Domain Analysis and Description

Dines Bjørner

Fredsvej 11, DK–2840 Holte, Denmark

June 30, 2014

It's life that matters, nothing but life –
the process of discovering, the everlasting and perpetual process,
not the discovery itself, at all.

`Fyodor Dostoyevsky, The Idiot, 1868, Part 3, Sect. V`

Summary

- We present a new approach to *domain analysis &*[1] *description*.
  - A '**domain**'[2] is seen as a mapping
    - from *entities*
    - to *qualities*,

    that is, a mapping
    - from manifest phenomena
    - to usually non-manifest qualities.

[1] When we write `A & B` we mean that the conjunction of the two concepts `A` and `B` form "a somehow" inseparable" concept.

[2] This is the first of a number of complementary characterisations of the concept of a domain. All these occurrences are marked by footnotes: Footnotes 2, 4 on slide 7, 5 on slide 9 and 18 on slide 249.

- This idea is then concretised, considerably, in the form of
  - analysing *entities* into either
    - **endurant** *entities* and
    - **perdurant** *entities*.
- We shall then analyse *endurant entities* into either
  - **discrete entities**,

    and, when so, into either

    - **atomic entities** or **composite entities**,

    or into
  - **continuous entities**.
- For *discrete endurants* we shall analyse qualities into

  - **unique identity**, **mereology** and **attributes**.

- We shall distinguish between
  - **external qualities** and
  - **internal qualities**.
- External qualities are such which are observable:
  - whether endurant or perdurant,
  - whether discrete or continuous,
  - whether atomic or composite.
- Internal qualities are (usually) not observable:
  - unique identity,
  - mereology and
  - attributes.

- The terms *endurant entities* and *perdurant entities* are terms we prefer in the context of domains.
- In the context of computing one may think of
  - *endurant entities* as *data* and
  - *perdurant entities* as *processes*.

- The study behind this seminar represents an ambitious undertaking:
  - Within the still daunting span of some 337 slides
  - it surveys a rather comprehensive set of techniques and tools
  - for the description of non-trivial, non-"toy-example" domains.
- The reader must, please, be prepared, to cast aside
  - previous conceptions of how domains may be specified;
  - the present proposal "starts all over, from basic principles".
- Perhaps a
  - short monograph rather
  - than a "longish" paper

  would be a better way to reach a reasonable audience,
- We shall see.

Introduction

- We offer a solution to the problem of
  - precisely describing, informally and formally,
  - spatio-temporal[3], human- and artifact-assisted domains.[4]

Example 1 . Domains: *Examples of domains are:*

- *air traffic,*
- *banks,*
- *container lines,*
- *documents,*
- *hospitals,*
- *manufacturing,*
- *pipelines,*
- *railways* and
- *road nets.*

■

[3] Spatio-temporal: Having both spatial and temporal properties

[4] This sentence loosely characterises the kind of domains for which our analysis & description techniques are well suited. It is our second characterisation of the concept of domain.

1.1. The Problem Area

- The problem area is
  - broadly that of
    methods for software development
  - and more narrowly that of
    methods for domain analysis & domain description.

1.1.1. Domains and Entities

- By a 'domain'[5] we shall here understand
  - an area of human activity
  - characterised by observable phenomena[6]:
    - entities,
      - whether endurants (manifest parts and materials)
      - or perdurants (actions, events and behaviours),
    - whether
      - discrete or
      - continuous,
    - and, if discrete, whether
      - atomic or
      - composite.
    - and of their qualities.

[5] This is our third characterisation of the concept of domain.

[6] As to what characterises an observable phenomenon we shall not speculate. This would entail a longer discourse [Aristotle,Russell1905,Peirce2,Whitehead1919,Rus18-19,Whitehead1920,Russell1922,RudolfCarnap1928,BarrySmith1993,PhilCA1997,ChrisFox2000,JohnSowa200(

1.1.2. The TripTych Approach to Software Engineering

- We suggest a **TripTych** view of software engineering:
  - *before software can be designed and coded*
  - *we must have a reasonable grasp of "its" requirements;*
  - *before requirements can be prescribed*
  - *we must have a reasonable grasp of "the underlying" domain.*

- To us, therefore, software engineering consists of three sub-disciplines:
  - domain engineering,
  - requirements engineering and
  - software design.

- This seminar focus on aspects of a **methodology** for **domain analysis** & **domain description**.
- References [dines:ugo65:2008]
  - show how to "refine" **domain descriptions** into **requirements prescriptions**,

  and reference [DomainsSimulatorsDemos2011]
  - indicates more general relations between **domain descriptions** and
    - **domain demos**,
    - **domain simulators** and
    - more general **domain specific software**.

1.1.3. Method

- By a 'method' we shall understand
  - a "somehow structured" set of **principles**
  - for **selecting** and **applying**
  - a number of **techniques** and **tools**
  - for **analysing** problems and **synthesizing** solutions
  - for a given domain.

- The 'somehow structuring' amounts,
  - in this treatise on domain analysis & description,
  - to the techniques and tools being related to a set of
  - domain analysis & description "prompts",
  - "issued by the method",
  - prompting the domain analyser,
  - hence carried out by the **domain analyser** & **describer**[7] —
  - conditional upon the result of other prompts.

[7] We shall use the term **domain engineer** to cover both the analyser & the describer.

- **A Critique:**
  - There may be other 'definitions' of the term 'method'.
  - The above is the one that will be adhered to in this seminar.
  - The main idea is that
    - there is a clear understanding of what we mean by, as here,
      - a software development method,
      - in particular a *domain analysis & description method.*

• • •

- A main contribution of this seminar is therefore
  - that of "painstakingly" elucidating the
    - principles,
    - techniques and
    - tools

    of the domain analysis & description method.

1.1.4. Methodology

- By 'methodology' we shall understand
  - the study and knowledge
  - about one or more methods.[8]

[8] Please note our distinction between method and methodology. We often find the two, to us, separate terms used interchangeably.

1.1.5. Towards a Methodology of Domain Analysis & Description

- By a '**domain description**' we shall understand a text which describes
  - the **entities** of the domain:
    - whether **endurant** or **perdurant**,
    - and when endurant whether
      - **discrete** or **continuous**,
      - **atomic** or **composite**;
    - or when perdurant whether
      - **action**s,
      - **event**s or
      - **behaviour**s.
  - as well as the **qualities** of these **entities**.

1.1.5.1 Practicalities of Domain Analysis & Description

- How does one go about analysing & describing a domain ?
  - Well, for the first,
    - one has to designate one or more **domain analysers** cum
    - **domain describers**,
    - i.e., trained **domain scientists** cum **domain engineers**.
  - How does one get hold of a **domain engineer** ?
    - One takes a **software engineer** and *educates* and *trains* that person in
      - **domain science** &
      - **domain engineering**
    - A derivative purpose of this seminar is to unveil aspects of **domain science & domain engineering**.

- The education and training consists in bringing forth
  - a number of scientific and engineering issues
    - of **domain analysis** and
    - of **domain description**.
  - Among the engineering issues are such as:
    - *what do I do when confronted*
      - *with the task of domain analysis ?* and
      - *with the task of description ?* and
    - *when, where and how do I*
      - `select` *and* `apply`
      - *which* `techniques` *and which* `tools` ?

- Finally, there is the issue of
  - *how do I, as a domain describer, choose appropriate*
    - *abstractions and*
    - *models ?*

1.1.5.2 The Four Domain Analysis & Description "Players"

- We can say that there are four 'players' at work here.
  - the domain,
  - the domain analyser & describer,
  - the domain analysis & description method, and
  - the evolving domain analysis & description.

- The *domain* is there.
  - The domain analyser & describer cannot change the domain.
  - Analysing & describing the domain does not change it.
  - In a meta-physical sense it is inert.
  - In the physical sense the domain will usually contain
    - parts that are static (i.e., constant), and
    - parts that are dynamic (i.e., variable).

- The *domain analyser & domain describer* is a human,
  - preferably a scientist/engineer[9],
  - well-educated and trained in domain science & engineering.
  - The domain analyser & describer
    - observes the domain,
    - analyses it according to a method and
    - thereby produces a domain description.

[9] At the present time domain analysis appears to be partly an art, partly a scientific endeavour. Until such a time when domain analysis & description principles, techniques and tools have matured it will remain so.

- As a concept the ***method*** is here considered "fixed".
  - By 'fixed' we mean that its principles, techniques and tools do not change during a domain analysis & description.
  - The domain analyser & describer
    - may very well apply these principles, techniques and tools
    - more-or-less haphazardly,
    - flaunting the method,
    - but that method remains invariant.
  - The method, however, may vary
    - from one domain analysis & description (project)
    - to another domain analysis & description (project).
  - Domain analysers & describer do become wiser from a project to the next.

- Finally there is the evolving *domain analysis & description*.
  - That description is a text, usually both informal and formal.
  - Applying a *domain description synthesiser* to the domain
    - yields an *additional domain description text*
    - which is added to the thus evolving *domain description*.
  - One may speculate of the rôle of the "input" domain description.
    - Does it change ?
    - Does it help determine the additional domain description text ?
    - Etcetera.
  - Without loss of generality we can assume
    - that the "input" domain description is changed and
    - that it helps determine the added text.

1.1.5.3 An Interactive Domain Analysis & Description Dialogue

- We see domain analysis & description
  - as a process involving the above-mentioned four 'players',
  - that is, as a dialogue
  - between the domain analyser & describer and the domain,
  - where the dialogue is guided by the method
  - and the result is the description.
- We see the method as a 'player' which issues prompts:
  - alternating between:
  - *"analyse this"* (analysis prompts) and
  - *"describe that"* (synthesis or, rather, description prompts).

1.1.5.4 Prompts

- In this paper we shall suggest
  - a number of *domain analysis prompts* and
  - a number of *domain description prompts*.
- The **'domain analysis prompt'**s,
  - schematically `analyse_named_condition(p)`,
  - direct the analyser to inquire
  - as to the truth of whatever the prompt "names"
  - at wherever part (or material), p, in the domain the prompt so designates.

- Based on the truth value of an analysed part the domain analyser may then be prompted to describe that part (or material).
- The **'domain description prompt'**s,
  - schematically `describe_type_or_quality(e)`,
  - direct the (analyser cum) describer to formulate
  - both an informal and a formal description
  - of the type or qualities of the entity (part or material) designated by the prompt.
- The prompts form languages, and there are thus two languages at play here.

1.1.5.5 A Domain Analysis & Description Language

- The 'Domain Analysis & Description Language' thus consists of a number of meta-functions, the prompts.
  - The meta-functions have names (say `is_endurant`) and types,
  - but have no formal definition.
  - They are not computable.
  - They are "performed" by the domain analysers & describers.
  - These meta-functions are systematically introduced and informally explained in Sect. 4.

1.1.5.6 The Domain Description Language

- The 'Domain Description Language' is RSL [RSL], the RAISE Specification Language [RaiseMethod].
- With suitable, simple adjustments it could also be either of
  - Alloy [alloy],
  - Event B [JRAbrial:TheBBooks],
  - VDM-SL [e:db:Bj78bwo,e:db:Bj82b,JohnFitzgerald+PeterGormLarsen] or
  - Z [m:z:jd+jcppw96].
- We have chosen RSL because of its simple provision for
  - defining sorts,
  - expressing axioms, and
  - postulating observers over sorts.

1.1.5.7 Domain Descriptions: Narration & Formalisation

- Descriptions
  - *must* be readable and
  - *should* be mathematically precise[10].
- For that reason we decompose domain description fragments into clearly identified "pairs" of
  - narrative texts and
  - formal texts.
- We refer to Slides 79–82.

[10] In order to be able to verify that domain descriptions satisfy a number of qualities not explicitly formulated as well as in order to very that requirements prescriptions satisfy domain descriptions we must insist on formalised domain descriptions.

1.2. Our Contribution

- The contribution, we *claim*, consists of three sets of concepts:
  - a set of domain description components,
  - a corresponding collection of domain observer functions, and
  - a set of domain analysis & description prompts.
- These are covered "triple-by-triple" in Sect. 3.

1.2.1. Domain Engineering as a Separate Activity

- There is, however, a more important contribution hidden here:
  - it is that of promoting domain analysis & description
  - as a prerequisite development activity
    separate from and prior to
    requirements prescription;
  - in fact, more generally, as a "free-standing" activity:
    - one that can be pursued whether or not
    - we subsequently pursue requirements prescription.
  - A number of publications discuss these issues.

1.2.2. Domain Description Components

- We suggest an approach to domain analysis & description which, in this seminar, focus on
  - domains as consisting of **endurant entities**,
    - their **atomicity** or **compositionality**,
    - the **sort**s (i.e., **abstract type**s) of **part**s and **material**s, and
    - **subpart**s of parts;
  - and their **qualities**:
    - **identity**,
    - how **composite part**s are composed, that is, **mereology**, and
    - their **attribute**s.

- Perdurants will also be covered in this seminar,
  - but not systematically,
  - with no enumeration of prompts and
  - with no explicit listing of domain description schemas.

1.2.3. Domain Observer Functions

- We consider as novel the following dual (a, b) aspects of this seminar.
  - First (a) the separation of parts and materials, hence
    - (a_1) the notions of *part sorts* and *material sorts*,
    - (a_2) the concepts of *part sort observer functions* and *material sort observer functions* (**obs**_P and **obs**_M), and
    - (a_3) the concepts of concrete part types (P = Type_Expr).

    That is, firstly the focus on external qualities of endurants.

- Then (b) the systematic treatment of part and material qualities:
  - (b_1) the notions of *unique part identifiers* (**uid**_) and *unique part identifier types* (ιP).
  - (b_2) the notion of *mereology: part-hood relations* (**mereo**_P) and *mereology types* ($\mathcal{E}(\iota P_1,\iota P_2,...,\iota P_m)$), and
  - (b_3) the notion of *part attribute types* (say A, B, ..., C) and *part attribute type observers* (**attr**_A, **attr**_B, ..., **attr**_C).

That is, secondly the focus on internal qualities of endurants.

1.2.4. The Domain Analysis & Description Prompts

- We claim that the above structuring of the domain analysis is new:
  - a clear separation of
    - parts and materials (external quality) analysis from
    - the (internal quality) analysis of their qualities; and
  - a clear provision of tools and analysis techniques "manifest" in the form of (first) domain analysis prompts:
    - `is_endurant`,
    - `is_perdurant`,
    - `is_discrete`,
    - `is_continuous`
    - (i.e., `is_part`,
    - `is_material`),
    - `is_atomic_part`,
    - `is_composite_part`,
    - `has_concrete_type`,
    - `has_material`,

- and then (second) domain description prompts:
  - obs_part_sorts,
  - obs_part_type,
  - obs_material_sorts,
  - obs_unique_identifier,
  - obs_mereology,
  - attribute_names and
  - obs_attributes.

• • •

- Section 6.2 reviews the above claims.

Formal Concept Analysis

2.1. A Formalisation

Some Notation:

- By $\mathcal{E}$ we shall understand the type of entities, i.e., an entity;
- by $\mathbb{E}$ we shall understand an entity of type $\mathcal{E}$;
- by $\mathcal{Q}$ we shall understand the type of qualities;
- by $\mathbb{Q}$ we shall understand a quality of type $\mathcal{Q}$;
- by $\mathcal{E}$**-set** we shall understand the type of sets of entities;
- by $\mathbb{ES}$ we shall understand a set of entities, i.e., a value of type $\mathcal{E}$**-set**;
- by $\mathcal{Q}$**-set** we shall understand a set of qualities, i.e., the type of sets of qualities; and
- by $\mathbb{QS}$ we shall understand a value of type $\mathcal{Q}$**-set**.

Definition: 1 Formal Context:

- A 'formal context' $\mathbb{K} := (\mathbb{ES}, \mathbb{I}, \mathbb{QS})$ consists of two sets;
  - $\mathbb{ES}$ of entities and
  - $\mathbb{QS}$ of qualities,

  and a
  - relation $\mathbb{I}$ between $\mathbb{E}$ and $\mathbb{Q}$.

- To express that $\mathbb{E}$ is in relation $\mathbb{I}$ to a Quality $\mathbb{Q}$ we write
  - $\mathbb{E} \cdot \mathbb{I} \cdot \mathbb{Q}$, which we read as
  - *"entity* $\mathbb{E}$ ***has*** *quality* $\mathbb{Q}$*"*.

- Example endurant entities are
  - a specific vehicle,
  - another specific vehicle,
  - ętcetera;
  - a specific street segment (link),
  - another street segment,
  - etcetera;
  - a specific road intersection (hub),
  - another specific road intersection,
  - etcetera,
  - a monitor.
- Example endurant entity qualities are
  - has mobility,
  - has velocity ($\geq$0),
  - has acceleration ($\geq$0),
  - has length ($>$0),
  - has location,
  - has traffic state,
  - etcetera.

Definition: 2 Qualities Common to a Set of Entities:

- For any subset, $s\mathbb{ES} \subseteq \mathbb{ES}$, of entities we can define $\mathcal{DQ}$ for "derive set of qualities".

$$\mathcal{DQ} : \mathcal{E}\textbf{-set} \to (\mathcal{E}\textbf{-set} \times \mathcal{I} \times \mathcal{Q}\textbf{-set}) \to \mathcal{Q}\textbf{-set}$$
$$\mathcal{DQ}(s\mathbb{ES})(\mathbb{ES}, \mathbb{I}, \mathbb{QS}) \equiv \{\mathbb{Q} \mid \mathbb{Q}{:}\mathcal{Q}, \mathbb{E}{:}\mathcal{E} \cdot \mathbb{E} \in s\mathbb{ES} \wedge \mathbb{E} \cdot \mathbb{I} \cdot \mathbb{Q}\}$$
$$\textbf{pre}: s\mathbb{ES} \subseteq \mathbb{ES}$$

The above expresses: *"the set of qualities common to entities in* $s\mathbb{ES}$*"*.

