_lui) ;
270(b)iiC      automobile$_{a_{ui}}$(a_ui,({},ruis,{}),rno)
270(b)iiC                     (atH(l_ui,th_ui,nxt_lui)) **end**)
270(b)i    **end**)
270c     ⊓
270d       **stop**
270(b)iA  increment: Fract → Fract
```

A.4.2.2.3 Hub Behaviour

271. 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 Sli. 260.

value
```
271   hub$_{h_{ui}}$(h_ui,(,(luis,vuis)),h$\omega$)(h$\sigma$,ht) ≡
271a     ⌷
271b       { let m = ba_r_ch[h_ui,v_ui] ? in
271c           assert: m=(_,atHub(_,h_ui,_))
271d         let ht′ = ht † [h_ui ↦ ⟨m⟩⁀ht(h_ui)] in
271e         hub$_{h_{ui}}$(h_ui,(,(luis,vuis)),(h$\omega$))(h$\sigma$,ht′)
271f       | v_ui:V_UI•v_ui∈vuis end end }
271g    post: ∀ v_ui:V_UI•v_ui ∈ dom ht′⇒time_ordered(ht′(v_ui))
```

A.4.2.2.4 Link Behaviour

272. The link behaviour non-deterministically, externally offers
273. to accept timed vehicle positions —
274. which will be on the link, from some vehicle, v_ui.
275. The timed vehicle link position is appended to the front of that vehicle's entry in the link's traffic table;
276. whereupon the link proceeds as a link behaviour with the updated link traffic table.
277. The link behaviour offers to accept from any vehicle.
278. 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 Sli. 264.

```
272  link$_{l_{ui}}$(l_ui,(_,(huis,vuis),_),lω)(lσ,lt) ≡
272    □
273      { let m = ba_r_ch[ l_ui,v_ui ] ? in
274            assert: m=(_,onLink(_,l_ui,_,_))
275        let lt′ = lt † [ l_ui ↦ ⟨m⟩⁀lt(l_ui) ] in
276        link$_{l_{ui}}$(l_ui,(huis,vuis),hω)(hσ,lt′)
277      | v_ui:V_UI•v_ui∈vuis end end }
278    post: ∀ v_ui:V_UI•v_ui ∈ dom lt′⇒time_ordered(lt′(v_ui))
```

A.5 System Initialisation

A.5.1 Initial States

value

hs:H-**set** ≡ ≡ obs_sH(obs_SH(obs_RN(rts)))
ls:L-**set** ≡ ≡ obs_sL(obs_SL(obs_RN(rts)))
bcs:BC-**set** ≡ obs_BCs(obs_SBC(obs_FV(obs_RN(rts))))
bs:B-**set** ≡ ∪{obs_Bs(bc)|bc:BC•bc ∈ bcs}
as:A-**set** ≡ obs_BCs(obs_SBC(obs_FV(obs_RN(rts))))

A.5.2 Initialisation

- We are reaching the end of this domain modeling example.
  - Behind us there are narratives and formalisations.
  - Based on these we now express
    - ⋆ the signature and
    - ⋆ the body of the definition
  - of a *"system build and execute"* function.

279. The system to be initialised is
    (a) the parallel compositions (‖) of
    (b) the distributed parallel composition (‖{...|...}) of all hub behaviours,
    (c) the distributed parallel composition (‖{...|...}) of all link behaviours,
    (d) the distributed parallel composition (‖{...|...}) of all bus company behaviours,
    (e) the distributed parallel composition (‖{...|...}) of all bus behaviours, and
    (f) the distributed parallel composition (‖{...|...}) of all automobile behaviours.

```
value
279  initial_system: Unit → Unit
279  initial_system() ≡
279b      || { hub$_{h_{ui}}$(h_ui,me,h$\omega$)(htrf,h$\sigma$)
279b         | h:H•h ∈ $hs$, h_ui:H_UI•h_ui=uid_H(h), me:HMetL•me=mereo_H(h),
279b           htrf:H_Traffic•htrf=attr_H_Traffic_H(h),
279b           h$\omega$:H$\Omega$•h$\omega$=attr_H$\Omega$(h), h$\sigma$:H$\Sigma$•h$\sigma$=attr_H$\Sigma$(h)∧h$\sigma$ ∈ h$\omega$ }
279a      ||
279c      || { link$_{l_{ui}}$(l_ui,me,l$\omega$)(ltrf,l$\sigma$)
279c           l:L•l ∈ $ls$, l_ui:L_UI•l_ui=uid_L(l), me:LMet•me=mereo_L(l),
279c           ltrf:L_Traffic•ltrf=attr_L_Traffic_H(l),
279c           l$\omega$:L$\Omega$•l$\omega$=attr_L$\Omega$(l), l$\sigma$:L$\Sigma$•l$\sigma$=attr_L$\Sigma$(l)∧l$\sigma$ ∈ l$\omega$ }
279a      ||
279d      || { bus_company$_{bc_{ui}}$(bcui,me)(btt)
279d           bc:BC•bc ∈ $bcs$, bc_ui:BC_UI•bc_ui=uid_BC(bc), me:BCMet•me=mereo_BC(bc),
279d           btt:BusTimTbl•btt=attr_BusTimTbl(bc) }
279a      ||
279e      || { bus$_{b_{ui}}$(b_ui,me)(ln,btt,bpos)
279e           b:B•b ∈ $bs$, b_ui:B_UI•b_ui=uid_B(b), me:BMet•me=mereo_B(b), ln:LN:pln=attr_LN(b),
279e           btt:BusTimTbl•btt=attr_BusTimTbl(b), bpos:BPos•bpos=attr_BPos(b) }
279a       ||
279f      || { automobile$_{a_{ui}}$(a_ui,me,rn)(apos)
279f           a:A•a ∈ $as$, a_ui:A_UI•a_ui=uid_A(a), me:AMet•me=mereo_A(a),
279f           rn:RegNo•rno=attr_RegNo(a), apos:APos•apos=attr_APos(a) }  ■
```

Appendix B. Pipelines

B.1 Endurants: External Qualities

We follow the ontology of Fig. **??** on Slide ??, the lefthand dashed box labelled *External Qualities*.

B.1.1 Parts

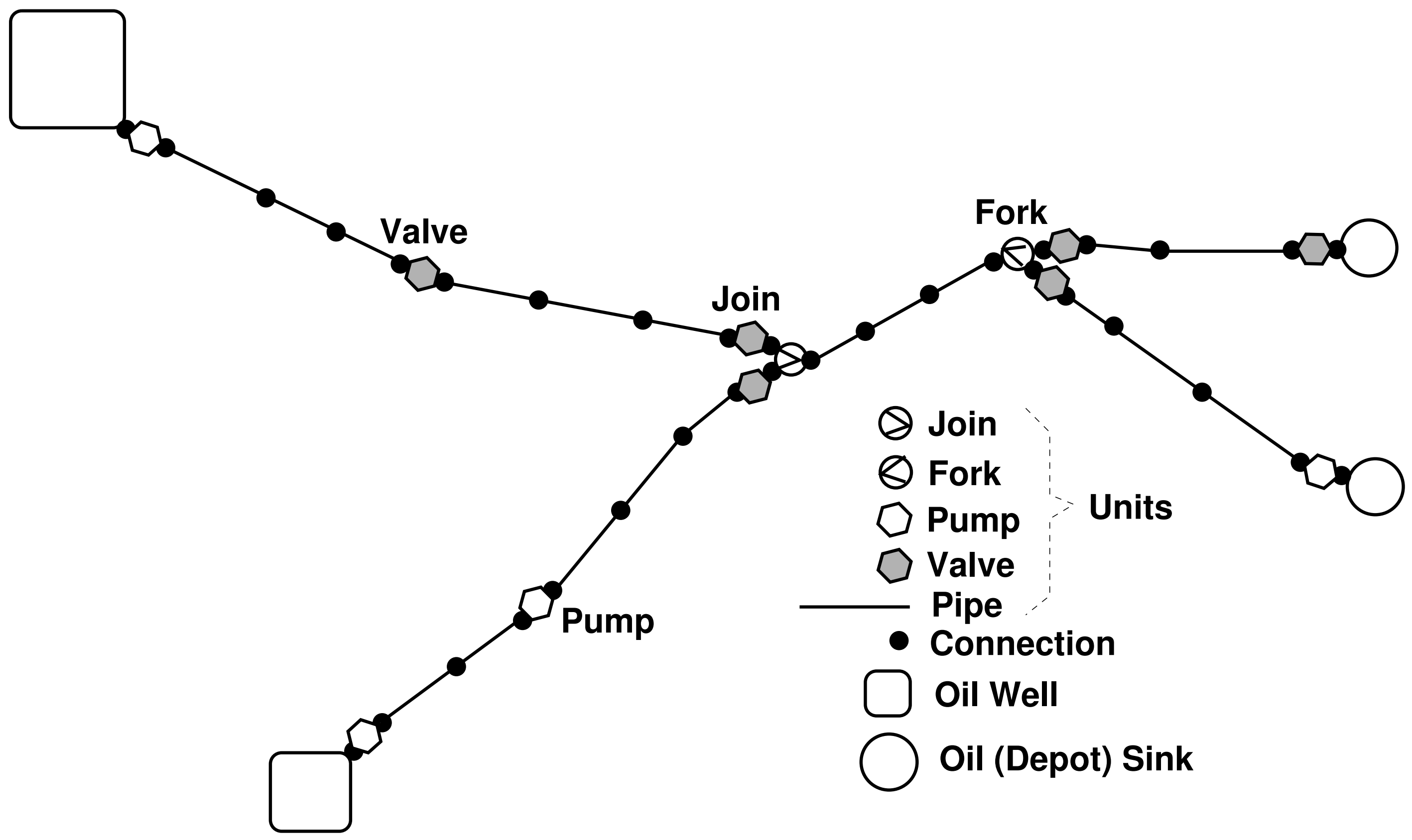

Figure B.1: An example pipeline system

280. A pipeline system contains a set of pipeline units and a pipeline system monitor.
281. The well-formedness of a pipeline system depends on its mereology and the routing of its pipes.

282. A pipeline unit is either a well, a pipe, a pump, a valve, a fork, a join, a plate[1], or a sink unit.
283. We consider all these units to be distinguishable, i.e., the set of wells, the set pipe, etc., the set of sinks, to be disjoint.

[1]A *plate* unit is a usually circular, flat steel plate used to “begin” or “end” a pipe segment.

