.

Lecture Day 1, Lecture 1

Opening

Course Agenda

- **1. Tuesday: 6.11: Course Appetizer + Example**
  - **Lectures 1–2:** [Bjø18a] Sects. 1 & 8 pp. 2–27, pp. 285–382
  - **Course Project:** up to 2 hours
- **2. Wednesday: 7.11: Entities, Endurants and Perdurants**
  - **Lectures 3–4:** [Bjø18a] Sect. 2 pp. 28–53
  - **Course Project:** up to 2 hours
  - **Excursion** to APM Terminals at SECT, the Waigaoqiao port area of Shanghai
- **3. Thursday: 8.11: The Endurant Analysis Calculus**
  - **Lectures 5–6:** [Bjø18a] Sects. 3.1–3.3, 3.5–3.7 pp. 55–112
  - **Course Project:** up to 2 hours
- **4. Friday: 9.11: The Endurant Description Calculus**
  - **Lectures 7–8:** [Bjø18a] Sect. 4 pp. 114–140
  - **Course Project:** up to 2 hours
- **5. Monday: 12.11: Unique Identifiers, Mereology**
  - **Lectures 9–10:** [Bjø18a] Sects. 5., 5.1–5,2 pp. 142–160
  - **Course Project:** up to 2 hours

- **6. Wednesday: 14.11: Attributes and summary**
  - **Lectures 11–12:** [Bjø18a] Sects. 5.3–5.6 pp. 162–212
  - **Course Project:** up to 2 hours
- **7. Friday: 16.11: Transcendental Deduction**
  - **Lectures 13–14:** [Bjø18a] Sect. 6. pp. 214–221
  - **Course Project:** up to 2 hours
- **8. Monday: 19.11: Perdurants, I**
  - **Lectures 15–16:** [Bjø18a] Sects. 7.1–7.3 pp. 223–231
  - **Course Project:** up to 2 hours
- **9. Wednesday: 21.11: Perdurants, II**
  - **Lectures 17–18:** [Bjø18a] Sects. 7.4–7.6 pp. 232–283
  - **Course Project:** up to 2 hours
- **10. Thursday: 22.11: Course Review**
  - **Lecture 19:** [Bjø18a] Sect. 9 pp. 384–410
  - **Course Project:** up to 2 hours: Evaluation

A Domain Analysis & Description Method

Principles, Techniques and a Modeling Language

Dines Bjørner
Fredsvej 11, DK-2840 Holte and DTU, DK-2800 Kgs. Lyngby, Denmark.
e-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~dibj

August 22, 2018: 16:59

The ECNU November 2018 Lectures

0 Summary

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

- These are
  - **endurants** (“still”), existing in space,
  - as well as **perdurants** (“alive”), existing also in time.
- Emphasis is placed on **“human-assistedness”**,
  - that is, that there is *at least one (man-made)* **artifact**
  - and that **humans** are a primary cause for
    - change of endurant **states**
    - as well as perdurant **behaviours**
- *Domain science & engineering* marks a new area of *computing science.*
  - Just as we are *formalising*
  - the *syntax and semantics of programming languages*,
  - so we are *formalising*
  - the *syntax and semantics of human-assisted domains.*

- Just as *physicists* are studying *mother nature*,
  - endowing it with *mathematical models*,
  - so we, *computing scientists*, are studying these *domains*,
  - endowing them with *mathematical models*,
- A difference between the endeavours of physicists and ours lies in the models:
  - the physics models are based on
    classical mathematics, *differential equations* and *integrals*, etc.,
  - our models are based on
    mathematical logic, *set theory*, and *algebra.*

1 Introduction

1.1 Foreword

- Dear student!
  - You are about to embark on a journey.
  - The lectures in front of us are many !
  - But it is not the number, 410, of lecture slides,
  - or duration of my unfolding the slides that I am referring to.

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

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

- But there is an even more fundamental issue "at play" here.
  - It is that of philosophy.
  - Let us briefly review some aspects of philosophy.
- *Metaphysics*
  - is a branch of *philosophy* that explores fundamental questions, including the nature of concepts like
  - *being*, *existence*, and *reality* ■[1]
- Traditional metaphysics seeks to answer,
  - in a "suitably abstract and fully general manner",
  - the questions:
    - *What is there ?* and
    - *And what is it like ?*[2].

[1] ■ is used to signal the end of a characterisation, a definition, or an example.
[2] https://en.wikipedia.org/wiki/Metaphysics

- Topics of metaphysical investigation include
  - existence,
  - objects and their properties,
  - space and time,
  - cause and effect, and
  - possibility.
- *Epistemology*
  - is the branch of philosophy concerned with
  - the theory of knowledge[3] ■

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

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

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

- We shall base some of our modelling decisions of Kai Sørlander's Philosophy [Sør94, Sør97, Sør02, Sør16].
- A main contribution of Kai Sørlander is, on the philosophical basis of the *possibility of truth*
  (in contrast to Kant's *possibility of self-awareness*),
  - to *rationally* and *transcendentally deduce*
  - **the absolutely necessary conditions for describing any world**.

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

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

1.2 Precursor

- The present lectures are based on a revision of the published [Bjø16e].
  - The revision considerably simplifies and considerably extends the domain analysis & description calculi of [Bjø16e].
  - The major revision that prompts this complete rewrite is due to a serious study of Kai Sørlander's Philosophy.
  - As a result we extend [Bjø16e]'s ontology of endurants: describable phenomena that exists in space, to not only cover those of **physical phenomena**, but also those of **living species**, notably **humans**, and, as a result of that, our understanding of discrete endurants is refined into those of **natural parts** and **artifacts**.
  - A new contribution is that of **intentional "pull"** akin to the *gravitational pull* of physics.

  - Both these lectures and [Bjø16e] are the result of extensive "non-toy" example case studies, see Example 1 on Slide 36.
  - These were carried out in the years since [Bjø16e] was first submitted (i.e., 2014).
  - The present lectures omit the extensive introduction and closing of [Bjø16e], sections, as well as the very many "interwoven" examples of [Bjø16e].
  - Instead Sect. 8 (Slides 285–382) shows one, rather comprehensive, larger example that illustrates many aspects of the methodology.
  - Most notably, however, is a clarified view on the transition from **parts** to **behaviours**, a **transcendental deduction** from *domain space* to *domain time*.

1.3 What are these Lectures About ?

- We present a *method* for *analysing &*[5] *describing domains.*
- By a **domain** we shall understand
  - a **rationally describable** segment of
  - a **human assisted** reality, i.e., of the world,
    - its **physical parts**,
      - **natural** ["God-given"] and
      - **artifactual** ["man-made"],
    - and **living species**:
      - **plants** and
      - **animals** including, predominantly, **humans**.

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

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

Definition 1 Domain Description: By a **domain description** we shall understand

- a combination of **narration** and **formalisation** of a domain.
- A **formal specification** is a collection of
  - *sort*, or *type* definitions,
  - *function* and *behaviour* definitions,
  - together with *axioms* and *proof obligations* constraining the definitions.
- A **specification narrative** is a natural language text which in terse statements introduces
  - the names of (in this case, the domain),
  - and, in cases, also the definitions, of
  - sorts (types), functions, behaviours and axioms;
  - not anthropomorphically, but by emphasizing their properties ■

- *Domain descriptions* are (to be) void of any reference to future, contemplated software, let alone IT systems, that may support entities of the domain.
  - As such *domain models*[6]
    - can be studied separately,
    - for their own sake,
    - for example as a basis for investigating possible domain theories, or
  - can, subsequently, form the basis for requirements engineering
  - with a view towards development of ('future') software, etc.
- Our aim is to provide a method for the precise analysis and the formal description of domains.

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

1.4 Structure of these Lectures

- Sections 2–7 form the core of these lectures.
- Section 8 brings a "large" example that is forward-referred to in Sects. 2–7 and refers (backwards) to Sects. 2–7.
- Section 2 introduces the first concepts of domain phenomena: *endurants* and *perdurants*. Their characterisation, in the form of "definitions", cannot be mathematically precise, as is usual in computer science lectures.

- Section 3
  - analyses the so-called *external qualities* of *endurants* into
    - *natural parts*,
    - *components*,
    - *living species* and
    - *structures*,
    - *materials*,
    - *artifacts*.
  - In doing so it covers the *external qualities analysis prompts*.
- Section 4 covers the *external qualities description prompts*
- Section 5
  - analyses the so-called *internal qualities* of *endurants* into
    - *unique identification*,
    - *mereology* and
    - *attributes*.
  - In doing so it covers both the *internal qualities analysis prompts* and the *internal qualities description prompts*

- Sections 3–5 have covered
  what these lectures has to say about *endurants.*
- Section 6 "bridges" Sects. 3–5 and Sect. 7
  by introducing the concept of *transcendental deduction.*
  These deductions allow us to "transform" *endurants* into *perdurants*:
  "passive" entities into "active" ones.

- The essence of Sects. 6–7 is to
  - *"translate"* endurant parts
  - into perdurant behaviours.
- Section 7 – although "only" half as long as the three sections on *endurants* –
  covers the analysis & description method for *perdurants.*
  - We shall model perdurants, notably *behaviours*,
    in the form of CSP [Hoa85].
  - Hence we introduce the CSP notions of
    - *channels* and
    - channel *input/output.*
  - Section 7 then "derives"
    the types of the behaviour arguments from
    the internal endurant qualities.
- Section 9 summarises the achievements and discusses open issues.

.

Lecture Day 2, Lectures 3–4

Entities, Endurants and Perdurants

2 Entities: Endurants and Perdurants

2.1 A Generic Domain Ontology

- Figure 1 on Slide 30 shows a so-called "upper ontology" for manifest domains.
  - By ontologies we shall here understand
  - *formal representations*
    of a set of concepts within a domain
    and the relationships between those concepts.

- Kai Sørlander's Philosophy justifies our organising the *entities* of any describable domain, for example[7], as follows:
  - There are
    - *describable* phenomena and there are
    - phenomena that we cannot describe.
    - The former we shall call *entities*.
  - The *entities* are
    - either *endurants*, "still" entities – existing in *space*,
    - or *perdurants*, "alive" entities – existing also in *time*.

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

.

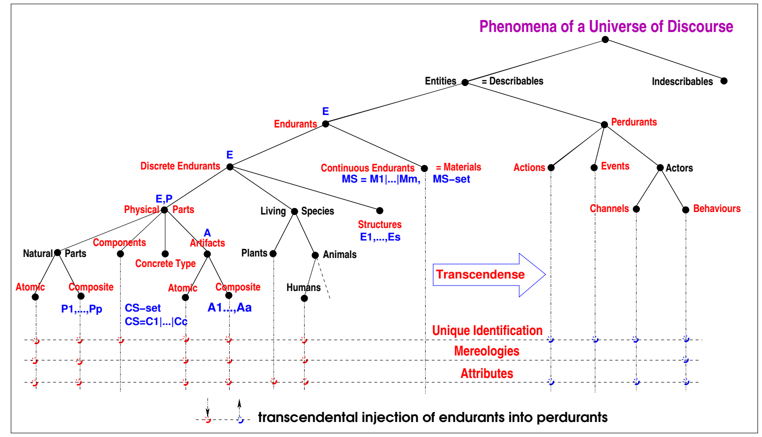


Figure 1: An Upper Ontology for Domains

- *Endurants* are
  - either *discrete*
  - or *continuous* –
  - in which latter case we call them *materials*[8].
- *Discrete* endurants are
  - *physical parts*,
  - *living species*, or are
  - *structures*.

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

- *Structures* consist of one or more endurants.
- *Physical parts* are
  - either *naturals*,
  - or *artifacts*, i.e. man-made,
  - or *components*[9],
  - or *sets of parts*.
- *Living Species* are
  - either *plants*
  - or *animals*.
- Among animals we have the *humans*.
- *Naturals* and *artifacts* are
  - either atomic or
  - or composite – consisting of two or more differently typed parts.

[9] Whether a discrete endurant as we shall soon see, is treated as a part or a component is a matter of pragmatics. Again cf. Footnote 8.

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

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

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

- Our reference, here, to Kai Sørlander's Philosophy, is very terse.
  - We refer to a detailed research report:
  - **A Philosophy of Domain Science & Engineering**,
  - **http://www.imm.dtu.dk/~dibj/2018/philosophy/filo.pdf**,
  - for carefully reasoned arguments.
- That report is under continued revision:
  - It reviews the **domain analysis & description** method;
  - translates many of Kai Sørlander's arguments
  - and relates, in detail, the "options"
  - of the **domain analysis & description** approach to Sørlander's Philosophy.

2.2 Universes of Discourse

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

Example 1 Universes of Discourse:

- We refer to a number of Internet accessible experimental reports[10] of descriptions of the following domains:
  - **railways** [Bjø00, BGP02, Bjø03],
  - **container shipping** [Bjø07],
  - **stock exchange** [Bjø10c],
  - **document systems** [Bjø17b],
  - **oil pipelines** [Bjø13a],
  - **"The Market"** [Bjø02],
  - **Web systems** [Bjø10b],
  - **weather information** [Bjø16d],
  - **credit card systems** [Bjø16a],
  - **urban planning** [Bjø17c],
  - **swarms of drones** [Bjø17a],
  - et cetera, et cetera ■

[10] These are **draft** reports, more-or-less complete. The writing of these reports was finished when sufficient evidence, conforming or refuting one or another aspect of the **domain analysis & description method**.

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

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

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

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

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

2.3 Entities

Characterisation 1 Entity:

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

Analysis Prompt 1 *is_entity:*

- *The domain analyser analyses "things" (θ) into entities or non-entities.*
- *The method can thus be said to provide the* **domain analysis prompt***:*
  - *is_entity – where* is_entity(θ) *holds if θ is an entity*[11] ■
- is_entity is said to be a *prerequisite prompt* for all other prompts.

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

- The *entities* that we are concerned with
  - are those with which Kai Sørlander's Philosophy is likewise concerned.
  - They are the ones that are *unavoidable* in any
  - any description of any possible world.
- And then, which are those entities ?
  - In both [Sør94] and [Sør16]
  - rationally deduces that these entities
  - must be in *space* and *time*,
  - must satisfy laws of physics – like those of Newton and Einstein,
  - but among them are also *living species: plants* and *animals* and hence *humans*.
  - The *living species*, besides still
  - being in *space* and *time*, and satisfying laws of physics,
  - must satisfy further properties – which we shall outline later.

2.4 Endurants and Perdurants

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

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

Characterisation 2 Endurant:

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

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

Example 2 Geography Endurants:

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

Example 3 Railway System Endurants:

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

Analysis Prompt 2 `is_endurant`:

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

Characterisation 3 Perdurant:

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

Example 4 Geography Perdurants:

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

Example 5 Railway System Perdurants:

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

Analysis Prompt 3 *is_perdurant:*

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

.

Lecture Day 3, Lectures 5–6

Parts, Components and Materials Analysis

3 Endurants: Analysis of External Qualities

3.1 Discrete and Continuous Endurants

Characterisation 4 Discrete Endurant:

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

  in form or concept ■
- The notion of *discreteness* is not extended to *perdurants*.

Example 6 Discrete Endurants:

- The individual endurants of Example 3 on Slide 48 were all discrete.
- Here are examples of discrete endurants of pipeline systems.
  - A pipeline and
  - its individual units:
    - pipes,
    - valves,
    - pumps,
    - forks,
    - etc.

Analysis Prompt 4 *is_discrete:*

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

Characterisation 5 Continuous Endurant:

- By a **continuous endurant** we shall understand an endurant which is
  - prolonged, without interruption,
  - in an unbroken series or pattern ■
- We shall prefer to refer to continuous endurants as *materials*
- and otherwise cover materials in Sect. 3.6.
- The notion of a *continuous endurant* is not extended to *perdurants*.

Example 7 Materials:

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

Analysis Prompt 5 *is_continuous:*

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

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

3.2 Discrete Endurants

- We analyse discrete endurants into
  - *physical parts*,
  - *living species* and
  - *structures*.

3.2.1 Physical Parts

Characterisation 6 Physical Parts:

- By a *physical part* we shall understand
  - a discrete endurant existing in time and
  - subject to laws of physics,
  - including the *causality principle* and
  - *gravitational pull*[12].

[12]This characterisation is the result of our study of relations between philosophy and computing science, notably influenced by Kai Sørlander's Philosophy. We refer to our research report [Bjø18b, www.imm.dtu.dk/˜dibj/2018/philosophy/filo.pdf].

Analysis Prompt 6 `is_physical_part:`

- *The domain analyser analyses "things" (η) into physical part.*
- *The method can thus be said to provide the* **domain analysis prompt***:*
  - `is_physical_part` *– where* `is_physical_part`*(η) holds if η is a physical part* ■
- Section 3.3 continues our treatment of physical parts.

3.2.2 Living Species

Definition 2 Living Species, I:

- By a *living species* we shall understand
  - a discrete endurant existing in time,
  - subject to laws of physics, and
  - additionally subject to *causality of purpose*[13]
- Definition 8 on Slide 91 elaborates.

[13]See Footnote 12 on Slide 62.

Analysis Prompt 7 *is_living_species:*

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

- Living species
  - have a *form* they can *develop* to reach;
  - they are *causally* determined to *maintain* this form;
  - and they do so by *exchanging matter* with an *environment*.
  - We refer to [Bjø18b] for details.
- Section 3.4 continues our treatment of living species.