Definition: 3 Entities Common to a Set of Qualities:

- For any subset, $s\mathbb{QS} \subseteq \mathbb{QS}$, of qualities we can define $\mathcal{DE}$ for "derive set of entities".

$$\mathcal{DE}: \ \mathcal{Q}\textbf{-set} \rightarrow (\mathcal{E}\textbf{-set} \times \mathcal{I} \times \ \mathcal{Q}\textbf{-set}) \rightarrow \mathcal{E}\textbf{-set}$$
$$\mathcal{DE}(s\mathbb{QS})(\mathbb{ES}, \mathbb{I}, \mathbb{QS}) \equiv \{\mathbb{E} \mid \mathbb{E}{:}\mathcal{E}, \ \mathbb{Q}{:}\mathcal{Q} \cdot \mathbb{Q} \in s\mathbb{Q} \wedge \mathbb{E} \cdot \mathbb{I} \cdot \mathbb{Q} \ \},$$
$$\textbf{pre}: \ s\mathbb{QS} \subseteq \mathbb{QS}$$

The above expresses: *"the set of entities which have all qualities in* $s\mathbb{QS}$*"*.

Definition: 4 Formal Concept:

- A 'formal concept' of a context $\mathbb{K}$ is a pair:
  - $(s\mathbb{Q}, s\mathbb{E})$ where
    - $\mathcal{DQ}(s\mathbb{E})(\mathbb{E}, \mathbb{I}, \mathbb{Q}) = s\mathbb{Q}$ and
    - $\mathcal{DE}(s\mathbb{Q})(\mathbb{E}, \mathbb{I}, \mathbb{Q}) = s\mathbb{E}$;
  - $s\mathbb{Q}$ is called the 'intent' of $\mathbb{K}$ and $s\mathbb{E}$ is called the 'extent' of $\mathbb{K}$.

■

2.2. Types Are Formal Concepts

- Now comes the "crunch":
  - *In the `TripTych` domain analysis*
  - *we strive to find formal concepts*
  - *and, when we think we have found one,*
  - *we assign a type (or a sort)*
  - *and qualities to it !*

2.3. Practicalities

- There is a little problem.
  - To search for all those entities of a domain
  - which each have the same sets of qualities
  - is not feasible.
- So we do a combination of two things:
  - we identify a small set of entities
    - all having the same qualities
    - and tentatively associate them with a type, and
  - we identify certain nouns of our national language
    - and if such a noun
      - does indeed designate a set of entities
      - all having the same set of qualities
    - then we tentatively associate the noun with a type.

- Having thus, tentatively, identified a type
  - we conjecture that type
  - and search for counterexamples,
    - that is, entities which
    - refutes the conjecture.
- This "process" of conjectures and refutations is iterated
  - until some satisfaction is arrived at
  - that the postulated type constitutes a reasonable conjecture.

2.4. Formal Concepts: A Wider Implication

- The formal concepts of a domain form Galois Connections [GanterWille:ConceptualAnalysis1999].
  - We gladly admit that this fact is one of the reasons that we emphasise formal concept analysis.
  - At the same time we must admit that this seminar does not do justice to this fact.
  - We have experimented with the analysis & description of a number of domains
  - and have noticed such Galois connections
  - but it is, for us, too early to report on this.
- Thus we invite the student to study this aspect of domain analysis.

Endurant Entities
3.1. General

Definition 1 . Entity:

- *By an 'entity' we shall understand a 'phenomenon', i.e., something*
  - *that can be observed, i.e., be*
    - *seen or*
    - *touched*

    by humans,
  - *or that can be conceived*
    - *as an abstraction*
    - *of an entity* ■

Endurant Analysis Prompt 1. is_entity:

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

3.2. Endurants and Perdurants

Definition 2 . Endurant:

- *By an 'endurant' we shall understand an entity*
  - ⋈ *that can be observed or conceived,*
  - ⋈ *as a "complete thing",*
  - ⋈ *at no matter which given snapshot of time.*

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

Example 2 . Traffic System Endurants

Examples of traffic system endurants are:

- *traffic system,*
- *road nets,*
- *fleets of vehicles,*
- *sets of hubs,*
- *sets of links,*
- *hubs,*
- *links,*
- *vehicles.* ■

Definition 3 . Perdurant:

- *By a* **'perdurant'** *we shall understand an entity*
  - ⟐ *for which only a fragment exists*
    if we look at or touch them
    at any given snapshot in time, that is,
  - ⟐ *where we to freeze time we would only see or touch*
    a fragment of the perdurant.

Example 3 . Traffic System Perdurants

Examples of road net perdurants are:

- *insertion and removals of hubs or links (actions),*
- *disappearance of links (events),*
- *vehicles entering or leaving the road net (actions),*
- *vehicles crashing (events) and*
- *road traffic (behaviour).* ■

Endurant Analysis Prompt 2. is_endurant:

- *The domain analyser analyses entities e into endurants 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` ■

Endurant Analysis Prompt 3. is_perdurant:

- *The domain analyser analyses entities e into perdurants as prompted by the* **'domain analysis prompt'**:
  - ⟐ `is_perdurant` — *ϕ is a perdurant if* `is_perdurant(`ϕ`)` *holds.*
- `is_entity` *is a* **'prerequisite prompt'** *for* `is_perdurant` ■

3.3. Endurants

Definition 4 . Parts:

- *By a* **'discrete endurant'** *we shall understand something which is*
  - *separate or distinct in form or concept,*
  - *consisting of distinct or separate parts.*

  We use the term **'part'** *for discrete endurants.*

Example 4 . Parts: *Example*

- *2 on slide 54 illustrated,*

and examples

- *7 on slide 67 and*
- *8 on slide 69 illustrate*

discrete endurants. ■

Definition 5 . Material:

- *By a* **'continuous endurant'** *we shall understand something which is*
  - *prolonged without interruption,*
  - *in an unbroken series or pattern* ■

  We use the term **'material'** *for continuous endurants* ■

Example 5 . Materials *Examples of material endurants are:*

- *air of an air conditioning system,*
- *grain of a silo,*
- *gravel of a barge,*
- *oil (or gas) of a pipeline,*
- *sewage of a waste disposal system, and*
- *water of a hydro-electric power plant.* ■

Endurant Analysis Prompt 4. is_discrete:

- *The domain analyser analyse endurants e into discrete entities as prompted by the* **'domain analysis prompt'**:
  - ⟐ `is_discrete` — *e is discrete if* `is_discrete(e)` *holds* ■

Endurant Analysis Prompt 5. is_continuous:

- *The domain analyser analyse endurants e into continuous entities as prompted by the* **'domain analysis prompt'**:
  - ⟐ `is_continuous` — *e is continuous if* `is_continuous(e)` *holds* ■

- We shall call
  - discrete endurants 'part's,
    i.e., $\texttt{is\_part}(e) \equiv \texttt{is\_discrete}(e)$ and
  - continuous endurants 'material's,
    i.e., $\texttt{is\_material}(e) \equiv \texttt{is\_continuous}(e)$
- Discrete endurants, i.e., parts, may contain continuous endurants, i.e., material.

Example 6 . Parts Containing Materials

- *Pipeline units, u, are here considered discrete, i.e., parts.*
- *Pipeline units serve to convey material, i.e., continuous endurants.* ■

- So which is the result of applying the `is_discrete` prompt to a pipeline unit u ?
- The answer is: it all depends !
- That is, it is the domain analyser who decides on
  - which one phenomenon is subordinated the other,
  - that is, whether the material is an aspect of a part,
  - or the part is subservient to the material !

3.4. Atomic and Composite Parts

Definition 6 . Discrete Endurant:

- *By a* ′**part**′*, to recall, we mean a discrete endurant* ■
- A distinguishing quality
  - of discrete endurants,
  - is whether they are atomic or composite.

Definition 7 . Atomic Part:

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

Example 7 . Atomic Parts *Examples of atomic parts of the above mentioned domains are:*

- *aircraft* *(of air traffic),*
- *demand/deposit accounts* *(of banks),*
- *containers* *(of container lines),*
- *documents* *(of document systems),*
- *hubs, links and vehicles* *(of road traffic),*
- *patients, medical staff and beds* *(of hospitals),*
- *pipes, valves and pumps* *(of pipeline systems), and*
- *rail units and locomotives* *(of railway systems).* ■

Definition 8 . Composite Part:

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

Example 8 . Composite Parts *Examples of atomic parts of the above mentioned domains are:*

- *airports and air lanes (of air traffic),*
- *banks (of a financial service industry),*
- *container vessels (of container lines),*
- *dossiers of documents (of document systems),*
- *routes (of road nets),*
- *medical wards (of hospitals),*
- *pipelines (of pipeline systems), and*
- *trains, rail lines and train stations (of railway systems).* ■

Endurant Analysis Prompt 8 . is_atomic:

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

Endurant Analysis Prompt 9 . is_composite:

- *The domain analyser analyses a discrete endurant, i.e., a part p into a composite endurant:*
  - ⟐ *is_composite(p): p is a composite endurant if is_composite(p) holds* ■

- is_discrete is a **'prerequisite prompt'** of both is_atomic and is_composite.

3.5. On Observing Part Sorts

3.5.1. Types and Sorts

- We use the term 'sort'
  - when we wish to speak of an abstract type,
  - that is, a type for which we have no model[11].
  - We shall use the term 'type' to cover both
    - abstract types and
    - concrete types.

[11] for example, in terms of the concrete types:

* sets,
* Cartesians,
* lists,
* maps,

or other.

3.5.2. On Discovering Part Sorts

- Recall from the section on *formal concept analysis* that we "equate" a formal concept with a type (i.e., a sort).
  - Thus, to us, a part sort is a set of all those entities
  - which all have exactly the same qualities.
- Our aim
  - is to present the basic principles that let
  - the domain analyser decide on part sorts.

- We observe parts (i.e., discrete endurants), one-by-one.
  - *(α) Our analysis of parts concludes when we have*
    - *"lifted" our examination of a particular part instance*
    - *to the conclusion that it is of a given sort,*
    - *that is, reflects, or is, a formal concept.*
- Thus there is, in this analysis, a "eureka",
  - a step where we shift focus
  - from the concrete to the abstract,
  - from observing specific part instances
  - to postulating a sort:
    - from one to the many.

Endurant Analysis Prompt 10 . observe_parts:

- *The* 'domain analysis prompt'*:*
  - `observe_parts`(p)
- *directs the domain analyser to observe the subparts of* p*;*
- *let us say they are:* $\{p_1,p_2,\ldots,p_m\}$ ■
- (β) The analyser analyses, for each of these parts, p_{i_k},
  - which formal concept, i.e., sort it belongs to;
  - let us say that it is of sort P_k;
  - thus the subparts of p are of sorts $\{P_1,P_2,\ldots,P_m\}$.
- Some P_k may be atomic sorts, some may be composite sorts.

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

    of the same, respective, part sorts.
  - *(γ) It is therefore concluded, that is, decided, that $\{p_i, p_j, p_\ell, \ldots, p_n\}$ are all of the same part sort P with observable part sub-sorts $\{P_1, P_2, \ldots, P_m\}$.*

- Above we have *type-font-highlighted* three sentences: (α, β, γ).
- When you analyse what they "prescribe" you will see that they entail a "depth-first search" for part sorts.
  - The β sentence says it rather directly:
  - *"The analyser analyses, for each of these parts, p_k, which formal concept, i.e., part sort it belongs to."*
  - To do this analysis in a proper way, the analyser must ("recursively") analyse the parts "down" to their atomicity,
  - and from the atomic parts decide on their part sort,
  - and work ("recurse") their way "back",
  - through possibly intermediate composite parts,
  - to the p_ks.

3.5.3. Part Sort Observer Functions

- The above analysis amounts to the analyser
  - first "applying" the domain analysis prompt
  - `is_composite`(p) to a discrete endurant,
  - where we now assume that the obtained truth value is **true**.
  - Let us assume that parts p:P consists of subparts of sorts $\{P_1,P_2,\ldots,P_m\}$.
  - Since we cannot automatically guarantee that our domain descriptions secure that
    - P and each P_i ([$1 \leq i \leq$ m])
    - denotes disjoint sets of entities

    we must prove it.

Domain Description Prompt 1

observe_part_sortsobserve-part-sorts

- *If* `is_composite`(p) *holds, then the analyser "applies" the description language observer prompt*

  ⟴ `observe_part_sorts`(p) [a.]

  resulting in the analyser writing down the part sorts and part sort observers *domain description text according to the followig schema:*

1. observe_part_sorts schema

Narration:

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

Formalisation:

type
[*s*] P,
[*s*] P_i [$1 \leq i \leq m$] **comment:** P_i [$1 \leq i \leq m$] *abbreviates* P_1, P_2, ..., P_m

value
[*o*] **obs_**P_i: $P \rightarrow P_i$ [$1 \leq i \leq m$]
[*i*] **is_**P_i: $P_i \rightarrow$ **Bool** [$1 \leq i \leq m$]

proof obligation [*Disjointness of part sorts*]
[*p*] $\forall\ p{:}(P_1|P_2|...|P_m)$ ·
[*p*] $\bigwedge \{\textbf{is\_}P_i(p) \equiv \bigvee \sim \{\textbf{is\_}P_j(p) \mid j \in \{1..m\} \setminus \{i\}\} \mid i \in \{1..m\}\}$

Example 9. Composite and Atomic Part Sorts of Transportation

- *The following example illustrates the multiple use of the* `observe_part_sorts` *function:*
  - *first to δ, a specific transport domain, Item 1,*
  - *then to an $n : N$, the net of that domain, Item 2, and*
  - *then to an $f : F$, the fleet of that domain, Item 3.*

1 A transportation domain is composed from a net, a fleet (of vehicles) and a monitor.

2 A transportation net is composed from a collection of hubs and a collection of links.

3 A fleet is a collection of vehicles.

- *The monitor is considered an atomic part.*

type
1. N, F, M
value
1. **obs**_N:$\Delta \to N$, **obs**_F:$\Delta \to F$, **obs**_M:$\Delta \to M$
type
2. HC, LC
value
2. **obs**_HC:$N \to HC$, **obs**_LC:$N \to LC$
type
3. VC
value
3. **obs**_VC:$F \to VC$

- *A proof obligation has to be discharged,*
  - *one that shows disjointness of sorts N, F and M.*
  - *An informal sketch is:*
    - *entities of sort N are composite and consists of two parts:*
    - *aggregations of hubs, HS, and aggregations of links, LS.*
    - *Entities of sort F consists of an aggregation, VS, of vehicles.*
    - *So already that makes N and F disjoint.*
    - *M is an atomic entity — where N and F are both composite.*
    - *Hence the three sorts N, F and M are disjoint.* ■

3.5.4. On Discovering Concrete Part Types

Endurant Analysis Prompt 11 . has_concrete_type:

- *The domain analyser*
  - *may decide that it is expedient, i.e., pragmatically sound,*
  - *to render a part sort, P, whether atomic or composite, as a concrete type, T.*
  - *That decision is prompted by the holding of the* **'domain analysis prompt'**:
    - `has_concrete_type`(p).
  - `is_discrete` *is a* **'prerequisite prompt'** *of* `has_concrete_type` ■

Domain Description Prompt 2. *observe_part_type*:

- *Then the domain analyser applies the* **'domain description prompt'**:
  - ⟴ *observe_part_type*[12] [b.]
- *to parts p:P which then yield the* **part type and part type observers** *domain description text according to the followig schema:*

[12] has_concrete_type is a **'prerequisite prompt'** of observe_part_type.

2. observe_part_type schema

Narration:

[t_1] ... *narrative text on types* ...
[t_2] ... *narrative text on types* ...
[o] ... *narrative text on type observers* ...

Formalisation:

type
[t_1] Q, R, ..., S
[t_2] $T = \mathcal{E}(Q,R,...,S)$
value
[o] **obs_**T: $P \rightarrow T$

- *where* Q, R, ..., S *may be any types, including part sorts*
- *and* $\mathcal{E}(Q,R,...,S)$ *is a type expression* ■

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

Example 10 . Concrete Part Types of Transportation

We continue Example 9 on slide 80:

4 A collection of hubs is a set of hubs and a collection of links is a set of links.

5 Hubs and links are, until further analysis, part sorts.

6 A collection of vehicles is a set of vehicles.

7 Vehicles are, until further analysis, part sorts.

type

4. $Hs = H$-**set**, $Ls = L$-**set**

5. H, L

6. $Vs = V$-**set**

7. V

value

4. **obs**_$Hs{:}HC \to H$-**set**, **obs**_$Ls{:}LC \to L$-**set**

6. **obs**_$Vs{:}VC \to V$-**set** ■

3.5.5. Forms of Part Types

- Usually it is wise to restrict the part type definitions, $\mathsf{T}_i = \mathcal{E}_i(\mathsf{Q},\mathsf{R},...,\mathsf{S})$, to simple type expressions.
  - T=A-**set** or
  - T=A* or
  - T=ID $\xrightarrow[m]{}$A or
  - T=$\mathsf{A}_t|\mathsf{B}_t|...|\mathsf{C}_t$

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

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

3.5.6. Part Sort and Type Derivation Chains

- Let P be a composite sort.
- Let P_1, P_2, ..., P_m be the part sorts "discovered" by means of `observe_part_sorts(p)` where p:P.
- We say that P_1, P_2, ..., P_m are (immediately) **'derived'** from P.
- If P_k is derived from P_j and P_j is derived from P_i, then, by transitivity, P_k is **'derived'** from P_i.

3.5.6.1 No Recursive Derivations

- We "mandate" that
  - if P_k is derived from P_j
  - then there
    - can be no P derived from P_j
    - such that P is P_j,
    - that is, P_j cannot be derived from P_j.
- That is, we do not allow recursive domain sorts.
- It is not a question, actually of allowing recursive domain sorts.
  - It is, we claim to have observed,
  - in very many domain modelling experiments,
  - that there are no recursive domain sorts!

3.5.7. Names of Part Sorts and Types

- The domain analysis and domain description text prompts
  - observe_part_sorts,
  - observe_material_sorts and
  - observe_part_type

— as well as the

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

prompts introduced below — "yield" type names.