```
type
280.  PLS′, U, M
281.  PLS = {| pls:PLS′•wf_PLS(pls) |}
value
281.  wf_PLS: PLS → Bool
281.  wf_PLS(pls) ≡
281.  wf_Mereology(pls)∧wf_Routes(pls)∧wf_Metrics(pls)[2]
280.  obs_Us: PLS → U-set
280.  obs_M: PLS → M
type
282.  U = We | Pi | Pu | Va | Fo | Jo | Pl | Si
283.  We :: Well
283.  Pi :: Pipe
283.  Pu :: Pump
283.  Va :: Valv
283.  Fo :: Fork
283.  Jo :: Join
283.  Pl :: Plate
283.  Si :: Sink
```

[2] wf_Mereology, wf_Routes and wf_Metrics will be explained in Sects. B.2.2.2 on Slide 545, B.2.3.2 on Slide 551, and B.2.4.3 on Slide 567.

B.1.2 An Endurant State

284. For a given pipeline system
285. we exemplify an endurant state σ
286. composed of the given pipeline system and all its manifest units, i.e., without plates.

value
284. pls:PLS
variable
285. σ := collect_state(pls)
value
286. collect_state: PLS
286. collect_state(pls) $\equiv$ {pls} $\cup$ obs_Us(pls) \ Pl

B.2 Endurants: Internal Qualities

We follow the ontology of Fig. ?? on Slide ??, the lefthand vertical and horisontal lines.

B.2.1 Unique Identification

287. The pipeline system, as such,
288. has a unique identifier, distinct (different) from its pipeline unit identifiers.
289. Each pipeline unit is uniquely distinguished by its unit identifier.
290. There is a state of all unique identifiers.

```
  type
288.    PLSI
289.    UI
  value
287.    pls:PLS
288.    uid_PLS: PLS → PLSI
289.    uid_U: U → UI
  variable
290.    σ_uid := { uid_PLS(pls) } ∪ xtr_UIs(pls)
  axiom
289.    ∀ u,u′:U·{u,u′}⊆obs_Us(pls)⇒u≠u′⇒uid_UI(u)≠uid_UI(u′)
289.  ∧  uid_PLS(pls) ∉ {uid_UI(u)|u:U·u ∈ obs_Us(pls)}
```

291. From a pipeline system one can observe the set of all unique unit identifiers.

value

291. xtr_UIs: PLS → UI-**set**

291. xtr_UIs(pls) ≡ {uid_UI(u)|u:U·u ∈ obs_Us(pls)}

292. We can prove that the number of unique unit identifiers of a pipeline system equals that of the units of that system.

theorem:

292. $\forall$ pls:PLS·**card** obs_Us(pl)=**card** xtr_UIs(pls)

B.2.2 Mereology

B.2.2.1 PLS Mereology

293. The mereology of a pipeline system is the set of unique identifiers of all the units of that system.

type
293. PLS_Mer = UI-**set**iptyPLS_Merpls-mer-00
value
293. mereo_PLS: PLS → PLS_Meripobmereo_PLSpls-mer-00
axiomiptyWellformed Mereologiespls-mer-00
293. ∀ uis:PLS_Mer · uis = **card** xtr_UIs(pls)

B.2.2.2 Unit Mereologies

294. Each unit is connected to zero, one or two other existing input units and zero, one or two other existing output units as follows:
    (a) A well unit is connected to exactly one output unit (and, hence, has no "input").
    (b) A pipe unit is connected to exactly one input unit and one output unit.
    (c) A pump unit is connected to exactly one input unit and one output unit.
    (d) A valve is connected to exactly one input unit and one output unit.
    (e) A fork is connected to exactly one input unit and two distinct output units.
    (f) A join is connected to exactly two distinct input units and one output unit.
    (g) A plate is connected to exactly one unit.
    (h) A sink is connected to exactly one input unit (and, hence, has no "output").

```
     type
294.     MER = UI-set × UI-set
     value
294.     mereo_U: U → MER
     axiom
294.      wf_Mereology: PLS → Bool
294.      wf_Mereology(pls) ≡
294.         ∀ u:U•u ∈ obs_Us(pls)⇒
294.           let (iuis,ouis) = mereo_U(u) in iuis ∪ ouis ⊆ xtr_UIs(pls) ∧
294.              case (u,(card uius,card ouis)) of
294a.                  (mk_We(we),(0,1)) → true,
294b.                  (mk_Pi(pi),(1,1)) → true,
294c.                  (mk_Pu(pu),(1,1)) → true,
294d.                  (mk_Va(va),(1,1)) → true,
294e.                  (mk_Fo(fo),(1,1)) → true,
294f.                 (mk_Jo(jo),(1,1)) → true,
294f.                 (mk_Pl(pl),(0,1)) → true, "begin"
294f.                 (mk_Pl(pl),(1,0)) → true, "end"
294h.                  (mk_Si(si),(1,1)) → true,
294.                _ → false end end
```

B.2.3 Pipeline Concepts, I

B.2.3.1 Pipe Routes

295. A route (of a pipeline system) is a sequence of connected units (of the pipeline system).
296. A route descriptor is a sequence of unit identifiers and the connected units of a route (of a pipeline system).

type

295. $\mathsf{R}' = \mathsf{U}^\omega$

295. $\mathsf{R} = \{|\ \mathsf{r{:}Route'{\cdot}wf\_Route(r)}\ |\}$

296. $\mathsf{RD} = \mathsf{UI}^\omega$

axiom

296. $\forall\ \mathsf{rd{:}RD} \cdot \exists\ \mathsf{r{:}R{\cdot}rd{=}descriptor(r)}$

value

296. $\mathsf{descriptor{:}\ R} \rightarrow \mathsf{RD}$

296. $\mathsf{descriptor(r)} \equiv \langle \mathsf{uid\_UI(r[\,i\,])} | \mathsf{i}{:}\mathbf{Nat}{\cdot}1{\leq}\mathsf{i}{\leq}\mathbf{len}\ \mathsf{r} \rangle$

297. Two units are adjacent if the output unit identifiers of one shares a unique unit identifier with the input identifiers of the other.

value

297. adjacent: U $\times$ U $\rightarrow$ **Bool**

297. adjacent(u,u′) $\equiv$ **let** (,ouis)=mereo_U(u),(iuis,)=mereo_U(u′) **in** ouis $\cap$ iuis $\neq$ {} **en**

298. Given a pipeline system, pls, one can identify the (possibly infinite) set of (possibly infinite) routes of that pipeline system.
   (a) The empty sequence, $\langle\rangle$, is a route of pls.
   (b) Let u, u' be any units of pls, such that an output unit identifier of u is the same as an input unit identifier of u' then $\langle u, u'\rangle$ is a route of pls.
   (c) If r and r' are routes of pls such that the last element of r is the same as the first element of r', then $r^\frown \mathbf{tl} r'$ is a route of pls.
   (d) No sequence of units is a route unless it follows from a finite (or an infinite) number of applications of the basis and induction clauses of Items 298a–298c.

```
value
298.   Routes: PLS → RD-infset
298.   Routes(pls) ≡
298a.      let rs = ⟨⟩ ∪
298b.          {⟨uid_UI(u),uid_UI(u′)⟩|u,u′:U·{u,u′}⊆obs_Us(pls) ∧ adjacent(u,u′)}
298c.        ∪ {r⁀tl r′|r,r′:R·{r,r′}⊆rs}
298d.      in rs end
```

B.2.3.2 Well-formed Routes

299. A route is acyclic if no two route positions reveal the same unique unit identifier.