3.2.3 Structures

Definition 3 Structure: By a **structure** we shall understand

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

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

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

Example 8 Structures:

- As shown in the main example, Sect. 8,
  - a model of transport is structured into
  - a *road net structure* and
  - an *automobile structure*.
- The *road net structure* is then structured as a pair:
  - a *structure of hubs* and
  - a *structure of links*.
- These latter structures are then modelled as set of hubs, respectively links.

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

- The concept of *structure* is new.
- That is, it was not present in [Bjø16e].
  - Whether to analyse & describe a discrete endurant into a structure or a physical part is a matter of choice.
  - If we choose to analyse a discrete endurant into a *physical part* then it is because we are interested in endowing the part with *qualities*, the
    - unique identifiers,
    - mereology and
    - one or more attributes.
  - If we choose to analyse a discrete endurant into a *structure* then it is because we are **not** interested in endowing the endurant with *qualities*.

Analysis Prompt 8 *is_structure:*

- *The domain analyser analyse endurants, e, into structure entities as prompted by the* **domain analysis prompt***:*
  - *is_structure* ■
- We shall now treat the external qualities of discrete endurants:
  - *physical parts* (Sect. 3.3) and
  - *living species* (Sect. 3.4).
- After that we cover
  - *components* (Sect. 3.5),
  - *materials* (Sect. 3.6) and
  - *artifacts* (physical man-made parts, Sect. 3.3.2) .

- We remind the listener
  - that in this section, i.e. Sect. 3, we cover only the *analysis calculus* for *external qualities*;
  - the *description calculus* for *external qualities* is treated in Sect. 4.
- The analysis and description calculi for internal qualities is covered in Sect. 5.

3.3 Physical Parts

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

- The categorisation of physical parts into these four is pragmatic.
  - *Physical parts* follow from Kai Sørlander's Philosophy.
  - *Natural parts* are what Sørlander's Philosophy is initially about.
  - *Artifacts* follow from *humans* acting according to their *purpose* in making "physical parts".
  - *Components* is a simplification of natural and man-made parts.
  - *Set of parts* is a simplification of composite natural and composite man-made parts as will be made clear in Sect. 4.2.

3.3.1 Natural Parts

Characterisation 7 Natural Parts:

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

Definition 4 Natural Part:

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

3.3.2 Artifacts

Characterisation 8 Man-made Parts: Artifacts:

- Artifacts are man-made either discrete or continuous endurants.
  - In this section we shall only consider discrete endurants.
  - Man-made continuous endurants are not treated separately but are “lumped” with [natural] materials.
  - Artifacts are
    - are in *space* and *time*;
    - are subject to the *laws of physics*,
    - and also subject to
      - ∗ the *principle of causality*
      - ∗ and *gravitational pull*.

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

Definition 5 Artifact:

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

- We shall assume, cf. Sect. 5.3 [*Attributes*],
  - that *artifacts* all come with an *attribute*
  - of kind *intent*, that is,
  - a set of purposes for which the artifact was constructed,
  - and for which it is intended to serve.

3.3.3 Parts

Example 9 Parts:

- The examples of
  - Example 2 on Slide 47 are all natural parts, and of
  - Example 3 on Slide 48 are all artifacts ■
- Except for the *intent* attribute of artifacts, we shall, in the following, treat
  - *natural* and
  - *artifactual*

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

Analysis Prompt 9 *is_part:*

- *The domain analyser analyse endurants, e, into part entities as prompted by the* **domain analysis prompt***:*
  - *is_part* ■

3.3.4 Atomic and Composite Parts

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

- This is a choice.
  - The choice is based on pragmatics.
  - It is still the domain analyser cum describers' choice whether to consider a discrete endurant
    - a part
    - or a component,
    - or a structure.
  - If the domain engineer wishes to investigate
    - the details of a discrete endurant
    - then the domain engineer choose to model[14]
    - the discrete endurant as a part
    - otherwise as a component.

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

3.3.5 Atomic Parts

Definition 6 Atomic Part:

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

Analysis Prompt 10 `is_atomic`**:**

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

Example 10 Atomic Road Net Parts:

- From one point of view all of the following can be considered atomic parts:
  - hubs, links[15], and
  - automobiles.

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

3.3.6 Composite Parts

Definition 7 Composite Part:

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

Analysis Prompt 11 `is_composite`**:**

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

Example 11 Composite Automobile Parts:

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

3.4 Living Species

- We refer to Sect. 3.2.2
  for our first characterisation (Slide 64) of the concept of *living species*[16]:
  - a discrete endurant existing in time,
  - subject to laws of physics, and
  - additionally subject to *causality of purpose*[17]

[16] See analysis prompt 7 on Slide 65.
[17] See Footnote 12 on Slide 62.

Definition 8 Living Species, II:

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

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

Analysis Prompt 12 `is_living_species:`

- *The domain analyser analyse discrete endurants, e, into living species entities as prompted by the* **domain analysis prompt**:
  - `is_living_species` ■

3.4.1 Plants

Example 12 Plants:

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

Analysis Prompt 13 `is_plant`:

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

3.4.2 Animals

Definition 9 Animal: We refer to the initial definition of *living species* above – while ephasizing the following traits:

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

Analysis Prompt 14 `is_animal`:

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

Example 13 Animals:

- Although we have not yet come across domains for which the need to model the living species of animals, in general, were needed, we give some examples anyway:
  - dolphin,
  - goose
  - cow
  - dog,
  - lion,
  - fly.
- We have not decided, for these lectures,
  - whether to model animals singly
  - or as sets[18] of such.

18
- school of dolphins,
- flock of geese,
- herd of cattle,
- pack of dogs,
- pride of lions,
- swarm of flies,

3.4.3 Humans

Definition 10 Human:

- A *human* (a *person*) is an *animal*, cf. Definition 9, with the additional properties of having
  - *language*,
  - being *conscious* of *having knowledge* (of its own situation), and
  - *responsibility*.

Analysis Prompt 15 *is_human:*

- *The domain analyser analyses "things" (ℓ) into a human.*
- *The method can thus be said to provide the* **domain analysis prompt**:
  - *is_human – where is_human(ℓ) holds if ℓ is a human* ■
- The predicate `is_animal(ℓ)` is a prerequisite for `is_human(ℓ)`.
- We refer to [Bjø18b, Sects. 10.4–10.5]
  - for a specific treatment of living species, animals and humans,
  - and to [Bjø18b] in general for the philosophy background for rationalising the treatment of living species, animals and humans.

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

- We have found, in these experimental domain modelling efforts
  - that we often confer anthropomorphic qualities on artifacts[19],
  - that is, that these artifacts have human characeristics.
- You, the listener are reminded
  - that when some programmers try to explain their programs
  - they do so using such phrases as
  - *and here the program does ...* so-and-so !

[19] Cf. Sect. 3.7 below.

3.5 Components

Definition 11 Component:

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

- Components are discrete endurants.
  - Usually they come in sets.
  - That is, sets of sets of components of different sorts (cf. Sect. 4.4 on Slide 134).
  - A discrete endurant can (itself) "be" a set of components.
  - But physical parts may contain (`has_components`) components:
    - natural parts may contain natural components,
    - artifacts may contain natural and artifactual components.
  - We leave it to the listener to provide analysis predicates for natural and artifactual "componentry".

Example 14 Components:

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

Analysis Prompt 16 *has_components:*

- *The domain analyser analyses discrete endurants e into component entities as prompted by the* **domain analysis prompt***:*
  - *has_components* ■
- We refer to Sect. 4.4 on Slide 134 for further treatment of the concept of *components*.

3.6 Continuous Endurants ≡ Materials

Definition 12 Material:

- By a **material** we shall understand a continuous endurant ■
- Materials are continuous endurants.
  - Usually they come in sets.
  - That is, sets of of materials of different sorts (cf. Sect. 4.5 on Slide 137).
  - So an endurant can (itself) "be" a set of materials.
  - But physical parts may contain (`has_materials`) materials:
    - natural parts may contain natural materials,
    - artifacts may contain natural and artifactual materials.
  - We leave it to the listener to provide analysis predicates for natural and artifactual "materials".

Example 15 Natural and Man-made Materials:

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

Analysis Prompt 17 *has_materials:*

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

- We refer to Sect. 4.5 on Slide 137 for further treatment of the concept of *materials*.
- We shall soon define the terms unique identification, mereology and attributes.

3.7 Artifacts

Definition 13 Artifacts:

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

Example 16 More Artifacts:

- We have already, in Example 9 on Slide 82, referred to some examples of artifacts.
- Here are some more:
  - ship,
  - container vessels,
  - container,
  - container stack,
  - container terminal port,
  - harbour.

Analysis Prompt 18 *`is_artifact`*:

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

3.8 States

Definition 14 State:

- By a *state* we shall understand any number of
  - physical parts or
  - materials.

Example 17 Artifactual States:

- The following endurants are examples of states (including being elements of state compounds):
  - pipe units (pipes, valves, pumps, etc.) of pipe-lines;
  - hubs and links of road nets (i.e., street intersections and street segments);
  - automobiles (of transport systems).

.

Lecture Day 4, Lectures 7–8

Parts, Components and Materials Description

4 Endurants: The Description Calculus

4.1 Parts: Natural or Man-made

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

4.1.1 On Discovering Endurant Sorts

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

- We observe parts one-by-one.
- *(α) Our analysis of parts concludes when we have*
  - *"lifted" our examination of a particular part instance*
  - *to the conclusion that it is of a given sort,*
  - *that is, reflects 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

Analysis Prompt 19 *observe_endurant:*

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

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

- The domain analyser continues to examine a finite number of other composite parts: $\{p_j, p_\ell, \ldots, p_n\}$.
  - It is then "discovered", that is, decided, that they all consists of the same number of sub-parts
  - $\{e_{i_1}, e_{i_2}, \ldots, e_{i_m}\}$,
  - $\{e_{j_1}, e_{j_2}, \ldots, e_{j_m}\}$,
  - $\{e_{\ell_1}, e_{\ell_2}, \ldots, e_{\ell_m}\}$,
  - ...,
  - $\{e_{n_1}, e_{n_2}, \ldots, e_{n_m}\}$, of the same, respective, endurant sorts.
- *(γ) It is therefore concluded, that is, decided, that $\{e_i, e_j, e_\ell, \ldots, e_n\}$ are all of the same endurant sort P with observable part sub-sorts $\{E_1, E_2, \ldots, E_m\}$.*

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

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

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

4.1.2 Endurant Sort Observer Functions

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

    we must prove it.

Domain Description Prompt 1 *observe_endurant_sorts*:

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

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

1. observe_endurant_sorts schema

Narration:
[s] ... narrative text on sorts ...
[o] ... narrative text on sort observers ...
[η] ... narrative text on sort type observers ...
[i] ... narrative text on sort recognisers ...
[p] ... narrative text on proof obligations ...
Formalisation:
type
[s] E,
[s] E_i i:[1..m] **comment:** E_i i:[1..m] abbreviates E_1, E_2, ..., E_m
value
[o] **obs_endurant_sorts**_E_i: E → E_i i:[1..m]
[η] **if** is_part(e_i): η(e_i) ≡ ≪ E_i ≫ i:[1..m]
[i] **is**_E_i: $(E_1|E_2|\ldots|E_m) \rightarrow$ **Bool** i[1..m]
proof obligation [Disjointness of endurant sorts]
[p] $\mathscr{PO}$: $\forall$ e:($E_1|E_2|\ldots|E_m$) · $\bigwedge$ {**is**_E_i(e) ≡ $\bigwedge$ {∼**is**_E_j(p)|j:[1..m] \ {i}}|i:[1..m]}

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

4.2 Concrete Part Types

Analysis Prompt 20 `has_concrete_type`:

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

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

Domain Description Prompt 2 `observe_part_type`:

- *Then the domain analyser applies the* **domain description prompt**:
  - `observe_part_type`(*p*)[20]
- *to parts p:P which then yield the part type and part type observers domain description text according to the following schema:*

[20] `has_concrete_type` is a *prerequisite prompt* of `observe_part_type`.

2. `observe_part_type` **schema**

Narration:

[t1] ... narrative text on sorts and types S_i ...
[t2] ... narrative text on types T ...
[t3] ... narrative text on type of value observer
[o] ... narrative text on type observers ...

Formalisation:

type
[t1] $S_1, S_2, ..., S_m, ..., S_n,$
[t2] $T = \mathscr{E}(S_1,S_2,...,S_n)$
[t3] $\eta(s_i) \equiv \lll S \ggg$, i:[1..n],s_i:S_i
value
[o] **obs_part**_T: P → T ■

- Usually it is wise to restrict the part type definitions, $T_i = \mathscr{E}_i$(Q,R,...,S), to simple type expressions.[21]

[21]

- T=A-**set** or
- T=A* or
- T=ID $\underset{m}{\rightarrow}$ A or
- T=A_t|B_t|...|C_t

where

- ID is a sort of unique identifiers,
- T=A_t|B_t|...|C_t defines the disjoint types
  - A_t==mkA_t(s:A_s),
  - B_t==mkB_t(s:B_s), ...,
  - C_t==mkC_t(s:C_s),

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

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

4.3 On Endurant Sorts

4.3.1 Derivation Chains

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

4.3.2 No Recursive Derivations

- We "mandate" that
  - if E_k is derived from E_j
  - then there
    - E_j is different from E_k and there
    - can be no E_k derived from E_j,
    - that is, E_k cannot be derived from E_k.
- That is, we do not "provide for" recursive domain sorts.
- It is not a question, actually of allowing recursive domain sorts.
  - It is, we claim to have observed,
  - in very many *analysis & description* experiments,
  - that there are no recursive domain sorts ![22]

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

4.3.3 Names of Part Sorts and Types

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

  – as well as the further below defined

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

  prompts introduced below – "yield" type names.

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

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

4.4 Components

- We refer to Sect. 3.5 on Slide 102 for our initial treatment of 'components'.

Domain Description Prompt 3 `observe_component_sorts`:

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

3. `observe_component_sorts` schema

Narration:
[s] ... narrative text on component sorts ...
[o] ... narrative text on component observers ...
[i] ... narrative text on component sort recognisers ...
[u] ... narrative text on unique identifier ...
[p] ... narrative text on component sort proof obligations ...

Formalisation:
type
[s] K1, K2, ..., Kn
[s] K = K1| K2 | ... | Kn
[s] KS = K-**set**
value
[o] **obs_components**_P: P → KS
[i] **is**_K_i: $(K_1|K_2|...|K_n) \to$ **Bool** i:[1..n]
[u] **uid**_K_i

Proof Obligation: [Disjointness of Component Sorts]
[p] $\mathscr{PO}$: $\forall\, k_i{:}(K_1|K_2|...|K_n) \cdot \bigwedge \{\mathbf{is}\_K_i(k_i) \equiv \bigwedge\{\sim\mathbf{is}\_K_j(k_j)|j{:}[1..n] \setminus \{i\}\}\}$ i:[1..n] ■

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

4.5 Materials

- We refer to Sect. 3.6 on Slide 106
  for our initial treatment of 'materials'.
- Continuous endurants (i.e., **materials**) are entities, m, which satisfy:
  - is_material(e) ≡ is_continuous(e)
- If is_material(e) holds
  - then we can apply the **domain description prompt**:
  - observe_material_sorts(e).

Domain Description Prompt 4 *observe_material_sorts*:

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

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

4. observe_material_sorts 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] M1, M2, ..., Mn
[s] M = M1 | M2 | ... | Mn
[s] MS = M-**set**
[a] Ai = A11 | A12 | ... | A1n
value
[o] **obs_mat_sort**_M$_i$: P → M, [i:1..n]
[o] **obs_materials**_P: P → MS
[i] **is**_M$_i$: M → **Bool** [i:1..n]
[a] **attr**_A$_{i_j}$: M$_i$ → A$_{i_j}$ [i:...,j:...]
proof obligation [Disjointness of Material Sorts]
[p] $\mathscr{PO}$: ∀ m_i:M · ⋀ {**is**_M$_i$(m_i) ≡ ⋀{∼**is**_M$_j$(m_j)|j ∈ {1..m} \ {i}}|i:[1..n]}

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

  we must prove it ■

.

Lecture Day 5, Lecture 9–10

Unique Identifiers and Mereology

5 Endurants: Analysis & Description of Internal Qualities

- We remind the listener that internal qualities cover
  - *unique Identifiers* (Sect. 5.1),
  - *mereology* (Sect. 5.2) and
  - *attributes* (Sect. 5.3).

5.1 Unique Identifiers

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

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

Representation of Unique Identifiers:

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

Domain Description Prompt 5 *observe_unique_identifier*:

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

5. observe_unique_identifier schema

Narration:

[s] ... narrative text on unique identifier sort PI ...
[u] ... narrative text on unique identifier observer **uid**_P ...
[η] ... narrative text on type name, an RSL$^+$Text observer ...
[a] ... axiom on uniqueness of unique identifiers ...

Formalisation:

type
[s] PI
value
[u] **uid**_P: P $\rightarrow$ PI
[u] η PI $\rightarrow \lll$ P $\ggg$
axiom [Disjointness of Domain Identifier Types]
[a] $\mathscr{A}$: $\mathscr{U}$(PI,PI_i,PI_j,...,PI_k) ■

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

    and
  - to all components.
- We find, from our many experiments, cf. Example 1 on Slide 36,
  - that we really focus on those domain entities which are
    - artifactual endurants and
    - their behavioural "counterparts".