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

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

Example 11 . Container Line Sorts

- *Our example is that of a container line*
  - *with container vessels and*
  - *container terminal ports.*

8 A container line contains a number of container vessels and a number of container terminal ports, as well as other components.

9 A container vessel contains a container stowage area, etc.

10 A container terminal port contains a container stowage area, etc.

11 A container stowage ares contains a set of uniquely identified container bays.

12 A container bay contains a set of uniquely identified container rows.

13 A container row contains a set of uniquely identified container stacks.

14 A container stack contains a stack, i.e., a first-in, last-out sequence of containers.

15 Containers are further undefined.

- *After a some slight editing we get:*

```
type
  CL
  VS, VI, V, Vs = VI →m V,
  PS, PI, P, Ps = PI →m P
value
  obs_VS: CL → VS
  obs_Vs: VS → Vs
  obs_PS: CL → PS
  obs_Ps: CTPS → CTPs
type
  CSA
value
  obs_CSA: V → CSA
  obs_CSA: P → CSA
```

```
type
  BAYS, BI, BAY, Bays=BI →m BAY
  ROWS, RI, ROW, Rows=RI →m ROW
  STKS, SI, STK, Stks=SI →m STK
  C
value
  obs_BAYS: CSA → BAYS,
  obs_Bays: BAYS → Bays
  obs_ROWS: BAY → ROWS,
  obs_Rows: ROWS → Rows
  obs_STKS: ROW → STKS,
  obs_Stks: STKS → Stks
  obs_Stk: STK → C*
```

- *Note that* `observe_part_sorts(v:V)` *and* `observe_part_sorts(p:P)` *both yield* CSA ■

3.5.8. More On Part Sorts and Types

- The above "experimental example" motivates the below.
  - We can always assume that composite parts $p{:}P$ abstractly consists of a definite number of subparts.
    - **Example 12**. We comment on Example 9 on slide 80: parts of type Δ and N are composed from three, respectively two abstract subparts of distinct types ■
  - Some of the parts, say p_{i_z} of $\{p_{i_1},p_{i_2},\ldots,p_{i_m}\}$, of $p{:}P$, may themselves be composite.
    - **Example 13**. We comment on Example 9 on slide 80: parts of type N, F, HC, LC and VC are all composite ■

- There are, pragmatically speaking, two cases for such compositionality.
  - Either the part, p_{i_z}, of type t_{i_z}, is is composed from a definite number of abstract or concrete subparts of distinct types.
    - **Example 14**. We comment on Example 9 on slide 80: parts of type N are composed from three subparts ■
  - Or it is composed from an indefinite number of subparts of the same sort.
    - **Example 15**. We comment on Example 9 on slide 80: parts of type HC, LC and VC are composed from an indefinite numbers of hubs, links and vehicles, respectively ■

Example 16 . Pipeline Parts

16 A pipeline consists of an indefinite number of pipeline units.

17 A pipeline units is either a well, or a pipe, or a pump, or a valve, or a fork, or a join, or a sink.

18 All these unit sorts are atomic and disjoint.

type
16. *PL, U, We, Pi, Pu, Va, Fo, Jo, Si*
16. *Well, Pipe, Pump, Valv, Fork, Join, Sink*
value
16. **obs**_*Us: PL* → *U*-**set**
type
17. *U* == *We* | *Pi* | *Pu* | *Va* | *Fo* | *Jo* | *Si*
18. *We::Well, Pi::Pipe, Pu::Pump, Va::Valv, Fo:Fork, Jo::Join, Si::Sink* ■

3.5.8.1 Derivation Lattices

- Derivation chains
  - start with the domain name, say Δ, and
  - (definitively) end with the name of an atomic sort.
- Sets of derivation chains form join lattices [Birk67].

Example 17 . Derivation Chains

- *Figure 1 on the following slide illustrates*
  - *two part sort and type derivation chains.*
  - *based on Examples 9 on slide 80 and 11 on slide 92, respectively.*

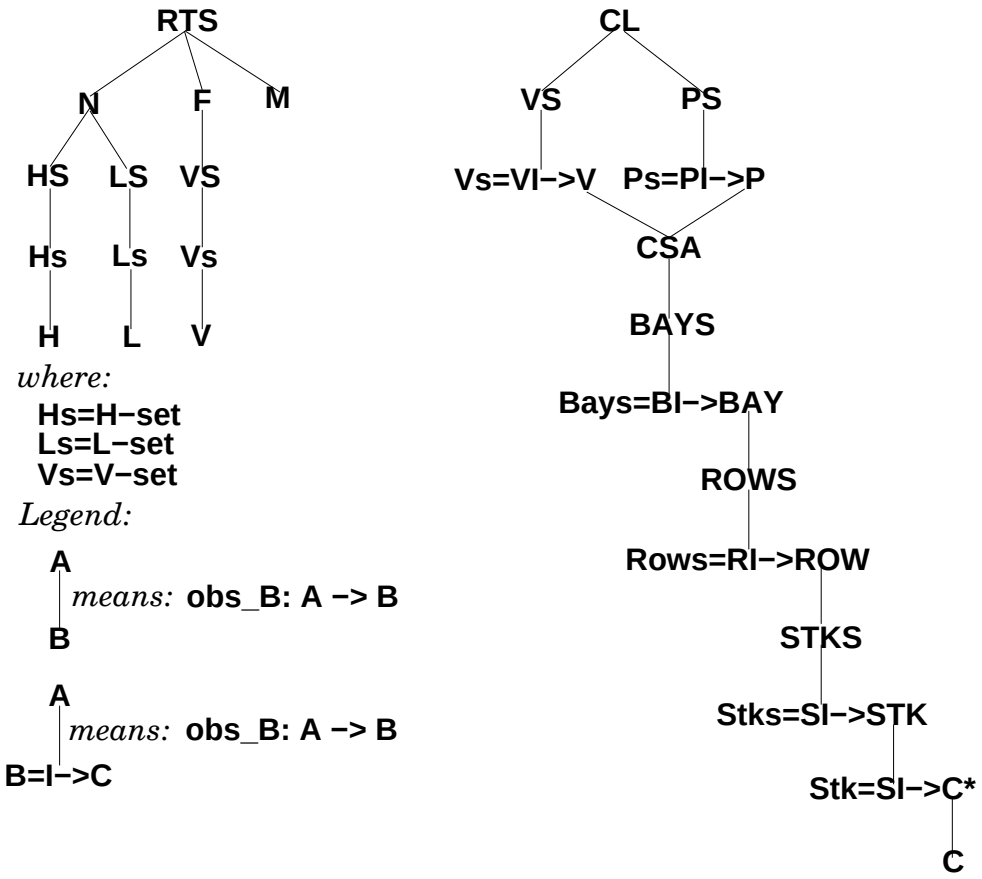

Figure 1: Two Domain Lattices: Examples 9 on slide 80 and 11 on slide 92

- *The "–>" of Fig. 1 stands for* $\overrightarrow{m}$ ■

3.6. Syntactic and Semantic Qualities of Endurants

- By an 'external endurant quality' we shall understand the
  - ◈ `is atomic`,
  - ◈ `is composite`,
  - ◈ `is discrete` and
  - ◈ `is continuous`

  qualities.
- We consider these to reflect 'syntactic qualities'.
- By an 'internal endurant quality' we shall understand the endurant qualities to be outlined in the next sections:
  - ◈ `unique ids`,
  - ◈ `mereology` and
  - ◈ `attributes`.
- By 'part qualities' we mean the sum total of
  - ◈ `external endurant` and
  - ◈ `internal endurant`

  qualities.
- We consider these to reflect 'semantic qualities'.

3.7. Three Categories of Semantic Qualities

- We suggest that the internal qualities of parts be analysed into three categories:
  - (i) a category of unique part identifiers,
  - (ii) a category of mereological quantities and
  - (iii) a category of general attributes.

- Part mereologies are about **sharing** qualities between parts.
  - Some such **sharing** expresses spatio-topological properties of how parts are organised.
  - Other part **sharing** aspects express relations (like equality) of part attributes.
  - We base our modelling of mereologies on the notion of unique part identifiers.
  - Hence we cover semantic qualities in the order (i–ii–iii).

3.8. Unique Part Identifiers

- We can assume, without any loss of generality,
  - that all parts, p, of any domain P, have unique identifiers,
  - that unique identifiers (of parts p:P) are abstract values (of the unique identifier sort PI of P),
  - such that distinct part sorts, P_i and P_j, have distinctly named unique identifier sorts, say PI_i and PI_j, and
  - that all π_i:PI_i and π_j:PI_j are distinct, and
  - that the observer function **uid**_P applied to p yields the unique identifier, say π:PI, of p.

Domain Description Prompt 3 . *observe_unique_identifier*:

- *We can therefore apply the* 'domain description prompt'*:*

  ◈ *observe_unique_identifier* [c.]

- *to parts* p:P *resulting in the analyser writing down the* unique identifier types and observers *domain description text according to the followig schema:*

3. observe_unique_identifier schema

Narration:

[s] *... narrative text on unique identifier sorts ...*
[u] *... narrative text on unique identifier observers ...*
[a] *... axiom on uniqueness of unique identifiers ...*

Formalisation:

type
[s] PI
value
[u] **uid**_P: $P \rightarrow PI$
axiom
[a] $\mathcal{U}$

- $\mathcal{U}$ *is a predicate over part sorts and unique part identifier sorts.*
- *The unique part identifier sort,* PI*, is unique, as are all part sort names,* P*.*

Example 18 . Unique Transportation Net Part Identifiers

We continue Example 9 on slide 80.

19 Links and hubs have unique identifiers

20 and unique identifier observers.

type

19. *LI, HI*

value

20. **uid**_*LI*: $L \to LI$

20. **uid**_*HI*: $H \to HI$

axiom [*Well–formedness of Links, L, and Hubs, H*]

19. $\forall\ l,l':L \cdot l \neq l' \Rightarrow$ **uid**_$LI(l) \neq$ **uid**_$LI(l')$,

19. $\forall\ h,h':H \cdot h \neq h' \Rightarrow$ **uid**_$HI(h) \neq$ **uid**_$HI(h')$ ■

3.9. 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 [Lesniewski1,Lesniewski3,Lesniewski5].

3.9.1. Part Relations

- Which are the relations that can be relevant for part-hood ?
- We give some examples.
  - Two otherwise distinct parts may share attribute values.

Example 19 . Shared Attribute Mereology: *Examples:*

- *(i) two or more distinct public transport busses may run according to the same, thus "shared", bus time table;*
- *(ii) all vehicles in a traffic participate in that traffic each with their "share": that is, position on links or at hubs – as observed by the (thus postulated) traffic observer.*

etcetera. ■

- Two otherwise distinct parts may be said to, for example, be topologically "adjacent" or one "embedded" within the other.

Example 20 . Topological Connectedness Mereology:
Examples:

- *(i) two rail units may be connected (i.e., adjacent),*
- *(ii) a road link may be connected to two road hubs;*
- *(iii) a road hub may be connected to zero or more road links;*

etcetera. ■

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

3.9.2. Part Mereology: Types and Functions

- If a composite part p
  - consists of a definite number of sub-parts: $a, ..., c$,
  - then that is the mereology of any p with respect to any $a, ..., c$,
  - and we need not bother about unique identifiers for p.
- In this case we have the situation that:

type
  P, A, ..., C
value
  obs_A: P → A, ..., **obs**_C: P → C.

- If, however, a composite part p
  - ⋈ consists of an indefinite number of sub-parts, $q_1, ..., q_n$,
  - ⋈ that is, if we have the situation that:

```
type
    P, Q, Ps = Q-set
value
    obs_Ps: P → Q-set,
```

  - ⋈ then the mereology need to be further elaborated.

Endurant Analysis Prompt 12 . has_mereology:

- *To do so the analyser can be said to endow a truth value* **true** *to the* '**domain analysis prompt**'*:*
  - ⋈ *has_mereology*

- When the domain analyser decides that
  - some parts are related in a specifically enunciated mereology,
  - then the analyser has to decide on suitable
    - mereology types and
    - mereology (i.e., part relation) observers.
- We can define a 'mereology type' as a type $\mathcal{E}$xpression over unique [part] identifier types.
  - We generalise to unique [part] identifier over a definite collection of part sorts, P1, P2, ..., Pn,
  - where the parts p1:P1, p2:P2, ..., pn:Pn are not necessarily (immediate) sub-parts of some part p:P.

type
  PI1, PI2, ..., PIn
  MT = $\mathcal{E}$(PI1, PI2, ..., PIn),

Domain Description Prompt 4. `observe_mereology`:

- *If* `has_mereology`(p) *holds for parts* p *of type* P,
  - *then the analyser can apply the* **'domain description prompt'**:
    - `observe_mereology` [d.]
  - *to parts of that type*
  - *and write down the* **mereology types and observers** *domain description text according to the followig schema:*

4. observe_mereology schema

Narration:

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

Formalisation:

type
[t] $MT = \mathcal{E}(PI1,PI2,...,PIm)$
value
[m] **mereo**_P: $P \rightarrow MT$
axiom [*Well–formedness of Domain Mereologies*]
[a] $\mathcal{A}(MT)$

- *Here* $\mathcal{E}$*(PI1,PI2,...,PIm) is a type expression over possibly all unique identifier types of the domain description,*
- *and* $\mathcal{A}$*(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.*
- `has_mereology` *is a* **'prerequisite prompt'** *for* `observe_mereology` ■

Example 21 . Road Net Part Mereologies *We continue Example 9 on slide 80 and Example 18 on slide 106.*

21 Links are connected to exactly two distinct hubs.

22 Hubs are connected to zero or more links.

23 For a given net the link and hub identifiers of the mereology of hubs and links must be those of links and hubs, respectively, of the net.

type
21. $LM = HI\text{-}\mathbf{set}$, $LM = \{|his{:}HI\text{-}\mathbf{set} \cdot \mathbf{card}(his){=}2|\}$
22. $HM = LI\text{-}\mathbf{set}$

value
21. **mereo**$\_L{:}\ L \to LM$
22. **mereo**$\_H{:}\ H \to HM$

axiom [*Well–formedness of Road Nets, N*]
23. $\forall\ n{:}N,l{:}L,h{:}H \cdot\ l \in$ **obs**$\_Ls($**obs**$\_LC(n)) \wedge h \in$ **obs**$\_Hs($**obs**$\_GC(n))$
23. **let** $his{=}mereology\_H(l)$, $lis{=}mereology\_H(h)$ **in**
23. $his \subseteq \cup\{$**uid**$\_H(h) \mid h \in$ **obs**$\_Hs($**obs**$\_HC(n))\}$
23. $\wedge\ lis \subseteq \cup\{$**uid**$\_H(l) \mid l \in$ **obs**$\_Ls($**obs**$\_LC(n))\}$ **end** ■

Example 22 . Pipeline Parts Mereology

- *We continue Example 16 on slide 97.*
- *Pipeline units serve to conduct fluid or gaseous material.*
- *The flow of these occur in only one direction: from so-called input to so-called output.*

24 Wells have exactly one connection to an output unit.

25 Pipes, pumps and valves have exactly one connection from an input unit and one connection to an output unit.

26 Forks have exactly one connection from an input unit and exactly two connections to distinct output units.

27 Joins have exactly one two connection from distinct input units and one connection to an output unit.

28 Sinks have exactly one connection from an input unit.

29 Thus we model the mereology of a pipeline unit as a pair of disjoint sets of unique pipeline unit identifiers.

```
type
29.  UM′=(UI-set×UI-set)
29.  UM={|(iuis,ouis):UI-set×UI-set·iuis ∩ ouis={}|}
value
29.  mereo_U: UM
axiom [ Well–formedness of Pipeline Systems, PLS (0)]
   ∀ pl:PL,u:U · u ∈ obs_Us(pl) ⇒
      let (iuis,ouis)=mereo_U(u) in
      case (card iuis,card ouis) of
24.          (0,1) → is_We(u),
25.          (1,1) → is_Pi(u)∨is_Pu(u)∨is_Va(u),
26.          (1,2) → is_Fo(u),
27.          (2,1) → is_Jo(u),
28.          (1,0) → is_Si(u)
      end end ■
```

3.9.3. Update of Mereologies

- We normally consider a part's mereology to be constant.
- There may, however, be cases where the mereology of a part changes.
- In order to update mereology values the description language offers the "built-in" operator:

Mereology Update Function: **upd_mereology**

⟴ **upd_mereology**: P → M → P

for all relevant M and P.

- The meaning of **upd_mereology** is, informally:

type
  P, M
value
  upd_mereology: P → M → P
  upd_mereology(p)(m) **as** p′
    post: **mereo_**(p′) = m

- The above is a simplification.
  - It lacks explaining that all other aspects of the part p:P are left unchanged.
  - It also omits mentioning some proof obligations.
    - The updated mereology must, for example,
    - only specify such unique identifiers of parts
    - that are indeed existing parts.
  - A proper formal explication requires
  - that we set up a formal model of the
  - domain/method/analyser/description quadrangle.

Example 23 . Mereology Update

- *The example is that of updating the mereology of a hub.*
- *Cf. Example 21 on slide 116.*

30 Inserting a link, l:L, between two hubs, ha:H,hb:H require the update of the mereologies of these two existing hubs.

*31 The unique identifier of the inserted link, l:L, is li, li=***uid**_*L(l) and h is either ha or hb;*

32 li is joined to the mereology of either ha or hb; and respective hubs are updated accordingly.

value

30. *update_hub_mereology:* $H \to LI \to H$

31. *update_hub_mereology(h)(li)* $\equiv$

32. **let** $m = \{li\} \cup$ **mereo**_(h) **in upd_mereology**$(h)(m)$ **end** ■

3.10. Discrete Endurant Attributes
3.10.1. Inseparability of Attributes from Endurants

- Endurants are
  - typically recognised because of their spatial form (parts or materials)
  - and are otherwise characterised by their intangible, but measurable attributes.
- We learned from our exposition of *formal concept analysis* that
  - a formal concept, that is, a type, consists of all the entities
  - which all have the same qualities.
- Thus removing a quality from an entity makes no sense:
  - the entity of that type
  - either becomes an entity of another type
  - or ceases to exist (i.e., becomes a non-entity)!