```
value
299.  is_acyclic_Route: R → Bool
299.  is_acyclic_Route(r) ≡ ∼∃ i,j:Nat·{i,j}⊆inds r ∧ i≠j ∧ r[i]=r[j]
```

300. A pipeline system is well-formed if none of its routes are circular (and all of its routes embedded in well-to-sink routes).

value

```
300.  wf_Routes: PLS → **Bool**
300.  wf_Routes(pls) ≡
300.      non_circular(pls) ∧ are_embedded_Routes(pls)

300.  is_non_circular_PLS: PLS → **Bool**
300.  is_non_circular_PLS(pls) ≡
300.      ∀ r:R·r ∈ routes(p)∧acyclic_Route(r)
```

301. We define well-formedness in terms of well-to-sink routes, i.e., routes which start with a well unit and end with a sink unit.

```
value
301.  well_to_sink_Routes: PLS → R-set
301.  well_to_sink_Routes(pls) ≡
301.    let rs = Routes(pls) in
301.    {r|r:R•r ∈ rs ∧ is_We(r[1]) ∧ is_Si(r[len r])} end
```

302. A pipeline system is well-formed if all of its routes are embedded in well-to-sink routes.

```
302.  are_embedded_Routes: PLS → **Bool**
302.  are_embedded_Routes(pls) ≡
302.     let wsrs = well_to_sink_Routes(pls) in
302.     ∀ r:R · r ∈ Routes(pls) ⇒
302.        ∃ r′:R,i,j:Nat ·
302.          r′ ∈ wsrs
302.         ∧ {i,j}⊆inds r′∧i≤j
302.         ∧ r = ⟨r′[ k ]|k:Nat·i≤k≤j⟩ end
```

B.2.3.3 Embedded Routes

303. For every route we can define the set of all its embedded routes.

```
  value
303.  embedded_Routes: R → R-set
303.  embedded_Routes(r) ≡ {⟨r[ k ]|k:Nat·i≤k≤j⟩ | i,j:Nat· i {i,j}⊆inds(r) ∧ i≤j}
```

B.2.3.4 A Theorem

304. The following theorem is conjectured:
   (a) the set of all routes (of the pipeline system)
   (b) is the set of all well-to-sink routes (of a pipeline system) and
   (c) all their embedded routes

theorem:

```
304.   ∀ pls:PLS ·
304.   let rs = Routes(pls),
304.       wsrs = well_to_sink_Routes(pls) in
304a.  rs =
304b.      wsrs ∪
304c.      ∪ {{r′|r′:R · r′ ∈ is_embedded_Routes(r″)} | r″:R · r″ ∈ wsrs}
303.   end
```

B.2.3.5 Fluids

305. The only fluid of concern to pipelines is the gas[3] or liquid[4] which the pipes transport[5].

```
 type
305.    GoL [ = M ]
 value
305.    obs_GoL: U → GoL
```

[3]Gaseous materials include: air, gas, etc.
[4]Liquid materials include water, oil, etc.
[5]The description of this document is relevant only to gas or oil pipelines.

B.2.4 Attributes

B.2.4.1 Unit Flow Attributes

306. A number of attribute types characterise units:
   (a) estimated current well capacity (barrels of oil, etc.),
   (b) pump height (a static attribute),
   (c) current pump status (not pumping, pumping; a programmable attribute),
   (d) current valve status (closed, open; a programmable attribute) and
   (e) flow (barrels/second, a biddable attribute).

```
  type
306a.    WellCap
306b.    Pump_Height
306c.    Pump_State == {|not_pumping,pumping|}
306d.    Valve_State == {|closed,open|}
306e.    Flow
```

307. Flows can be added and subtracted,
308. added distributively and
309. flows can be compared.

```
  value
307.    ⊕,⊖: Flow×Flow → Flow
308.    ⊕: Flow-set → Flow
309.    <,≤,=,≠,≥,>: Flow × Flow → Bool
```

310. Properties of pipeline units include

(a) estimated current well capacity (barrels of oil, etc.) [a biddable attribute],
(b) pipe length [a static attribute],
(c) current pump height [a biddable attribute],
(d) current valve open/close status [a programmable attribute],
(e) current [$\mathcal{L}$aminar] in-flow at unit input [a monitorable attribute],
(f) current $\mathcal{L}$aminar] in-flow leak at unit input [a monitorable attribute],
(g) maximum [$\mathcal{L}$aminar] guaranteed in-flow leak at unit input [a static attribute],

(h) current [$\mathcal{L}$aminar] leak unit interior [a monitorable attribute],
(i) current [$\mathcal{L}$aminar] flow in unit interior [a monitorable attribute],
(j) maximum $\mathcal{L}$aminar] guaranteed flow in unit interior [a monitorable attribute],
(k) current [$\mathcal{L}$aminar] out-flow at unit output [a monitorable attribute],
(l) current [$\mathcal{L}$aminar] out-flow leak at unit output [a monitorable attribute] and
(m) maximum guaranteed $\mathcal{L}$aminar out-flow leak at unit output [a static attribute.

```
type
310e  In_Flow = Flow
310f  In_Leak = Flow
310g  Max_In_Leak = Flow
310h  Body_Flow = Flow
310i  Body_Leak = Flow
310j  Max_Flow = Flow
310k  Out_Flow = Flow
310l  Out_Leak = Flow
310m  Max_Out_Leak = Flow
value
310a  attr_WellCap: We → WellCap
310b  attr_LEN: Pi → LEN
310c  attr_Height: Pu → Height
310d  attr_ValSta: Va → VaSta
310e  attr_In_Flow: U → UI → Flow
310f  attr_In_Leak: U → UI → Flow
310g  attr_Max_In_Leak: U → UI → Flow
310h  attr_Body_Flow: U → Flow
310i  attr_Body_Leak: U → Flow
310j  attr_Max_Flow: U → Flow
310k  attr_Out_Flow: U → UI → Flow
310l  attr_Out_Leak: U → UI → Flow
310m  attr_Max_Out_Leak: U → UI → Flow
```

311. Summarising we can define a two notions of flow:
    (a) static and
    (b) monitorable.

```
type
311a  Sta_Flows = Max_In_Leak×In_Max_Flow>Max_Out_Leak
311b  Mon_Flows = In_Flow×In_Leak×Body_Flow×Body_Leak×Out_Flow×Out_Leak
```

B.2.4.2 Unit Metrics

- Pipelines are laid out in the terrain.
  - Units have length and diameters.
  - Units are positioned in space: have altitude, longitude and latitude positions of its one, two or three connection PoinTs[6].

312. length (a static attribute),
313. diameter (a static attribute) and
314. position (a static attribute).

[6] 1 for *wells*, *plates* and *sinks*; 2 for *pipes*, *pumps* and *valves*; 1+2 for *forks*, 2+1 for *joins*.

```
type
312.  LEN
313.  ◯
314.  POS == mk_One(pt:PT) | mk_Two(ipt:PT,opt:PT)
314.          | mk_OneTwo(ipt:PT,opts:(lpt:PT,rpt:PT))
314.          | mk_TwoOne(ipts:(lpt:PT,rpt:PT),opt:PT)
314.  PT = Alt × Lon × Lat
314.  Alt, Lon, Lat = ...
value
312.  attr_LEN: U → LEN
313.  attr_◯: U → ◯
314.  attr_POS: U → POS
```

- We can summarise the metric attributes:

315. Units are subject to either of four (mutually exclusive) metrics:
    (a) Length, diameter and a one point position.
    (b) Length, diameter and a two points position.
    (c) Length, diameter and a one+two points position.
    (d) Length, diameter and a two+one points position.

type

```
315.  Unit_Sta = Sta1_Metric | Sta2_Metric | Sta12_Metric | Sta21_Metric
315a  Sta1_Metric = LEN × Ø × mk_One(pt:PT)
315b  Sta2_Metric = LEN × Ø × mk_Two(ipt:PT,opt:PT)
315c  Sta12_Metric = LEN × Ø × mk_OneTwo(ipt:PT,opts:(lpt:PT,rpt:PT))
315d  Sta21_Metric = LEN × Ø × mk_TwpOne(ipts:(lpt:PT,rpt:PT),opt:PT)
```

B.2.4.3 Wellformed Unit Metrics

- The points positions of neighbouring units must “fit” one-another.

316. Without going into details we can define a predicate, wf_Metrics, that applies to a pipeline system and yields **true** iff neighbouring units must “fit” one-another.

value

316. wf_Metrics: PLS $\rightarrow$ **Bool**

316. wf_Metrics(pls) $\equiv$ ...

B.2.4.4 Summary

- We summarise the static, monitorable and programmable attributes for each manifest part of the pipeline system:

type