5.2 Mereology

- Mereology is the study and knowledge of parts and part relations.
  - Mereology, as a logical/philosophical discipline, can perhaps best be attributed to the Polish mathematician/logician Stanisław Leśniewski [CV99, Bjø14].

5.2.1 Part Relations

- Which are the relations that can be relevant for part-hood ?
- We give some examples.
  - (i) Two otherwise distinct parts may *"share"* values.
    - By *'sharing'* values we shall, as a generic example, mean that two parts of different sorts has the same attributes
    - but that one *'defines'* the attribute, like, for example *'programming'* its values, cf. Defn.8 Page175,
    - whereas the other *'uses'* these values, like, for example considering them *'inert'*, cf. Defn.3 Page173.
  - (ii) Two otherwise distinct parts may be said to, for example, be topologically "adjacent" or one "embedded" within the other.

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

5.2.2 Part Mereology: Types and Functions

Analysis Prompt 21 `has_mereology`:

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

1 We define a **mereology type** of a part $p{:}P$ as a triplet type expression over set of unique [part] identifiers.

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

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

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

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

- The three part mereology is just a suggestion.
  - As it is formulated here
    we mean the three 'sets' to be disjoint.
  - Other forms of expressing a mereology should be considered
  - for the particular domain and for the particular parts of that domain.
- We leave out further characterisation of
  - the seemingly vague notion "is_of_interest".
  - It is exemplified in Sect. 8.1.8 Slide 304.

type
2 iPI = iPI1 | iPI2 | ... | iPIm
3 ioPI = ioPI1 | ioPI2 | ... | ioPIn
4 oPI = oPI1 | oPI2 | ... | oPIo
1 MT = iPI-**set** × ioPI-**set** × oPI-**set**
axiom
5 ∀ (iset,ioset,oset):MT ·
5 **card** iset + **card** ioset + **card** oset = **card** ∪{iset,ioset,oset}
5 ∪{iset,ioset,oset} ⊆ unique_identifiers(uod)
value
5 unique_identifiers: P → UI-**set**
5 unique_identifiers(p) ≡ ...

Domain Description Prompt 6 `observe_mereology`:

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

6. `observe_mereology` 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[23]
value
[m] **obs_mereo**_P: P → MT
axiom [Well−formedness of Domain Mereologies]
[a] $\mathscr{A}$: $\mathscr{A}$(MT)

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

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

5.2.3 Formulation of Mereologies

- The `observe_mereology` domain descriptor, Slide 156,
  - may give the impression that the mereo type MT can be described
  - "at the point of issue" of the `observe_mereology` prompt.
  - Since the MT type expression may depend on any part sort
  - the mereo type MT can, for some domains,
  - "first" be described when all part sorts have been dealt with.

5.2.4 Some Modelling Observations

- It is, in principle, possible to find examples of mereologies of natural parts:
  - rivers: their confluence, lakes and oceans; and
  - geography: mountain ranges, flat lands, etc.
- But in our experimental case studies cf. Example 1 on Slide 36, we have found no really interesting such cases.
- All our experimental case studies appears to focus on the mereology of artifacts.

- And, finally, in modelling humans,
  - we find that their mereology encompass
    - all other humans
    - and all artifacts
  - Humans cannot be tamed to refrain from interacting with everyone and everything.

.

Lecture Day 6, Lecture 11–12

Attributes and Summary of Internal Qualities

5.3 Attributes

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

5.3.1 Technical Issues

- We divide Sect. 5.3 into two subsections:
  - *technical issues*, the present one, and
  - *modelling issues*, Sect. 5.3.2.

5.3.1.1 Inseparability of Attributes from Parts and Materials:

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

5.3.1.2 Attribute Quality and Attribute Value:

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

Analysis Prompt 22 *attribute types:*

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

5.3.1.3 Attribute Types and Functions:

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

Domain Description Prompt 7 *observe_attributes***:**

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

7. observe_attributes schema

Narration:

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

Formalisation:

```
type
  [t] A_i [1≤i≤n]
value
  [o] attr_A_i:P→A_i i:[1..n]
  [v] obs_attrib_values_P(p) ≡ { attr_A_1(p),attr_A_2(p),...,attr_A_n(p) }
  [i] is_A_i:(A_1|A_2|...|A_n)→Bool i:[1..n]
proof obligation [Disjointness of Attribute Types]
  [p] 𝒫𝒪: let P be any part sort in [the domain description]
  [p]        let a:(A_1|A_2|...|A_n) in is_A_i(a) ≠ is_A_j(a) end end [i≠i, i,j:[1..n]]
```

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

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

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

  Therefore we must prove it.

5.3.1.4 Attribute Categories:

- Michael A. Jackson [Jac95] has suggested a hierarchy of attribute categories:
  - static or
  - dynamic values – and within the dynamic value category:
    - inert values or
    - reactive values or
    - active values – and within the dynamic active value category:
      - autonomous values or
      - biddable values or
      - programmable values.
- We now review these attribute value types. The review is based on [Jac95, M.A. Jackson].

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

Attribute Category: 1 • By a **static attribute**, a:A, `is_static_attribute`(a), we shall understand an attribute whose values
- are constants,
- i.e., cannot change.

Attribute Category: 2 • By a **dynamic attribute**, a:A, `is_dynamic_attribute`(a), we shall understand an attribute whose values
- are variable,
- i.e., can change.

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

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

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

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

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

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

- change (also) of its own volition.

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

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

- whose values change value only "on their own volition".[24]

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

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

- *are prescribed*
- *but may fail to be observed as such.*

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

- can be prescribed.

- Figure 2 captures an attribute value ontology.

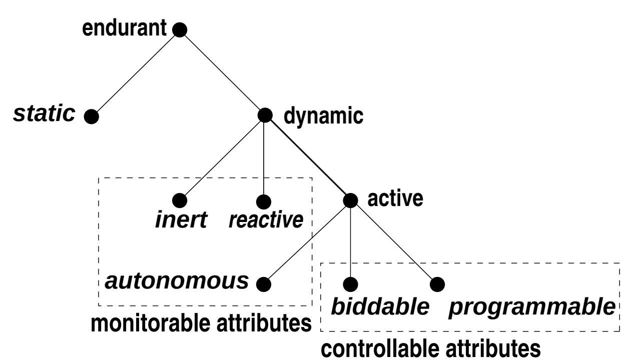


Figure 2: Attribute Value Ontology

5.3.1.5 Calculating Attributes:

6 Given a part *p* we can calculate its static attributes.

7 Given a part *p* we can calculate its controllable, i.e., the biddable and programmable attributes.

8 And given a part *p* we can calculate its monitor-able attributes, i.e., the inert, reactive and autonomous attributes.

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


```
value
6   stat_attr_typs: P → ≪ SA1×SA2×...×SAs ≫
7   ctrl_attr_typs:  P → ≪ CA1×CA2×...×CAc ≫
8   mon_attr_typs: P → ≪ MA1×MA2×...×MAm ≫
axiom
9   ∀ p:P ·
9       let ≪ SA1×SA2×...×SAs ≫ = stat_attr_typs(p),
9           ≪ CA1×CA2×...×CAc ≫ = ctrl_attr_typs(p),
9           ≪ MA1×MA2×...×MAm ≫ = mon_attr_typs(p) in
9       card{SA1,SA2,...,SAs}+card{CA1,CA2,...,CAc}+card{MA1,MA2,...,
9       = card{SA1,SA2,...,SAs,CA1,CA2,...,CAc,MA1,MA2,...,MAm} end
```


10 Given a part *p* we can calculate its static attribute values.

11 Given a part *p* we can calculate its controllable, i.e., the biddable and programmable attribute values.

```
value
10  stat_attr_vals: P → SA1×SA2×...×SAs
10  stat_attr_vals(p) ≡
10      let ≪ SA1×SA2×...×SAs ≫ = stat_attr_typs(p) in
10      (attr_SA1(p),attr_SA2(p),...,attr_SAs(p)) end

11  ctrl_attr_vals: P → CA1×CA2×...×CAc
11  ctrl_attr_vals(p) ≡
11      let ≪ CA1×CA2×...×CAc ≫ = ctrl_attr_typs(p) in
11      (attr_CA1(p),attr_CA2(p),...,attr_CAc(p)) end
```


5.3.2 Basic Principles for Ascribing Attributes

- Section 5.3.1 dealt with technical issues of expressing attributes.
- This section will indicate some modelling principles.

5.3.2.1 Natural Parts

- are in space and time – and are subject to laws of physics.
- So basic attributes focus on physical (including chemical) properties.
- These attributes cover the full spectrum of attribute categories outlined in Sect. 5.3.1.

5.3.2.2 Materials:

- are in space and time – and are subject to laws of physics.
- So basic attributes focus on physical, especially chemical properties.
- These attributes cover the full spectrum of attribute categories outlined in Sect. 5.3.1.

• • •

- The next paragraphs, **living species**, **animate entities** and **humans**, reflect Sørlander's Philosophy [Sør16, pp 14–182].

• • •

Causality of Purpose:

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

Living Species:

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

5.3.2.3 Living Species:

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

5.3.2.4 Animate Entities:

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

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

5.3.2.5 Humans:

Consciousness and Learning:

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

Language:

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

Knowledge:

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

Responsibility:

- In this way one can deduce that humans
  - can thus have *memory*
  - and hence can have *responsibility*,
  - be *responsible*.
  - Further deductions lead us into *ethics*.

•••

5.3.2.6 Intentionality

- *Intentionality is*
  - *a philosophical concept*
  - *and is defined by the* `Stanford Encyclopedia of Philosophy`[25] *as*
    - *“the power of minds to be about, to represent, or to stand for,*
    - *things, properties and states of affairs.”*

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

Definition 15 Intentional Pull:

- Two or more artifactual parts
  - of different sorts, but with overlapping sets of intents
  - may excert an *intentional "pull"* on one another ■

- This *intentional "pull"* may take many forms.
  - Let $p_x : X$ and $p_y : Y$
  - be two parts of *different sorts* (X,Y),
  - and with *common intent*, ι.
  - *Manifestations* of these, their common intent
  - must somehow be *subject to constraints*,
  - and these must be *expressed predicatively*.
- See Sect. 8.3.6, pp. 379–382, for an example.

•••

5.3.2.7 Artifacts:

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

•••

5.3.2.8 Assignment of Attributes:

- So what can we deduce from the above, a little more than a page ?
- The attributes of **natural parts** and **natural materials**
  - are generally of such concrete types –
  - expressible as some **real** with a dimension[26] of
  - the International System of Units:
  - `https://physics.nist.gov/cuu/Units/units.html`.
- Attribute values usually enter *differential equations* and *integrals*,
- that is, classical calculus.

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

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

Examples:

- These are examples of human intents:
  - they create *roads* and *automobiles* with the intent of *transport*.
  - they create *houses* with the intents of *living*, *offices*, *production*, etc., and
  - they create *pipelines* with the intent of *oil* or *gas transport* ■
- Human attribute values usually enter into *modal logic* expressions.

- **Artifacts, including Man-made Materials:**
  - Artifacts, besides those of parts,
    - significantly includes a usually singleton set of *intents*.

Examples:

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

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

5.4 The Unfolding of an Ontology

- We have unfolded an ontology of domain endurants.
- Figure 3 illustrates this "unfolding":

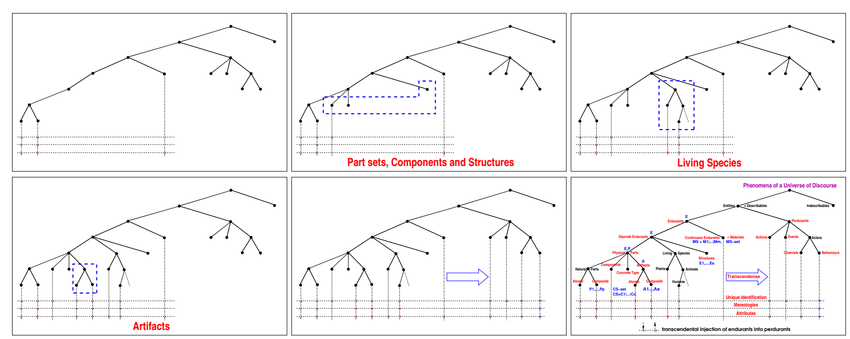


Figure 3: Five Stages of Ontology Development

- The upper left diagram shows the ontology of *part* and *material* endurants and of perdurants.
- The upper middle diagram shows the ontology addition of *concrete part sets* and *structures*.
- The upper right diagram shows the ontology addition of *living species*.
- The lower left diagram shows the ontology addition of *artifacts*.
- The lower middle diagram shows the ontology with the *transcendentally deduced* "coupling" of *internal endurant qualities* with *perdurant behaviour arguments*.
- The lower rightmost diagram shows the fully annotated ontology – and that diagram is the same as Fig. 1 on Slide 30.

5.5 Some Axioms and Proof Obligations

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

- We refer to axioms in Item [a] of domain description prompts of
  - *unique identifiers:* 5 on Slide 146 and of
  - *mereologies:* 6 on Slide 156.

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

- Proof obligations supplement axioms.
- We refer to proof obligations in
  - Item [p] of domain description prompts about
  - *endurant sorts:* 1 on Slide 124, about
  - *components sorts:* 3 on Slide 135, about
  - *materials sorts:* 4 on Slide 139, and about
  - *attribute types:* 7 on Slide 168.
- The difference between expressing axioms and expressing proof obligations is this:

- **We use axioms**
  - when our formula cannot otherwise express it simply,
  - but when physical or other *properties of the domain*[27]
  - dictates property constraints.
- **We use proof obligations**
  - where necssary constraints
  - are not necessarily physically impossible.
- **Proof obligations** finally arise
  - in the transition from endurants to perdurants
  - where endurant axioms
  - become properties that must be proved to hold.

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

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

- When considering a part with, for example,
  a programmable attribute, at two different instances of time
  - we expect the particular programmable attribute
  - to enjoy any expressed well-formedness properties.
- We shall, as from Slide 223,
  - see how these programmable attributes
  - re-occur as explicit behaviour parameters,
  - “programmed” to possibly new values
  - passed on to recursive invocations of the same behaviour.

- If well-formedness axioms were expressed
  - for the part on which the behaviour is based,
  - then a *proof obligation* arises,
  - one that must show that new values of the programmed attribute
  - satisfies the part attribute axiom.
- This is, but one relation between *axioms* and *proof obligations*.
- We refer to remarks made in the bullet (•) named **Biddable Access** Slide 258.

5.6 Discussion of Endurants

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

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

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

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

.

Lecture Day 7, Lecture 13

Transcendentality

6 A Transcendental Deduction

6.1 An Explanation

- It should be clear to the reader that in **domain analysis & description**
  - we are reflecting on a number of philosophical issues.
  - First and foremost on those of *epistemology* and *ontology*.
  - In this section on a sub-field of epistemology,
  - namely that of a number of issues of *transcendental* nature.

Definition 16 Transcendental: By **transcendental** we shall understand the philosophical notion: **the a priori or intuitive basis of knowledge, independent of experience.**

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

Definition 17 Transcendental Deduction: By a **transcendental deduction** we shall understand the philosophical notion: **a transcendental "conversion" of one kind of knowledge into a seemingly different kind of knowledge.**

Definition 18 Transcendentality: By **transcendentality** we shall here mean the philosophical notion: the state or condition of being transcendental.

Example 18 Transcendentality:

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

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

- The three *senses* are:

  (i) as an **endurant** (here a *part*),
  (ii) as a **perdurant** (as we shall see a *behaviour*), and
  (iii) as an **attribute**[28] ■

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

- Example 18, we claim, reflects *transcendentality* as follows:

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

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

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

- In other words:
  - two (i–ii) kinds of different knowledge;
  - that they relate *must indeed* be based on *a priori knowledge*.
  - Someone claims that they relate !
- The two statements (i–ii) are claimed to relate transcendentally.[29]

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

6.2 Some Special Notation

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

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

- Where in the function definition bodies
  - we write ⟪⟫ function_expression
  - we mean just rsl$^+$_text
  - emanating from function_expression.
  - That is:
    - ⟪⟫ function_expression ≡ function_expression and
    - ⟪⟫ ⟪⟫ ≡ ⟪⟫.
- Where in the function definition bodies
  - we write { ≪ $f(x)$ ≫| x:RSL$^+$Text}
  - we mean the “expansion” of the RSL$^+$Text $f(x)$,
  - in arbitrary, linear text order,
  - for appropriate RSL$^+$Texts x.

.

Lecture Day 8, Lectures 15–16

Perdurants: Functions, Channels and Signatures

7 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*.

7.1 States, Actors, Actions, Events and Behaviours

7.1.1 States

Definition 19 Domain States: By a **state** we shall understand

- any collection of **parts**
- or **components**
- or **materials** ■

- We refer to Slide 295.

7.1.2 Functions: Actors, Actions, Events and Behaviours

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

  behaviours.

7.1.3 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.

7.1.4 Actors

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

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

- We shall, in principle, associate an actor with each part[30].
  - These actors will be described as behaviours.
  - These behaviours evolve around a state.
  - The state is
    - the set of qualities, in particular the dynamic attributes, of the associated parts
    - and/or any possible components or materials of the parts.

