PhD Lectures, TU Wien, 19–30 October 2015

195.085: Domains and Requirements: Science & Engineering

Dines Bjørner

Prof., Dr., ...

- Course Objective
- Course Aims
- Lecture Material
- Lecture Schedule
- Übungen: Topics

- **Course Objective:**
  - To introduce students to
    a rather different, initial approach to software development
  - to such a degree that the students can, afterwards,
  - themselves undertake "that" new approach
  - to software development
    in an "experimental engineering" fashion
    while reflecting on research issues.

- **Course Aims:**
  - To cover
    - principles and
    - techniques of, and
    - tools for
    - domain analysis & description

    and
    - principles and
    - techniques of,
    - and tools for
    - the systematic "derivation"
      of requirements prescriptions from domain descriptions.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

- **Lecture Material:**
  - **Paper: Manifest Domains: Analysis & Description**
    [D] www.imm.dtu.dk/~dibj/2015/tuv/faoc-md-aad.pdf
  - **Paper: From Domain Descriptions to Requirements Prescriptions: A Different Approach to Requirements Engineering**
    [R] www.imm.dtu.dk/~dibj/2015/tuv/faoc-req.pdf
  - **Slides:** Covering Domains and Requirements, 700 Slides
    www.imm.dtu.dk/~dibj/2015/tuv/md-aad-slides.pdf

- **Day-by-day Schedule:** approximate !

Domains

- Mon.19.10:
  Introduction [D:1.1–1.9] Slides 1–82 and
  An Example [R:2] Slides 397–447
- Tue.20.10:
  Entities [D:2] Slides 83–100 and
  Parts, Components, Materials [D:3.1] Slides 101–150
- Wed.21.10:
  Unique Identifiers [D.3.2] Slides 151–155
  Mereology [D.3.3] Slides 156–169
  Attributes [D.3.4] Slides 170–234
- Thu.22.10:
  Perdurants: Actions and Events [D:4.1–4.10] Slides 235–282
- Fri.23.10:
  Perdurants: Behaviours [D:4.11–4.13] Slides 283–310
  Closing [D:5] Slides 311–371

Weekend ! study and project work !

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

Requirements

- Mon.26.10:
  Introduction [R:1] Slides 375–396 and
  Requirements [R:3] Slides 448–486
- Tue.27.10:
  Domain Requirements [R:4.1–4.2] Slides 487–527
- Wed.28.10:
  Domain Requirements [R:4.3–4.6] Slides 528–588
- Thu.29.10:
  Interface Requirements [R:5] Slides 589–623
- Fri.30.10:
  Machine Requirements [R:6] Slides 624–696 and
  Conclusion [R:7] Slides 697–716

- **Übungen – Course Projects**
  - **Topics:**
    - **Documents:** Create, Edit, Read, Copy, Authorise, Refer, ...
    - **Railway Systems:** Net, Trains, Traffic, ...
    - **Water Supply Systems:** Sources, Net, Sinks – and all between!
    - **Health Care Systems:** Hospitals, Clinics, Pharmacies, Insurance, ...
    - **Energy:** Coal, Gas, Nuclear, Solar, Water, Wind, ...
    - **Container Lines:** Containers, Vessels, Ports, Routes, ...
    - **Or ?**
  - **Teams:** 3–4 [PhD] Students per project
  - **Daily "Laboratory"**
    - **Lecturer** guides teams
    - **Teams** present copy of hand-written description before each "laboratory"
    - **Class + Lecturer** reviews respective descriptions in class
    - **Lecturer** guides teams for next stage and steps

Manifest Domains

Analysis & Description

Dines Bjørner

Technical University of Vienna

19–30 October 2015

Summary

- We show that manifest domains,
  - ◈ an understanding of which are
  - ◈ a prerequisite for software requirements prescriptions,

  can be precisely described:
  - ◈ narrated and
  - ◈ formalised.

- We show that manifest domains can be understood as a collection of
  - endurant, that is, basically spatial entities:
    - parts,
    - components and
    - materials,

    and
  - perdurant, that is, basically temporal entities:
    - actions,
    - events
    - and behaviours.

- We show that parts can be modeled in terms of
  - external qualities whether:
    - atomic or
    - composite

    parts,
- having internal qualities:
  - unique identifications,
  - mereologies, which model relations between parts, and
  - attributes.