```
  PLS_Sta = PLS_net×...
  PLS_Mon = ...
  PLS_Prg = PLS_Σ×...
  Well_Sta = Sta1_Metric×Sta_Flows×Orig_Cap×...
  Well_Mon = Mon_Flows×Well_Cap×...
  Well_Prg = ...
  Pipe_Sta = Sta2_Metric×Sta_Flows×LEN×...
  Pipe_Mon = Mon_Flows×In_Temp×Out_Temp×...
  Pipe_Prg = ...
  Pump_Sta = Sta2_Metric×Sta_Flows×Pump_Height×...
  Pump_Mon = Mon_Flows×...
  Pump_Prg = Pump_State×...
```

Valve_Sta = Sta2_Metric×Sta_Flows×...
Valve_Mon = Mon_Flows×In_Temp×Out_Temp×...
Valve_Prg = Valve_State×...
Fork_Sta = Sta12_Metric×Sta_Flows×...
Fork_Mon = Mon_Flows×In_Temp×Out_Temp×...
Fork_Prg = ...
Join_Sta = Sta21_Metric×Sta_Flows×...
Join_Mon = Mon_Flows×In_Temp×Out_Temp×...
Join_Prg = ...
Sink_Sta = Sta1_Metric×Sta_Flows×Max_Vol×...
Sink_Mon = Mon_Flows×Curr_Vol×In_Temp×Out_Temp×...
Sink_Prg = ...

317. Corresponding to the above three attribute categories we can define "collective" attribute observers:

value

317. sta_A_We: We $\rightarrow$ Sta1_Metric×Sta_Flows×Orig_Cap×...
317. mon_A_We: We $\rightarrow$ ηMon_Flows×ηWell_Cap×ηIn_Temp×ηOut_Temp×...
317. prg_A_We: We $\rightarrow$ ...
317. sta_A_Pi: Pi $\rightarrow$ Sta2_Metric×Sta_Flows×LEN×...
317. mon_A_Pi: Pi $\rightarrow$ $\mathcal{N}$Mon_Flows×ηIn_Temp×ηOut_Temp×...
317. prg_A_Pi: Pi $\rightarrow$ ...
317. sta_A_Pu: Pu $\rightarrow$ Sta2_Metric×Sta_Flows×LEN×...
317. mon_A_Pu: Pu $\rightarrow$ $\mathcal{N}$Mon_Flows×ηIn_Temp×ηOut_Temp×...
317. prg_A_Pu: Pu $\rightarrow$ Pump_State×...
317. sta_A_Va: Va $\rightarrow$ Sta2_Metric×Sta_Flows×LEN×...
317. mon_A_Va: Va $\rightarrow$ $\mathcal{N}$Mon_Flows×ηIn_Temp×ηOut_Temp×...
317. prg_A_Va: Va $\rightarrow$ Valve_State×...
317. sta_A_Fo: Fo $\rightarrow$ Sta12_Metric×Sta_Flows×...

317. mon_A_Fo: Fo $\rightarrow$ $\mathcal{N}$Mon_Flows×ηIn_Temp×ηOut_Temp×...
317. prg_A_Fo: Fo $\rightarrow$ ...
317. sta_A_Jo: Jo $\rightarrow$ Sta21_Metric×Sta_Flows×...
317. mon_A_Jo: Jo $\rightarrow$ Mon_Flows×ηIn_Temp×ηOut_Temp×...
317. prg_A_Jo: Jo $\rightarrow$ ...
317. sta_A_Si: Si $\rightarrow$ Sta1_Metric×Sta_Flows×Max_Vol×...
317. mon_A_Si: Si $\rightarrow$ $\mathcal{N}$Mon_Flows×ηIn_Temp×ηOut_Temp×...
317. prg_A_Si: Si $\rightarrow$ ...

317. $\mathcal{N}$Mon_Flows $\equiv$ (ηIn_Flow,ηIn_Leak,ηBody_Flow,ηBody_Leak,ηOut_Flow,ηOut_Leak

- Monitored flow attributes
  - are [to be] passed as arguments to behaviours *by reference*
  - so that their monitorable attribute values can be sampled.

B.2.4.5 Fluid Attributes

- Fluids, we here assume, oil, as it appears in the pipeline units
  - have no unique identity,
  - have not mereology,
  - but does have attributes: hydrocarbons consisting predominantly of
    - ⋆ aliphatic,
    - ⋆ alicyclic and
    - ⋆ aromatic hydrocarbons.
  - It may also contain small amounts of
    - ⋆ nitrogen,
    - ⋆ oxygen, and
    - ⋆ sulfur

    compounds

318. We shall simplify, just for illustration, crude oil fluid of units to have these attributes:

(a) volume,
(b) viscosity,
(c) temperature,
(d) paraffin content (%age),
(e) naphtenes content (%age),

type		**value**	
318.	Oil	318b.	obs_Oil: U → Oil
318a.	Vol	318a.	attr_Vol: Oil → Vol
318b.	Visc	318b.	attr_Visc: Oil → Visc
318c.	Temp	318c.	attr_Temp: Oil → Temp
318d.	Paraffin	318d.	attr_Paraffin: Oil → Paraffin
318e.	Naphtene	318e.	attr_Naphtene: Oil → Naphtene

B.2.4.6 Pipeline System Attributes

- The “root” pipeline system is a compound.
- In its transcendentally deduced behavioral form
  - it is, amongst other “tasks”, entrusted with the monitoring and control of all its units.
    - ⋆ To do so it must, as a basically static attribute
    - ⋆ possess awareness, say in the form of a net diagram
      - · of how these units are interconnected,
      - · together with all their internal qualities,
      - · by type and by value.
  - Next we shall give a very simplified account of the possible pipeline system attribute.

319. We shall make use, in this example, of just a simple pipeline state, pls_ω.

- The pipeline state, pls_ω, embodies all the information that is relevant to the monitoring and control of an entire pipeline system, whether static or dynamic.

type
319. PLS_Ω

B.2.5 Pipeline Concepts, II: Flow Laws

320. "What flows in, flows out !". For $\mathcal{L}$aminar flows: for any non-well and non-sink unit the sums of input leaks and in-flows equals the sums of unit and output leaks and out-flows.

Law:

```
320.    ∀ u:U\We\Si ·
320.        sum_in_leaks(u) ⊕ sum_in_flows(u) =
320.        attr_body_Leak_𝓛(u) ⊕
320.        sum_out_leaks(u) ⊕ sum_out_flows(u)
```

```
value
  sum_in_leaks: U → Flow
  sum_in_leaks(u) ≡ let (iuis,) = mereo_U(u) in ⊕ {attr_In_Leak$_\mathcal{L}$(u)(ui)|ui:UI•ui ∈ iuis} end
  sum_in_flows: U → Flow
  sum_in_flows(u) ≡ let (iuis,) = mereo_U(u) in ⊕ {attr_In_Flow$_\mathcal{L}$(u)(ui)|ui:UI•ui ∈ iuis} end
  sum_out_leaks: U → Flow
  sum_out_leaks(u) ≡ let (,ouis) = mereo_U(u) in ⊕ {attr_Out_Leak$_\mathcal{L}$(u)(ui)|ui:UI•ui ∈ ouis} end
  sum_out_flows: U → Flow
  sum_out_flows(u) ≡ let (,ouis) = mereo_U(u) in ⊕ {attr_Out_Leak$_\mathcal{L}$(u)(ui)|ui:UI•ui ∈ ouis} end
```

321. "What flows out, flows in !". For $\mathcal{L}$aminar flows: for any adjacent pairs of units the output flow at one unit connection equals the sum of adjacent unit leak and in-flow at that connection.

Law:

```
321. ∀ u,u′:U•adjacent(u,u′) ⇒
321.     let (,ouis)=mereo_U(u), (iuis′,)=mereo_U(u′) in
321.     assert: uid_U(u′) ∈ ouis ∧ uid_U(u) ∈ iuis′
321.     attr_Out_Flow_ℒ(u)(uid_U(u′)) =
321.     attr_In_Leak_ℒ(u)(uid_U(u))⊕attr_In_Flow_ℒ(u′)(uid_U(u)) end
```

- These "laws" should hold for a pipeline system without plates.

B.3 Perdurants

We follow the ontology of Fig. ?? on Slide ??, the righthand dashed box labelled *Perdurants* and the righthand vertical and horisontal lines.

B.3.1 State

- We introduce concepts of *manifest* and *structure* endurants.
  - The former are such compound endurants (Cartesians of sets) to which we ascribe internal qualities;
  - the latter are such compound endurants (Cartesians of sets) to which we **do not** ascribe internal qualities.
- The distinction is pragmatic.

322. For any given pipeline system we suggest the state to consist of the manifest endurants of all its non-plate units.

value

322. σ = obs_Us(pls)

B.3.2 Channel

323. There is a [global] array channel
indexed by a "set pair" of distinct manifest endurant part identifiers – signifying the possibility of the syncharonisation and communication between any pair of pipeline units and between these and the pipeline system, cf. last, i.e., bottom-most diagram of Fig. B.11 on Slide 610.

channel
323. { ch[{i,j}] | {i,j}:(PLSI|UI) · {i,j}$\subseteq\sigma_{id}$ }

B.3.3 Actions

- These are, informally, some of the actions of a pipeline system:

324. **start pumping**: from a state of not pumping to a state of pumping “at full blast !”.[7]
325. **stop pumping**: from a state of (full) pumping to a state of no pumping at all.
326. **open valve**: from a state of a fully closed valve to a state of fully open valve.[8]
327. **close valve**: from a state of a fully opened valve to a state of fully closed valve.

- We shall not define these actions in this paper.
- But they will be referred to in the *pipeline_system* (Items 346a, 346b, 346c), the *pump* (Items 349a, 349b) and the *valve* (Items 352a, 352b) behaviours.

[7]– that is, we simplify, just for the sake of illustration, and do not consider “intermediate” states of pumping.
[8]– cf. Footnote 7.

B.3.4 Behaviours

B.3.4.1 Behaviour Kinds

- There are eight kinds of behaviours:

328. the pipeline system behaviour;[9]
329. the [generic] well behaviour,
330. the [generic] pipe behaviour,
331. the [generic] pump behaviour,
332. the [generic] valve behaviour,
333. the [generic] fork behaviour,
334. the [generic] join behaviour,
335. the [generic] sink behaviour.

[9]This "PLS" behaviour summarises the either global, i.e., *SCADA*[10]-like behaviour, or the fully distributed, for example, manual, human-operated behaviour of the monitoring and control of the entire pipeline system.

[10]Supervisory Control And Data Acquisition

B.3.4.2 Behaviour Signatures

336. The *pipeline_system* behaviour, *pls*,
337. The *well* behaviour signature lists the unique well identifier, the well mereology, the static well attributes, the monitorable well attributes, the programmable well attributes and the channels over which the well [may] interact with the pipeline system and a pipeline unit.
338. The *pipe* behaviour signature lists the unique pipe identifier, the pipe mereology, the static pipe attributes, the monitorable pipe attributes, the programmable pipe attributes and the channels over which the pipe [may] interact with the pipeline system and its two neighbouring pipeline units.

339. The *pump* behaviour signature lists the unique pump identifier, the pump mereology, the static pump attributes, the monitorable pump attributes, the programmable pump attributes and the channels over which the pump [may] interact with the pipeline system and its two neighbouring pipeline units.
340. The *valve* behaviour signature lists the unique valve identifier, the valve mereology, the static valve attributes, the monitorable valve attributes, the programmable valve attributes and the channels over which the valve [may] interact with the pipeline system and its two neighbouring pipeline units.

341. The *fork* behaviour signature lists the unique fork identifier, the fork mereology, the static fork attributes, the monitorable fork attributes, the programmable fork attributes and the channels over which the fork [may] interact with the pipeline system and its three neighbouring pipeline units.
342. The *join* behaviour signature lists the unique join identifier, the join mereology, the static join attributes, the monitorable join attributes, the programmable join attributes and the channels over which the join [may] interact with the pipeline system and its three neighbouring pipeline units.

343. The *sink* behaviour signature lists the unique sink identifier, the sink mereology, the static sing attributes, the monitorable sing attributes, the programmable sink attributes and the channels over which the sink [may] interact with the pipeline system and its one or more pipeline units.

value

```
336. pls: plso:PLSI → pls_mer:PLS_Mer → PLS_Sta → PLS_Mon →
336.                PLS_Prg → { ch[ {plsi,ui} ] | ui:UI · ui ∈ σ_ui } Unit
337. well: wid:WI → well_mer:MER → Well_Sta → Well_mon →
337.                Well_Prgr →  { ch[ {plsi,ui} ] | wi:WI · ui ∈ σ_ui } Unit
338. πipe: UI → pipe_mer:MER → Pipe_Sta → Pipe_mon →
338.                Pipe_Prgr →  { ch[ {plsi,ui} ] | ui:UI · ui ∈ σ_ui } Unit
339. pump: pi:UI → pump_mer:MER → Pump_Sta → Pump_Mon →
339.                Pump_Prgr →  { ch[ {plsi,ui} ] | ui:UI · ui ∈ σ_ui } Unit
340. valve: vi:UI → valve_mer:MER → Valve_Sta → Valve_Mon →
340.                Valve_Prgr →  { ch[ {plsi,ui} ] | ui:UI · ui ∈ σ_ui } Unit
341. fork: fi:FI → fork_mer:MER → Fork_Sta → Fork_Mon →
341.                 Fork_Prgr →  { ch[ {plsi,ui} ] | ui:UI · ui ∈ σ_ui } Unit
342. join: ji:JI → join_mer:MER → Join_Sta → Join_Mon →
342.                Join_Prgr →  { ch[ {plsi,ui} ] | ui:UI · ui ∈ σ_ui } Unit
343. sink: si:SI → sink_mer:MER → Sink_Sta → Sink_Mon →
343.                Sink_Prgr →  { ch[ {plsi,ui} ] | ui:UI · ui ∈ σ_ui } Unit
```

B.3.4.2.1 Behaviour Definitions

- We show the definition of only three behaviours:
- the pipe_line_system behaviour,
- the pump behaviour and
- the valve behaviour.

B.3.4.2.2 The Pipeline System Behaviour

344. The pipeline system behaviour
345. calculates, based on its programmable state, its next move;
346. if that move is [to be] an action on a named
    (a) pump, whether to start or stop pumping, then the named pump is so informed, whereupon the pipeline system behaviour resumes in the new pipeline state; or
    (b) valve, whether to open or close the valve, then the named valve is so informed, whereupon the pipeline system behaviour resumes in the new pipeline state; or
    (c) unit, to collect its monitorable attribute values for monitoring, whereupon the pipeline system behaviour resumes in the further updated pipeline state;
    (d) et cetera;

```
value
344.  pls(plsi)(uis)(pls_msta)(pls_mon)(pls_$\omega$) ≡
345.    let (to_do,pls_$\omega'$) = calculate_next_move(plsi,pls_mer,pls_msta,pls_mon,pls_prgr
346.    case to_do of
346a      mk_Pump(pi,$\alpha$) →
346a          ch[{plsi,pi}] ! $\alpha$ assert: $\alpha$ ∈ {stop_pumping,pump};
346a          pls(plsi)(pls_mer)(pls_msta)(pls_mon)(pls_$\omega'$),
346b      mk_Valve(vi,$\alpha$) →
346b          ch[{plsi,vi}] ! $\alpha$ assert: $\alpha$ ∈ {open_valve,close_valve};
346b          pls(plsi)(pls_mer)(pls_msta)(pls_mon)(pls_$\omega'$),
346c      mk_Unit(ui,monitor) →
346c          ch[{plsi,ui}] ! monitor;
346c          pls(plsi)(pls_mer)(pls_msta)(pls_mon)(update_pls_$\omega$(ch[{plsi,ui}] ?,ui)(pls_$\omega$
346d      ... end
344     end
```

- We leave it to the reader to define the calculate_next_move function !

B.3.4.2.3 The Pump Behaviours

347. The [generic] pump behaviour internal non-deterministically alternates between
348. doing own work (...), or
349. accepting pump directives from the pipeline behaviour.
    (a) If the directive is either to start or stop pumping, then that is what happens – whereupon the pump behaviour resumes in the new pumping state.
    (b) If the directive requests the values of all monitorable attributes, then these are *gathered*, communicated to the pipeline system behaviour – whereupon the pump behaviour resumes in the "old" state.

```
value
347.  pump(π)(pump_mer)(pump_sta)(pump_mon)(pump_prgr) ≡
348.    ...
349.   ⨅ let α = ch[{plsi,π}] ? in
349.      case α of
349a.          stop_pumping ∨ pump
349a.             → pump(π)(pump_mer)(pump_sta)(pump_mon)(α)[11] end,
349b.          monitor
349b.             → let mvs = gather_monitorable_values(π,pump_mon) in
349b.               ch[{plsi,π}] ! mvs;
349b.               pump(π)(pump_mer)(pump_sta)(pump_mon)(pump_prgr) end
349.      end
```

- We leave it to the reader to defined the gather_monitorable_values function.

[11] Updating the programmable pump state to either **stop_pumping** or **pump** shall here be understood to mean that the pump is set to not pump, respectively to pump.

B.3.4.2.4 The Valve Behaviours

350. The [generic] valve behaviour internal non-deterministically alternates between
351. doing own work (...), or
352. accepting valve directives from the pipeline system.
    (a) If the directive is either to open or close the valve, then that is what happens – whereupon the pump behaviour resumes in the new valve state.
    (b) If the directive requests the values of all monitorable attributes, then these are *gathered*, communicated to the pipeline system behaviour – whereupon the valve behaviour resumes in the "old" state.