3.10.2. Attribute Quality and Attribute Value

- We distinguish between
  - an attribute, as a logical proposition, and
  - an attribute value, as a value in some value space.

Example 24 . Attribute Propositions and Other Values

- *A particular street segment (i.e., a* **link***), say* ℓ*,*
  - *satisfies the proposition (attribute)* **has_length***, and*
  - *may then have value* **length 90 meter** *for that attribute.*
- *A particular road transport domain,* δ*,*
  - *has three immediate sub-parts: net,* n*, fleet,* f*, and monitor* m*;*
  - *typically nets* `has_net_name` *and* `has_net_owner` *proposition attributes*
  - *with, for example,* `US Interstate Highway System` *respectively* `US Department of Transportation` *as values for those attributes* ■

3.10.3. Endurant Attributes: Types and Functions

- Let us recall that attributes cover qualities other than unique identifiers and mereology.
- Let us then consider that parts have one or more attributes.
  - These attributes are qualities
  - which help characterise "what it means" to be a part,
  - that is, a discrete endurant.

Example 25 . Atomic Part Attributes

- *Examples of attributes of atomic parts such as a human are:*
  - *name,*
  - *gender,*
  - *birth-date,*
  - *birth-place,*
  - *nationality,*
  - *height,*
  - *weight,*
  - *eye colour,*
  - *hair colour,*

  etc.
- *Examples of attributes of transport net links are:*
  - *length,*
  - *location,*
  - *1 or 2-way link,*
  - *link condition,*

  etc. ■

Example 26 . Composite Part Attributes

- *Examples of attributes of composite parts such as a road net are:*
  - *owner,*
  - *public or private net,*
  - *free-way or toll road,*
  - *a map of the net,*

  etc.
- Examples of attributes of a group of people could be: *statistic distributions of*
  - *gender,*
  - *age,*
  - *income,*
  - *education,*
  - *nationality,*
  - *religion,*

  etc. ■

- We now assume that all parts (and materials, see further on), have attributes.
- The question is now, in general, how many and, particularly, which.

Endurant Analysis Prompt 13 . attribute_names:

- *The* **'domain analysis prompt'** `attribute_names`
  - *when applied to a part p*
  - *yields the set of names of its attribute types:*
  - `attribute_names`(p): $\{\eta A_1, \eta A_2, ..., \eta A_n\}$.
- η *is a type operator. Applied to a type A it yields is name*[13] ■

[13] Normally, in non-formula texts, type A is referred to by ηA. In formulas A denote a type, that is, a set of entities. Hence, when we wish to emphasize that we speak of the name of that type we use ηA. But often we omit the distinction

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

  Therefore we must prove it.

3.10.3.1 The Attribute Value Observer

- The "built-in" description language operator
  - ⟐ **attr**_A
- applies to parts, p:P, where ηA∈attribute_names(p).
- It yields the value of attribute A of p.

3.10.3.2 A Comprehensive Attributes Observer

- The "built-in", part sort independent description language operator
  - **obs_attribs**
- when applied, **obs_attribs**(e), to an endurant entity (part or material) e of type P (or M)
  - yields the set, attrs:P_ATTRS, of attributes of that part such that
  - if A_1, A_2, ..., A_n are the types of the n attributes of e
  - then for all i ($1 \leq i \leq n$)
  - **attr**_$\mathsf{A}_i(e)$ = **attr**_A_i(**obs_attribs**(e)).

Domain Description Prompt 5 . `observe_attributes`:

- *The domain analyser experiments, thinks and reflects about part attributes.*
- *That process is initated by the* **'domain description prompt'**:
  - ⟴ `observe_attributes`. [e.]
- *The result of that* **'domain description prompt'** *writes down the* **attribute types and observers** *domain description text according to the followig schema:*

e. observe_attributes schema

Narration:

[t] ... *narrative text on attribute sorts* ...
[o] ... *narrative text on attribute sort observers* ...
[i] ... *narrative text on attribute sort recognisers* ...
[p] ... *narrative text on attribute sort proof obligations* ...

Formalisation:

type
[t] A_i [$1 \leq i \leq n$]
[t] P_ATTRS
value
[o] **obs_attribs**$:P \to P\_ATTRS$
[o] **attr**$\_A_i:(P|P\_ATTRS) \to A_i$ [$1 \leq i \leq n$]
[i] **is**$\_A_i:A_i \to$ **Bool** [$1 \leq i \leq n$]
axiom $\forall\ i$:**Nat** $\cdot\ 1 \leq i \leq n \Rightarrow$ **attr**$\_A_i(p) =$ **attr**$\_A_i($**obs_attribs**$(p))$
proof obligation [*Disjointness of Attribute Types*]
[p] $\forall\ \delta:\Delta$
[p] **let** *P be any part sort* **in** *[the* Δ *domain description]*
[p] **let** $a:(A_1|A_2|...|A_n)$ **in** **is**$\_A_i(a) \neq$ **is**$\_A_j(a)$ **end end** [$i \neq j,\ 1 \leq i,j \leq n$]

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

  are then "automatically" given:
  - *if a part (type* P*) has an attribute* A_i
  - *then there is postulated, "by definition" [eureka] an attribute observer function* **attr**_A_i:$(P|P\_ATTRS)\rightarrow A_i$ *etcetera* ■

- The fact that, for example, A_1, A_2, ..., A_n are attributes of p:P, means that the propositions
  - $\texttt{has\_attribute\_A}_1(p)$,
    $\texttt{has\_attribute\_A}_2(p)$,
    ..., and
    $\texttt{has\_attribute\_A}_n(p)$

  holds.
- Thus the observer functions **attr**_A_1, **attr**_A_2, ..., **attr**_A_n
  - can be applied to ps in P
  - and yield attribute values a_1:A_1, a_2:A_2, ..., a_n:A_n respectively.

Example 27 . Road Hub Attributes *After some analysis a domain analyser may arrive at some interesting hub attributes:*

33 hub state: from which links (by reference) can one reach which links (by reference),

34 hub state space: the set of all potential hub states that a hub may attain,

35 such that

- *a. the links referred to in the state are links of the hub mereology*
- *b. and the state is in the state space.*

36 Etcetera — i.e., there are other attributes not mentioned here.

type
33. $H\Sigma = (LI \times LI)set$
34. $H\Omega = H\Sigma$**-set**
value
33. **attr**_$H\Sigma$:$H \to H\Sigma$
34. **attr**_$H\Omega$:$H \to H\Omega$
axiom [*Well–formedness of Hub States,* $H\Sigma$]
35. $\forall\ h{:}H \cdot$ **let** $lis =$ **mereo**_$H(h)$ **in**
35. **let** $h\sigma =$ **attr**_$H\Sigma(h)$ **in**
35a.. $\{li,li'|li,li'{:}LI \cdot (li,li') \in h\sigma\} \subseteq lis$
35b.. $\wedge\ h\sigma \in$ **attr**_$H\Omega(h)$
35. **end end**
type
36. ..., ...
value
36. **attr**_..., ... ■

3.10.4. Attribute Categories

- One can suggest a hierarchy of part attribute categories:
  - discrete or continuous (including chaotic) values,
  - static or dynamic values (and within the dynamic value category:
    - inert values or
    - reactive values or
    - active values (and within the dynamic active value category:
      - autonomous values or
      - biddable values or
      - programmable values)).
  - We now review [M.A. Jackson] these attribute value types.

Part attributes are either **discrete** or **continuous** or **chaotic** attributes.

- A part attribute is said to be a **'discrete attribute'**,
  - `is_discrete_attribute`,
  - if it takes on a finite or countably infinite number
  - of distinct or unconnected elements.
- A part attribute is said to be a **'continuous attribute'**,
  - `is_continuous_attribute`,
  - if a suitable abstract model
  - describes it as a continuous function
    - from some point set value type A (A could be time)
    - to some not necessarily point set value type (B): $A \rightarrow B$.
- We shall not explain concepts of chaotic attributes.

Discrete or continuous part attributes are either constant or a variable, i.e., **static** or **dynamic** attributes.

- By a **'static attribute'**, `is_static_attribute`, we shall understand an attribute whose values
  - are constants,
  - i.e., cannot change.
- By a **'dynamic attribute'**, `is_dynamic_attribute`, we shall understand an attribute whose values
  - are variable,
  - i.e., can change.

Dynamic attributes are either inert, reactive or active attributes.

- By an **'inert attribute'**, `is_inert_attribute`, we shall understand a dynamic attribute whose values
  - only change as the result of external stimuli where
  - these stimuli prescribe exactly these new values.
- By a **'reactive attribute'**, `is_reactive_attribute`, we shall understand a dynamic attribute whose values,
  - if they vary, change value in response to
  - the change of other attribute values.
- By an **'active attribute'**, `is_active_attribute`, we shall understand a dynamic attribute whose values
  - change (also) of its own volition.

Example 28 . Inert and Reactive Attributes

- *Busses (i.e., vehicles) have a* timetable *attribute which is dynamic, i.e., can change, namely when the operator of the bus decides so, thus the bus timetable attribute is inert.*
- *Pipeline valve units include the two attributes of* valve opening *(*`open, close`*) and* internal flow *(measured, say* `gallons` *per* `second`*).*
  - *The valve opening attribute is of the programmable attribute category.*
  - *The flow attribute is reactive (flow changes with valve opening/closing)* ■

Active attributes are either autonomous, biddable or programmable attributes.

- By an **'autonomous attribute'**, `is_autonomous_attribute`, we shall understand a dynamic active attribute
  - whose values change value only "on their own volition".[14]
- By a **'biddable attribute'**, `is_biddable_attribute` (of a part) we shall understand a dynamic active attribute whose values
  - may be subject to a contract
  - as to which values it is expected to exhibit.
- By a **'programmable attribute'**, `is_programmable_attribute`. we shall understand a dynamic active attribute whose values
  - can be accurately prescribed.

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

Example 29 . Static and Dynamic Link Attributes

37 Some link attributes

a. length, *b. name,*

can be considered static,

38 whereas other link attributes

a. state, *b. state space*

can be considered programmable,

39 Finally link attributes

a. link state–of–repair, *b. date last maintained,*

can be considered inert.

```
type
37a.. LEN
value
37a.. obs_LEN: L → LEN
type
37b.. Name
value
37b.. obs_Name: L → Name
type
38a.. LΣ′=(HI×HI)-set
38a.. LΣ={|lσ:LΣ • card lσ ≤ 2|}
value
38a.. obs_LΣ: L → LΣ
type
38b.. LΩ′=LΣ-set
38b.. LΩ={|lω:LΩ • card lω = 1|}
value
38b.. obs_LΩ: L → LΩ
type
39a.. LSoR
39b.. DLM
value
39a.. obs_LSoR: L → LSoR
39b.. obs_DLM: L → DLM
```

Example 30 . Autonomous and Programmable Hub Attributes *We continue Example 29.*

- *Time progresses autonomously,*
- *Hub states are programmed (traffic signals):*
  - *changing*
    - *from red to green via yellow,*
    - *in one pair of (co-linear) directions,*
  - *while changing, in the same time interval,*
    - *from green via yellow to red*
    - *in the "perpendicular" directions* ■

3.10.5. Shared Attributes

- Normally part attributes of different part sorts are distinctly named.
- If, however, `observe_attributes`(p_{ik}:P_i) and `observe_attributes`($\mathsf{p}_{j\ell}$:P_j),
  - for any two distinct part sorts, P_i and P_j, of a domain,
  - "discovers" identically named attributes, say A,
  - then we say that parts p_i:P_i and p_j:P_j **'share'** attribute A.
  - that is, that a:**attr**_A(p_i) (and a′:**attr**_A(p_j)) is a **'shared attribute'**
  - (with a=a′ always (□) holding).

Example 31. Shared Attributes. Examples of shared attributes:

- Bus **timetable attributes** have the same value as the regional transport system **timetable attribute**.
- Bus **clock attributes** have the same value as the regional transport system **clock attribute**.
- Bus **owner attributes** have the same value as the regional transport system **owner attribute**.
- Bank customer **passbooks** record bank transactions on, for example, demand/deposit accounts share values with the bank general ledger **passbook** entries.
- A link incident upon or emanating from a hub shares the **connection** between that link and the hub as an attribute.
- Two pipeline units[15], p_i, p_j, that are **connected**, such that an outlet π_j of p_i "feeds into" an inlet π_i of p_j, are said to share the connection (modelled by, e.g., $\{(\pi_i, \pi_j)\}$.

[15] See upcoming Example 22 on slide 118

Example 32 . Shared Timetables

- *The fleet and vehicles of Example 9 on slide 80 and Example 10 on slide 87 is that of a bus company.*

40 From the fleet and from the vehicles we observe unique identifiers.

41 Every bus mereology records the same one unique fleet identifier.

42 The fleet mereology records the set of all unique bus identifiers.

43 A bus timetable is a share fleet and bus attribute.

type

40. FI, VI, BT

value

40. **uid**_F: $F \to FI$

40. **uid**_V: $V \to VI$

41. **mereo**_F: $F \to VI$-**set**

42. **mereo**_V: $V \to FI$

43. **attr**_BT: $(F|V) \to BT$

axiom

$\square \forall\ f{:}F \Rightarrow$

$\forall\ v{:}V \cdot v \in$ **obs**_$Vs($**obs**_$VC(f)) \cdot$ **attr**_$BT(f) =$ **attr**_$BT(v)$

[which is the same as]

$\square \forall\ f{:}F \Rightarrow$

$\{$**attr**_$BT(f)\} = \{$**attr**_$BT(v){:}v{:}V \cdot v \in$ **obs**_$Vs($**obs**_$VC(f))\}$ ■

- Part attributes of one sort, P_i, may be simple type expressions such as
  - **A-set**,
  - where A may be an attribute of some other part sort, P_j,
  - in which case we say that part attributes
    - **A-set** and
    - A

    are shared.

Example 33 . Shared Passbooks

44 A banking system contains

- *an administration and*
- *a set of customers.*

45 The administration contains a general ledger.

46 An attribute of a general ledger is a set of passbooks.

47 An attribute of a customer is that of a passbook.

48 Passbooks are uniquely identified by unique customer identifiers.

type

44. [*parts*] $BS, AD, GL, CS, Cs = C$**-set**

47. [*attributes*] PB

value

44. **obs**_AD: $BS \to AD$

45. **obs**_GL: $AD \to GL$

46. **attr**_PBs: $GL \to PB$**-set**

44. **obs**_CS: $BS \to CS$

44. **obs**_Cs: $BS \to Cs$

47. **attr**_PB: $C \to PB$

48. **uid**_PB: $PB \to PBI$

axiom

$\square \forall\ bs{:}BS \cdot$

attr_PBs(**attr**_GL(**obs**_AD(bs)))

$= \{$**attr**_PB(c)$|\ c{:}C \cdot c \in$ **obs**_Cs(**obs**_CS(bs))$\}$ ∎

3.10.6. Update of Dynamic Attributes

- In order to update values of dynamic attributes the description language offers the "built-in" operator:

Attribute Update Function: **upd_attr**

⟴ **upd_attr**: P × A → P

for all relevant A and P.

- The meaning of **upd_attr** is, informally:

type
  P, A
value
  upd_attr: P × A → P
  upd_attr(p)(a) **as** p′
    pre: ηA ∈ `attribute_names`(p)
    post: **attr**_A(p′) = a ∧ no other qualities of P other than A are affected

- As for the mereology update operation (Slide 122) the above is a simplification.
  - We shall presently leave it with the above!
  - Remarks similar to those given for **upd_mereology** apply (Slide 122).
- We shall presently leave it with the above!

Example 34 . Hub State Update *We continue Example 27 on slide 137.*

- *Hub states can be considered abstractions of road intersection signal values* **red**/**yellow**/**green**.

49 We consider **set_hub_state** *a primitive operation*

a. which takes a hub, h, and a hub state, hσ, as arguments and "delivers the same" hub, h′,

b. with unchanged hub state space (etcetera)

c. but with hσ being the new value of the HΣ attribute of h.

value

49. set_hub_state:
49a.. H → HΣ → H
49. set_hub_state(h)(hσ) **as** *h′*
49a.. pre: hσ ∈ **attr**_*HΩ(h)*
49b.. post: hσ ∈ **attr**_*HΩ(h′)*
49c.. ∧ h′ = upd_HΣ_of_H(h)(hσ) ■

3.11. Materials: Continuous Endurants

- Continuous endurants (i.e., 'material's) are entities, m, which satisfy:
  - $\texttt{is\_material}(m) \equiv \texttt{is\_endurant}(m) \wedge \texttt{is\_continuous}(m)$

Example 35 . Parts and Materials

- *We observe materials as components of parts:*
  - *Thus liquid or gaseous materials are observed in pipeline units.*
- *We can also observe parts immersed in materials:*
  - *container vessels "floats" on the oceans;*
  - *aircrafts flies in the air!* ■
- We shall in this seminar not cover the case of parts being immersed in materials.

- We now complement the `observe_part_sorts` (of earlier).
- We assume, without loss of generality, that only atomic parts may contain materials.
- Let $p{:}P$ be some atomic part.

Endurant Analysis Prompt 14 . has_materials:

- *The* **'domain analysis prompt'**:
  - ⟐ *has_materials*(p)
- *yields* **true** *if a component* m *of the atomic* $p{:}P$ *satisfies* *is_material*(m), *otherwise false* ■

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

  we must prove it.