- We show that the manifest domain analysis endeavour can be supported by a calculus of manifest domain analysis prompts:
  - `is_entity`,
  - `is_endurant`,
  - `is_perdurant`,
  - `is_part`,
  - `is_component`,
  - `is_material`,
  - `is_atomic`,
  - `is_composite`,
  - `has_components`,
  - `has_materials`,
  - `has_concrete_type`,
  - `attribute_names`,
  - `is_stationary`, etcetera.

- We show how the manifest domain description endeavour can be supported by a calculus of manifest domain description prompts:
  - observe_part_sorts,
  - observe_part_type,
  - observe_components,
  - observe_materials,
  - observe_unique_identifier,
  - observe_mereology,
  - observe_attributes,
  - observe_location and
  - observe_position.

- We show how to model attributes, essentially following Michael Jackson, [Jac95a], but with a twist:
  - The model of attributes introduces the attribute analysis prompts
    - `is_static_attribute`,
    - `is_dynamic_attribute`,
    - `is_inert_attribute`,
    - `is_reactive_attribute`,
    - `is_active_attribute`,
    - `is_autonomous_attribute`,
    - `is_biddable_attribute` and
    - `is_programmable_attribute`.

- The twist suggests ways of modeling "access" to the values of these kinds of attributes:
  - the static attributes by simply *"copying"* them, once,
  - the programmable attributes by *"carrying"* them as function parameters whose values are kept always updated, and
  - the remaining, the `external_attribute`s, by inquiring, when needed, as to their value, as if they were always offered on `CSP`-like channels [Hoa85].

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

- We show how to model essential aspects of perdurants in terms of their signatures based on the concepts of endurants.
- And we show how one can "compile"
  - descriptions of endurant parts into
  - descriptions of perdurant behaviours.
- We do not show prompt calculi for perdurants.
- The above contributions express a method
  - with principles, techniques and tools
  - for constructing domain descriptions.

1. Introduction

- The broader subject of these lectures is that of software development.
- The narrower subject is that of manifest domain engineering.
- We shall see software development
  in the context of the `TripTych` approach (next section).

- The contribution of these lectures is twofold:
  - the propagation of manifest domain engineering
    - as a first phase of the development of
    - a large class of software —

  and

    - a set of principles, techniques and tools
    - for the engineering of the analysis & descriptions
    - of manifest domains.

- These principles, techniques and tools are embodied in a set of analysis and description prompts.
  - We claim that this embodiment
  - — in the form of prompts —
  - is novel.

1.1. The TripTych Approach to Software Engineering

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

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

- This seminar contributes, we claim, to a **methodology** for **domain analysis** &[1] **domain description**.
- References [dines:ugo65:2008]
  - show how to "refine" **domain descriptions** into **requirements prescriptions**,

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

[1] When, as here, we write A & B we mean A & B to be one subject.

- In branches of engineering based on natural sciences
  - professional engineers are educated in these sciences.
  - Telecommunications engineers know Maxwell's Laws.
    - Maybe they cannot themselves "discover" such laws,
    - but they can "refine" them into designs,
    - for example, for mobile telephony radio transmission towers.
  - Aeronautical engineers know laws of fluid mechanics.
    - Maybe they cannot themselves "discover" such laws,
    - but they can "refine" them into designs,
    - for example, for the design of airplane wings.
  - And so forth for other engineering branches.

- Our point is here the following:
  - software engineers must domain specialise.
  - This is already done, to a degree, for designers of
    - compilers,
    - operating systems,
    - database systems,
    - Internet/Web systems,

    etcetera.
  - But is it done for software engineering
    - banking systems,
    - traffic systems,
    - health care,
    - insurance, etc. ?
  - We do not think so, but we claim it should be done.

- The concept of **systems engineering** arises naturally in the TripTych approach.
  - First: *domains can be claimed to be systems*.
  - Secondly: *requirements are usually not restricted to software, but encompasses all the human and technological "assists" that must be considered*.
  - Other than that we do not wish to consider domain analysis & description principles, techniques and tools specific to "systems engineering".

1.2. Method and Methodology

1.2.1. Method

- By a **method** we shall understand
  - a "somehow structured" set of `principles`
  - for `selecting` and `apply`ing
  - a number of `techniques` and `tools`
  - for `analysing` problems and `synthesizing` solutions
  - for a given domain ■[2]

[2]Definitions and examples are delimited by ■ respectively ■ symbols.