```
value
350.  valve(vi)(valv_mer)(valv_sta)(valv_mon)(valv_prgr) ≡
351.    ...
352.   ⨅ let $\alpha$ = ch[{plsi,$\pi$}] ? in
352.      case $\alpha$ of
352a.        open_valve ∨ close_valve
352a.          → valve(vi)(val_mer)(val_sta)(val_mon)($\alpha$)[12] end,
352b.        monitor
352b.          → let mvs = gather_monitorable_values(vi,val_mon) in
352b.            ch[{plsi,$\pi$}] ! (vi,mvs);
352b.            valve(vi)(val_mer)(val_sta)(val_mon)(val_prgr) end
352.      end
```

[12]Updating the programmable valve state to either **open_valve** or **close_valve** shall here be understood to mean that the valve is set to open, respectively to closed position.

B.3.4.3 Sampling Monitorable Attribute Values

- Static and programmable attributes are, as we have seen, *passed by value* to behaviours.
- Monitorable attributes "surreptitiously" change their values so, as a technical point, these are *passed by reference* –
- by *passing attribute type names*.

353. From the name, ηA, of a monitorable attribute and
the unique identifier, u_i, of the part having the named
monitorable attribute
one can then, "dynamically", "on-the-fly",
as the part behaviour "moves-on", retrieve the value of the
monitorable attribute. This can be illustrated as follows:

354. The unique identifier u_i is used in order to retrieve, from the global parts state, σ, that identified part, p.

355. Then attr_A is applied to p.

value

353. retr_U: UI $\rightarrow \Sigma \rightarrow$ U
353. retr_U(ui)(σ) $\equiv$ **let** u:U $\cdot$ u $\in \sigma \wedge$uid_U(u)=ui **in** u **end**
354. retr_AttrVal: UI $\times$ ηA $\rightarrow \Sigma \rightarrow$ A
355. retr_AttrVal(ui)(ηA)(σ) $\equiv$ attr_A(retr_U(ui)(σ))

- retr_AttrVal(...)(...)(...) can now be applied in the body of the behaviour definitions, for example in gather_monitorable_values.

B.3.4.4 System Initialisation

- System initialisation means to "morph" all manifest parts
  - into their respective behaviours,
  - initialising them with their respective attribute values.

356. The *pipeline system* behaviour is initialised and "put" in parallel with the parallel compositions of
357. all initialised *well*,
358. all initialised *pipe*,
359. all initialised *pump*,
360. all initialised *valve*,
361. all initialised *fork*,
362. all initialised *join* and
363. all initialised *sink* behaviours.[13]

[13]Plates are treated as are structures, i.e., not "behaviourised"!

value

356. pls(uid_PLS(pls))(mereo_PLS(pls))((pls))((pls))((pls))
357. $\|$ $\|$ { well(uid_U(we))(mereo_U(we))(sta_A_We(we))(mon_A_We(we))(prg_A_We(we)) | we:Well · w $\in \sigma$ }
358. $\|$ $\|$ { pipe(uid_U(pi))(mereo_U(pi))(sta_A_Pi(pi))(mon_A_Pi(pi))(prg_A_Pi(pi)) | pi:Pi · pi $\in \sigma$ }
359. $\|$ $\|$ { pump(uid_U(pu))(mereo_U(pu))(sta_A_Pu(pu))(mon_A_Pu(pu))(prg_A_Pu(pu)) | pu:Pump · pu $\in \sigma$ }
360. $\|$ $\|$ { valv(uid_U(va))(mereo_U(va))(sta_A_Va(va))(mon_A_Va(va))(prg_A_Va(va)) | va:Well · va $\in \sigma$ }
361. $\|$ $\|$ { fork(uid_U(fo))(mereo_U(fo))(sta_A_Fo(fo))(mon_A_Fo(fo))(prg_A_Fo(fo)) | fo:Fork · fo $\in \sigma$ }
362. $\|$ $\|$ { join(uid_U(jo))(mereo_U(jo))(sta_A_Jo(jo))(mon_A_J(jo))(prg_A_J(jo)) | jo:Join · jo $\in \sigma$ }
363. $\|$ $\|$ { sink(uid_U(si))(mereo_U(si))(sta_A_Si(si))(mon_A_Si(si))(prg_A_Si(si)) | si:Sink · si $\in \sigma$ }

- The sta_..., mon_..., and prg_A... functions are defined in Items 317 on Slide 570.
- Note: $\|$ { f(u)(...) | u:U · u $\in$ {} } $\equiv$ ().

B.4 Index

B.5 Illustrations of Pipeline Phenomena

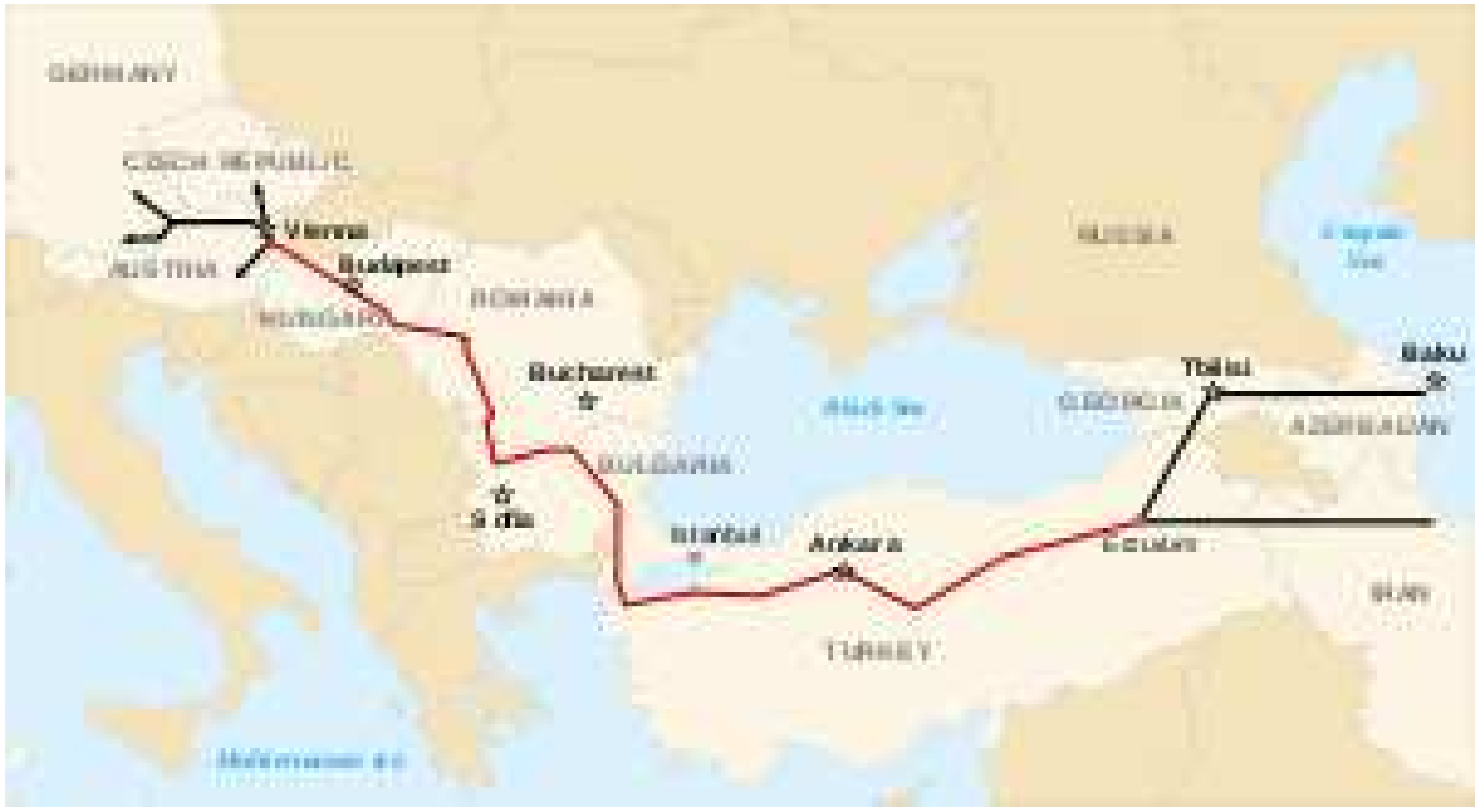

Figure B.2: The Planned Nabucco Pipeline: http://en.wikipedia.org/wiki/Nabucco_Pipeline

Figure B.3: Pipeline Construction

Figure B.4: Pipe Segments

Figure B.5: Valves

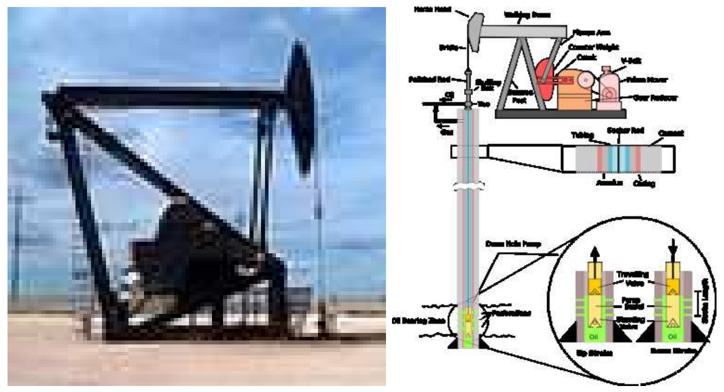

Figure B.6: Oil Pumps

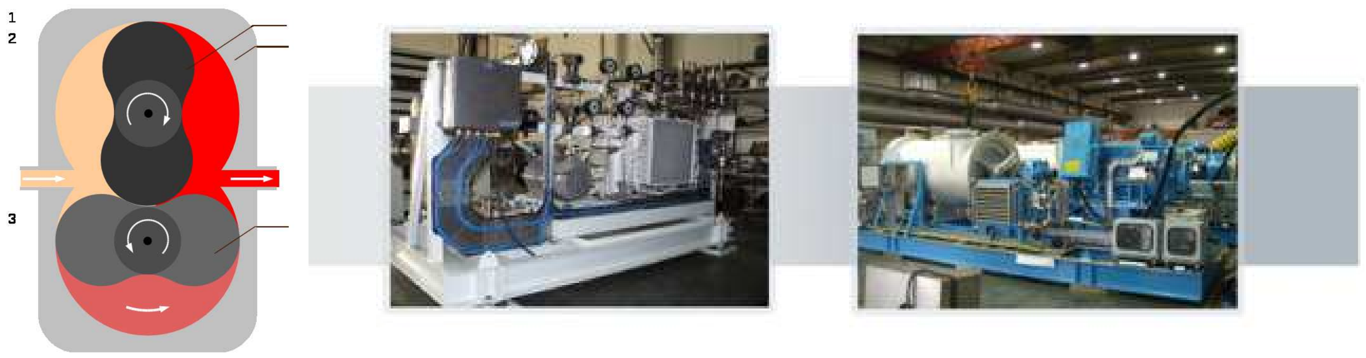

Figure B.7: Gas Compressors

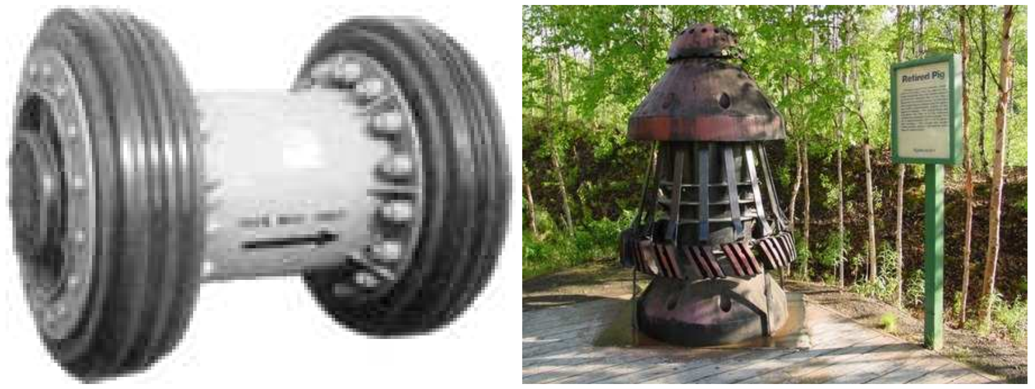
Figure B.8: New and Old Pigs

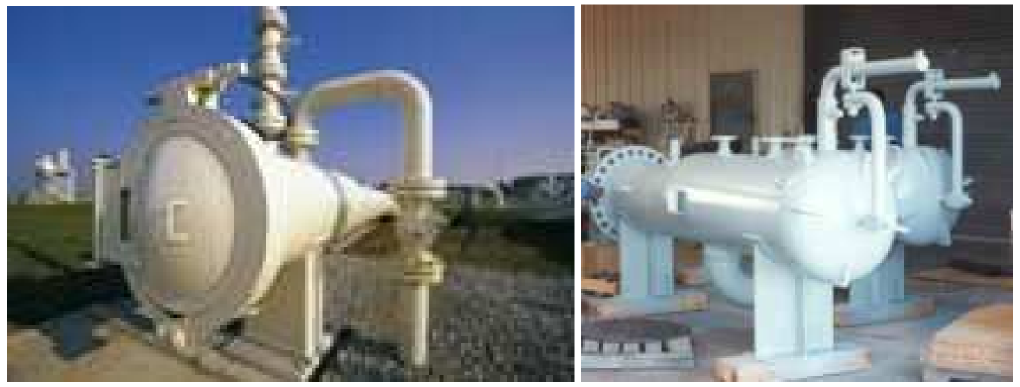
Figure B.9: Pig Launcher, Receiver

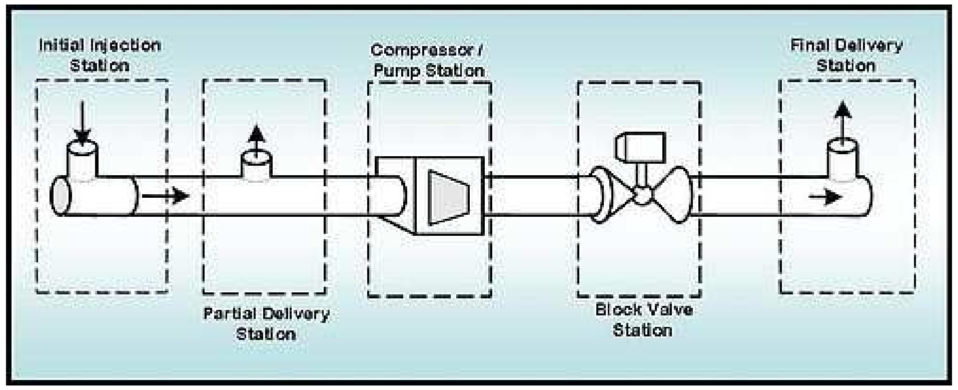

Figure B.10: Leftmost: A Well. 2nd from left: a Fork. Rightmost: a Sink

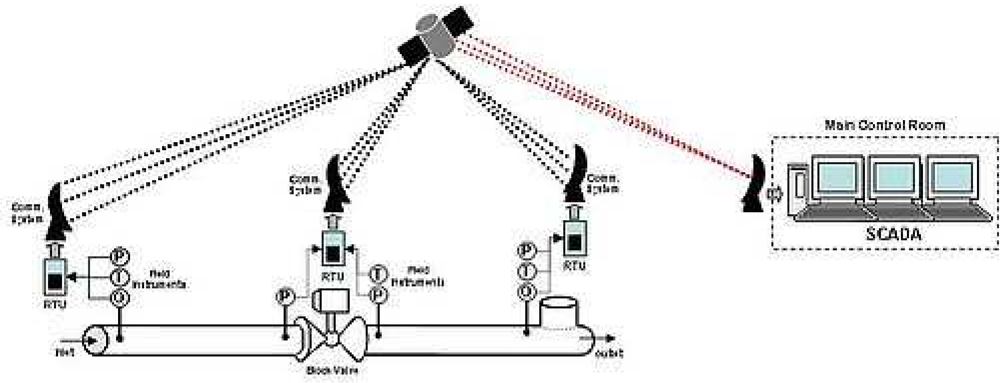

Figure B.11: A SCADA [Supervisory Control And Data Acquisition] Diagram

Appendix C. Bibliography

Bibliography

[1] Rober Audi. *The Cambridge Dictionary of Philosophy*. Cambridge University Press, The Pitt Building, Trumpington Street, Cambridge CB2 1RP, England, 1995.