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

7.1.5 Discrete Actions

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

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

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

7.1.6 Discrete Events

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

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

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

7.1.7 Discrete Behaviours

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

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

- Discrete behaviours now become the *focal point* of our investigation.
  - To every part we associate, by transcendental deduction, a behaviour.
  - We shall express these behaviours as CSP *processes* [Hoa85].
    - For those behaviours we must therefore establish their means of *communication* via *channels*;
    - their *signatures*; and
    - their *definitions* – as *translated* from endurant parts.

7.2 Channels and Communication

7.2.1 The CSP Story:

- Behaviours
  - sometimes synchronise
  - and usually communicate.
- We use the CSP [Hoa85] notation (adopted by RSL) to introduce and 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 modeled as similarly indexed channels:

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

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

7.2.2 From Mereologies to Channel Declarations:

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

- Figure 4 shows two dotted rectangle box diagrams.

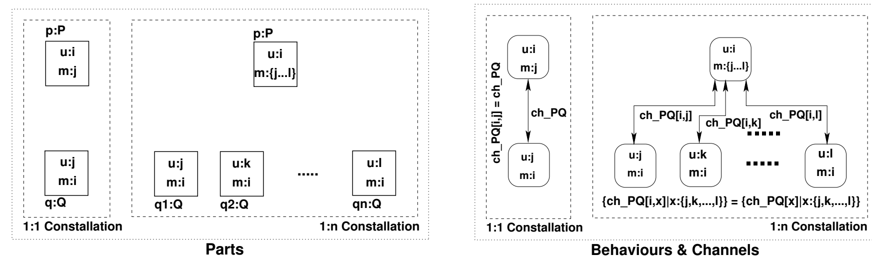


Figure 4: Two Part and Channel Constallations.
u:p unique id. p; *m:p* mereology p

- The left fragment of the figure intends to show a `1:1 Constallation` of a single p:P box and a single q:Q part, respectively, indicating, within these parts, their unique identifiers and mereologies.
- The right fragment of the figure intends to show a `1:n Constallation` of a single p:P box and a set of q:Q parts, now with arrowed lines connecting the p part with the q parts.
- These lines are intended to show channels.
- We show them with two way arrows.
- We could instead have chosen one way arrows, in one or the other direction.
- The directions are intended to show a direction of value transfer.
- We have given the same channel names to all examples, ch_PQ.
- We have ascribed channel message types MPQ to all channels.[31]

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

- Figure 5 shows an arrangement similar to that of Fig. 4 on Slide 238, but for an `m:n Constallation`.

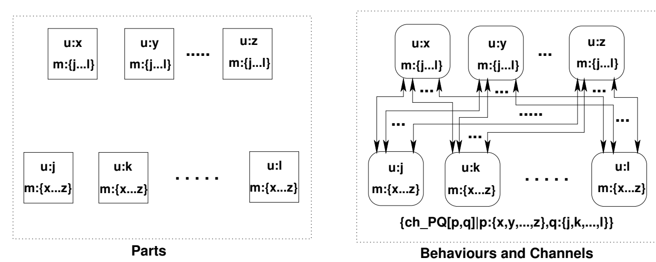


Figure 5: Multiple Part and Channel Arrangements:
u:p unique id. p; *m:p* mereology p

- The channel declarations corresponding to Figs. 4 and 5 are:

channel

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

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

channel

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


12 The following description identities holds:

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

12  { ch_PQ[p,q]:MPQ | p:{x,y,...,z}, q:{j,k,...,l} } ≡
12     ch_PQ[x,j],ch_PQ[x,k],...,ch_PQ[x,l],
12     ch_PQ[y,j],ch_PQ[y,k],...,ch_PQ[y,l],
12     ...,
12     ch_PQ[z,j],ch_PQ[z,k],...,ch_PQ[z,l]
```

- We can sketch a diagram
- similar to Figs. 4 on Slide 238 and 5 on Slide 240
- for the case of composite parts.

7.2.3 Continuous Behaviours

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

7.3 Perdurant Signatures

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

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

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

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

- a pair of *type expressions*.
- separated by a *function type constructor*
  - either $\rightarrow$ (for **total function**)
  - or $\stackrel{\sim}{\rightarrow}$ (for **partial function**) ■
- The *type expressions*
  - are part sort or type, or material sort or type, or component sort or type, or attribute type names,
  - but may, occasionally be expressions over respective type names involving **-set**, $\times$, *, $\overrightarrow{m}$ and | type constructors.

7.3.1 Action Signatures and Definitions

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

    action: VAL $\rightarrow \Sigma \stackrel{\sim}{\rightarrow} \Sigma$
    action(v)(σ) **as** σ'
    pre: $\mathscr{P}$(v,σ)
    post: $\mathscr{Q}$(v,σ,σ')
  - expresses that a selection of the domain,
  - as provided by the Σ type expression,
  - is acted upon and possibly changed.

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

- Which could be the argument values, v:VAL, of actions ?
  - Well, there can basically be only the following kinds of argument values:
    - parts, components and materials, respectively
    - unique part identifiers, mereologies and attribute values.
  - 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[32],
  - ascribing signatures to action functions, and
  - determining
    - action pre-conditions and
    - action post-conditions.

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

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

7.3.2 Event Signatures and Definitions

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

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

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

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

7.3.3 Discrete Behaviour Signatures

Signatures:

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

  behaviour: ... → ... **Unit**

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

  behaviour: **Unit** → ...

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

  behaviour: ... → **in** ch ...

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

  behaviour: ... → **out** ch ...

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

  behaviour: ARG → ...

- where ARG can be any type expression:

  T, T→T, T→T→T, etcetera

  where T is any type expression.

7.3.4 Attribute Access

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

- **Static attributes** designate constants, cf. Defn. 1 Slide 172.
  As such they can be "compiled" into behaviour definitions.
  We choose, instead to list them,
  in behaviour signatures, as arguments.
- **Inert attributes** designate values provided by external stimuli,
  cf. Defn. 3 Slide 173,
  that is, must be obtained by channel input: attr_Inert_A_ch ?.
- **Reactive attributes** are functions of other attribute values,
  cf. Defn. 4 Slide 173.

- **Autonomous attributes** must be input,
  cf. Defn. 6 Slide 174,
  like inert attributes: attr_Autonomous_A_ch ?.
- **Programmable attribute** values are calculated by their behaviours,
  cf. Defn. 8 Slide 175.
  We list them as behaviour arguments.
  The behaviour definitions may then specify new values. These are provided in the position of the programmable attribute arguments in *tail recursive* invocations of these behaviours.
- **Biddable attributes** are like programmable attributes, but when provided in possibly tail recursive invocations of their behaviour the calculated biddable attribute value is *modified*, usually by some *perturbation*[33] of the calculated value – to reflect that although they *are prescribed* they *may fail to be observed as such*, cf. Defn. 7 Slide 175.

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

7.3.5 Calculating In/Output Channel Signatures

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

13 The function calc_i_o_chn_refs apply to parts and yield RSL$^+$Text.

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

b. If there the mereology is not void and/or the are monitorable values then a (Currying[34]) right pointing arrow, →, is inserted.[35]

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

d. Similarly for the input/output mereology;

e. and for the output mereology.

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

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

value
13 calc_i_o_chn_refs: P → RSL$^+$Text
13 calc_i_o_chn_refs(p) ≡
13a. **let** ui = **uid**_P(p),
13a. (ics,iocs,ocs) = **obs_mereo**_(p),
13a. atrvs = **obs_attrib_values**_P(p) **in**
13b. **if** ics ∪ iocs ∪ ocs ∪ atrvs ≠ {}
13b. **then** ≪ → ≫ **end**
13c. **if** ics ∪ atrvs ≠{}
13c. **then** ≪**in**≫ calc_attr_chn_refs(ui,atrvs), calc_chn_refs(ui,ichs) **end**
13d. **if** iocs≠{}
13d. **then** ≪**in,out**≫ calc_chn_refs(ui,iochs) **end**
13e. **if** ocs≠{}
13e. **then** ≪**out**≫ calc_chn_refs(ui,ochs) **end end**

14 The function calc_attr_chn_refs

a. apply to a set, mas, of monitorable attribute types and yield RSL$^+$Text.
b. If achs is empty no text is generated.
Otherwise a channel declaration attr_A_ch
is generated for each attribute type whose name, A,
which is obtained by applying η
to an observed attribute value, ηa.

14a. calc_attr_chn_refs: UI × A-**set** → RSL$^+$Text
14b. calc_attr_chn_refs(ui,mas) ≡
14b. { ≪ attr_ηa_ch[ui] ≫ | a:A·a ∈ mas }

15 The function calc_chn_refs

a. apply to a pair, (ui,uis) of a unique part identifier and a set of unique part identifiers and yield RSL$^+$Text.
b. If uis is empty no text is generated. Otherwise an array channel declaration is generated.

15a. calc_chn_refs: P_UI × Q_UI-**set** → RSL$^+$Text
15b. calc_chn_refs(pui,quis) ≡ { ≪ η(pui,qui)_ch[pui,qui] ≫ | qui:Q_UI·qui ∈ quis }

16 The function calc_all_chn_dcls

a. apply to a pair, (pui,quis) of a unique part identifier and a set of unique part identifiers and yield RSL$^+$Text.
b. If quis is empty no text is generated. Otherwise an array channel declaration
   - { ≪ η(pui,qui)_ch[pui,qui]:η(pui,qui)M ≫ | qui:Q_UI·qui ∈ quis }

   is generated.

16a. calc_all_chn_dcls: P_UI × Q_UI-**set** → RSL$^+$Text
16a. calc_all_chn_dcls(pui,quis) ≡
16a. { ≪η(pui,qui)_ch[pui,qui]:η(pui,qui)M ≫ | qui:Q_UI·qui ∈ quis }

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

17 The overloaded η operator is here applied to a pair of unique identifiers.

17 η: (UI → RSL$^+$Text)|((X_UI×Y_UI) → RSL$^+$Text)
17 η(x_ui,y_ui) ≡ (≪(η x_ui η y_ui≫))

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

  {PQ_ch[$p_{1_{ui}}$,qui]|$p_{1_{ui}}$:P_UI,qui:Q_UI·qui ∈ quis}
  {PQ_ch[$p_{2_{ui}}$,qui]|$p_{2_{ui}}$:P_UI,qui:Q_UI·qui ∈ quis}
  ...
  {PQ_ch[$p_{n_{ui}}$,qui]|$p_{n_{ui}}$:P_UI,qui:Q_UI·qui ∈ quis}

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

  { PQ_ch[pui,qui] | pui:P_UI,qui:Q_UI : −pui ∈ puis,qui ∈ quis }
  where puis = { $p_{1_{ui}}$,$p_{2_{ui}}$,...,$p_{n_{ui}}$}

- As an example we have already calculated the array channels for Fig. 5 Slide 240 – cf. the left, the **Parts**, of that figure – cf. Items [1–3] Pages 241–242.
- The identities Item 12 Slide 242 apply.

.

Lecture Day 9, Lectures 17–18

Perdurants: Discrete Behaviour Definitions

7.4 Discrete Behaviour Definitions

- We associate with each part, p:P, a behaviour name $\mathcal{M}_P$.
- Behaviours have as first argument their unique part identifier: **uid**_P(p).
- Behaviours evolves around a state, or, rather, a set of values:
  - its possibly changing mereology, mt:MT and
  - the attributes of the part.[36]

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

- A behaviour signature is therefore:

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

  where

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

- Let P be a composite sort defined in terms of endurant[37] sub-sorts E_1, E_2, . . . , E_n.
  - The behaviour description *translated* from p:P, is composed from
    - a behaviour description, $\mathcal{M}_P$, relying on and handling the unique identifier, mereology and attributes of part *p*
    - to be *translated* with behaviour descriptions $\beta_1, \beta_2, \ldots, \beta_n$ where
      - β_1 is *translated* from e_1:E_1,
      - β_2 is *translated* from e_2:E_2,
      - ..., and
      - β_n is *translated* from e_n:E_n.
- The domain description *translation* schematic below "formalises" the above.

[37]– structures or composite

Process Schema 1

Abstract `is_composite(p)`

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


- Expression $\mathcal{B}_P$(ui,me,sta,pa) stands for the *behaviour definition body* in which the names ui, me, sta, pa are bound to the *behaviour definition head*, i.e., the left hand side of the ≡.
- Endurant sorts E_1, E_2, ..., E_n are obtained from the `observe_endurant_sorts` prompt, Slide 124.
- We informally explain the **Translate**$_{P_i}$ function.
  - It takes endurants and produces RSL$^+$Text.
  - Resulting texts are bracketed: ≪rsl_text≫

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

Process Schema 2 **Abstract** is_structure(e)

value
  Translate$_P$(p) ≡
    Translate$_{P_1}$(**obs_endurant_sorts**_P$_1$(p))
  ⫴ Translate$_{P_2}$(**obs_endurant_sorts**_P$_2$(p))
  ⫴ ...
  ⫴ Translate$_{P_n}$(**obs_endurant_sorts**_P$_n$(p))

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

Process Schema 3 **Concrete** is_composite(p)

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

Process Schema 4 **Atomic** is_atomic(p)

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

Process Schema 5

Core Process

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

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

$\mathscr{F}_P$: P_UI×MT×ST → CT→ in_out_chns(me) → MT×CT

- We refer to [Bjø16e, Process Schema V: Core Process (II), Page 40] for possible forms of $\mathscr{F}_P$.

7.5 Running Systems

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

- The choice
  - as to which parts, i.e., behaviours,
  - are to represent an initial, i.e., a start system behaviour,
  - cannot be "formalised",
  - it really depends on the "deeper purpose"
  - of the system.
- In other words:
  - requires careful analysis and is
  - beyond the scope of the present lectures.
- We refer to the example, Slides 371–378.

7.6 Concurrency: Communication and Synchronisation

- Process Schemas I, II, III and V
  (Slides 271, 273, 275 and 277),
  reveal
  - that two or more parts, which temporally coexist (i.e., at the same time),
  - imply a notion of *concurrency*.
- Process Schema IV, Page 276,
  - through the `RSL/CSP` language expressions ch ! v and ch ?,
  - indicates the notions of *communication* and *synchronisation*.
- Other than this
  we shall not cover these crucial notion related to *parallelism*.

7.7 Summary and Discussion of Perdurants

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

7.7.1 Summary

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

7.7.2 Discussion

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

.

Lecture Day 1, Lecture 2

Example

8 A Methodology Example: A Road Transport System

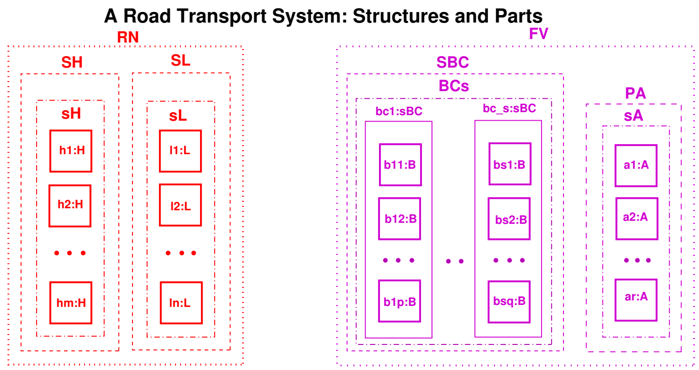


Figure 6: A Road Transport System

8.1 Endurants

8.1.1 The Discourse

- The universe of discourse is *road transport systems*.
  - We analyse & describe not the class of all road transport systems
  - but a representative subclass, UoD, is *structured* into such notions as
    - a road net, RN, of hubs, H, (intersections) and links, L, (street segments between intersections);
    - a fleet of vehicles, FV, *structured* into companies, BC, of buses, B, and pools, PA, of private automobiles, A (et cetera);
    - et cetera.
  - See Fig. 6 on the preceding slide

8.1.2 Structures & Parts

18 There is the *universe of discourse*, UoD.
It is structured into

19 a *road net*, RN, a structure, and

20 a *fleet of vehicles*, FV, a structure.

type
18 UoD **axiom** ∀ uod:UoD · is_structure(uod).
19 RN **axiom** ∀ rn:RN · is_structure(rn).
20 FV **axiom** ∀ fv:FV · is_structure(fv).
value
19 obs_RN: UoD → RN
20 obs_FV: UoD → FV

8.1.3 Parts

21 The road net consists of

a. a structure, SH, of hubs and

b. a structure, SL, of links.

22 The fleet of vehicles consists of

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

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

type
21a. SH **axiom** ∀ sh:SH · is_structure(sh)
21b. SL **axiom** ∀ sl:SL · is_structure(sl)
22a. SBC **axiom** ∀ sbc:SBC · is_structure(bc)
22b. PA **axiom** ∀ pa:PA · is_structure(pa)
value
21a. obs_SH: RN → SH
21b. obs_SL: RN → SL
22a. obs_BC: FV → BC
22b. obs_PA: FV → PA

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

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

25 The structure of busses is a set, sBC, of composite bus companies, BC.

26 The composite bus companies, BC, are sets of busses, sB.

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

type
23 H, sH = H-**set** **axiom** ∀ h:H · is_atomic(h)
24 L, sL = L-**set** **axiom** ∀ l:L · is_atomic(l)
25 BC, BCs = BC-**set** **axiom** ∀ bc:BC · is_composite(bc)
26 B, Bs = B-**set** **axiom** ∀ b:B · is_atomic(b)
27 A, sA = A-**set** **axiom** ∀ a:A · is_atomic(a)
value
23 obs_sH: SH → sH
24 obs_sL: SL → sL
25 obs_sBC: SBC → BCs
26 obs_Bs: BCs → Bs
27 obs_sA: SA → sA

8.1.4 Components

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

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

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

value
28 has_components: H → **Bool**
type
29 Sand, Gravel, CobbleStones, Asphalt, Cement, ...
29 KS = (Sand|Gravel|CobbleStones|Asphalt|Cement|...)**-set**
value
28 obs_components_H: H → KS
28 **pre**: obs_components_H(h) ≡ **card** mereo(h) = 1

8.1.5 Materials

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

30 Links may contain material.

31 That material is water, W.

type
31 W
value
30 obs_material: L → W
30 **pre**: obs_material(l) ≡ has_material(h)

8.1.6 States

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

From that state we can calculate other states.

33 The set of all hubs, *hs*.

34 The set of all links, *ls*.

35 The set of all hubs and links, *hls*.

36 The set of all bus companies, *bcs*.

37 The set of all busses, *bs*.

38 The map from the unique bus company identifiers, see Item 44c. Slide 297, to the set of all the identifies bus company's buses, $bc_{ui}bs$.

39 The set of all private automobiles, *as*.

40 The set of all parts, *ps*.

value
32 *rts*:UoD
33 *hs*:H**-set** ≡ ≡ obs_sH(obs_SH(obs_RN(*rts*)))
34 *ls*:L**-set** ≡ ≡ obs_sL(obs_SL(obs_RN(*rts*)))
35 *hls*:(H|L)**-set** ≡ *hs*∪*ls*
36 *bcs*:BC**-set** ≡ obs_BCs(obs_SBC(obs_FV(obs_RN(*rts*))))
37 *bs*:B**-set** ≡ ∪{obs_Bs(bc)|bc:BC·bc ∈ *bcs*}
38 $bc_{ui}bs$:(BC_UI $\overrightarrow{m}$ B**-set**) ≡
38 [uid_BC(bc) ↦ obs_Bs(bc) | bc:BC · bc ∈ *bcs*]
39 *as*:A**-set** ≡ obs_BCs(obs_SBC(obs_FV(obs_RN(*rts*))))
40 *ps*:(H|L|BC|B|A)**-set** ≡ *hls*∪*bcs*∪*bs*∪*as*

8.1.7 Unique Identifiers

8.1.7.1 Part Identifiers:

41 We assign unique identifiers to all parts.

42 By a road identifier we shall mean a link or a hub identifier.

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

44 Unique identifiers uniquely identify all parts.

a. All hubs have distinct [unique] identifiers.

b. All links have distinct identifiers.

c. All bus companies have distinct identifiers.

d. All busses of all bus companies have distinct identifiers.

e. All automobiles have distinct identifiers.

f. All parts have distinct identifiers.

type

41 H_UI, L_UI, BC_UI, B_UI, A_UI

42 R_UI = H_UI | L_UI

43 V_UI = B_UI | A_UI

value

44a. uid_H: H → H_UI

44b. uid_L: H → L_UI

44c. uid_BC: H → BC_UI

44d. uid_B: H → B_UI

44e. uid_A: H → A_UI

8.1.7.2 Extract Parts from Their Unique Identifiers:

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

type

45 P = H | L | BC | B | A

value

45 $\wp$: H_UI→H | L_UI→L | BC_UI→BC | B_UI→B | A_UI→A

45 $\wp$(ui) ≡ **let** p:(H|L|BC|B|A)·p∈*ps*∧uid_P(p)=ui **in** p **end**

8.1.7.3 Unique Identifier Constants:

We can calculate:

46 the *s*et, $h_{ui}s$, of *u*nique *h*ub *i*dentifiers;

47 the *s*et, $l_{ui}s$, of *u*nique *l*ink *i*dentifiers;

48 the *m*ap, $hl_{ui}m$, from *u*nique *h*ub *i*dentifiers to the *s*et of *u*nique *l*ink *i*identifiers of the links connected to the zero, one or more identified hubs,

49 the *m*ap, $lh_{ui}m$, from *u*nique *l*ink *i*dentifiers to the *s*et of *u*nique *h*ub *i*identifiers of the two hubs connected to the identified link;

50 the *s*et, $r_{ui}s$, of all *u*nique hub and link, i.e., *r*oad *i*dentifiers;

51 the *s*et, $bc_{ui}s$, of *u*nique *b*us *c*ompany *i*dentifiers;

52 the *s*et, $b_{ui}s$, of *u*nique *b*us *i*dentifiers;

53 the *s*et, $a_{ui}s$, of *u*nique private *a*utomobile *i*dentifiers;

54 the *s*et, $v_{ui}s$, of *u*nique bus and automobile, i.e., *v*ehicle *i*dentifiers;

55 the *m*ap, $bcb_{ui}m$, from *u*nique *b*us *c*ompany *i*dentifiers to the *s*et of its *u*nique *b*us *i*dentifiers; and

56 the (*b*ijective) *m*ap, $bbc_{ui}bm$, from *u*nique *b*us *i*dentifiers to their *u*nique *b*us *c*ompany *i*dentifiers.

value

46 $h_{ui}s$:H_UI-**set** ≡ {uid_H(h)|h:H•h ∈ hs}
47 $l_{ui}s$:L_UI-**set** ≡ {uid_L(l)|l:L•l ∈ ls}
50 $r_{ui}s$:R_UI-**set** ≡ $h_{ui}s$∪$l_{ui}s$
48 $hl_{ui}m$:(H_UI $\overrightarrow{m}$ L_UI-**set**) ≡
48 [h_ui↦luis|h_ui:H_UI,luis:L_UI-**set**•h_ui∈$h_{ui}s$∧(_,luis,_)=mereo_H(η(h_ui))] [cf. Item 63]
49 $lh_{ui}m$:(L+UI $\overrightarrow{m}$ H_UI-**set**) ≡
49 [l_ui↦huis | h_ui:L_UI,huis:H_UI-**set** • l_ui∈$l_{ui}s$ ∧ (_,huis,_)=mereo_L(η(l_ui))] [cf. Item 64]
51 $bc_{ui}s$:BC_UI-**set** ≡ {uid_BC(bc)|bc:BC•bc ∈ bcs}
52 $b_{ui}s$:B_UI-**set** ≡ ∪{uid_B(b)|b:B•b ∈ bs}
53 $a_{ui}s$:A_UI-**set** ≡ {uid_A(a)|a:A•a ∈ as}
54 $v_{ui}s$:V_UI-**set** ≡ $b_{ui}s$ ∪ $a_{ui}s$
55 $bcb_{ui}m$:(BC_UI $\overrightarrow{m}$ B_UI-**set**) ≡
55 [bc_ui ↦ buis | bc_ui:BC_UI, bc:BC • bc∈bcs ∧ bc_ui=uid_BC(bc) ∧ (_,_,buis)=mereo_BC(bc)]
56 $bbc_{ui}bm$:(B_UI $\overrightarrow{m}$ BC_UI) ≡
56 [b_ui ↦ bc_ui | b_ui:B_UI,bc_ui:BC_ui • bc_ui=**dom**$bcb_{ui}m$∧b_ui∈$bcb_{ui}m$(bc_ui)]

8.1.7.4 Uniqueness of Part Identifiers:

See Sect. 5.5 Slide 202.

- We must express the following axioms:

57 All hub identifiers are distinct.
58 All link identifiers are distinct.
59 All bus company identifiers are distinct.
60 All bus identifiers are distinct.
61 All private automobile identifiers are distinct.
62 All part identifiers are distinct.

axiom

57 **card** hs = **card** $h_{ui}s$
58 **card** ls = **card** $l_{ui}s$
59 **card** bcs = **card** $bc_{ui}s$
60 **card** bs = **card** $b_{ui}s$
61 **card** as = **card** $a_{ui}s$
62 **card** {$h_{ui}s$∪$l_{ui}s$∪$bc_{ui}s$∪$b_{ui}s$∪$a_{ui}s$}
62 = **card** $h_{ui}s$+**card** $l_{ui}s$+**card** $bc_{ui}s$+**card** $b_{ui}s$+**card** $a_{ui}s$

8.1.8 Mereology

We refer to S;ide 152.

63 The mereology of hubs is a triple: (i) the set of all bus and automobile identifiers[38], (ii) the set of unique identifiers of the links that it is connected to and the set of all unique identifiers of all vehicle (buses and private automobiles).[39], and (iii) an empty set.[40]

64 The mereology of links is a triple: (i) the set of all bus and automobile identifiers, (ii) the set of the two distinct hubs they are connected to, and (iii) an empty set.

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

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

[40] ... the hubs are not "proactive", i.e., that the universe of discourse have no parts that are `interested` in the hub.

65 The mereology of of a bus company is a triple: (i) an empty set, (ii) empty set, and (iii) and set the unique identifiers of the buses operated by that company.

66 The mereology of a bus is a triple: (i) the set of the one single unique identifier of the bus company it is operating for, (ii) an empty set, and (iii) the unique identifiers of all links and hubs[41].

67 The mereology of an automobiles is a triple: (i) an empty set, (ii) an empty set, and (iii) the set of the unique identifiers of all links and hubs[42].

68 Empty sets are modeled as empty sets of tokens where tokens are further undefined.

[41] that the bus might pass through
[42] that the automobile might pass through

type
68 ES = TOKEN-**set**
68 **axiom** ∀ es:ES·es={}
63 H_Mer = V_UI-**set**×L_UI-**set**×ES
63 **axiom** ∀ (vuis,luis,_):H_Mer · luis⊆$l_{ui}s$ ∧ vuis=$v_{ui}s$
64 L_Mer = V_UI-**set**×H_UI-**set**×ES
64 **axiom** ∀ (vuis,huis,_):L_Mer ·
64 vuis=$v_{ui}s$ ∧ huis⊆$h_{ui}s$ ∧ **card**huis=2
65 BC_Mer = ES×ES×B_UI-**set**
65 **axiom** ∀ (_,_,buis):H_Mer · buis = $b_{ui}s$
66 B_Mer = BC_UI×ES×R_UI-**set**
66 **axiom** ∀ (bc_ui,_,ruis):H_Mer · bc_ui∈$bc_{ui}s$ ∧ ruis=$r_{ui}s$
67 A_Mer = ES×ES×R_UI-**set**
67 **axiom** ∀ (_,ruis,_):A_Mer · ruis=$r_{ui}s$

value
63 mereo_H: H → H_Mer
64 mereo_L: L → L_Mer
65 mereo_BC: BC → BC_Mer
66 mereo_B: B → B_Mer
67 mereo_A: A → A_Mer

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

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

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

axiom
69 ∀ h:H,l:L · h ∈ hs ∧ l ∈ ls ⇒
let (_,luis,_) = mereo_H(h), (_,huis,) = mereo_L(l) **in**
70 uid_L(l) ∈ luis ⇒ uid_H(h) ∈ huis **end**

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

8.1.9 Attributes

- We treat part attributes, sort by sort.

Hubs: We show just a few attributes:

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

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

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

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