Domain Description Prompt 6 . `observe_material_sorts`:

- *The* '**domain description prompt**'*:*
  - `observe_material_sorts`(e) [f.]

  yields the material sorts and material sort observers *domain description text according to the followig schema:*

6. observe_material_sorts schema

Narration:

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

Formalisation:

type
[s] M_i [$1 \leq i \leq n$]
value
[o] **obs**_M_i: $P \rightarrow M_i$ [$1 \leq i \leq n$]
[i] **is**_M_i: $M \rightarrow$ **Bool** [$1 \leq i \leq n$]
proof obligation [*Disjointness of Material Sorts*]
[p] $\forall\ m_i{:}(M_1|M_2|...|M_n)\ \cdot$
[p] $\bigwedge\ \{\textbf{is\_}M_i(m_i) \equiv \bigvee\sim\{\textbf{is\_}M_j(m_i)|j \in \{1..m\}\backslash\{i\}\}|i \in \{1..m\}\}$

- *The M_i are all distinct* ■

Example 36 . Pipeline Material *We continue Example 16 on slide 97 and Example 22 on slide 118.*

- *Previously we may have left the impression that pipeline units, u:U , were atomic. Now we consider them composite all satisfying* `has_material`*(u).*

50 When we apply `obs_material_sorts_U` *to any unit* `u:U` *we obtain*

a. a type clause stating the material sort `LoG` *for some further undefined liquid or gaseous material, and*

b. a material observer function signature.

type

50a. *LoG*

value

50b. **obs**_*LoG*: *U* → *LoG* ■

3.11.1. Material Qualities

- It seems that we do not need to model
  - unique identifier nor
  - mereology

  qualities of materials[16].
- But materials do have attributes.
  - We extend the usual attribute-related analysis and synthesis prompts (`observe_attributes`) to apply also to materials.

[16] We might be persuaded to call the isotope marking of a liquid (for the purposes of tracing the sources or sinks of that liquid) a unique identifier.

Example 37 . Pipeline Material Attributes *We continue Examples 16, 22 and 36 on slide 164.*

- *One possible attribute of the liquid material of pipelines are liquid type:* `organic oil` *or* `mineral oil`,
- *For, for example, mineral oil there are many, many petrochemical attributes.*
- *We refer to standard work on petrochemistry for details* ■

3.11.2. Materials-related Part Attributes

- It seems that the "interplay" between parts and materials
  - is an area where domain analysis
  - in the sense of this seminar
  - is relevant.

Example 38 . Pipeline Material Flow *We continue Examples 16, 22, 36 and 37.*

- *Let us postulate a[n attribute] sort* **Flow**.
- *We now wish to examine the flow of liquid (or gaseous) material in pipeline units.*
- *We use two types*

  51 **F** *for "productive" flow, and* **L** *for wasteful leak.*
- *Flow and leak is measured, for example, in terms of volume of material per second.*
- *We then postulate the following unit attributes*
  - *"measured" at the point of in- or out-flow*
  - *or in the interior of a unit.*

52 current flow of material into a unit input connector,

53 maximum flow of material into a unit input connector while maintaining laminar flow,

54 current flow of material out of a unit output connector,

55 maximum flow of material out of a unit output connector while maintaining laminar flow,

56 current leak of material at a unit input connector,

57 maximum guaranteed leak of material at a unit input connector,

58 current leak of material at a unit input connector,

59 maximum guaranteed leak of material at a unit input connector,

60 current leak of material from "within" a unit, and

61 maximum guaranteed leak of material from "within" a unit.

type
51. F, L

value
52. **attr**_cur_iF: $U \to UI \to F$
53. **attr**_max_iF: $U \to UI \to F$
54. **attr**_cur_oF: $U \to UI \to F$
55. **attr**_max_oF: $U \to UI \to F$
56. **attr**_cur_iL: $U \to UI \to L$
57. **attr**_max_iL: $U \to UI \to L$
58. **attr**_cur_oL: $U \to UI \to L$
59. **attr**_max_oL: $U \to UI \to L$
60. **attr**_cur_L: $U \to L$
61. **attr**_max_L: $U \to L$

- The maximum flow attributes are static attributes and are typically provided by the manufacturer as indicators of flows below which laminar flow can be expected.
- The current flow attributes are dynamic attributes ■.

3.11.3. Laws of Material Flows and Leaks

- It may be difficult or costly, or both,
  - to ascertain flows and leaks in materials-based domains.
  - But one can certainly speak of these concepts.
  - This casts new light on **domain modelling**.
  - That is in contrast to
    - incorporating such notions of flows and leaks
    - in **requirements modelling**
  - where one has to show implementability.
- Modelling flows and leaks is important to the modelling of materials-based domains.

Example 39 . Pipelines: Intra Unit Flow and Leak Law

62 For every unit of a pipeline system, except the well and the sink units, the following law apply.

63 The flows into a unit equal

a. the leak at the inputs

b. plus the leak within the unit

c. plus the flows out of the unit

d. plus the leaks at the outputs.

axiom [*Well–formedness of Pipeline Systems, PLS (1)*]

62. ∀ *pls:PLS,b:B\We\Si,u:U* ·
62. *b* ∈ **obs_***Bs*(*pls*)∧*u*=**obs_***U*(*b*)⇒
62. **let** (*iuis,ouis*) = **mereo_***U*(*u*) **in**
63. *sum_cur_iF*(*iuis*)(*u*) =
63a.. *sum_cur_iL*(*iuis*)(*u*)
63b.. ⊕ **attr_***cur_L*(*u*)
63c.. ⊕ *sum_cur_oF*(*ouis*)(*u*)
63d.. ⊕ *sum_cur_oL*(*ouis*)(*u*)
62. **end**

64 The sum_cur_iF (cf. Item 63) sums current input flows over all input connectors.

65 The sum_cur_iL (cf. Item 63a.) sums current input leaks over all input connectors.

66 The sum_cur_oF (cf. Item 63c.) sums current output flows over all output connectors.

67 The sum_cur_oL (cf. Item 63d.) sums current output leaks over all output connectors.

64. sum_cur_iF: UI-**set** → U → F
64. sum_cur_iF(iuis)(u) ≡ ⊕ {**attr**_cur_iF(ui)(u)|ui:UI•ui ∈ iuis}
65. sum_cur_iL: UI-**set** → U → L
65. sum_cur_iL(iuis)(u) ≡ ⊕ {**attr**_cur_iL(ui)(u)|ui:UI•ui ∈ iuis}
66. sum_cur_oF: UI-**set** → U → F
66. sum_cur_oF(ouis)(u) ≡ ⊕ {**attr**_cur_iF(ui)(u)|ui:UI•ui ∈ ouis}
67. sum_cur_oL: UI-**set** → U → L
67. sum_cur_oL(ouis)(u) ≡ ⊕ {**attr**_cur_iL(ui)(u)|ui:UI•ui ∈ ouis}
    ⊕: (F|L) × (F|L) → F ■

Example 40 . Pipelines: Inter Unit Flow and Leak Law

68 For every pair of connected units of a pipeline system the following law apply:

a. the flow out of a unit directed at another unit minus the leak at that output connector

b. equals the flow into that other unit at the connector from the given unit plus the leak at that connector.

axiom [*Well–formedness of Pipeline Systems, PLS (2)*]

$$
\begin{array}{ll}
68. & \forall\ pls{:}PLS, b, b'{:}B, u, u'{:}U\cdot \\
68. & \quad \{b,b'\}\subseteq \mathbf{obs\_}Bs(pls) \wedge b\neq b' \wedge u'=\mathbf{obs\_}U(b') \\
68. & \quad \wedge\ \mathbf{let}\ (iuis,ouis)=\mathbf{mereo\_}U(u),(iuis',ouis')=\mathbf{mereo\_}U(u'), \\
68. & \quad\quad ui=\mathbf{uid\_}U(u), ui'=\mathbf{uid\_}U(u')\ \mathbf{in} \\
68. & \quad\quad ui \in iuis \wedge ui' \in ouis' \Rightarrow \\
68a.. & \quad\quad\quad \mathbf{attr\_}cur\_oF(u')(ui') - \mathbf{attr\_}leak\_oF(u')(ui') \\
68b.. & \quad\quad = \mathbf{attr\_}cur\_iF(u)(ui) + \mathbf{attr\_}leak\_iF(u)(ui) \\
68. & \quad\quad \mathbf{end} \\
68. & \quad \mathbf{comment:}\ b'\ precedes\ b
\end{array}
$$

■

- From the above two laws one can prove the **theorem:**
  - what is pumped from the wells equals
  - what is leaked from the systems plus what is output to the sinks.
- We need formalising the flow and leak summation functions.

3.12. "No Junk, No Confusion"

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

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

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

- So, since one naturally wishes "no junk, no confusion" what does one do ?
- The answer to that is
  - *one proceeds with great care !*
- To avoid junk we have stated a number of sort well-formedness axioms, for example:
  - Slide 106 for *Well-formedness of Links, L, and Hubs, H*,
  - Slide 114 for *Well-formedness of Domain Mereologies*,
  - Slide 117 for *Well-formedness of Road Nets, N*,
  - Slide 119 for *Well-formedness of Pipeline Systems, PLS (0)*,
  - Slide 138 for *Well-formedness of Hub States, HΣ*,
  - Slide 173 for *Well-formedness of Pipeline Systems, PLS (1)*,
  - Slide 175 for *Well-formedness of Pipeline Systems, PLS (2)*,
  - Slide 182 for *Well-formedness of Pipeline Route Descriptors* and
  - Slide 186 for *Well-formedness of Pipeline Systems, PLS (3)*.

- To avoid confusion we have stated a number of proof obligations:
  - Slide 79 for *Disjointness of Part Sorts*,
  - Slide 134 for *Disjointness of Attribute Types* and
  - Slide 163 for *Disjointness of Material Sorts*.

Example 41 . No Pipeline Junk

- *We continue Example 16 on slide 97 and Example 22 on slide 118.*

69 We define a proper pipeline route to be a sequence of pipeline units.

a. such that the i^{th} and $i+1^{st}$ units in sequences longer than 1 are (forward) adjacent, in the sense defined below, and

b. such that the route is acyclic, in the sense also defined below.

To formalise the above we describe some auxiliary notions.

3.12.0.1 Pipe Routes

70 A route descriptor is the sequence of unit identifiers of the units of a route (of a pipeline system).

type

69. R′ = U^ω
69. R = {| r:Route′·wf_Route(r) |}
70. RD = UI^ω

axiom [Well−formedness of Pipeline Route Descriptors, RD]

70. ∀ rd:RD · ∃ r:R·rd=descriptor(r)

value

70. descriptor: R → RD
70. descriptor(r) ≡ ⟨**uid**_UI(r[i])|i:**Nat**·1≤i≤**len** r⟩

71 Two units are (forward) adjacent if the output unit identifiers of one shares a unique unit identifier with the input identifiers of the other.

```
value
71.   adjacent: U × U → Bool
71.   adjacent(u,u′) ≡
71.     let (,ouis)=mereo_U(u),
71.         (iuis,)=mereo_U(u′) in
71.     ouis ∩ iuis ≠ {} end
```

72 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 be a unit of pls, then $\langle u\rangle$ is a route of pls.
c. Let u, u' be adjacent units of pls then $\langle u, u'\rangle$ is a route of pls.
d. 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.
e. No sequence of units is a route unless it follows from a finite number of applications of the basis and induction clauses of Items 72a.–72d..

```
value
72.     Routes: PLS → R-infset
72.     Routes(pls) ≡
72a..       let rs = ⟨⟩
72b..           ∪ {⟨u⟩|u:U•u ∈ obs_Us(pls)}
72c..           ∪ {⟨u,u′⟩|u,u′:U•{u,u′}⊆obs_Us(pls) ∧ adjacent(u,u′)}
72d..           ∪ {r⌢tl r′|r,r′:R•{r,r′}⊆rs∧r[len r]=hd r′}
72e..       in rs end
```

3.12.0.2 Well-formed Routes

73 A route is acyclic if no two route positions reveal the same unique unit identifier.

value

73. acyclic_Route: R → **Bool**
73. acyclic_Route(r) ≡ ∼∃ i,j:**Nat**·{i,j}⊆**inds** r ∧ i≠j ∧ r[i]=r[j]

3.12.0.3 Well-formed Pipeline Systems

74 A pipeline system is well-formed if

a. none of its routes are circular and

b. all of its routes embedded in well-to-sink routes.

axiom [Well−formedness of Pipeline Systems, PLS (3)]
74. ∀ pls:PLS ·
74a.. non_circular(pls)
74b.. ∧ are_embedded_in_well_to_sink_Routes(pls)
value
74. non_circular_PLS: PLS → **Bool**
74. non_circular_PLS(pls) ≡
74. ∀ r:R·r ∈ routes(p)∧acyclic_Route(r)

75 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
75. well_to_sink_Routes: PLS → R-set
75. well_to_sink_Routes(pls) ≡
75.     let rs = Routes(pls) in
75.     {r|r:R·r ∈ rs ∧ is_We(r[1]) ∧ is_Si(r[len r])} end
```

76 A pipeline system is well-formed if all of its routes are embedded in well-to-sink routes.

```
76.  are_embedded_in_well_to_sink_Routes: PLS → Bool
76.  are_embedded_in_well_to_sink_Routes(pls) ≡
76.      let wsrs = well_to_sink_Routes(pls) in
76.      ∀ r:R · r ∈ Routes(pls) ⇒
76.          ∃ r′:R,i,j:Nat ·
76.              r′ ∈ wsrs
76.            ∧ {i,j}⊆inds r′∧i≤j
76.            ∧ r = ⟨r′[ k ]|k:Nat·i≤k≤j⟩ end
```

3.12.0.4 Embedded Routes

77 For every route we can define the set of all its embedded routes.