[2] Dines Bjørner. Formal Software Techniques in Railway Systems. In Eckehard Schnieder, editor, *9th IFAC Symposium on Control in Transportation Systems*, pages 1–12, Technical University, Braunschweig, Germany, 13–15 June 2000. VDI/VDE-Gesellschaft Mess– und Automatisieringstechnik, VDI-Gesellschaft für Fahrzeug– und Verkehrstechnik. Invited talk.

[3] Dines Bjørner. Domain Models of "The Market" — in Preparation

for E–Transaction Systems. In *Practical Foundations of Business and System Specifications (Eds.: Haim Kilov and Ken Baclawski)*, The Netherlands, December 2002. Kluwer Academic Press. www2.imm.dtu.dk/˜dibj/themarket.pdf.

[4] Dines Bjørner. Dynamics of Railway Nets: On an Interface between Automatic Control and Software Engineering. In *CTS2003: 10th IFAC Symposium on Control in Transportation Systems*, Oxford, UK, August 4-6 2003. Elsevier Science Ltd. Symposium held at Tokyo, Japan. Editors: S. Tsugawa and M. Aoki. www2.imm.dtu.dk/˜dibj/ifac-dynamics.pdf.

[5] Dines Bjørner. A Container Line Industry Domain. www.imm.dtu.dk/ db/container-paper.pdf. Techn. report, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, Denmark, June 2007.

[6] Dines Bjørner. On Development of Web-based Software: A Divertimento of Ideas and Suggestions. Technical, Technical

University of Vienna, August–October 2010. www.imm.dtu.dk/˜dibj/wfdftp.pdf.

[7] Dines Bjørner. The Tokyo Stock Exchange Trading Rules www.imm.dtu.dk/˜db/todai/tse-1.pdf, www.imm.dtu.dk/˜db/todai/tse-2.pdf. R&D Experiment, Techn. Univ. of Denmark, Fredsvej 11, DK-2840 Holte, Denmark, 2010.

[8] Dines Bjørner. Pipelines – a Domain www.imm.dtu.dk/˜dibj/pipe-p.pdf. Experimental Research Report 2013-2, DTU Compute and Fredsvej 11, DK-2840 Holte, Denmark, Spring 2013.

[9] Dines Bjørner. *A Rôle for Mereology in Domain Science and Engineering*. Synthese Library (eds. Claudio Calosi and Pierluigi Graziani). Springer, Amsterdam, The Netherlands, October 2014.

[10] Dines Bjørner. A Credit Card System: Uppsala Draft www.imm.dtu.dk/˜dibj/2016/credit/accs.pdf. Technical Report:

Experimental Research, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, Denmark, November 2016.

[11] Dines Bjørner. Weather Information Systems: Towards a Domain Description www.imm.dtu.dk/˜dibj/2016/wis/wis-p.pdf. Technical Report: Experimental Research, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, Denmark, November 2016.

[12] Dines Bjørner. A Space of Swarms of Drones. www.imm.dtu.dk/˜dibj/2017/swarms/swarm-paper.pdf. Research Note, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, Denmark, December 2017.

[13] Dines Bjørner. What are Documents ? www.imm.dtu.dk/˜dibj/2017/docs/docs.pdf. Research Note, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, Denmark, July 2017.

[14] Dines Bjørner. Container Terminals. www.imm.dtu.dk/ dibj/2018/yangshan/maersk-pa.pdf. Technical report, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, Denmark, September 2018. An incomplete draft report; currently 60+ pages.

[15] Dines Bjørner. *An Assembly Plant Domain* – Analysis & Description, www.imm.dtu.dk/ dibj/2021/assembly/assembly-line.pdf. Technical report, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, Denmark, September 2019.

[16] Dines Bjørner. A Retailer Market: Domain Analysis & Description. A Comparison Heraklit/DS&E Case Study. www.imm.dtu.dk/ dibj/2021/Retailer/BjornerHeraklit27January2021.pdf. Technical Report, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, Denmark, January 2021.

[17] Dines Bjørner. Automobile Assembly Plants.

www.imm.dtu.dk/~dibj/2021/assembly/assembly-line.pdf. Technical Report, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, Denmark, Summer 2021.

[18] Dines Bjørner. *Domain Science & Engineering – A Foundation for Software Development*. EATCS Monographs in Theoretical Computer Science. Springer, 2021.

[19] Dines Bjørner. Rivers and Canals. www.imm.dtu.dk/~dibj/2021/Graphs/Rivers-and-Canals.pdf. Technical Report, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, Denmark, March 2021.

[20] Dines Bjørner. Shipping. www.imm.dtu.dk/~dibj/2021/ral/ral.pdf. Technical Report, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, Denmark, April 2021.

[21] Dines Bjørner and Asger Eir. Compositionality: Ontology and Mereology of Domains. Some Clarifying Observations in the

Context of Software Engineering in July 2008, eds. Martin Steffen, Dennis Dams and Ulrich Hannemann. In *Festschrift for Prof. Willem Paul de Roever Concurrency, Compositionality, and Correctness*, volume 5930 of *Lecture Notes in Computer Science*, pages 22–59, Heidelberg, July 2010. Springer.