```
type
71  HΣ = (L_UI×L_UI)-set                                  [programmable, Df.8 Pg.175]
axiom
71  ∀ h:H • obs_HΣ(h) ∈ obs_HΩ(h)
type
72  HΩ = HΣ-set                                            [static, Df.1 Pg.172]
73  H_Traffic                                              [programmable, Df.8 Pg.175]
73  H_Traffic = (A_UI|B_UI) →m (𝒯 × VPos)*
axiom
73  ∀ ht:H_Traffic,ui:(A_UI|B_UI)•ui ∈ dom ht
73      ⇒ time_ordered(ht(ui))
value
71  attr_HΣ: H → HΣ
72  attr_HΩ: H → HΩ
73  attr_H_Traffic: : → H_Traffic
axiom
74  ∀ h:H • h ∈ hs ⇒
74      let hσ = attr_HΣ(h) in
74      ∀ (l_ui i,li_ui i′):(L_UI×L_UI) • (l_ui i,l_ui i′) ∈ hσ
74          ⇒ {l_ui_i,l′_ui_i} ⊆ l_ui s end
value
73      time_ordered: 𝒯* → Bool
73      time_ordered(tvpl) ≡ ...
```


Links: We show just a few attributes:

75 There is a link state. It is a set of pairs, (h_f,h_t), of distinct hub identifiers, where these hub identifiers are in the mereology of the link. The meaning of a link state in which (h_f,h_t) is an element is that the link is open, **"green"**, for traffic *f*rom hub h_f *t*o hub h_t. Link states can have either 0, 1 or 2 elements.

76 There is a link state space. It is a set of link states. The meaning of the link state space is that its states are all those the which the link can attain. The current link state must be in its state space. If a link state space is empty then the link is (permanently) closed. If it has one element then it is a one-way link. If a one-way link, *l*, is imminent on a hub whose mereology designates that link, then the link is a "trap", i.e., a "blind cul-de-sac".

77 Since we can think rationally about it, it can be described, hence it can model, as an attribute of links a history of its traffic: the recording, per unique bus and automobile identifier, of the time ordered positions along the link (from one hub to the next) of these vehicles.

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


```
type
75  LΣ = H_UI-set                                        [programmable, Df.8 Pg.175
axiom
75  ∀ lσ:LΣ·card lσ=2
75  ∀ l:L · obs_LΣ(l) ∈ obs_LΩ(l)
type
76  LΩ = LΣ-set                                          [static, Df.1 Pg.172]
77  L_Traffic                                            [programmable, Df.8 Pg.175
77  L_Traffic = (A_UI|B_UI) →m (𝒯×(H_UI×Frac×H_UI))*
77  Frac = Real, axiom frac:Fract · 0<frac<1
value
75  attr_LΣ: L → LΣ
76  attr_LΩ: L → LΩ
77  attr_L_Traffic: : → L_Traffic
axiom
77  ∀ lt:L_Traffic,ui:(A_UI|B_UI)·ui ∈ dom ht
77        ⇒ time_ordered(ht(ui))

78  ∀ l:L · l ∈ ls ⇒
78      let lσ = attr_LΣ(l) in
78      ∀ (h_ui i,h_ui i′):(H_UI×K_UI) ·
78          (h_ui i,h_ui i′) ∈ lσ ⇒ {h_ui_i,h′_ui_i} ⊆ h_ui s end
```


Bus Companies:

- Bus companies operate a number of lines that service passenger transport along routes of the road net. Each line being serviced by a number of busses.

79 Bus companies have a physical, i.e., "real, actual" time attribute.

80 Bus companies create, maintain, revise and distribute
[to the public (not modeled here), and to busses]
bus time tables, not further defined.