```
value
77. embedded_Routes: R → R-set
77. embedded_Routes(r) ≡
77.     {⟨r[k]|k:Nat·i≤k≤j⟩ | i,j:Nat· i {i,j}⊆inds(r) ∧ i≤j}
```

3.12.0.5 A Theorem

78 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:

78. ∀ pls:PLS ·
78. **let** rs = Routes(pls),
78. wsrs = well_to_sink_Routes(pls) **in**
78a.. rs =
78b.. wsrs ∪
78c.. ∪ {{r′|r′:R · r′ ∈ embedded_Routes(r″)} | r″:R · r″ ∈ wsrs}
77. **end** ■

- The above example,
  - besides illustrating one way of coping with "junk",
  - also illustrated the need for introducing a number of auxiliary notions:
    - types,
    - functions,
    - axioms and
    - theorems.

3.13. Discussion of Endurants

- In Sect. 4.2.2 a "depth-first" search for part sorts was hinted at.
- It essentially expressed
  - that we discover domains epistemologically
  - but understand them ontologically.
- The Danish philosopher Søren Kirkegaard (1813–1855) expressed it this way:
  - *Life is lived forwards,*
  - *but is understood backwards.*
- The presentation of the of the **'domain analysis prompt'**s and the **'domain description prompt'**s is based on resulting in a domain description which is ontological.
- The "depth-first" search recognizes the epistemological nature of bringing about understanding.

- This "depth-first" search
  - that ends with the analysis of atomic part sorts
  - can be guided, i.e., hastened (shortened),
  - by postulating composite sorts
  - that "correspond" to vernacular nouns:
  - everyday nouns that stand for classes of endurants.

- We could have chosen our **′domain analysis prompt′**s and **′domain description prompt′**s to reflect
  - a "bottom-up" epistemology,
  - one that reflected how we composed composite understandings
  - from initially atomic parts.
  - We leave such a collection of **′domain analysis prompt′**s and **′domain description prompt′**s to the student.

Perdurants

- We shall give only a cursory overview of perdurants.
- That is, we shall not systematically present
  - a set of 'domain analysis prompt's and
  - a set of 'domain description prompt's

  leading to description language,
  i.e., RSL texts describing perdurant entities.

- The reason for giving this albeit cursory overview of perdurants
  - is that, through this cursory overview, we can justify our detailed study of endurants,
    - their part and subparts,
    - their unique identifiers, attributes and mereology.
- This justification is manifested
  - in expressing the types of signatures,
  - in basing behaviours on parts,
  - in basing the for need CSP-oriented inter-behaviour communications on shared part attributes,
  - in indexing behaviours as are parts, i.e., on unique identifiers,

  and
  - in directing inter-behaviour communications across channel arrays indexed as per the mereology of the part behaviours.

- These are all notions related to endurants
  and are now justified by their use in describing perdurants.
- Perdurants can perhaps best be explained in terms of
  - a notion of **state** and
  - a notion of **time**.
- We shall, in this seminar, not detail notions of **time**.

4.1. States

Definition 9 . State: *By a 'state' we shall understand*

- *any collection of endurants*
- *each of which has at least one dynamic attribute.*

Example 42 . States *Some examples of states are:*

- *A road hub can be a state,*
  cf. Hub State, LΣ, Example 27 on slide 137.
- *A road net can be a state – since its hubs can be.*
- *Container stowage areas. CSA, Example 11 on slide 92, of container vessels and container terminal ports can be states as containers can be removed from and put on top of container stacks* ■

4.2. Actions, Events and Behaviours

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

  behaviours.

4.2.1. Time Considerations

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

- Instead we shall refer to two seminal monographs:
  - Specifying Systems [Leslie Lamport, 2002] and
  - Duration Calculus: A Formal Approach to Real-Time Systems [Zhou ChaoChen and Michael Reichhardt Hansen, 2004].

4.2.2. Actors

Definition 10 . Actor: *By an 'actor' we shall understand*

- *something that is capable of initiating and/or carrying out*
  - *actions,*
  - *events or*
  - *behaviours.*
- We shall, in principle, associate an actor with each part.
  - These actors will be described as behaviours.
  - These behaviours evolve around a state.
  - The state is the set of qualities,
    in particular the dynamic attributes,
    of the associated part.

Example 43 . Actors *We refer to the road transport and the pipeline systems examples of earlier.*

- *The fleet, each vehicle and the road management of the Transportation system of Examples 9 on slide 80 and 32 on slide 150 can be considered actors;*
- *so can the net and its links and hubs.*
- *The pipeline monitor and each pipeline unit of the Pipeline System, Example 16 on slide 97 and Examples 16 on slide 97 and 22 on slide 118 will be considered actors.*
- *The bank general ledger and each bank customer of the Shared Passbooks example, Example 33 on slide 153, will be considered actors* ■

4.2.3. Parts, Attributes and Behaviours

- Example 43 on the previous slide focused on what shall soon become a major relation within domains:
  - that of parts being also considered actors,
  - or more specifically, being also considered to be behaviours.

Example 44 . Parts, Attributes and Behaviours

- *Consider the term 'train'.*
- *It has several possible "meanings".*
  - *the train as a part, viz., as standing on a platform;*
  - *the train as listed in a timetable (an attribute of a transport system part),*
  - *the train as a behaviour: speeding down the rail track* ■

4.3. Discrete Actions

Definition 11 . Discrete Action: *By a 'discrete action' [WilsonScpall2012] we shall understand*

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

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

Example 45 . Road Net Actions

- *Examples of road net actions initiated by the net actor are:*
  - *insertion of hubs,*
  - *insertion of links,*
  - *removal of hubs,*
  - *removal of links,*
  - *setting of hub states.*
- *Examples of traffic system actions initiated by vehicle actors are:*
  - *moving a vehicle along a link,*
  - *stopping a vehicle,*
  - *starting a vehicle,*
  - *entering a hub and*
  - *leaving a hub* ■

4.4. Discrete Events

Definition 12 . Event: *By an 'event' we shall understand*

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

- Events can be characterised by
  - a pair of (before and after) states,
  - a predicate over these
  - and, optionally, a time or time interval.
- The notion of event continues to puzzle philosophers [Dretske1967,Quinton1979,Mellor1980,Davidson1980] [Hacker1982b,Badiou1988,JaegwonKim93,CasatiVarzi1996] [TonyPi99,CasatiVarzi2010].
- We note, in particular, [Davidson1980,Badiou1988,JaegwonKim93].

Example 46 . Road Net and Road Traffic Events

- *Some road net events are:*
  - *"disappearance" of a hub or a link,*
  - *failure of a hub state to change properly when so requested, and*
  - *occurrence of a hub state leading traffic into "wrong-way" links.*
- *Some road traffic events are:*
  - *the crashing of one or more vehicles (whatever 'crashing' means),*
  - *a car moving in the wrong direction of a one-way link, and*
  - *the clogging of a hub with too many vehicles* ■

4.5. Discrete Behaviours

Definition 13 . **Discrete Behaviour:** *By a* **'discrete behaviour'** *we shall understand*

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

Example 47 . Behaviours

- *Examples of behaviours:*
  - *Road Nets: A sequence of hub and link insertions and removals, link disappearances, etc.*
  - *Road Traffic: A sequence of movements of vehicles along links, entering, circling and leaving hubs, crashing of vehicles, etc.*
  - *Pipelines: A sequence of pipeline pump and valve openings and closings, and failures to do so (events), etc.*
  - *Container Vessels and Ports: Concurrent sequences of movements (by cranes) of containers from vessel to port (unloading), with sequences of movements (by cranes) from port to vessel (loading), with dropping of containers by cranes, etcetera* ■

4.5.1. Channels and Communication

- Behaviours usually communicate. We use CSP to model behaviour communication.
  - Communication is abstracted as
    - the sending (ch ! m) and
    - receipt (ch ?)
    - of messages, m:M,
    - over channels, ch.

      type M
      channel ch M
  - Communication between (unique identifier) indexed behaviours have their channels modelled as similarly indexed channels:

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

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

4.5.2. Relations Between Attribute Sharing and Channels

- We shall now interpret
  - the syntactic notion of attribute sharing with
  - the semantic notion of channels.
- This is in line with the above hinted interpretation of
  - parts with behaviours, and,

  as we shall soon see
  - part attributes with behaviour states.

- Thus, for every pair of parts, p_{ik}:P_i and $\mathsf{p}_{j\ell}$:P_j, of distinct sorts, P_i and P_j which share attribute values in A
  - ⋈ we are going to associate a channel.
    - ⊚ If there is only one pair of parts, p_{ik}:P_i and $\mathsf{p}_{j\ell}$:P_j, of these sorts, then just a simple channel, say ch_{P_i,P_j}.

      channel ch_{P_i,P_j}:A.
    - ⊚ If there is only one part, p_i:P_i, but a definite set of parts p_{jk}:P_j, with shared attributes, then a *vector* of channels.
      - ∗ Let $\{p_{j1}, p_{j2}, ..., p_{jn}\}$ be all the part of the domain of sort P_j.
      - ∗ Then $uids : \{uid_{p_{j1}}, uid_{p_{j2}}, ..., uid_{p_{jn}}\}$ is the set of their unique identifiers.
      - ∗ Now a schematic channel array declaration can be suggested:

        channel $\{\mathrm{ch}[\{\pi_i,\pi_j\}]|\pi_i$=**uid**_$\mathrm{P}_i(\mathrm{p}_i)\wedge\pi_j \in \mathrm{uids}\}$:A.

Example 48 . Bus System Channels

- *We extend Examples 9 on slide 80 and 32 on slide 150.*
- *We consider the* **fleet** *and the* **vehicles** *to be behaviours.*

79 We assume some **transportation system**, δ. *From that system we observe*

80 the **fleet** *and*

81 the **vehicles**.

82 The fleet to vehicle channel array is indexed by the 2-element sets of the unique fleet identifier and the unique vehicle identifiers. We consider **bus timetables** *to be the only message communicated between the* **fleet** *and the* **vehicle** *behaviours.*

```
value
79.   δ:Δ,
80.   f:F = obs_F(δ),
81.   vs:V-set = obs_Vs(obs_VC((obs_F(δ))))
channel
82.   {fch[{uid_F(f),uid_V(v)}]|v:V·v ∈ vs}:BT ■
```

Example 49 . Bank System Channels

- *We extend Example 33 on slide 153.*
- *We consider the general ledger and the customers to be behaviours.*

83 We assume some bank system. From the bank system

84 we observe the general ledger.

85 and the set of customers.

86 We consider passbooks to be the only message communicated between the general ledger and the customer behaviours.

value

83. bs:BS
*84. gl=***obs_***GL(***obs_***AD(bs)):GL*
*85. cs=***obs_***Cs(***obs_***CS(bs)):C-***set**

channel

*86. {bsch[{***uid_***GL(gl),***uid_***C(c)}]|c:C·c ∈ cs}:PB* ■

4.6. Continuous Behaviours

- By a 'continuous behaviour' we shall understand
  - a continuous time
  - sequence of state changes.
- We shall not go into what may cause these state changes.

Example 50 . Flow in Pipelines

- *We refer to Examples 22, 36, 38, 39 and 40.*
- *Let us assume that oil is the (only) material of the pipeline units.*
- *Let us assume that there is a sufficient volume of oil in the pipeline units leading up to a pump.*
- *Let us assume that the pipeline units leading from the pump (especially valves and pumps) are all open for oil flow.*
- *Whether or not that oil is flowing, if the pump is pumping (with a sufficient* **head***) then there will be oil flowing from the pump outlet into adjacent pipeline units.* ■

- To describe the flow of material (say in pipelines) requires knowledge about a number of material attributes — not all of which have been covered in the above-mentioned examples.
- To express flows one resorts to the mathematics of fluid-dynamics using such second order differential equations as first derived by Bernoulli (1700–1782) and Navier–Stokes (1785–1836 and 1819–1903).

4.7. Perdurant Signatures and Definitions

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

4.7.1. Function Signatures

Definition 14 . **Function Signature:** *By a 'function signature' we shall understand*

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

Definition 15 . **Function Type Expression:** *By a 'function type expression' we shall understand*

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

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

4.7.2. Action Signatures and Definitions

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

$$\begin{aligned}&\text{action: VAL} \to \Sigma \xrightarrow{\sim} \Sigma\\&\text{action(v)}(\sigma)\ \textbf{as}\ \sigma'\\&\quad \textbf{pre:}\quad \mathcal{P}(\text{v},\sigma)\\&\quad \textbf{post:}\quad \mathcal{Q}(\text{v},\sigma,\sigma')\end{aligned}$$

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

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

Example 51 . Insert Hub Action Formalisation *We formalise aspects of the above-mentioned hub and link actions:*

87 Insertion of a hub requires

88 that no hub exists in the net with the unique identifier of the inserted hub,

89 and then results in an updated net with that hub.

value

87. $insert\_H: H \rightarrow N \xrightarrow{\sim} N$

87. $insert\_H(h)(n)$ **as** n'

88. **pre**: $\sim\exists\ h':H \cdot h' \in$ **obs**$\_Hs($**obs**$\_HS(n)) \cdot$**uid**$\_H(h) =$**uid**$\_H(h')$

89. **post**: **obs**$\_Hs($**obs**$\_HS(n')) =$**obs**$\_Hs($**obs**$\_HS(n)) \cup \{h\}$ ■

- Which could be the argument values, v:VAL, of actions ?
  - Well, there can basically be only two kinds of argument values:
    - parts and materials, respectively
    - unique part identifiers, mereologies and attribute values.
  - It basically has to be so
    - since there are no other kinds of values in domains.
  - There can be exceptions to the above
    - (Booleans,
    - natural numbers),

    but they are rare !

- **Perdurant (action) analysis thus proceeds as follows:**
  - identifying relevant actions,
  - assigning names to these,
  - delineating the "smallest" relevant state[17],
  - ascribing signatures to action functions, and
  - determining
    - action pre-conditions and
    - action post-conditions.
  - Of these, ascribing signatures is, perhaps, the most crucial:
    - In the process of determining the action signature
    - one oftentimes discovers
    - that part or material attributes have been left "undiscovered".

[17] Experience shows that the domain analyser cum describer should strive for identifying the smallest state.

- Example 52 shows examples of signatures whose arguments are
  - either parts,
  - or parts and unique identifiers,
  - or parts and unique identifiers and attributes.

Example 52 . Some Function Signatures

- *Inserting a link between two identified hubs in a net:*

  value *insert_L: L* × *(HI* × *HI)* → *N* $\xrightarrow{\sim}$ *N*

- Removing a hub and removing a link:

  value remove_H: HI → N $\xrightarrow{\sim}$ N
  remove_L: LI → N $\xrightarrow{\sim}$ N

- Changing a hub state.

  value change_HΣ: HI × HΣ → N $\xrightarrow{\sim}$ N ■

4.7.3. Event Signatures and Definitions

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

value
  event: $\Sigma \times \Sigma \rightarrow$ **Bool**
  event(σ,σ') **as true**$\lceil\rceil$**false**
    pre: $P(\sigma)$
    post: $Q(\sigma,\sigma')$

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

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

Example 53 . Link Disappearence Formalisation *We formalise aspects of the above-mentioned link disappearance event:*

90 The result net is not well-formed.

91 For a link to disappear there must be at least one link in the net;

92 and such a link may disappear such that

93 it together with the resulting net makes up for the "original" net.

value

90. *link_diss_event:* $N \times N \times$ **Bool**
90. *link_diss_event(n,n′)* **as** *tf*
91. **pre:** **obs_**Ls(**obs_**$LS(n)$)$\neq\{\}$
92. **post:** $\exists$ *l:L*$\cdot l \in$ **obs_**Ls(**obs_**$LS(n)$) $\Rightarrow$
93. $l \notin$ **obs_**Ls(**obs_**$LS(n')$) $\wedge$ $n' \cup \{l\} =$ **obs_**Ls(**obs_**$LS(n)$) ■

4.7.4. Discrete Behaviour Signatures and Definitions

- We shall only cover behaviour signatures when expressed in RSL/CSP [RSL].
- The behaviour functions are now called processes.
- As shown in [dines:urbino:2012] we can, without loss of generality, associate with each part and sub-part a behaviour;
  - parts which share attributes
  - and are therefore referred to in some parts' mereology,
  - can communicate (their "sharing") via channels.

- A behaviour signature is therefore:

  behaviour: π:Π $\times$ p:P $\times$ VAL $\rightarrow$ **out** ochs **in** ichns $\rightarrow$ **process**

  where

  - π:Π is the unique identifier of part p, i.e., π=**uid_**P(p), and
  - ochs, ichs are channel expressions, generally of the form

    ochs,ichs: {ch[i]|i:IDE·$\mathcal{P}$(i)}

    where $\mathcal{P}$(i) is some predicate expression.

- Let P be a composite sort.
  - Let P be defined in terms of sub-sorts PA, PB, ..., PC.
  - Proces p "derived" from p:P, is composed from
    - a process, $\mathcal{M}_P$, relying on and handling the attributes of process p as defined by P
    - operating in parallel with processes $p_a, p_b, \ldots, p_c$ where
      - p_a is "derived" from PA,
      - p_b is "derived" from PB,
      - ..., and
      - p_c is "derived" from PC.
- The domain description "compilation" schematic below "formalises" the above.

Process Schema I

```
value
    p_a:PA=obs_PA(p),
    p_b:PB=obs_PB(p),
    ...,
    p_c:PC=obs_PC(p),
    comp_process(p) ≡
        p: π:Π × p:P × attrs:P_ATTRS →
              in,out {ch[{π,pi}]·pi ∈ mereo_P(p)} process
        p(π:uid_P(p),p,attrs:obs_attribs(p)) ≡
            M_P(π,p,attrs)
          ‖ comp_process(p_a)
          ‖ comp_process(p_b)
          ‖ ...
          ‖ comp_process(p_c)
```

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

Process Schema II