[22] Dines Bjørner, Chris W. George, and Søren Prehn. Computing Systems for Railways — A Rôle for Domain Engineering. Relations to Requirements Engineering and Software for Control Applications. In *Integrated Design and Process Technology. Editors: Bernd Kraemer and John C. Petterson*, P.O.Box 1299, Grand View, Texas 76050-1299, USA, 24–28 June 2002. Society for Design and Process Science. www2.imm.dtu.dk/˜dibj/pasadena-25.pdf.

[23] Dines Bjørner. Urban Planning Processes. www.imm.dtu.dk/˜dibj/2017/up/urban-planning.pdf. Research Note, Technical University of Denmark, Fredsvej 11, DK-2840

Holte, Denmark, July 2017.

[24] Wayne D. Blizard. A Formal Theory of Objects, Space and Time. *The Journal of Symbolic Logic*, 55(1):74–89, March 1990.

[25] Roberto Casati and Achille C. Varzi. *Parts and Places: the structures of spatial representation*. MIT Press, 1999.

[26] Asger Eir. *Construction Informatics — issues in engineering, computer science, and ontology*. PhD thesis, Dept. of Computer Science and Engineering, Institute of Informatics and Mathematical Modeling, Technical University of Denmark, Building 322, Richard Petersens Plads, DK–2800 Kgs.Lyngby, Denmark, February 2004.

[27] Asger Eir. *Formal Methods and Hybrid Real-Time Systems*, chapter Relating Domain Concepts Intensionally by Ordering Connections, pages 188–216. Springer (LNCS Vol. 4700, Festschridt: Essays in Honour of Dines Bjørner and Zhou Chaochen on the Occasion of Their 70th Birthdays), 2007.

[28] David John Farmer. *Being in time: The nature of time in light of McTaggart's paradox*. University Press of America, Lanham, Maryland, 1990. 223 pages.

[29] Carlo A. Furia, Dino Mandrioli, Angelo Morzenti, and Matteo Rossi. *Modeling Time in Computing*. Monographs in Theoretical Computer Science. Springer, 2012.

[30] Charles Anthony Richard Hoare. *Communicating Sequential Processes*. C.A.R. Hoare Series in Computer Science. Prentice-Hall International, 1985. Published electronically: usingcsp.com/cspbook.pdf (2004).

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

[32] W. Little, H.W. Fowler, J. Coulson, and C.T. Onions. *The Shorter Oxford English Dictionary on Historical Principles*. Clarendon Press, Oxford, England, 1973, 1987. Two vols.

[33] J. M. E. McTaggart. The Unreality of Time. *Mind*, 18(68):457–84, October 1908. New Series. See also: [34].

[34] Robin Le Poidevin and Murray MacBeath, editors. *The Philosophy of Time*. Oxford University Press, 1993.

[35] Karl R. Popper. *Logik der Forschung*. Julius Springer Verlag, Vienna, Austria, 1934 (1935). English version [36].

[36] Karl R. Popper. *The Logic of Scientific Dicovery*. Hutchinson of London, 3 Fitzroy Square, London W1, England, 1959,. . . ,1979. Translated from [35].

[37] Arthur Prior. *Changes in Events and Changes in Things*, chapter in [34]. Oxford University Press, 1993.

[38] Arthur N. Prior. *Logic and the Basis of Ethics*. Clarendon Press, Oxford, UK, 1949.

[39] Arthur N. Prior. *Formal Logic*. Clarendon Press, Oxford, UK, 1955.

[40] Arthur N. Prior. *Time and Modality*. Oxford University Press, Oxford, UK, 1957.

[41] Arthur N. Prior. *Past, Present and Future*. Clarendon Press, Oxford, UK, 1967.

[42] Arthur N. Prior. *Papers on Time and Tense*. Clarendon Press, Oxford, UK, 1968.

[43] Gerald Rochelle. *Behind time: The incoherence of time and McTaggart's atemporal replacement*. Avebury series in philosophy. Ashgate, Brookfield, Vt., USA, 1998. vii + 221 pages.

[44] Kai Sørlander. *Det Uomgængelige – Filosofiske Deduktioner [The Inevitable – Philosophical Deductions, with a foreword by Georg Henrik von Wright]*. Munksgaard · Rosinante, Copenhagen, Denmark, 1994. 168 pages.

[45] Kai Sørlander. *Under Evighedens Synsvinkel [Under the viewpoint of eternity]*. Munksgaard · Rosinante, Copenhagen,

Denmark, 1997. 200 pages.

[46] Kai Sørlander. *Den Endegyldige Sandhed [The Final Truth]*. Rosinante, Copenhagen, Denmark, 2002. 187 pages.

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

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

[49] Steven Weintraub. *Galois Theory*. Springer, 2009.

[50] Johan van Benthem. *The Logic of Time*, volume 156 of *Synthese Library: Studies in Epistemology, Logic, Methhodology, and Philosophy of Science (Editor: Jaakko Hintika)*. Kluwer Academic Publishers, P.O.Box 17, NL 3300 AA Dordrecht, The Netherlands, second edition, 1983, 1991.