```
type
79  𝒯                                    [inert, Df.3 Pg.173]
80  BusTimTbl                            [programmable, Df.8 Pg.175]
value
79  attr_T: BC → 𝒯
80  attr_BusTimTbl: BC → BusTimTbl
```

- There are two notions of time at play here:
  - the inert "real" or "actual" time as an inert attribute provided by some outside "agent"; and
  - the calendar, hour, minute and second time designation occurring in some textual form in, e.g., time tables..

Busses: We show just a few attributes:

79 Buses have a time attribute.

81 Busses run routes, according to their line number, ln:LN, in the

82 bus time table, btt:BusTimTbl obtained from their bus company, and and keep, as inert attributes, their segment of that time table.

83 Busses occupy positions on the road net:

a. either *at a hub* identified by some h_ui,

b. or *on a link*, some *fraction*, f:Fract, down an *identified link*, *l_ui*, from one of its *identified connecting hub*s, fh_ui, in the direction of the other *identified hub*, th_ui.

84 Et cetera.

type

79 $\mathscr{T}$ [inert, Df.3 Pg.173]

81 LN [programmable, Df.8 Pg.175]

82 BusTimTbl [inert, Df.3 Pg.173]

83 BPos == atHub | onLink [programmable, Df.8 Pg.175]

83a. atHub :: h_ui:H_UI

83b. onLink :: fh_ui:H_UI×l_ui:L_UI×frac:Fract×th_ui:H_UI

83b. Fract = **Real**, **axiom** frac:Fract · 0<frac<1

84 ...

value

79 attr_T: B → $\mathscr{T}$

82 attr_BusTimTbl: B → BusTimTbl

83 attr_BPos: B → BPos

Private Automobiles: We show just a few attributes:

- We illustrate but a few attributes:

79 Automobiles have a time attribute.

85 Automobiles have static number plate registration numbers.

86 Automobiles have dynamic positions on the road net:

[83a.] either *at a hub* identified by some h_ui,

[83b.] or *on a link*, some *fraction*, *frac:Fract* down an *identified link*, *l_ui*, from one of its *identified connecting hub*s, fh_ui, in the direction of the other *identified hub*, th_ui.

type

79 $\mathscr{T}$ [inert, Df.3 Pg.173]

85 RegNo [static, Df.1 Pg.172]

86 APos == atHub | onLink [programmable, Df.8 Pg.175]

83a. atHub :: h_ui:H_UI

83b. onLink :: fh_ui:H_UI × l_ui:L_UI × frac:Fract × th_ui:H_UI

83b. Fract = **Real**, **axiom** frac:Fract · 0<frac<1

value

79 attr_T: A → $\mathscr{T}$

85 attr_RegNo: A → RegNo

86 attr_APos: A → APos

- Obvious attributes that are not illustrated are those of
  - velocity and acceleration,
  - forward or backward movement,
  - turning right, left or going straight,
  - etc.

- The *acceleration*, *deceleration*, *even velocity*, or *turning right*, *turning left*, *moving straight*, or *forward* or *backward* are seen as *command actions*.
  - As such they denote actions by the automobile —
  - such as pressing the accelerator, or lifting accelerator pressure or *braking*, or *turning the wheel* in one direction or another, etc.
  - As actions they have a kind of counterpart in the velocity, the acceleration, etc. attributes.

8.1.10 Discussion

- Observe that bus companies each have their own distinct *bus time table*, and that these are modeled as *programmable*, Item 79 on Slide 315, Page 315.
- Observe then that busses each have their own distinct *bus time table*, and that these are model-led as *inert*, Item 82 on Slide 317, Page 317.
- In Items 117–118b. Slide 363 we shall see how the busses communicate with their respective bus companies in order for the busses to obtain the *programmed* bus time tables "in lieu" of their *inert* one !

- In Items 73 Slide 310 and 77 Slide 313, we illustrated an aspect of domain analysis & description that may seem, and at least some decades ago would have seemed, strange: namely that if we can think, hence speak, about it, then we can model it "as a fact" in the domain. The case in point is that we include among hub and link attributes their histories of the timed whereabouts of buses and automobiles.[43]

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

8.1.11 Some Axioms and Proof Obligations

- Examples of axioms are given in
  - Items 57 – 62 Slide 302,
  - Items 69 – 70 Slide 308,
  - Item 73, and in
  - Item 77.
- We shall give an example of a **proof obligation** expressed as a **post** condition,
  - related to the last two of the above axioms,
  - in Items 109g. Slide 358 and 116 Slide 360

- Those proof obligations reflect an aspect of the concept of **transcendental deduction:**
  - that *axioms* over, as here, *internal qualities* of *endurants*
  - via *post conditions* of *perdurants*
  - become *proof obligations !*

8.1.12 Discussion of Endurants

- We have chosen to model some discrete endurants
  - as structures
  - others as parts (usually composite).
- Those choices are made mostly to illustrate that the *domain analyser & describer* has a choice.
  - If a choice is made to model a discrete endurant as a structure
    - then it entails that the *domain analyser & describer* does not wish to "implement" that discrete endurant as a behaviour separate from its sub-endurants;
  - If the choice is made to model a discrete endurant as a part
    - then it entails that the *domain analyser & describer* wishes to "implement" that discrete endurant as a behaviour separate from its sub-endurants.

- The following discrete endurants which are modeled as structures above, could, instead, if modeled as parts, have the entailed behaviours reflect the following possibilities:
  - *road net*, rn:RN: The road net behaviour could be that of a road net authority charged with building, servicing, operating and maintaining the road net. Building and maintaining the road net could mean the insertion of new or removal of old links or hubs. Operating the road net could mean the gathering of bus and automobile traffic statistics, the setting of hub states (traffic signal monitoring and control), etc.
  - *aggregate of bus companies*, sbc:SBC: The composite aggregate of bus companies could be that of a public transport authority charged with establishing, servicing, operating and maintaining a common bus time table, etc.
  - *aggregate of private automobiles*, ps:PA: The aggregate of private automobiles could be that of one or more *automobile clubs*, etc.

8.2 Transcendentality

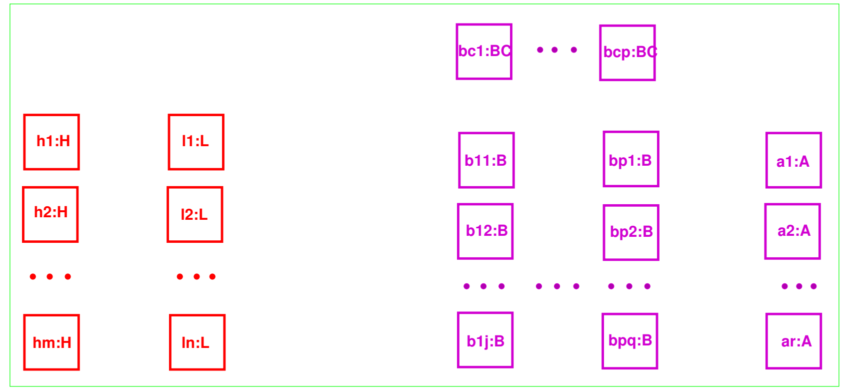
- We refer to Sect. 6 on Slide 214 Defn. 18 Page 216.

Example 19 **A Case of Transcendentality**:

- We refer to the following example:
  - We can speak of a bus in at least three *senses:*
    - The bus as it is being maintained, serviced, refueled;
    - the bus as it "speeds" down its route; and
    - the bus as it "appears" (listed) in a bus time table.
  - The three *senses* are:
    - as a part,
    - as a behaviour, and
    - as an attribute[44] ■

[44] in this case rather: as a fragment of an attribute

8.3 Perdurants



- In the figure above
- we "symbolically", i.e., the "...", show the following parts:
  - each individual hub,
  - each individual link,
  - each individual bus company,
  - each individual bus, and
  - each individual automobile
  - – and all of these.

- The idea is that those are the parts for which we shall define behaviours.
- That figure, however, and in contrast to Fig. 6 Slide 285,
  - shows the composite parts as not containing their atomic parts,
  - but as if they were "free-standing, atomic" parts.
- That shall visualise the transcendental interpretation
  - as atomic part behaviours
  - not being somehow embedded in composite behaviours,
  - but operating concurrently, in parallel.

8.3.1 Constants and States

8.3.1.1 Constants:

We refer to Sect. 7.1.1 Slide 224, and to App. 8.1.6 Slide 295

- We assume, as a constant, an arbitrarily selected universe of discourse, *uod*,
- and calculate from *uod* all its endurants.

value

32 $rts:\mathsf{UoD}$ [32]
33 $hs:\mathsf{H}\textbf{-set} \equiv :\mathsf{H}\textbf{-set} \equiv \mathsf{obs\_sH}(\mathsf{obs\_SH}(\mathsf{obs\_RN}(rts)))$ [33]
34 $ls:\mathsf{L}\textbf{-set} \equiv :\mathsf{L}\textbf{-set} \equiv \mathsf{obs\_sL}(\mathsf{obs\_SL}(\mathsf{obs\_RN}(rts)))$ [34]
35 $hls:(\mathsf{H}|\mathsf{L})\textbf{-set} \equiv hs \cup ls$ [35]
36 $bcs:\mathsf{BC}\textbf{-set} \equiv \mathsf{obs\_BCs}(\mathsf{obs\_SBC}(\mathsf{obs\_FV}(\mathsf{obs\_RN}(rts))))$ [36]
37 $bs:\mathsf{B}\textbf{-set} \equiv \cup\{\mathsf{obs\_Bs}(bc)|bc:\mathsf{BC}\cdot bc \in bcs\}$ [37]
38 $as:\mathsf{A}\textbf{-set} \equiv \mathsf{obs\_BCs}(\mathsf{obs\_SBC}(\mathsf{obs\_FV}(\mathsf{obs\_RN}(rts))))$ [38]

8.3.1.2 Indexed States:

- We shall

87 index bus companies,
88 index buses, and
89 index automobiles

using the unique identifiers of these parts.

type

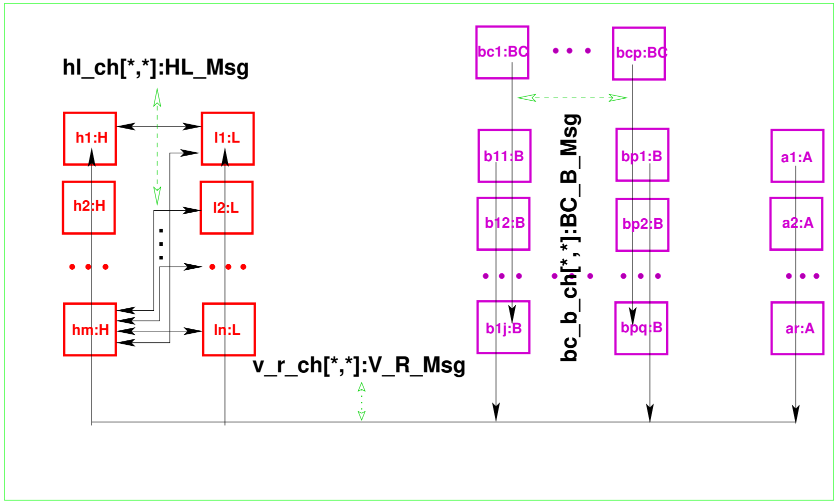
87 BC_{ui}
88 B_{ui}
89 A_{ui}

value

87 $ibcs:\mathsf{BC}_{ui}\textbf{-set} \equiv$
87 $\{ \mathsf{bc}_{ui} \mid \mathsf{bc:BC,bc:BC}_{ui}\mathsf{:BC}_{ui} \cdot \mathsf{bc} \in bcs \wedge ui = \mathsf{uid\_BC(bc)} \}$
88 $ibs:\mathsf{B}_{ui}\textbf{-set} \equiv$
88 $\{ \mathsf{b}_{ui} \mid \mathsf{b:B,b:B}_{ui}\mathsf{:B}_{ui} \cdot \mathsf{b} \in bs \wedge ui = \mathsf{uid\_B(b)} \}$
89 $ias:\mathsf{A}_{ui}\textbf{-set} \equiv$
89 $\{ \mathsf{a}_{ui} \mid \mathsf{a:A,a:A}_{ui}\mathsf{:A}_{ui} \cdot \mathsf{a} \in as \wedge ui = \mathsf{uid\_A(a)} \}$

8.3.2 Channels

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



8.3.2.1 Channel Message Types:

- We ascribe types to the messages offered on channels.

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

91 Hubs send one kind of messages, links another.

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

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

type

91 H_L_Msg, L_H_Msg
90 HL_Msg = H_L_Msg | L_F_Msg
92 BC_B_Msg = T × BusTimTbl
93 V_R_Msg = T × (BPos|APos)

8.3.2.2 Channel Declarations:

- ...

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

channel

94 { hl_ch[h_ui,l_ui]:H_L_Msg
94 | h_ui:H_UI,l_ui:L_UI•i ∈ $h_{ui}s$∧j ∈ $lh_{ui}m$(h_ui) }
94 ∪
94 { hl_ch[h_ui,l_ui]:L_H_Msg
94 | h_ui:H_UI,l_ui:L_UI•l_ui ∈ $l_{ui}s$∧i ∈ $lh_{ui}m$(l_ui) }

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

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

channel

95 {bc_b_ch[bc_ui,b_ui]:BC_B_Msg
95 | bc_ui:BC_UI, b_ui:B_UI •
95 bc_ui ∈ $bc_{ui}s$ ∧ b_ui ∈ $b_{ui}s$}
95 {bc_b_ch[bc_ui,b_ui]|bc_ui:BC_UI,b_ui:B_UI•bc_ui ∈ $bc_{ui}s$∧j ∈ $b_{ui}s$}: BC_B
95 {bc_b_ch[bc_ui,b_ui]|bc_ui:BC_UI,b_ui:B_UI•bc_ui ∈ $bc_{ui}s$∧j ∈ $b_{ui}s$}: BC_B

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

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

channel

96 {v_r_ch[v_ui,r_ui]:V_R_Msg | v_ui:V_UI,r_ui:R_UI•v_ui ∈ $v_{ui}s$∧r_ui ∈ $r_{ui}s$}

- The channel calculations are described on Slides 259–266.

8.3.3 Behaviour Signatures

- We first decide on names of behaviours.
  - In Sect. 7.4, Pages 268–276,
  - we gave schematic names to behaviours of the form $\mathscr{M}_P$.
  - We now assign mnemonic names: from part names to names of transcendentally interpreted behaviours
  - and then we assign signatures to these behaviours.

97 $\mathsf{hub}_{h_{ui}}$:

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

b. then there are the programmable attributes;

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

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


```
value
97    hub_{h_ui}:
097a.   h_ui:H_UI×(vuis,luis,_):H_Mer×HΩ
97b.     → (HΣ×H_Traffic)
97c.     → in,out { h_l_ch[h_ui,l_ui] | l_ui:L_UI:l_ui ∈ luis }
97d.           { ba_r_ch[h_ui,v_ui] | v_ui:V_UI•v_ui∈vuis } Unit
97a.      pre: vuis = v_ui s ∧ luis = l_ui s
```


98 $\mathsf{link}_{l_{ui}}$:

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

b. then there are the programmable attributes;

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

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

value

98 $\mathsf{link}_{l_{ui}}$:
98a. l_ui:L_UI×(vuis,huis,_):L_Mer×LΩ
98b. → (LΣ×L_Traffic)
98c. → **in**,**out** { h_l_ch[h_ui,l_ui] | h_ui:H_UI:h_ui ∈ huis }
98d. { ba_r_ch[l_ui,v_ui] | v_ui:(B_UI|A_UI)·v_ui∈vuis } **Unit**
98a. **pre**: vuis = $v_{ui}s$ ∧ huis = $h_{ui}s$

99 $\mathsf{bus\_company}_{bc_{ui}}$:

a. there is here just a “doublet” of arguments: unique identifier and mereology;
b. then there is the one programmable attribute;
c. and finally there are the input/output channel references: first the input time channel,
d. then the input/output allowing communication between the bus company and buses.

value

99 $\mathsf{bus\_company}_{bc_{ui}}$:
99a. bc_ui:BC_UI×(_,_,buis):BC_Mer
99b. → BusTimTbl
99c. → **in** attr_T_ch
99d. **in**,**out** {bc_b_ch[bc_ui,b_ui]|b_ui:B_UI·b_ui∈buis} **Unit**
99a. **pre**: buis = $b_{ui}s$ ∧ huis = $h_{ui}s$

100 $\mathsf{bus}_{b_{ui}}$:

a. there is here just a “doublet” of arguments: unique identifier and mereology;
b. then there are the programmable attributes;
c. and finally there are the input/output channel references: first the input time channel, and the input/output allowing communication between the bus company and buses,
d. and the input/output allowing communication between the bus and the hub and link behaviours.

value

100 $bus_{b_{ui}}$:
100a. b_ui:B_UI×(bc_ui,_,ruis):B_Mer
100b. → (LN × BTT × BPOS)
100c. → **in** attr_T_ch **in**,**out** bc_b_ch[bc_ui,b_ui],
100d. {ba_r_ch[r_ui,b_ui]|r_ui:(H_UI|L_UI)·ui∈$v_{ui}s$} **Unit**
100a. **pre**: ruis = $r_{ui}s$ ∧ bc_ui ∈ $bc_{ui}s$

101 $automobile_{a_{ui}}$:

a. there is the usual "triplet" of arguments: unique identifier, mereology and static attributes;
b. then there is the one programmable attribute;
c. and finally there are the input/output channel references: first the input time channel,
d. then the input/output allowing communication between the automobile and the hub and link behaviours.

value

101 $automobile_{a_{ui}}$:
101a. a_ui:A_UI×(_,_,ruis):A_Mer×rn:RegNo
101b. → apos:APos
101c. → **in** attr_T_ch
101d. **in**,**out** {ba_r_ch[a_ui,r_ui]|r_ui:(H_UI|L_UI)·r_ui∈ruis} **Unit**
101a. **pre**: ruis = $r_{ui}s$ ∧ a_ui ∈ $a_{ui}s$

8.3.4 Behaviour Definitions

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

8.3.4.1 Automobiles:

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

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

104 informing the hub behaviour,

105 or, internally non-deterministically,

- a. moves onto a link, tli, whose “next” hub, identified by th_ui, is obtained from the mereology of the link identified by tl_ui;
- b. informs the hub it is leaving and the link it is entering of its initial link position,
- c. whereupon the vehicle resumes the vehicle behaviour positioned at the very beginning (0) of that link,

106 or, again internally non-deterministically,

107 the vehicle “disappears — off the radar” !

102 automobile$_{a_{ui}}$(a_ui,({},(ruis,vuis),{}),rn)
102 (apos:atH(fl_ui,h_ui,tl_ui)) ≡
103 (ba_r_ch[a_ui,h_ui] ! (attr_T_ch?,atH(fl_ui,h_ui,tl_ui));
104 automobile$_{a_{ui}}$(a_ui,({},(ruis,vuis),{}),rn)(apos))
105 ⌈⌉
105a. (**let** ({fh_ui,th_ui},ruis′)=mereo_L(℘(tl_ui)) **in**
105a. **assert:** fh_ui=h_ui ∧ ruis=ruis′
102 **let** onl = (tl_ui,h_ui,0,th_ui) **in**
105b. (ba_r_ch[a_ui,h_ui] ! (attr_T_ch?,onL(onl)) ‖
105b. ba_r_ch[a_ui,tl_ui] ! (attr_T_ch?,onL(onl))) ;
105c. automobile$_{a_{ui}}$(a_ui,({},(ruis,vuis),{}),rn)
105c. (onL(onl)) **end end**)
106 ⌈⌉
107 **stop**

108 We abstract automobile behaviour on a Link.

- a. Internally non-deterministically, either
  - i the automobile remains, “idling”, i.e., not moving, on the link,
  - ii however, first informing the link of its position,
- b. or
  - i **if** if the automobile’s position on the link *has not yet reached the hub*, **then**
    - A then the automobile moves an arbitrary small, positive **Real**-valued *increment* along the link
    - B informing the hub of this,
    - C while resuming being an automobile ate the new position, or

  - ii **else**,
    - A while obtaining a “next link” from the mereology of the hub (where that next link could very well be the same as the link the vehicle is about to leave),
    - B the vehicle informs both the link and the imminent hub that it is now at that hub, identified by th_ui,
    - C whereupon the vehicle resumes the vehicle behaviour positioned at that hub;
- c. or
- d. the vehicle “disappears — off the radar” !