- The 'somehow structuring' amounts,
  - in this treatise on domain analysis & description,
  - to the techniques and tools being related to a set of
  - domain analysis & description "prompts",
  - "issued by the method",
  - prompting the domain engineer,
  - hence carried out by the **domain analyser** & **describer**[3] —
  - conditional upon the result of other prompts.

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

1.2.2. Discussion

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

- The **main principles** of the `TripTych` domain analysis and description approach are those of
  - abstraction and both
    - narrative and
    - formal
  - modeling.
  - This means that evolving domain descriptions
    - necessarily limit themselves to a subset of the domain
    - focusing on what is considered relevant, that is,
    - abstract "away" some domain phenomena.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

- The **main techniques** of the `TripTych` domain analysis and description approach are
  - besides those techniques which are in general associated with formal descriptions,
  - focus on the techniques that relate to the deployment of of the individual prompts.

- And the **main tools** of the `TripTych`
  domain analysis and description approach are
  - the analysis and description prompts and the
  - description language, here the `R`aise `S`pecification `L`anguage `RSL`.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

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

  of the domain analysis & description method.

1.2.3. Methodology

- By **methodology** we shall understand
  - the study and knowledge
  - about one or more methods[4] ■

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

1.3. Computer and Computing Science

- By **computer science** we shall understand
  - ⟐ the study and knowledge of
    - ⊚ the conceptual phenomena
    - ⊚ that "exists" inside computers
  - ⟐ and, in a wider context than just computers and computing,
    - ⊚ of the theories "behind" their
    - ⊚ formal description languages ■
- Computer science is often also referred to as theoretical computer science.

- By **computing science** we shall understand
  - the study and knowledge of
    - how to construct
    - and describe

    those phenomena ■
- Another term for computing science is programming methodology.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

- These lectures are about computing science.
  - They are concerned with the construction of domain descriptions.
  - They put forward a calculus for analysing and describing domains.
  - They do not theorize about this calculus.
  - There are no theorems about this calculus and hence no proofs.
  - We leave that to another study and set of lectures.

1.4. What Is a Manifest Domain ?

- We offer a number of complementary delineations of what we mean by a manifest domain.
- But first some examples, "by name" !

Example 1 **Names of Manifest Domains**: Examples of suggestive names of manifest domains are:

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

- A **manifest domain** is a
  - human- and
  - artifact-assisted
  - arrangement of
    - **endurant**, that is spatially "stable", and
    - **perdurant**, that is temporally "fleeting"

    entities.
  - Endurant entities are
    - either parts
    - or components
    - or materials.
  - Perdurant entities are
    - either actions
    - or events
    - or behaviours

Example 2 **Manifest Domain Endurants**: Examples of (names of) endurants are

- **Air traffic:** *aircraft, airport, air lane.*
- **Banks**: *client, passbook.*
- **Container lines:** *container, container vessel, terminal port.*
- **Documents:** *document, document collection.*
- **Hospitals:** *patient, medical staff, ward, bed, medical journal.*
- **Pipelines:** *well, pump, pipe, valve, sink, oil.*
- **Railways:** *simple rail unit, point, crossover, line, track, station.*
- **Road nets:** *link (street segment), hub (street intersection)* ■

Example 3 **Manifest Domain Perdurants**: Examples of (names of) perdurants are

- **Air traffic:** *start (ascend) an aircraft, change aircraft course.*
- **Banks:** *open, deposit into, withdraw from, close (an account).*
- **Container lines:** *move container off or on board a vessel.*
- **Documents:** *open, edit, copy, shred.*
- **Hospitals:** *admit, diagnose, treat (patients).*
- **Pipelines:** *start pump, stop pump, open valve, close valve.*
- **Railways:** *switch rail point, start train.*
- **Road nets:** *set a hub signal, sense a vehicle* ■

- A **manifest domain** is further seen as a mapping
  - from *entities*
  - to *qualities*,

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

Example 4 **Endurant Entity Qualities**: Examples of (names of) endurant qualities:

- **Pipeline:**
  - *unique identity of a pipeline unit,*
  - *mereology (connectedness) of a pipeline unit,*
  - *length of a pipe,*
  - *(pumping) height of a pump,*
  - *open/close status of a valve.*
- **Road net:**
  - *unique identity of a road unit (hub or link),*
  - *road unit mereology:*
    - *identity of neighbouring hubs of a link,*
    - *identity of links emanating from a hub,*
  - *and state of hub (traversal) signal* ■

Example 5 **Perdurant Entity Qualities**: Examples of (names of) perdurant qualities:

- **Pipeline:**
  - *the signature of an open (or close) valve action,*
  - *the signature of a start (or stop) pump action,*
  - *etc.*
- **Road net:**
  - *the signature of an insert (or remove) link action,*
  - *the signature of an insert (or remove) hub action,*
  - *the signature of a vehicle behaviour,*
  - *etc.* ■

We shall in the rest of this paper just write 'domain' instead of 'manifest domain'.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

1.5. What Is a Domain Description ?

- By a **domain description** we understand
  - a collection of pairs of
  - narrative and commensurate
  - formal

  texts, where each pair describes
  - either aspects of an endurant entity
  - or aspects of a perdurant entity

- What does it mean that some text describes a domain entity ?
- For a text to be a **description text** it must be possible
  - to either, if it is a narrative,
    - to reason, informally, that the ***designated*** entity
    - is described to have some properties
    - that the reader of the text can observe
    - that the described entities also have;
  - or, if it is a formalisation
    - to prove, mathematically,
    - that the formal text
    - ***denotes*** the postulated properties ■

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

- By a **domain description** we shall thus understand a text which describes
  - the **entities** of the domain:
    - whether **endurant** or **perdurant**,
    - and when endurant whether
      - **discrete** or **continuous**,
      - **atomic** or **composite**;
    - or when perdurant whether
      - **actions**,
      - **events** or
      - **behaviours**.
  - as well as the **qualities** of these **entities**.

- So the task of the domain analyser cum describer is clear:
  - There is a domain: right in front of our very eyes,
  - and it is expected that that domain be described.

1.6. Towards a Methodology of Domain Analysis & Description

1.6.1. Practicalities of Domain Analysis & Description.

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

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

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

1.6.2. The Four Domain Analysis & Description "Players".

- We can say that there are four 'players' at work here.
  - (i) the domain,
  - (ii) the domain analyser & describer,
  - (iii) the domain analysis & description method, and
  - (iv) the evolving domain analysis & description (document).

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

- The *domain* is there.
  - The domain analyser & describer cannot change the domain.
  - Analysing & describing the domain does not change it[5].
  - During the analysis & description process
    - the domain can be considered inert.
    - (It changes with the installation of the software
    - that has been developed from the
    - requirements developed from the
    - domain description.)
  - In the physical sense the domain will usually contain
    - entities that are static (i.e., constant), and
    - entities that are dynamic (i.e., variable).

[5] Observing domains, such as we are trying to encircle the concept of domain, is not like observing the physical world at the level of subatomic particles. The experimental physicists' instruments of observation change what is being observed.

- The domain analyser & domain describer is a human,
  - preferably a scientist/engineer[6],
  - well-educated and trained in domain science & engineering.
  - The domain analyser & describer
    - observes the domain,
    - analyses it according to a method and
    - thereby produces a domain description.

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

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

- Finally there is the evolving *domain analysis & description*.
  - That description is a text, usually both informal and formal.
  - Applying a *domain description prompt* to the domain
    - yields an *additional domain description text*
    - which is added to the thus evolving *domain description*.

- One may speculate of the rôle of the "input" domain description.
  - Does it change ?
  - Does it help determine the additional domain description text ?
  - Etcetera.
- Without loss of generality we can assume
  - that the "input" domain description is changed and
  - that it helps determine the added text.