```
type
    Qs = Q-set
value
    qs:Q-set = obs_Qs(p) in
    comp_process(p) ≡
        p: π:Π × p:P × attrs:P_ATTRS →
              in,out {ch[{π,pi}]·pi ∈ mereo_P(p)}  process
        p(π:uid_P(p),p,attrs:obs_attribs(p)) ≡
           M_P(π,p,attrs)
         ‖ ‖{comp_process(q)|q:Q·q ∈ qs}
```

Example 54 . Bus Timetable Coordination

- *We refer to Examples 9 on slide 80, 10 on slide 87, 32 on slide 150 and 48 on slide 215.*

94 δ *is the transportation system;* **f** *is the fleet part of that system;* **vs** *is the set of vehicles of the fleet;* **bt** *is the shared bus timetable of the fleet and the vehicles.*

95 The **fleet** *process is compiled as per Process Schema II (Slide 238)*

type
  Δ, F, VC *[Example 9 on slide 80]*
  V, Vs=V-**set** *[Example 10 on slide 87]*
  FI, VI, BT *[Example 32 on slide 150]*
channel
  $\{fch...\}$ *[Example 48 on slide 215]*
value
94. δ:Δ,
94. f:F = **obs_**$F(\delta)$,
94. vs:V-**set** = **obs_**Vs(**obs_**$VC(f)$),
94. bt:BT = **attr_**$BT(f)$
axiom
94. $\forall\ v{:}V{\cdot}v \in vs \Rightarrow bt$ = **attr_**$BT(v)$ *[Example 32 on slide 150]*
value
95. *fleet*: $fi{:}FI{\times}f{:}F{\times}BT \rightarrow$ **in**,**out** $\{fch[\,\{fi,$**uid_**$V(v)\}\,]|v{:}V{\cdot}v \in vs\}$ **process**
95. *fleet*$(fi,f,bt) \equiv \mathcal{M}_F(fi,f,bt) \parallel \parallel \{vehicle($**uid_**$V(v),v,bt)|v{:}V{\cdot}v \in vs\}$
95. *vehicle*: $vi{:}VI{\times}v{:}V{\times}bt{:}BT \rightarrow$ **in**,**out** $fch[\,\{fi,vi\}\,]$ **process**
95. *vehicle*$(vi,v,bt) \equiv \mathcal{M}_V(vi,v,bt)$

- Fleet and vehicle processes
  - $\mathcal{M}_F$ and
  - $\mathcal{M}_V$
- are both "never-ending" processes:

value

$\mathcal{M}_F$: FI×F×BT → **in**,**out** {fch[{fi,**uid**_V(v)|v:V·v ∈ vs} **process**
$\mathcal{M}_F$(fi,f,bt) ≡ **let** bt′ = $\mathcal{F}$(fi,f,bt) **in** $\mathcal{M}_F$(fi,f,bt′) **end**

$\mathcal{M}_V$: VI×V×BT → **in**,**out** fch[{fi,vi}] **process**
$\mathcal{M}_V$(vi,v,bt) ≡ **let** bt′ = $\mathcal{V}$(vi,v,bt) **in** $\mathcal{M}_V$(vi,v,bt′) **end**

- The "core" processes,
  - $\mathcal{F}$ and
  - $\mathcal{V}$,

  are simple actions.

- In this example we simplify them to change only bus timetables.
- The expression of actual synchronisation and communication between the fleet and the vehicle processes are contained in $\mathcal{F}$ and $\mathcal{V}$.

value

$\mathcal{F}$: FI×F×BT → **in**,**out** {fch[{fi,**uid**_V(v)|v:V·v ∈ vs} BT
$\mathcal{F}$(fi,f,bt) ≡ ...

$\mathcal{V}$: VI×V×BT → **in**,**out** fch[{fi,vi}] BT
$\mathcal{V}$(vi,v,bt) ≡ ...

- What the synchronisation and communication between the fleet and the vehicle processes consists of we leave to the reader! ■

Example 55 . Client Bank Transactions

- *We refer to Example 33 on slide 153.*

96 bs *is the bank system,*

97 gl *is the general ledger of the bank administration,*

98 pbs *is the set of passbooks attribute of the general ledger and*

99 cs *is the set of bank customers.*

100 bank *is the overall bank system behaviour.*

101 gen_ldgr *is the behaviour of the general ledger, that is, the demand/deposit activities of the bank.*

102 clients *is the overall behaviour of the ensemble of bank demand/deposit [account] customers.*

103 customer *is the behaviour of the individual bank customer. It is here simplified to just the customer behaviour with respect to the demand/deposit account as manifested by the passbook attribute.*

The processes are compiled as per Process Schema I (Slide 236) – two "compilations" !

type
  [parts] BS, AD, GL, CS, Cs, C [Example 33 on slide 153]
  [attribute] PB [Example 33 on slide 153]
value
96. *bs:BS*
97. *gl:GL* = **obs**_*GL*(**obs**_*AD*(*bs*)), *glid:GLI* = **uid**_*GL*(*gl*)
98. *pbs:PB*-**set** = **attr**_*PSs*(*gl*)
99. *cs:C*-**set** = **obs**_*Cs*(**obs**_*CS*(*bs*))
axiom
98. *pbs* = {**attr**_*PS*(*c*)|*c:C:0c* ∈ *cs*} *[Example 33 on slide 153]*

value

98. bank: bs:BS → **process**
98. bank(bs) ≡ gen_ldgr(glid)(gl)(pbs) ‖ clients(cs)
99. gen_ldgr: π:GLI × GL × PB**-set** → **in**,**out** {bch[π,**uid**_C(c)]|c:C·c ∈
99. gen_ldgr(π,gl,pbs) ≡ $\mathcal{M}_{GL}$(π,gl,pbs)
100. clients: C**-set** → **in**,**out** {bch[π,**uid**_C(c)]|c:C·c ∈ cs} **process**
100. clients(cs) ≡ ‖{customer(**uid**_C(c),c,**attr**_PB(c))|c:C·c ∈ cs}
101. customer: π:CI × C × PB → **in**,**out** {bsch[glid,π]} **process**
101. customer(**uid**_C(c),c,**attr**_PB(c)) ≡ $\mathcal{M}_{C}$(**uid**_C(c),c,**attr**_PB(c))

- The $\mathcal{M}_{GL}$ and $\mathcal{M}_C$ behaviours are seen as "never-ending".
- We leave their definition to the listener
  - who is expected to model
  - simple deposit, withdraw and inter-account transfer transactions.
- We have here assumed that each such transactions all lead to update of both the client and the general ledger passbooks ■

4.8. Summary and Discussion of Perdurants

4.8.1. Summary

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

4.8.2. Discussion

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

Conclusion

- Our last characterisation of the concept of domain can now be formulated:
  - By a **'domain'**[18] we shall understand
    - a spatially organised collection of manifest endurants and
    - a temporally understood collection of interacting observable perdurants,
    - including humans.
- Mathematically we can characterise a **'domain description'**
  - as denoting a heterogeneous algebra of
    - sorts and
    - operations.
  - The sorts model endurants and
  - the operations model actions, events and behaviours.

[18] This is our last characterisation of the concept of 'domain'.

- The (i) construction, (ii) existence, and (iii) propagation & acceptance of domain descriptions serve a number of rôles:
  - (i) a construction (of a domain analysis & description) process rôle is to explore, inquire and theorize about a domain;
  - (ii) an existence rôle is, for the domain description, to be read and discussed by colleagues and **domain stake-holders** in order to reach agreements, through this social process, that the description is (hopefully) an appropriate model of the domain;

- (ii) another existence rôle is for a domain description to serve as a basis for the development of various kinds of software: demos, simulators and actual production software [dines:ugo65:2008,DomainsSimulatorsDemos2011];
- (iii) a propagation & acceptance rôle is for a domain description to become a de facto standard for further propagation of the domain description, including its use in teaching and training — also in the ordinary (say, secondary) school system!

- This paper started with a quote from Dostovevsky's `The Idiot`.

It's life that matters, nothing but life –
the process of discovering, the everlasting and perpetual process,
not the discovery itself, at all.[19]

- I find that quote appropriate in the following, albeit rather mundane, sense:
  - It is the process of analysing and describing a domain
  - that exhilarates me:
  - that causes me to feel very happy and excited.
- There is beauty [E.W. Dijkstra] not only in the result but also in the process.

[19] `Fyodor Dostoyevsky, The Idiot, 1868, Part 3, Sect. V`

5.1. On Domain Description Languages

- We have in this seminar expressed the domain descriptions in the RAISE [RaiseMethod] specification language RSL [RSL].
- With what is thought of as basically inessential, editorial changes, one can reformulate these domain description texts in either of
  - Alloy [alloy] or
  - The B-Method [JRAbrial:TheBBooks] or
  - VDM [e:db:Bj78bwo,e:db:Bj82b,JohnFitzgerald+PeterGormLarsen] or
  - Z [m:z:jd+jcppw96].

- We did not go into much detail with respect to perdurants, let alone behaviours.
  - For all the very many domain descriptions, covered elsewhere, RSL (with its CSP sub-language) suffices.
  - But there are cases where we have conjoined our RSL domain descriptions with descriptions in
    - Petri Nets [m:petri:wr09] or
    - MSC [MSCall] or
    - StateCharts [Harel87].

- Since this seminar only focused on endurants there was no need, it appears, to get involved in temporal issues.
- When that becomes necessary, in a study or description of perdurants, then we either deploy
  - DC: The Duration Calculus [zcc+mrh2002] or
  - TLA+: Temporal Logic of Actions [Lamport-TLA+02].

5.2. A Review of Our Claims: Interpretation and Evaluation

We structure this review according to that of Sect.1.2 Slides 33–40.

5.2.1. Domain Description Components

- Section 3 reveal that we have in mind to describe domains model-theoretically.
  - Usually, in a model-theoretic specification,
    - one specifies "data", i.e., the type of data,
    - before specifying the operations on these.
  - (In a property-oriented specification, notably in algebraic specifications [SanellaTarlecki2012],
    - one postulates the data (e.g., "stacks") and
    - then characterises these data through the interaction of operations on these data
      (e.g., "empty", "push", "pop", "top").)

- In order to analyse a domain we must make some simplifying assumptions.
  - A first significant "reduction of domain complexity"
    - is the decision to "divide the world" into two kinds of endurants:
      - manifest discrete and
      - manifest continuous,
    - that is, into parts and materials.
  - The second 'reduction' is to allow
    - that parts may contain material(s), `part_has_material`,
      - that is, that these forms of endurants can be clearly separated,

      or, vice-versa,
    - that `material_has_parts`.

- The choice as to
  - whether a part contains material(s) or
  - whether a material contains parts

  is a matter of "style":
  - whichever way best expresses a "being";
  - the described domain is the same!

- The separation of external and internal qualities is justified purely on pragmatic grounds.
  - The issue here is that of separating concerns.
  - Those concerns which are "more-or-less" manifest are "relegated" to 'external qualities'.
  - Those that are not manifest but can be measured, "more-or-less" by gadgets of physics are "relegated" to 'internal qualities'.

- The choice for modelling internal qualities:
  - that is,
    - **unique identifiers**,
    - **mereology** and the
    - **attributes**,

    is "conventional abstraction" !
  - Unique identification is a postulated "figment of imagination" !
    - They are best modelled as sorts.

- Mereology can be modelled many ways.
  - Where mereology reflects spatial
    - "adjacency" or
    - "embeddedness"

    relations,
  - they could be modelled "geometrically"
  - but that would sometimes lead to unnecessarily intricate "models".
  - So we choose simple forms of expressions over unique identifiers.

- Attributes are usually simple, "measurable" phenomena.
  - As such they are modelled as we would model data types.
  - In modelling domain attributes one usually exerts restraint.
    - Usually one can associate "zillions" of attributes even with atomic parts.
    - We advocate selecting as few attributes as needed.

- The chosen domain description language, RSL [RSL], allows us to distinguish between sorts and types.
  - Characteristics of sort values are then given through domain axioms, as expressed in RSL.
  - This is the case in general, it is the very basis for the "built-in" domain observer functions, **obs**_P, **obs**_M, **uid**_P, **mereo**_P and **attr**_A, briefly reviewed below.
  - In this RSL "borrows"
    - more from algebraic specifications, [SanellaTarlecki2012],
    - than from model-oriented ones.

5.2.2. Domain Observer Functions

- The **obs_P**, **obs_M**, **uid_P**, **mereo_P** and **attr_A**, etcetera, literals are emphasized in bold face.
- They need not have been written in this way; obs_P, obs_M, uid_P, mereo_P and attr_A, etcetera, would be just as fine.
- They are postulated functions
  - that "become defined" through the axioms
  - that usually accompany their first text presentation.
- (Many other functions are postulated and defined through axioms.)
- We have chosen the **obs_P**, ..., **attr_A** forms in order to emphasize that they are "standard" sort observer functions, 'standard' in the sense of 'common' to all domain descriptions.
- We consider this repertoire of 'standard' observer functions to be novel.

5.3. Comparison to Other Work

- Section 3. outlined the `TripTych` modelling approach to domain endurants.
- We shall now compare that approach to
  - a number of techniques and tools
  - that are somehow related —
  - if only by the term 'domain'!

5.3.1. Ontological Engineering

- Ontological engineering [Benjamins+Fensel98] build ontologies.
- Ontologies are *"formal representations of a set of concepts within a domain and the relationships between those concepts"* — expressed usually in some logic.
- Published ontologies usually consists of thousands of logical expressions.
- These are represented in some, for example, low-level mechanisable form so that they can be interchanged between ontology research groups and processed by various tools.

- There does not seem to be a concern for "deriving" such ontologies into requirements for software.
- Usually ontology presentations
  - either start with the presentation of,
  - or makes reference to its reliance on,

  an **upper ontology**.
- Ontology databases
  - appear to be used for the computerised
  - discovery and analysis
  - of relations between ontologies.

- The concerns of `TripTych` domain science & engineering is based on that of algorithmic engineering.
  - The domains to which we are applying our analysis & description tools and techniques
    - are spatio-temporal, that is, can be observed, physically;
    - this is in contrast to such conceptual domains as various branches of
      - mathematics,
      - physics,
      - biology,
      - etcetera.
  - Domain science & engineering is not aimed at
    - letting the computer solve problems based on
    - the knowledge it may have stored.

- Instead it builds models based on knowledge of the domain.

- The TripTych form of domain science & engineering differs from conventional **ontological engineering** in the following, essential ways:
  - The TripTych domain descriptions rely essentially on a "built-in" **upper ontology**:
    - types, abstract as well as model-oriented (i.e., concrete) and
    - actions, events and behaviours.
  - Domain science & engineering is not, to a first degree, concerned with modalities, and hence do not focus on the modelling of
    - knowledge and belief,
    - necessity and possibility, i.e., alethic modalities,
    - epistemic modality (certainty),
    - promise and obligation (deontic modalities),

    etcetera.

5.3.2. Knowledge and Knowledge Engineering

- The concept of **knowledge** has occupied philosophers since Plato.
  - No common agreement on what 'knowledge' is has been reached.
  - From
    `[OED,cambridge.dict.phil95,mw2004,dictionary:brittanica]`
    we may learn that
    - *knowledge is a familiarity with someone or something;*
      - *it can include facts, information, descriptions, or skills acquired through experience or education;*
      - *it can refer to the theoretical or practical understanding of a subject;*
    - *knowledge is produced by socio-cognitive aggregates*
      - *(mainly humans)*
      - *and is structured according to our understanding of how human reasoning and logic works.*

- The aim of **knowledge engineering** was formulated, in 1983, by an originator of the concept, Edward A. Feigenbaum [Feigenbaum83]:
  - **knowledge engineering** is an engineering discipline
  - that involves integrating knowledge into computer systems
  - in order to solve complex problems
  - normally requiring a high level of human expertise.

- Knowledge engineering focus on
  - continually building up (acquire) large, shared data bases (i.e., knowledge bases),
  - their continued maintenance,
  - testing the validity of the stored 'knowledge',
  - continued experiments with respect to knowledge representation,
  - etcetera.

- Knowledge engineering can, perhaps, best be understood in contrast to algorithmic engineering:
  - In the latter we seek more-or-less conventional, usually imperative programming language expressions of algorithms
    - whose algorithmic structure embodies the knowledge
    - required to solve the problem being solved by the algorithm.
  - The former seeks to solve problems based on an interpreter inferring possible solutions from logical data. This logical data has three parts:
    - a collection that "mimics" the semantics of, say, the imperative programming language,
    - a collection that formulates the problem, and
    - a collection that constitutes the knowledge particular to the problem.
- We refer to `[BjornerNilsson1992]`.

• • •

- Domain science & engineering is not aimed at
  - letting the computer solve problems based on
  - the knowledge it may have stored.
- Instead it builds models based on knowledge of the domain.

- Finally,
  - the domains to which we are applying 'our form of' domain analysis
  - are domains which focus on spatio-temporal phenomena.
  - That is, domains which have concrete renditions:
    - air traffic,
    - banks,
    - container lines,
    - manufacturing,
    - pipelines,
    - railways,
    - road transport,
    - stock exchanges,
    - etcetera.

- In contrast
  - one may claim that the domains
  - described in classical ontologies and knowledge representations
  - are mostly conceptual:
    - mathematics,
    - physics,
    - biology,
    - etcetera.

5.3.3. Database Analysis

- There are different, however related "schools of database analysis".
  - DSD: the Bachman (or data structure) diagram model [Bach69],
  - RDM: the relational data model [codd70] and
  - ER: entity set relationshp model [peter-chen-1976] "schools".
  - DSD and ER aim at graphically specifying database structures.
  - Codd's RDM simplifies the data models of DSD and ER while offering two kinds of languages with which to operate on RDM databases: SQL and Relational Algebra.
- All three "schools" are focused
  - more on data modelling for databases
  - than on domain modelling both endurant and perdurant entities.

5.3.4. Domain Analysis

- Domain analysis, or product line analysis (see below), as it was then conceived in the early 1980s by James Neighbors [JamesNeighbors1980,JamesNeighbors1984],
  - is the analysis of related software systems in a domain
  - to find their common and variable parts.
  - It is a model of wider business context for the system.
- This form of domain analysis turns matters "upside-down":
  - it is the set of software "systems" (or packages)
  - that is subject to some form of inquiry,
  - albeit having some domain in mind,
  - in order to find common features of the software
  - that can be said to represent a named domain.

- In this section we shall mainly be comparing the TripTych approach to domain analysis to that of Reubén Prieto-Dĩaz's approach [Prieto-Diaz:1987,Prieto-Diaz:1990,Prieto-Diaz:1991].
- Firstly, our understanding of domain analysis basically coincides with Prieto-Dĩaz's.
- Secondly, in, for example, [Prieto-Diaz:1987], Prieto-Dĩaz's domain analysis is focused on the very important stages that precede the kind of domain modelling that we have described:
  - major concerns are
    - selection of what appears to be similar, but specific entities,
    - identification of common features,
    - abstraction of entities and
    - classification.
  - Selection and identification is assumed in our approach, but we suggest to follow the ideas of Prieto-Dĩaz.
  - Abstraction (from values to types and signatures) and classification into parts, materials, actions, events and behaviours is what we have focused on.

- All-in-all we find Prieto-Dĩaz's work very relevant to our work:
  - relating to it by providing guidance to pre-modelling steps,
  - thereby emphasising issues that are necessarily informal,
  - yet difficult to get started on by most software engineers.
- Where we might differ is on the following:
  - although Prieto-Dĩaz does mention a need for domain specific languages,
  - he does not show examples of domain descriptions in such DSLs.
  - We, of course, basically use mathematics as the DSL.
- In our approach
  - we do not consider requirements, let alone software components,
  - as do Prieto-Dĩaz,

  but we find that that is not an important issue.

5.3.5. Domain Specific Languages

- Martin Fowler[20] defines a *Domain-specific language* (DSL)
  - *as a computer programming language*
  - *of limited expressiveness*
  - *focused on a particular domain* [dsl2010].
- Other references are [dsl2005,dsl2001].
- Common to [dsl2001,dsl2005,dsl2010] is
  - that they define a domain in terms of classes of software packages;
  - that they never really "derive" the DSL from a description of the domain; and
  - that they certainly do not describe the domain in terms of that DSL, for example, by formalising the DSL.

[20] http://martinfowler.com/dsl.h

5.3.6. Feature-oriented Domain Analysis (FODA)

- Feature oriented domain analysis (FODA)
  - is a domain analysis method
  - which introduced feature modelling to domain engineering
  - FODA was developed in 1990 following several U.S. Government research projects.
  - Its concepts have been regarded as critically advancing software engineering and software reuse.
- The US Government supported report [KyoKang+et.al.:1990] states: "*FODA is a necessary first step*" for software reuse.

- To the extent that
  - TripTych domain engineering
  - with its subsequent requirements engineering

  indeed encourages reuse at all levels:
  - domain descriptions and
  - requirements prescription,

  we can only agree.
- Another source on FODA is [Czarnecki2000].
- Since FODA "leans" quite heavily on 'Software Product Line Engineering' our remarks in that section, next, apply equally well here.

5.3.7. Software Product Line Engineering

- Software product line engineering,
  earlier known as domain engineering,
  - is the entire process of **reusing domain knowledge** in the production of new software systems.
- Key concerns of **software product line engineering** are
  - **reuse**,
  - the building of repositories of **reusable software components**, and
  - **domain specific languages** with which to more-or-less automatically build software based on **reusable software components**.

- These are not the primary concerns of TripTych domain science & engineering.
  - But they do become concerns as we move from domain descriptions to requirements prescriptions.
  - But it strongly seems that software product line engineering is not really focused on the concerns of domain description — such as is TripTych domain engineering.