```
108        automobile_aui(a_ui,({},ruis,{}),rno)
108                        (vp:onL(fh_ui,l_ui,f,th_ui)) ≡
108(a.)ii   (ba_r_ch[thui,aui]!atH(lui,thui,nxt_lui) ;
108(a.)i     automobile_aui(a_ui,({},ruis,{}),rno)(vp))
108b.      ⌈⌉
108(b.)i    (if not_yet_at_hub(f)
108(b.)i      then
108(b.)iA       (let incr = increment(f) in
102              let onl = (tl_ui,h_ui,incr,th_ui) in
108(b.)iB         ba−r_ch[l_ui,a_ui] ! onL(onl) ;
108(b.)iC         automobile_aui(a_ui,({},ruis,{}),rno)
108(b.)iC                        (onL(onl))
108(b.)i          end end)
108(b.)ii      else
108(b.)iiA      (let nxt_lui:L_UI•nxt_lui ∈ mereo_H(℘(th_ui)) in
108(b.)iiB        ba_r_ch[thui,aui]!atH(l_ui,th_ui,nxt_lui) ;
108(b.)iiC        automobile_aui(a_ui,({},ruis,{}),rno)
108(b.)iiC                       (atH(l_ui,th_ui,nxt_lui)) end)
108(b.)i     end)
108c.      ⌈⌉
108d.        stop
108(b.)iA   increment: Fract → Fract
```


8.3.4.2 Hubs:

We model the hub behaviour vis-a-vis vehicles: buses and automobiles.

109 The hub behaviour

a. non-deterministically, externally offers
b. to accept timed vehicle positions —
c. which will be at the hub, from some vehicle, v_ui.
d. The timed vehicle hub position is appended to the front of that vehicle's entry in the hub's traffic table;
e. whereupon the hub proceeds as a hub behaviour with the updated hub traffic table.
f. The hub behaviour offers to accept from any vehicle.
g. A **post** condition expresses what is really a **proof obligation**: that the hub traffic, ht′ satisfies the **axiom** of the endurant hub traffic attribute Item 73 Slide 310.


```
value
109    hub_hui(h_ui,(,(luis,vuis)),hω)(hσ,ht) ≡
109a.       ⌈⌉
109b.         { let m = ba_r_ch[h_ui,v_ui] ? in
109c.              assert: m=(_,atHub(_,h_ui,_))
109d.           let ht′ = ht † [h_ui ↦ ⟨m⟩⌢ht(h_ui)] in
109e.           hub_hui(h_ui,(,(luis,vuis)),(hω))(hσ,ht′)
109f.         | v_ui:V_UI•v_ui∈vuis end end }
109g.     post: ∀ v_ui:V_UI•v_ui ∈ dom ht′⇒time_ordered(ht′(v_ui))
```


8.3.4.3 Links:

Similarly we model the link behaviour vis-a-vis vehicles.

110 The link behaviour non-deterministically, externally offers

111 to accept timed vehicle positions —

112 which will be on the link, from some vehicle, v_ui.

113 The timed vehicle link position is appended to the front of that vehicle's entry in the link's traffic table;

114 whereupon the link proceeds as a link behaviour with the updated link traffic table.

115 The link behaviour offers to accept from any vehicle.

116 A **post** condition expresses what is really a **proof obligation**: that the link traffic, lt′ satisfies the **axiom** of the endurant link traffic attribute Item 77 Slide 313.

```
110  link_lui(l_ui,(_,(huis,vuis),_),lω)(lσ,lt) ≡
110     ⌈⌉
111       { let m = ba_r_ch[l_ui,v_ui] ? in
112              assert: m=(_,onLink(_,l_ui,_,_))
113         let lt′ = lt † [l_ui ↦ ⟨m⟩⁀lt(l_ui)] in
114         link_lui(l_ui,(huis,vuis),hω)(hσ,lt′)
115       | v_ui:V_UI•v_ui∈vuis end end }
116    post: ∀ v_ui:V_UI•v_ui ∈ dom lt′⇒time_ordered(lt′(v_ui))
```


8.3.4.4 Bus Companies:

- We model bus companies very rudimentary.
  - Bus companies keep a fleet of buses.
  - Bus companies create, maintain, distribute bus time tables.
  - Bus companies deploy their buses to honor obligations of their bus time tables.
  - We shall basically only model the distribution of bus time tables to buses.
  - We shall not cover other aspects of bus company management, etc.

117 Bus companies non-deterministically, internally, chooses among
   a. updating their bus time tables
   b. whereupon they resume being bus companies, albeit with a new bus time table;

118 “interleaved” with
   a. offering the current time-stamped bus time table to buses which offer willingness to received them
   b. whereupon they resume being bus companies with unchanged bus time table.


```
99     bus_company_bcui(bcui,(_,buis,_))(btt) ≡
117a.    (let btt′ = update(btt,...) in
117b.     bus_company_bcui(bcui,(_,buis,_))(btt′) end )
118     ⌈⌉
118a.    ( ⌈⌉ {bc_b_ch[bc_ui,b_ui] ! btt | b_ui:B_UI•b_ui∈buis
118b.           bus_company_bcui(bcui,(_,buis,_))(attr_T_ch?,btt) } )
```

8.3.4.5 Buses:

- We model the interface between buses and their owning companies —
- as well as the interface between buses and the road net,
- the latter by almost "carbon-copying" all elements of the automobile behaviour(s).

119 The bus behaviour chooses to either

  a. accept a (latest) time-stamped buss time table from its bus company –

  b. where after it resumes being the bus behaviour now with the updated bus time table.

120 or, non-deterministically, internally,

  a. based on the bus position

    i if it is at a hub then it behaves as prescribed in the case of automobiles at a hub,

    ii else, it is on a link, and then it behaves as prescribed in the case of automobiles on a link.

119 $bus_{b_{ui}}$(b_ui,(_,(bc_ui,ruis),_))(ln,btt,bpos) ≡
119a. (**let** btt′ = b_bc_ch[b_ui,bc_ui] ? **in**
119b. $bus_{b_{ui}}$(b_ui,({},(bc_ui,ruis),{}))(ln,btt′,bpos) **end**)
120 ⌈⌉
120a. (**case** bpos **of**
120(a.)i atH(fl_ui,h_ui,tl_ui) →
120(a.)i atH_$bus_{b_{ui}}$(b_ui,(_,(bc_ui,ruis),_))(ln,btt,bpos),
120(a.)ii aonL(fh_ui,l_ui,f,th_ui) →
120(a.)ii onL_$bus_{b_{ui}}$(b_ui,(_,(bc_ui,ruis),_))(ln,btt,bpos)
120a. **end**)

- The atH_$bus_{b_{ui}}$ behaviour definition is a simple transcription of the
  - $automobile_{a_{ui}}$ (atH) behaviour definition:
    - mereology expressions being changed from to ,
    - programmed attributes being changed from atH(fl_ui,h_ui,tl_ui) to (ln,btt,atH(fl_ui,h_ui,tl_ui)),
    - channel references a_ui being replaced by b_ui, and
    - behaviour invocations renamed from $automobile_{a_{ui}}$ to $bus_{b_{ui}}$.
- So formula lines 103–108d. below presents "nothing new" !

120(a.)i atH_$bus_{b_{ui}}$(b_ui,(_,(bc_ui,ruis),_))
120(a.)i (ln,btt,atH(fl_ui,h_ui,tl_ui)) ≡
103 (ba_r_ch[b_ui,h_ui] ! (attr_T_ch?,atH(fl_ui,h_ui,tl_ui));
104 $bus_{b_{ui}}$(b_ui,({},(bc_ui,ruis),{}))(ln,btt,bpos))
119a. ⌈⌉
105a. (**let** ({fh_ui,th_ui},ruis′)=mereo_L(℘(tl_ui)) **in**
105a. **assert:** fh_ui=h_ui ∧ ruis=ruis′
102 **let** onl = (tl_ui,h_ui,0,th_ui) **in**
105b. (ba_r_ch[b_ui,h_ui] ! (attr_T_ch?,onL(onl)) ‖
105b. ba_r_ch[b_ui,tl_ui] ! (attr_T_ch?,onL(onl))) ;
105c. $bus_{b_{ui}}$(b_ui,({},(bc_ui,ruis),{}))
105c. (ln,btt,onL(onl)) **end end**)
108c. ⌈⌉
108d. **stop**

- The onL_bus$_{b_{ui}}$ behaviour definition is a similar simple transcription of the automobile$_{a_{ui}}$ (onL) behaviour definition.
- So formula lines 103–108d. below presents "nothing new" !

121 – this is the "almost last formula line" !


```
120(a.)ii    onL_bus_bui(b_ui,(_,(bc_ui,ruis),_))
120(a.)ii          (ln,btt,bpos:onL(fh_ui,l_ui,f,th_ui)) ≡
103          (ba_r_ch[b_ui,h_ui] ! (attr_T_ch?,bpos);
104           bus_bui(b_ui,({},(bc_ui,ruis),{}))(ln,btt,bpos))
119a.          ⌈⌉
108(b.)i        (if not_yet_at_hub(f)
108(b.)i          then
108(b.)iA            (let incr = increment(f) in
102                   let onl = (tl_ui,h_ui,incr,th_ui) in
108(b.)iB             ba−r_ch[l_ui,b_ui] ! onL(onl) ;
108(b.)iC             bus_bui(b_ui,({},(bc_ui,ruis),{}))
108(b.)iC                    (ln,btt,onL(onl))
108(b.)i              end end)
108(b.)ii          else
108(b.)iiA           (let nl_ui:L_UI•nxt_lui∈mereo_H(℘(th_ui)) in
108(b.)iiB            ba_r_ch[thui,b_ui]!atH(l_ui,th_ui,nxt_lui) ;
108(b.)iiC            bus_bui(b_ui,({},(bc_ui,ruis),{}))
108(b.)iiC                   (ln,btt,atH(l_ui,h_ui,nxt_lui))
108(b.)iiA            end)end)
108c.           ⌈⌉
121            stop
```


8.3.5 A Running System

8.3.5.1 Preliminaries:

- We recall the *hub*, *link*, *bus company*, *bus* and the *automobile states* first mentioned in Sect. 3.8 Page 296.

value

33 hs:H-**set** ≡ ≡ obs_sH(obs_SH(obs_RN(rts)))
34 ls:L-**set** ≡ ≡ obs_sL(obs_SL(obs_RN(rts)))
36 bcs:BC-**set** ≡ obs_BCs(obs_SBC(obs_FV(obs_RN(rts))))
37 bs:B-**set** ≡ ∪{obs_Bs(bc)|bc:BC•bc ∈ bcs}
39 as:A-**set** ≡ obs_BCs(obs_SBC(obs_FV(obs_RN(rts))))

8.3.5.2 Starting Initial Behaviours:

- We are reaching the end of this domain modeling example.
  - Behind us there are narratives and formalisations 18 Slide 287 – 121 Slide 369.
  - Based on these we now express the signature and the body of the definition
  - of a "*system build and execute*" function.

122 The system to be initialised is

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


```
value
122    initial_system: Unit → Unit
122    initial_system() ≡
122c.      ‖ { hub_{h_ui}(h_ui,me,hω)(htrf,hσ)
122c.         | h:H·h ∈ hs,
122c.           h_ui:H_UI·h_ui=uid_H(h),
122c.           me:HMetL·me=mereo_H(h),
122c.           hω:HΩ·hω=attr_HΩ(h),
122c.           htrf:H_Traffic·htrf=attr_H_Traffic_H(h),
122c.           hσ:HΣ·hσ=attr_HΣ(h)∧hσ ∈ hω
122c.         }
```



```
122a.      ‖
122d.      ‖ { link_{l_ui}(l_ui,me,lω)(ltrf,lσ)
122d.           l:L·l ∈ ls,
122d.           l_ui:L_UI·l_ui=uid_L(l),
122d.           me:LMet·me=mereo_L(l),
122d.           lω:LΩ·lω=attr_LΩ(l),
122d.           ltrf:L_Traffic·ltrf=attr_L_Traffic_H(l),
122d.           lσ:LΣ·lσ=attr_LΣ(l)∧lσ ∈ lω
122d.         }
```