- Of course, analysis & description is a trial-and-error, iterative process.
  - During a sequence of analyses,
  - that is, analysis prompts,
  - the analyser "discovers"
  - either more pleasing abstractions
  - or that earlier analyses or descriptions were wrong,
  - or that an entity either need be abstracted or made less abstract.
  - So they are corrected.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

1.6.3. An Interactive Domain Analysis & Description Dialogue.

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

1.6.4. Prompts

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

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

1.6.5. A Domain Analysis & Description Language.

- The 'Domain Analysis & Description Language' thus consists of a number of meta-functions, the prompts.
  - The meta-functions have names (say `is_endurant`) and types,
  - but have no formal definition.
  - They are not computable.
  - They are "performed" by the domain analysers & describers.
  - These meta-functions are systematically introduced and informally explained in Sects. 2–4.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

1.6.6. The Domain Description Language.

- The ‘Domain Description Language’ is RSL [GHH+92], the RAISE Specification Language [GHH+95].
- With suitable, simple adjustments it could also be either of
  - Alloy [Jac06],
  - Event B [Abr09],
  - VDM-SL [BJ78, BJ82, FL98] or
  - Z [WD96].
- We have chosen RSL because of its simple provision for
  - defining sorts,
  - expressing axioms, and
  - postulating observers over sorts.

1.6.7. Domain Descriptions: Narration & Formalisation

- Descriptions
  - *must* be readable and
  - *must* be mathematically precise.[7]
- For that reason we decompose domain description fragments into clearly identified "pairs" of
  - narrative texts and
  - formal texts.

[7] One must insist on formalised domain descriptions in order to be able to verify that domain descriptions satisfy a number of properties not explicitly formulated as well as in order to verify that requirements prescriptions satisfy domain descriptions.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

1.7. One Domain – Many Models ?

- Will two or more domain engineers cum scientists arrive at "the same domain description" ?
- No, almost certainly not !
- What do we mean by "the same domain description" ?
  - To each proper description we can associate a mathematical meaning, its semantics.
  - Not only is it very unlikely that the syntactic form of the domain descriptions are the same or even "marginally similar".
  - But it is also very unlikely that the two (or more) semantics are the same;
  - that is, that all properties that can be proved for one domain model can be proved also for the other.

- Why will different domain models emerge ?
  - Two different domain describers will, undoubtedly,
  - when analysing and describing independently,
  - focus on different aspects of the domain.
    - One describer may focus attention on certain phenomena,
    - different from those chosen by another describer.
    - One describer may choose some abstractions
    - where another may choose more concrete presentations.
    - Etcetera.

- We can thus expect that a set of domain description developments lead to a set of distinct models.
  - As these domain descriptions
    - are communicated amongst domain engineers cum scientists
    - we can expect that iterated domain description developments
    - within this group of developers
    - will lead to fewer and more similar models.
  - Just like physicists,
    - over the centuries of research,
    - have arrived at a few models of nature,
    - we can expect there to develop some consensus models of "standard" domains.

- We expect, that sometime in future, software engineers,
  - when commencing software development for a “standard domain”, that is,
  - one for which there exists one or more “standard models”,
  - will start with the development of a domain description
  - based on “one of the standard models” —
  - just like control engineers of automatic control
  - “repeat” an essence of a domain model for a control problem.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

Example 6 One Domain – Three Models:

- In this paper we shall bring many examples from a domain containing automobiles.
  - (i) One domain model may focus on roads and vehicles, with roads being modeled in terms of atomic hubs (road intersections) and atomic links (road sections between immediately neighbouring hubs), and with automobiles being modeled in terms of atomic vehicles.
  - (ii) Another domain model considers hubs of the former model as being composite, consisting, in addition to the "bare" hub, also of a signaling part — with automobiles remaining atomic vehicles,
  - (iii) A third model focuses on vehicles, now as composite parts consisting of composite and atomic sub-parts such as they are relevant in the assembly-line manufacturing of cars[8] ■

[8] The road nets of the first two models can be considered a zeroth model.

1.8. Formal Concept Analysis

- Domain analysis involves that of concept analysis.
- As soon as we have identified an entity for analysis we have identified a concept.
  - The entity is usually a spatio-temporal, i.e., a physical thing.
  - Once we speak of it, it becomes a concept.
- Instead of examining just one entity the domain analyser shall examine many entities.
- Instead of describing one entity the domain describer shall describe a class of entities.
- Ganter & Wille's [GW99] addresses this issue.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

1.8.1. A Formalisation

Some Notation:

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

Definition: 1 Formal Context:

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

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

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

Definition: 2 Qualities Common to a Set of Entities:

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

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

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

Definition: 3 Entities Common to a Set of Qualities:

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

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

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

Definition: 4 Formal Concept:

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

1.8.2. Types Are Formal Concepts

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

1.8.3. Practicalities

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

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

1.8.4. Formal Concepts: A Wider Implication

- The formal concepts of a domain form Galois Connections [GW99].
  - We gladly admit that this fact is one of the reasons why we emphasise **formal concept analysis**.
  - At the same time we must admit that this seminar does not do justice to this fact.
  - We have experimented with the analysis & description of a number of domains,
  - and have noticed such Galois connections,
  - but it is, for us, too early to report on this.
- Thus we invite the student to study this aspect of domain analysis.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

1.9. Structure of Seminar

- Sections 2.–4. are the main sections of this seminar.
  - They cover the analysis and description of
  - endurants and perdurants.
- Section 2. introduce the concepts of
  - entities,
  - endurant entities and
  - perdurant entities.