  - It seems that software product line engineering is primarily based, as is, for example, FODA: Feature-oriented Domain Analysis, on analysing features of software systems.
  - Our [DomainsSimulatorsDemos2011] puts the ideas of software product lines and model-oriented software development in the context of the TripTych approach.

5.3.8. Problem Frames

- The concept of **problem frames** is covered in [mja2001a].
- Jackson's prescription for software development focus on the "triple development" of descriptions of
  - the **problem world**,
  - the **requirements** and
  - the **machine** (i.e., the **hardware** and **software**) to be built.
- Here **domain analysis** means, the same as for us, the **problem world analysis**.

- In the **problem frame** approach the software developer plays three, that is, all the `TripTych` rôles:
  - **domain engineer**,
  - **requirements engineer** and
  - **software engineer**,

  "all at the same time",
- iterating between these rôles repeatedly.
- So, perhaps belabouring the point,
  - **domain engineering** is done only to the extent needed by the prescription of **requirements** and the **design** of **software**.
- These, really are minor points.

- But in "restricting" oneself to consider
  - only those aspects of the domain which are mandated by the requirements prescription
  - and software design

  one is considering a potentially smaller fragment [Jackson2010Facs] of the domain than is suggested by the TripTych approach.
- At the same time one is, however, sure to
  - consider aspects of the domain
  - that might have been overlooked when pursuing domain description development
  - in the "more general" TripTych approach.

5.3.9. Domain Specific Software Architectures (DSSA)

- It seems that the concept of DSSA
  - was formulated by a group of ARPA[21] project "seekers"
  - who also performed a year long study (from around early-mid 1990s);
  - key members of the DSSA project were Will Tracz, Bob Balzer, Rick Hayes-Roth and Richard Platek [dom:Trasz:1994].
- The [dom:Trasz:1994] definition of domain engineering is *"the process of creating a DSSA:*
  - *domain analysis and domain modelling*
  - *followed by creating a software architecture*
  - *and populating it with software components."*

[21] ARPA: The US DoD Advanced Research Projects Agency

- This definition is basically followed also by [Mettala+Graham:1992,Shaw+Garlan:1996,Medvidovic+Colbert:2004].
- Defined and pursued this way, DSSA appears,
  - notably in these latter references, to start with
  - the analysis of software components, "per domain",
  - to identify commonalities within application software,
  - and to then base the idea of software architecture
  - on these findings.

- Thus DSSA turns matter "upside-down" with respect to TripTych requirements development
  - by starting with software components,
  - assuming that these satisfy some requirements,
  - and then suggesting domain specific software
  - built using these components.
- This is not what we are doing:
  - we suggest that requirements
    - can be "derived" systematically from,
    - and related back, formally to domain descriptionss
    - without, in principle, considering software components,
    - whether already existing, or being subsequently developed.

- Of course, given a **domain description**
  - it is obvious that one can develop, from it, any number of **requirements prescription**s
  - and that these may strongly hint at shared, (to be) implemented **software component**s;
- but it may also, as well, be the case
  - two or more **requirements prescription**s
  - "derived" from the same **domain description**
  - may share no **software component**s whatsoever !

- It seems to this author that had the **DSSA** promoters
  - based their studies and practice on also using formal specifications,
  - at all levels of their study and practice,
  - then some very interesting insights might have arisen.

5.3.10. Domain Driven Design (DDD)

- Domain-driven design (DDD)[22]
  - *"is an approach to developing software for complex needs*
  - *by deeply connecting the implementation to an evolving model of the core business concepts;*
  - *the premise of domain-driven design is the following:*
    - *placing the project's primary focus on the core domain and domain logic;*
    - *basing complex designs on a model;*
    - *initiating a creative collaboration between technical and domain experts to iteratively cut ever closer to the conceptual heart of the problem."*[23]

[22] Eric Evans: http://www.domaindrivendesign.org/
[23] http://en.wikipedia.org/wiki/Domain-driven_design

- We have studied some of the DDD literature,
  - mostly only accessible on the Internet, but see also [Haywood2009],
  - and find that it really does not contribute to new insight into domains such as wee see them:
  - it is just "plain, good old software engineering cooked up with a new jargon.

5.3.11. Unified Modelling Language (UML)

- Three books representative of UML are [Booch98,Rumbaugh98,Jacobson99].
- The term **domain analysis** appears numerous times in these books,
  - yet there is no clear, definitive understanding
  - of whether it, the **domain**, stands for entities in the domain such as we understand it,
  - or whether it is wrought up, as in several of the 'approaches' treated in this section, to wit, Sects. 1, 1, 1, 1, 1 and 1, with
    - either **software design** (as it most often is),
    - or **requirements prescription**.

- Certainly, in UML,
  - in [Booch98,Rumbaugh98,Jacobson99] as well as
  - in most published papers claiming "adherence" to UML,
  - that domain analysis usually
    - is manifested in some UML text
    - which "models" some requirements facet.
  - Nothing is necessarily wrong with that,
  - but it is therefore not really the TripTych form of domain analysis
    - with its concepts of abstract representations of endurant and perdurants,
    - with its distinctions between domain and requirements, and
    - with its possibility of "deriving"
      - requirements prescriptions from
      - domain descriptions.

- The UML notion of class diagrams is worth relating to our structuring of the domain.
  - Class diagrams appear to be inspired by [] Bachman, 1969]Bach69 and [] Chen, 1976]peter-chen-1976.
  - It seems that
    - each part sort — as well as other than part (or material) sorts — deserves a class diagram (box),
    - that (assignable) attributes — as well as other non-part (or material) types — are written into the diagram box —
    - as are action signatures — as well as other function signatures.

- Class diagram boxes are line connected with annotations where some annotations are
  - as per the mereology of the part type and the connected part types
  - and others are not part related.
- The class diagrams are said to be object-oriented
  - but it is not clear how objects relate to parts
  - as many are rather implementation-oriented quantities.

• All this needs looking into a bit more, for those who care.

5.3.12. Requirements Engineering

- There are in-numerous books and published papers on requirements engineering.
  - A seminal one is [AvanLamsweerde2009].
  - I, myself, find [SorenLauesen2002] full of very useful, non-trivial insight.
  - [Dorfman+Thayer:1997:IEEEComp.Soc.Press] is seminal in that it brings a number or early contributions and views on requirements engineering.

- Conventional text books, notably `[Pfleeger2001,Pressman2001,Sommerville2006]` all have their "mandatory", yet conventional coverage of **requirements engineering**.
  - None of them "derive" requirements from domain descriptions,
    - yes, OK, from domains,
    - but since their description is not mandated
    - it is unclear what "the domain" is.
  - Most of them repeatedly refer to **domain analysis**
    - but since a written record of that **domain analysis** is not mandated
    - it is unclear what "domain analysis" really amounts to.

- Axel van Laamsweerde's book [AvanLamsweerde2009] is remarkable.
  - Although also it does not mandate descriptions of domains
  - it is quite precise as to the relationships between domains and requirements.
  - Besides, it has a fine treatment of the distinction between goals and requirements,
  - also formally.
- Most of the advices given in [SorenLauesen2002]
  - can beneficially be followed also in
  - TripTych requirements development.
- Neither [AvanLamsweerde2009] nor [SorenLauesen2002] preempts TripTych requirements development.

Summary of Comparisons:

- It should now be clear from the above that
  - basically only Jackson's *problem frames* really take
    - the same view of **domains** and,
    - in essence, basically maintain similar relations between
      - **requirements prescription** and
      - **domain description**.
  - So potential sources of, we should claim, mutual inspiration
    - ought be found in one-another's work —
    - with, for example, [ggjz2000,Jackson2010Facs],
    - and the present document,
    - being a good starting point.

- But none of the referenced works make the distinction between
  - discrete endurants (parts) and their
  - qualities, with their further distinctions between
    - **unique identifiers,**
    - **mereology** and
    - **attributes.**
- And none of them makes the distinction between
  - **parts** and
  - **materials.**
- Therefore our contribution can include
  - the mapping of parts into behaviours
  - interacting as per the part mereologies

  as highlighted in the **process schema**s of Sect. 5.4.4 Slides 236–238.

5.4. What Have We Not Covered ?

- The concept of domain science & engineering,
  - such as basically formulated in this seminar,
  - was covered, in some other form, in [TheSEBook3wo].
  - The domain analysis development stage,
    - such as covered in this seminar,
    - is, however, new.

5.4.1. Domain Facets

- A development stage
  - that has not been covered in this seminar
  - is that of **domain facet description**.
  - It was covered in [TheSEBook3wo]
  - and then further developed in [dines:facs:2008].
- In [dines:ictac:2007] we list
  - some research challenges based on that 2007–2008
  - understanding of **domain science & engineering**.

- By a **'domain facet'** we understand
  - one amongst a finite set of generic ways of analysing a domain:
  - a view of the domain,
  - such that the different facets cover conceptually different views,
  - and such that these views together cover the domain.

- We consider the following facets:
  - intrinsics,
  - support technology,
  - organisation & management,
  - rules & regulations,
  - script and
  - behaviour.

5.4.1.1 The Intrinsics Facet:

- By 'intrinsics' of a domain we understand
  - those phenomena and concepts of a domain which are basic to any of the other facets (listed above and cursorily described below),
  - with such domain intrinsics initially covering at least one specific, hence named, stake-holder view.

5.4.1.2 The Support Technology Facet:

- By the 'support technology facet' of a domain we understand
  - ways and means of implementing certain observed phenomena.

5.4.1.3 The Organisation & Management Facet:

- By the 'organisation & management facet' we understand such people (such decisions)
  - who (which) determine, formulate and thus set standards (cf. rules and regulations next) concerning
    - strategic, tactical and operational

    decisions;
  - who ensure that these decisions are passed on to (lower) levels of management, and to floor staff;
  - who make sure that such orders, as they were, are indeed carried out;
  - who handle undesirable deviations in the carrying out of these orders cum decisions;
  - and who "backstop" complaints from lower management levels and from floor staff.

5.4.1.4 The Rules & Regulations Facet:

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

5.4.1.5 The Script Facet:

- By the *'script facet'* we understand
  - the structured, almost, if not outright, formally expressed, wording of
  - somehow structured collection of rules or regulations
  - that has legally binding power,
  - that is, which may be contested in a court of law.

5.4.1.6 The Behaviour Facet:

- By the 'human behaviour facet' we understand
  - ◈ any of a quality spectrum of carrying out assigned work:
    - ◎ from *careful*, *diligent* and *accurate*,

    via

    - ◎ *sloppy* dispatch, and
    - ◎ *delinquent* work,

    to

    - ◎ outright *criminal* pursuit.
- By the 'support technology behaviour facet' we understand the ability of that technology
  - ◈ to faithfully carry out the prescriptions laid down in scripts for that technology, or

    - ◎ occasionally,
    - ◎ repeatedly, or
    - ◎ permanently

    failing to do so.

- The concept of domain facets is one of pragmatics.
  - It cannot be formalised.
  - The borderlines between the entities described through the various facet views are not precise.
  - The list of domain facets is a check-list.
  - The domain analyser & describer may be well served in checking the domain "against" this list.
  - The domain analyser & describer need not structure a domain description according to the check-list.
  - But doing so may sometimes help the reader of the domain description.

- We shall not cover the concept of domain facets further
  - than saying that the presentation of this concept in
  - [dines:facs:2008] and [dines:ictac:2007]
  - need be reviewed in the context of a much sharper understanding now
  - of the concept of domain analysis.

5.4.2. Laws of Domain Description Prompts

- Domain description prompts result in texts.
  - Typically `observe_part_sorts` applies to a composite part, p:P, and yield descriptions of one or more part sorts: p−1:P_1,p2:P_2,...,pm:P_m.
  - Let p−i:P_i,p$_j$:P_j,...,p$_k$:P_k (of these) be composite.
  - Now `observe_part_sorts`(p$_i$) and `observe_part_sorts`(p$_j$), etc., can be applied and yield texts *text*$_i$, respectively *text*$_j$.
  - A law of domain description prompts now expresses that the order in which the two or more observers is applied is immaterial, that is, they commute.
- In [Kiev:2010ptII] we made an early exploration of such laws of domain description prompts.
- More work, hear also next, need be done.

5.5. Future Work

- The previous sections indicated a number of topics that need further study. For example:
  - *Order of Analysis & Description*,
  - *Laws of Domain Description Prompts* and
  - *A Formal Understanding of Domain Facets*.
- Below we briefly mention some further topics.

5.5.1. Analysis of Perdurants

- We plan to carry out a study of perdurants, as detailed as that of our study of endurants.
- The difficulty, as we see it, is the choice of formalisms:
  - whereas the basic formalisms for the expression of endurants and their qualities was type theory and simple functions and predicates,
  - there is no such simple set of formal constructs that can "carry" the expression of behaviours.
    - Besides the textual `CSP`, `[Hoare85+2004]`, there is graphic notations of
    - Petri Nets, `[m:petri:wr09]`,
    - Message Sequence Charts, `[MSCall]`,
    - State-charts, `[Harel87]`, and others.

5.5.2. Commensurate Discrete and Continuous Models

- Section 5.3.7 Slides 218–220 hinted at
  - co-extensive descriptions of discrete and continuous behaviours,
  - the former in, for example, RSL,
  - the latter in, typically, the calculus mathematics of partial different equations (PDEs).
  - The problem that arises in this situation is the following:
    - there will be, say variable identifiers, e.g., x, y, ..., z
    - which in the RSL formalisation has one set of meanings, but
    - which in the PDE "formalisation" has another set of meanings.

- Current formal specification languages[24] do not cope with continuity.

- Some research is going on.
- But to substantially cover, for example, the proper description of laminar and turbulent flows in networks (e.g., pipelines, Example 50 on slide 219) requires more substantial results.

[24] Alloy [alloy], Event B [JRAbrial:TheBBooks], RSL [RSL], VDM-SL [e:db:Bj78bwo,e:db:Bj82b,JohnFitzgerald+PeterGormLarsen], Z, etc.

5.5.3. Interplay between Parts and Materials

- Examples 37 on slide 166, 38 on slide 168, 39 on slide 172, 40 on slide 175 and 50 on slide 219 revealed but a small fraction of the problems that may arise in connection with modelling the interplay between parts and materials.
- Subject to proper formal specification language and, for example PDE specification we may expect more interesting
  - laws, as for example those of Examples 39 on slide 172, 40 on slide 175,
  - and even proof of these as if they were theorems.
- Formal specifications have focused on verifying properties of requirements and software designs.
- With co-extensive (i.e., commensurate) formal specifications of both discrete and continuous behaviours we may expect formal specifications to also serve as bases for predictions.

5.5.4. Towards a Mathematical Model of Domain Analysis & Description

- There are two aspects to a precise description of the **'domain analysis prompt'**s and **'domain description prompt'**s.
  - There is that of describing
    - the individual prompts
    - as if they were "machine instructions"
    - for an albeit strange machine;
  - and there is that of describing
    - the interplay between prompts:
      - the sequencing of **'domain description prompt'**s
      - as determined by the outcome of the **'domain analysis prompt'**s.
- We have described and formalised the latter in [[] Processes]2013da-processes.

- And we are in the midst of describing and formalising the former in [[ ] Prompts]2013da-prompts.

5.5.5. Domains and Galois Connections

- Section 2.3 very briefly mentioned that formal concepts form Galois Connections.
- In the seminal `[GanterWille:ConceptualAnalysis1999]` a careful study is made of this fact and beautiful examples show the implications for domains.
- It seems that our examples have all been too simple.
- They do not easily lead on to the "discovery" of "new" domain concepts from appropriate concept lattices.
- Further study need be done.

5.5.6. Domain Theories

- An ultimate goal of domain science & engineering is to prove properties of domains.
  - Well, maybe not properties of domains, but then at least properties of domain descriptions.
- If one can be convinced that a posited domain description indeed is a faithful description of a domain,
  - then proofs of properties of the domain description
  - are proofs of properties of that domain.
- Ultimately domain science & engineering must embrace such studies of *laws of domains*.
- Here is a fertile ground for zillions of Master and PhD theses!

Example 56 . A Law of Train Traffic:

- *Let a transport net, n:N, be that of a railroad system.*
  - *Hubs are train stations.*
  - *Links are rail lines between stations.*
  - *Let a train timetable record train arrivals and train departures from stations.*
  - *And let such a timetable be modulo some time interval, say typically 24 hours.*

- *Now let us (idealistically) assume*
  - *that actual trains arrive at and depart from train stations according the train timetable and*
  - *that the train traffic includes all and only such trains as are listed in the train timetable.*
- *Now a law of train traffic expresses*
  - *Over the modulo time interval of a train timetable it is the case that*
    - *the number of trains arriving at a station*
    - *minus the number of trains ending their journey at that station*
    - *plus the number of trains starting their journey at that station*
    - *equals number of trains departing from that station.*

5.5.7. Precise Descriptions of Man-made Domains

- The focus on the principles, techniques and tools of domain analysis & description has been such domains in which humans play an active rôle.
  - Formal descriptions of domains may serve to
    - prove properties of domains,
    - in other words, to understand better these domains, and to
    - validate requirements derived from such domain descriptions, and
    - thereby to ensure that software derived from such requirements
      - is not only correct,
      - but also meet users expectations.

- Improved understanding of man-made domains —
  - without necessarily leading to new software

  — may serve to
  - improve the "business processes" of these domains,
  - make them more palatable for the human actors,
  - make them more efficient wrt. resource-usage.

- Descriptions of domains are descriptions of the syntax and semantics of the technical languages used in speaking about and in the domain.

- The domain analysis required for the design of programming languages is based on computability: mathematical logic and recursive function theory.
- The domain analysis required for "real-world" domains is not based on computability: that "world" is not computable.
- Requirements engineering based on domain descriptions is based on deriving computable subsets of refined domain descriptions.
- The classical theory and practice of programming language semantics and compiler development [p:db:Bj77b] and [[]Part VII (Chapters 16–19)]TheSEBook2wo can now be further developed into a theory and practice for deriving general software from formal domain descriptions [dines:ugo65:2008].

- Physicists study 'Mother Nature', the world without us.
- Domain scientists study man-made part and material based universes with which we interact — the world within and without us.
- Classical engineering builds on laws of physics to design and construct
  - buildings,
  - chemical compounds,
  - machines and
  - E&E products.

- So far software engineers have not expressed software requirements on any precise description of the basis domain.
- This seminar strongly suggests such a possibility.
- Regardless:
  - it is interesting to also formally describe domains;
  - and, as shown, it can be done.

5.6. Acknowledgements

- The writing of this seminar was begun in mid December 2012 after a PhD lecture tour of China where I presented a previous version of 'Domain Analysis'.
- Versions prior to the China PhD lectures were presented in the years 2007–2012:[25] Nancy, Graz, Saarland, Edinburgh, St. Andrews, Glasgow, Tokyo, Vienna, Budapest, Uppsala and Paris.
- The specific impetus to do a complete rewrite derived from my 'Domain Analysis' lectures at the University of Bergen, Norway, May 2012 and from remarks and observations made there by Prof. Magne Haveraaen.
- I especially thank Magne Haveraaen and also my many PhD lecture hosts around Europe, Japan and China for their kind support.

[25] You may find a list of places and references to these earlier versions at http://www2.compute.dtu.dk/~dibj/node4.html.

Thanks. Questions ?