```
122a.      ‖
122e.      ‖ { bus_company_{bc_ui}(bcui,me)(btt)
122e.           bc:BC·bc ∈ bcs,
122e.           bc_ui:BC_UI·bc_ui=uid_BC(bc),
122e.           me:BCMet·me=mereo_BC(bc),
122e.           btt:BusTimTbl·btt=attr_BusTimTbl(bc)
122e.         }
```

122a. ‖
122f. ‖ { $\text{bus}_{b_{ui}}$(b_ui,me)(ln,btt,bpos)
122f. b:B·b ∈ *bs*,
122f. b_ui:B_UI·b_ui=uid_B(b),
122f. me:BMet·me=mereo_B(b),
122f. ln:LN:pln=attr_LN(b),
122f. btt:BusTimTbl·btt=attr_BusTimTbl(b),
122f. bpos:BPos·bpos=attr_BPos(b)
122f. }

122a. ‖
122g. ‖ { $\text{automobile}_{a_{ui}}$(a_ui,me,rn)(apos)
122g. a:A·a ∈ *as*,
122g. a_ui:A_UI·a_ui=uid_A(a),
122g. me:AMet·me=mereo_A(a),
122g. rn:RegNo·rno=attr_RegNo(a),
122g. apos:APos·apos=attr_APos(a)
122g. }

8.3.6 Intentional "Pull"

- We illustrate the concept of *intentional "pull"* cf. definition on Slide 193:

123 *automobiles* include the *intent* of 'transport',
124 and so do *hubs* and *links*.

123 **attr**_Intent: A → ('transport'|...)-**set**
124 **attr**_Intent: H → ('transport'|...)-**set**
124 **attr**_Intent: L → ('transport'|...)-**set**

- *Manifestations* of 'transport' is reflected in
  - *automobiles* having the automobile position attribute, APos, Item 86 Slide 319,
  - *hubs* having the *hub traffic* attribute, H_Traffic, Item 73 Slide 310, and in
  - *links* having the *link traffic* attribute, L_Traffic, Item 77 Slide 313.

125 Seen from the point of view of an automobile there is its own traffic history, A_Hist, which is a (time ordered) sequence of timed automobile's positions;

126 seen from the point of view of a hub there is its own traffic history, H_Traffic Item 73 Slide 310, which is a (time ordered) sequence of timed maps from automobile identities into automobile positions; and

127 seen from the point of view of a link there is its own traffic history, L_Traffic Item 77 Slide 313, which is a (time ordered) sequence of timed maps from automobile identities into automobile positions.

- The *intentional "pull"* of these manifestations is this:

128 The union, i.e. proper merge of all automobile traffic histories, AllATH, must now be identical to the same proper merge of all hub, AllHTH, and all link traffic histories, AllLTH.

type

125 $\mathsf{A\_Hi} = (\mathscr{T} \times \mathsf{APos})^*$

73 $\mathsf{H\_Trf} = \mathsf{A\_UI} \overrightarrow{_m} (\mathscr{T} \times \mathsf{APos})^*$

77 $\mathsf{L\_Trf} = \mathsf{A\_UI} \overrightarrow{_m} (\mathscr{T} \times \mathsf{APos})^*$

128 $\mathsf{AllATH} = \mathscr{T} \overrightarrow{_m} (\mathsf{AUI} \overrightarrow{_m} \mathsf{APos})$

128 $\mathsf{AllHTH} = \mathscr{T} \overrightarrow{_m} (\mathsf{AUI} \overrightarrow{_m} \mathsf{APos})$

128 $\mathsf{AllLTH} = \mathscr{T} \overrightarrow{_m} (\mathsf{AUI} \overrightarrow{_m} \mathsf{APos})$

axiom

128 **let** allA=mrg_AllATH({(a,attr_A_Hi(a))|a:A·a $\in$ *as*}),

128 allH=mrg_AllHTH({attr_H_Trf(h)|h:H·h $\in$ *hs*}),

128 allL =mrg_AllLTH({attr_L_Trf(l)|l:L·h $\in$ *ls*}) **in**

128 allA = mrg_HLT(allH,allL) **end**

- We leave the definition of the merge functions to the listener !

- We now discuss the concept of *intentional "pull"*.
  - We endow
    - each automobile with its history of timed positions and
    - each hub and link with their histories of timed automobile positions.
  - These histories are facts !
  - They are not something that is laboriously recorded, where such recordings may be imprecise or cumbersome[45].
  - The facts are there, so we can (but may not necessarily) talk about these histories as facts.
  - It is in that sense that the purpose (`'transport'`)
    - for which man let automobiles, hubs and link be made
    - with their `'transport'` intent
    - are subject to an *intentional "pull"*.
- It can be no other way: if automobiles "record" their history, then hubs and links must together "record" identically the same history ! ■

[45] or thought technologically in-feasible – at least some decades ago!

.

Lecture Day 10, Lecture 19

Course Review

9 Closing

9.1 What Have We Achieved ?

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

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

- A concept of *transcendental deduction* has been introduced.
  - It is used to justify the interpretation
  - of *endurant parts*
  - as *perdurant behaviours* – a la CSP.
- A new concept of *intentional "pull"* has been introduced.
  - It applies, in the form of attributes, to humans and artifacts.
  - It "corresponds", in a way, to *gravitational pull*;
  - that concept invites further study.
- The pair of gravitational pull and intentional "pull"
  - appears to lie behind the determination
  - of the mereologies of parts;
  - that possibility invites further study.

- Finally it is shown
  - how CSP *channels* can be calculated from endurant mereologies, and
  - how the form of *behaviour arguments* can be calculated from respective attribute categorisations.
- The domain concepts outlined above
  - form a *domain ontology*
  - that applies to a wide variety of domains.

9.2 Issues of Philosophy

- Three issues of philosophy are of concern here:
  - the "nature" of the definition of the analysis prompts;
  - the *transcendental deduction* whereby *parts* are interpreted as *behaviours;* and
  - the *intentional "pull"* whereby seemingly "unrelated" parts are indeed "related" !
- They all relate to *what can be described.*

9.2.1 What Can Be Described

As for the first, consider the analysis prompts:

a. is_ entity, 6
b. is_ endurant, 6
c. is_ perdurant, 7
d. is_ discrete, 7
e. is_ continuous, 7
f. is_ physical_ part, 8
g. is_ living_ species, 8
h. is_ structure, 9
i. is_ part, 10
j. is_ atomic, 10
k. is_ composite, 10
l. is_ living_ species, 11
m. is_ plant, 11
n. is_ animal, 11
o. is_ human, 12
p. has_ components, 12
q. has_ materials, 13
r. is_ artifact, 13
s. observe_ endurants, 14
t. has_ concrete_ type, 15
u. has_ mereology, 19
v. attribute_ types, 21

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

- then you probably,
  - expecting to read a technical/scientific paper,
  - realise that those explanations are not precise in the sense
  - of such papers.
- Many of our definitions are taken
  - from [LFCO87, The Oxford Shorter English Dictionary] and
  - from the Internet based [Zal16, The Stanford Encyclopedia of Philosophy].

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

- This raises the more general question, such as we see it:
  - *"which are the absolutely necessary and unavoidable bases for describing the world ?"*
  - This is a question of philosophy.
  - We shall not develop the reasoning here.
  - Instead we refer to the forthcoming [Bjø18b, Philosophical Issues in Domain Modeling].
  - That work is based on [Sør94, Sør97, Sør02, Sør16].

9.2.2 The Transcendental Deduction

- The interpretation of *endurant parts* as *perdurant behaviours*
  - ∞ represents a *transcendental deduction* –
  - ∞ and must, somehow, be rationally justified.
  - ∞ the justification is here seen as exactly that:
  - ∞ a *transcendental deduction*
- It seems that transcendental deductions abound:
  - ∞ when compiling program texts into machine code,
  - ∞ in transitions from syntax to semantics to pragmatics, and
  - ∞ in any abstract interpretation of formal texts.
  - ∞ We refer to the forthcoming [Bjø18b, Philosophical Issues in Domain Modeling].

9.2.3 The Intentional "Pull"

- This last concept is merely a suggestion.
  - ∞ A serious paper cannot solve all issues.

9.3 Two Frequently Asked Questions

- *How much of a* DOMAIN *must or should we* ANALYSE & DESCRIBE ?
  - ∞ When this question is raised, after a talk of mine over the subject, and by a colleague researcher & scientist I usually reply:
  - ∞ *As large a domain as possible !*
  - ∞ This reply is often met by this *comment* (from the audience) *Oh ! No, that is not reasonable !*
  - ∞ To me that comment shows either or both of:
    - ∞ the questioner was not asking as a researcher/scientist, but as an engineer. Yes, an engineer needs only analyse & describe up to and slightly beyond the "border" of the domain-of-interest for a current software development – but
    - ∞ a researcher cum scientist is, of course, interested not only in a possible requirements engineering phase beyond domain engineering, but is also curious about the larger context of the domain, in possibly establishing a proper domain theory, etc.

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

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

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

9.4 On How to Pursue Domain Science & Engineering

- We set up a dogma and discuss a ramification.
  - ∞ One thing is the doctrine, the method for **domain analysis & description** outlined in this paper.
  - ∞ Another thing is its practice.
  - ∞ I find myself, when experimentally pursuing the modeling of domains, as, for example, reported in [Bjø00, BGP02, Bjø03, PSB03, SPB03, Bjø13b, Bjø13a, Bjø07, Bjø95, Bjø17a, Bjø16d, Bjø17c, Bjø17b], **not following the doctrine !**

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

  then the transcendental deduction of behaviours from parts;

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

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

- I remarked this situation to a dear friend and colleague, Dr. Ole N. Oest.
  - ∞ His remark stressed what was going on:
    - ∞ the **creative** engineer
      took possession,
    - ∞ the **exploring**, sometimes **sceptic** scientist
      entered the picture,
    - ∞ the well-trained engineer
      lost ground in the realm of imagination.
    - ∞ But perhaps, in the interest of **innovation etc.**
      it is necessary to be **creative** and **sceptic**
      and **loose ground** –
      for a while !
  - ∞ I knew that, but had sort-of-forgotten it !
- *I thank Ole N. Oest for this observation.*

9.5 Related Work

- The present lectures is but one in
  a series on the topic of *domain science & engineering*.
  - ∞ With these lectures the author expects to have laid a foundation.
  - ∞ With the many experimental case studies,
    referenced in Example 1 on Slide 36, the author seriously think
    that reasonably convincing arguments are given
    for this *domain science & engineering*.

- We comment on some previous publications:
  - [Bjø10a, Bjø18c] explores additional views on analysing & describing domains, in terms of *domain facets:*
    - *intrinsics,*
    - *support technologies,*
    - *rules & regulations,*
    - *scripts,*
    - *management & organisation,*
    - and *human behaviour.*
- [Bjø09a, Bjø18d] explores relations between Stanisław Leśhnieiski's mereology and ours.
- [Bjø08, Bjø16c] shows how to rigorously transform domain descriptions into software system requirements prescriptions.

- [Bjø16b] discusses various interpretations of domain models: as bases for
  - demos,
  - simulators,
  - real system monitors and
  - real system monitor & controllers.
- [Bjø09b] is a compendium of reports around the management and engineering of software development based in domain analysis & description.
  These reports were the result of a year at JAIST:
  Japan Institute of Science & Technology, Ishikawa, Japan.

9.6 Tony Hoare's Summary on 'Domain Modeling'

- In a 2006 e-mail, in response, undoubtedly to my steadfast – perhaps conceived as stubborn – insistence, on domain engineering,
- Tony Hoare summed up his reaction to domain engineering as follows, and I quote[46]:

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

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

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

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

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

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

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

- All of these issues were covered in [Bjø06, Part IV].

10 Acknowledgements

- I thank colleagues in China, Germany, France, Norway and Sweden:
  - Yamine Ait Ameur,
  - Dominique Méry,
  - Andreas Harmfeldt,
  - Magne Haveraaen,
  - Otthein Herzog,
  - Steve McKeever
  - and Yang ShaoFa.
- Their comments on recent papers and
- their acting as sounding boards
  for the case studies that lead to a number of
  - clarifications,
  - simplifications and
  - solidifications

  of the *domain analysis & description* method of [Bjø16e]
  now reported in the present paper are much appreciated.

- I thank Wang ShuLin of the Chinese Academy of Sciences
  for incisive questions answers to which are found,
  in particular, in Sect. 5.5 of this paper.
- And I thank Ole N. Oest for some remarks
  that lead to my remarks on Slide 403.

11 Bibliography

11.1 References

[BGP02] Dines Bjørner, Chris W. George, and Søren Prehn. Computing Systems for Railways — A Rôle for Domain Engineering. Relations to Requirements Engineering and Software for Control Applications. In *Integrated Design and Process Technology. Editors: Bernd Kraemer and John C. Petterson*, P.O.Box 1299, Grand View, Texas 76050-1299, USA, 24–28 June 2002. Society for Design and Process Science. Extended version.

[Bjø95] Dines Bjørner. Software Systems Engineering — From Domain Analysis to Requirements Capture: An Air Traffic Control Example. In *2nd Asia-Pacific Software Engineering Conference (APSEC '95)*. IEEE Computer Society, 6–9 December 1995. Brisbane, Queensland, Australia.

[Bjø00] Dines Bjørner. Formal Software Techniques in Railway Systems. In Eckehard Schnieder, editor, *9th IFAC Symposium on Control in Transportation Systems*, pages 1–12, Technical University, Braunschweig, Germany, 13–15 June 2000. VDI/VDE-Gesellschaft Mess– und Automatisieringstechnik, VDI-Gesellschaft für Fahrzeug– und Verkehrstechnik. Invited talk.

[Bjø02] Dines Bjørner. Domain Models of "The Market" — in Preparation for E–Transaction Systems. In *Practical Foundations of Business and System Specifications (Eds.: Haim Kilov and Ken Baclawski)*, The Netherlands, December 2002. Kluwer Academic Press. Final draft version.

[Bjø03] Dines Bjørner. Dynamics of Railway Nets: On an Interface between Automatic Control and Software Engineering. In *CTS2003: 10th IFAC Symposium on Control in Transportation Systems*, Oxford, UK, August 4-6 2003. Elsevier Science Ltd. Symposium held at Tokyo, Japan. Editors: S. Tsugawa and M. Aoki. Final version.

[Bjø06] Dines Bjørner. *Software Engineering, Vol. 3: Domains, Requirements and Software Design*. Texts in Theoretical Computer Science, the EATCS Series. Springer, 2006.

[Bjø07] Dines Bjørner. A Container Line Industry Domain. Techn. report, Fredsvej 11, DK-2840 Holte, Denmark, June 2007. Extensive Draft.

[Bjø08] Dines Bjørner. From Domains to Requirements. In *Montanari Festschrift*, volume 5065 of *Lecture Notes in Computer Science (eds. Pierpaolo Degano, Rocco De Nicola and José Meseguer)*, pages 1–30, Heidelberg, May 2008. Springer.

[Bjø09a] Dines Bjørner. On Mereologies in Computing Science. In *Festschrift: Reflections on the Work of C.A.R. Hoare*, History of Computing (eds. Cliff B. Jones, A.W. Roscoe and Kenneth R. Wood), pages 47–70, London, UK, 2009. Springer.

[Bjø09b] Dines Bjørner. *Domain Engineering: Technology Management, Research and Engineering*. A JAIST Press Research Monograph # 4, 536 pages, March 2009.

[Bjø10a] Dines Bjørner. Domain Engineering. In Paul Boca and Jonathan Bowen, editors, *Formal Methods: State of the Art and New Directions*, Eds. Paul Boca and Jonathan Bowen, pages 1–42, London, UK, 2010. Springer.

[Bjø10b] Dines Bjørner. On Development of Web-based Software: A Divertimento of Ideas and Suggestions. Technical, Technical University of Vienna, August–October 2010. http://www.imm.dtu.dk/˜dibj/wfdftp.pdf.

[Bjø10c] Dines Bjørner. The Tokyo Stock Exchange Trading Rules. R&D Experiment, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, Denmark, January and February, 2010. Version 1, Version 2.

[Bjø11] Dines Bjørner. Domains: Their Simulation, Monitoring and Control – A Divertimento of Ideas and Suggestions. In *Rainbow of Computer Science, Festschrift for Hermann Maurer on the Occasion of His 70th Anniversary.*, Festschrift (eds. C. Calude, G. Rozenberg and A. Saloma), pages 167–183. Springer, Heidelberg, Germany, January 2011.

[Bjø13a] Dines Bjørner. Pipelines – a Domain Description[47]. Experimental Research Report 2013-2, DTU Compute and Fredsvej 11, DK-2840 Holte, Denmark, Spring 2013.

[Bjø13b] Dines Bjørner. Road Transportation – a Domain Description[48]. Experimental Research Report 2013-4, DTU Compute and Fredsvej 11, DK-2840 Holte, Denmark, Spring 2013.

[47] http://www.imm.dtu.dk/˜dibj/pipe-p.pdf
[48] http://www.imm.dtu.dk/˜dibj/road-p.pdf

[Bjø14] Dines Bjørner. *A Rôle for Mereology in Domain Science and Engineering*. Synthese Library (eds. Claudio Calosi and Pierluigi Graziani). Springer, Amsterdam, The Netherlands, October 2014.

[Bjø16a] Dines Bjørner. A Credit Card System: Uppsala Draft. Technical Report: Experimental Research, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, Denmark, November 2016. http://www.imm.dtu.dk/˜dibj/2016/credit/accs.pdf.

[Bjø16b] Dines Bjørner. Domains: Their Simulation, Monitoring and Control – A Divertimento of Ideas and Suggestions. Technical report, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, Denmark, 2016. Extensive revision of [Bjø11]. http://www.imm.dtu.dk/˜dibj/2016/demos/faoc-demo.pdf.

[Bjø16c] Dines Bjørner. From Domain Descriptions to Requirements Prescriptions – A Different Approach to Requirements Engineering. Technical report, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, Denmark, 2016. Extensive revision of [Bjø08].

[Bjø16d] Dines Bjørner. Weather Information Systems: Towards a Domain Description. Technical Report: Experimental Research, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, Denmark, November 2016. http://www.imm.dtu.dk/˜dibj/2016/wis/wis-p.pdf.

[Bjø16e] Dines Bjørner. Manifest Domains: Analysis & Description. *Formal Aspects of Computing*, 29(2):175–225, Online: July 2016.

[Bjø17a] Dines Bjørner. A Space of Swarms of Drones. Research Note, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, Denmark, November–December 2017. http://www.imm.dtu.dk/˜dibj/2017/docs/docs.pdf.

[Bjø17b] Dines Bjørner. What are Documents ? Research Note, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, Denmark, July 2017. http://www.imm.dtu.dk/˜dibj/2017/docs/docs.pdf.

[Bjø17c] Dines Bjørner. Urban Planning Processes. Research Note, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, Denmark, July 2017. http://www.imm.dtu.dk/˜dibj/2017/up/urban-planning.pdf.

[Bjø18a] Dines Bjørner. A Domain Analysis & Description Method – Principles, Techniques and Modeling Languages. Research Note based on [Bjø16e], Technical University of Denmark, Fredsvej 11, DK-2840 Holte, Denmark, February 20 2018. http://www.imm.dtu.dk/ dibj/2018/tosem/Bjorner-TOSEM.pdf.

[Bjø18b] Dines Bjørner. A Philosophy of Domain Science & Engineering – An Interpretation of Kai Sørlander's Philosophy. Research Note, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, Denmark, Spring 2018. http://www.imm.dtu.dk/˜dibj/2018/philosophy/filo.pdf.

[Bjø18c] Dines Bjørner. Domain Facets: Analysis & Description. May 2018. Extensive revision of [Bjø10a]. http://www.imm.dtu.dk/˜dibj/2016/facets/faoc-facets.pdf.

[Bjø18d] Dines Bjørner. To Every Manifest Domain a CSP Expression — A Rôle for Mereology in Computer Science. *Journal of Logical and Algebraic Methods in Programming*, (94):91–108, January 2018.

[CV99] Roberto Casati and Achille C. Varzi. *Parts and Places: the structures of spatial representation*. MIT Press, 1999.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes*. C.A.R. Hoare Series in Computer Science. Prentice-Hall International, 1985. Published electronically: http://www.usingcsp.com/cspbook.pdf (2004).

[Jac95] Michael A. Jackson. *Software Requirements & Specifications: a lexicon of practice, principles and prejudices*. ACM Press. Addison-Wesley, Reading, England, 1995.

[LFCO87] W. Little, H.W. Fowler, J. Coulson, and C.T. Onions. *The Shorter Oxford English Dictionary on Historical Principles*. Clarendon Press, Oxford, England, 1973, 1987. Two vols.

[PSB03] Martin Pěnička, Albena Kirilova Strupchanska, and Dines Bjørner. Train Maintenance Routing. In *FORMS'2003: Symposium on Formal Methods for Railway Operation and Control Systems*. L'Harmattan Hongrie, 15–16 May 2003. Conf. held at Techn.Univ. of Budapest, Hungary. Editors: G. Tarnai and E. Schnieder, Germany. Final version.

[Sør94] Kai Sørlander. *Det Uomgængelige – Filosofiske Deduktioner [The Inevitable – Philosophical Deductions, with a foreword by Georg Henrik von Wright]*. Munksgaard · Rosinante, 1994. 168 pages.

[Sør97] Kai Sørlander. *Under Evighedens Synsvinkel [Under the viewpoint of eternity]*. Munksgaard · Rosinante, 1997. 200 pages.

[Sør02] Kai Sørlander. *Den Endegyldige Sandhed [The Final Truth]*. Rosinante, 2002. 187 pages.

[Sør16] Kai Sørlander. *Indføring i Filosofien [Introduction to The Philosophy]*. Informations Forlag, 2016. 233 pages.

[SPB03] Albena Kirilova Strupchanska, Martin Pěnička, and Dines Bjørner. Railway Staff Rostering. In *FORMS2003: Symposium on Formal Methods for Railway Operation and Control Systems*. L'Harmattan Hongrie, 15–16 May 2003. Conf. held at Techn.Univ. of Budapest, Hungary. Editors: G. Tarnai and E. Schnieder, Germany. Final version.

[Zal16] Edward N. Zalta. The Stanford Encyclopedia of Philosophy. 2016. Principal Editor: https://plato.stanford.edu/.