- Section 3. can be considered the main contribution of this seminar. It introduces
  - the external qualities of
    - parts,
    - components and
    - materials,

    and
    - the internal qualities of
      - unique part identifiers,
      - part mereologies and
      - part attributes.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

- Section 4. complements Sect. 3.
  - It covers a less systematic analysis and description of perdurants.
- Section 5. concludes the seminar.

2. Entities

2.1. General

Definition 1 Entity:

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

    by humans,
  - **or** *that can be* **conceived**
    - *as an* **abstraction**
    - *of an entity.*
  - *We further demand that an entity can be objectively described*[9]

[9] **Definitions** and **examples** are delimited by respectively

Analysis Prompt 1 *is_entity:*

- *The domain analyser analyses "things" (θ) into either entities or non-entities.*
- *The method can thus be said to provide the* **domain analysis prompt***:*
  - *is_entity — where is_entity(θ) holds if θ is an entity*[10]
- is_entity is said to be a ***prerequisite prompt*** for all other prompts.

[10] **Analysis** prompt definitions and **description** prompt definitions and schemes are delimited by respectively .

Whither Entities:

- The "demands" that entities
  - be observable and objectively describable

  raises some philosophical questions.
- Can sentiments, like feelings, emotions or "hunches" be objectively described ?
- This author thinks not.
- And, if so, can they be other than artistically described ?
- It seems that
  - psychologically and
  - aesthetically

  "phenomena" appears to lie beyond objective description.
- We shall leave these speculations for later.

2.2. Endurants and Perdurants

Definition 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.*

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

- That is, endurants "reside" in space.
- Endurants are, in the words of Whitehead (1920), continuants.

Example 7 Traffic System Endurants:
Examples of traffic system endurants are:

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

Definition 3 Perdurant:

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

- That is, perdurants "reside" in space and time.
- Perdurants are, in the words of Whitehead(1920), occurrents.

Example 8 Traffic System Perdurants:
Examples of road net perdurants are:

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

Analysis Prompt 2 *is_endurant:*

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

Analysis Prompt 3 *is_perdurant:*

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

- In the words of Whitehead(1920)
  - an endurant has stable qualities that enable its various appearances at different times to be recognised as the same individual;
  - a perdurant is in a state of flux that prevents it from being recognised by a stable set of qualities.

Necessity and Possibility:

- It is indeed possible to make the endurant/perdurant distinction.
- But is it necessary ?
- We shall argue that it is 'by necessity' that we make this distinction.
  - Space and time are fundamental notions.
  - They cannot be dispensed with.
  - So, to describe manifest domains
    without resort to space and time
    is not reasonable.

2.3. Discrete and Continuous Endurants

Definition 4 Discrete Endurant:

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

  in form or concept

Example 9 Discrete Endurants:

- Examples of discrete endurants are
  - a road net,
  - a link,
  - a hub,
  - a vehicle,
  - a traffic signal,
  - etcetera

Definition 5 Continuous Endurant:

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

Example 10 Continuous Endurants:

- Examples of continuous endurants are
  - water,
  - gas,
  - grain,
  - oil,
  - sand,
  - etcetera

- 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 and components
    shall be seen as 'continuous',

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* ■

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*

2.4. An Upper Ontology Diagram of Domains

- Figure 1 on the following slide shows a so-called upper ontology for manifest domains.
  - So far we have covered only a fraction of this ontology, as noted.
  - By ontologies we shall here understand
  - *"formal representations of a set of concepts within a domain and the relationships between those concepts"*.
  - In Sect. we shall review
    - relations between our approach to modeling domains and
    - that of many related modeling approaches,
    - including the so-called ontology approach based on AI-models.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

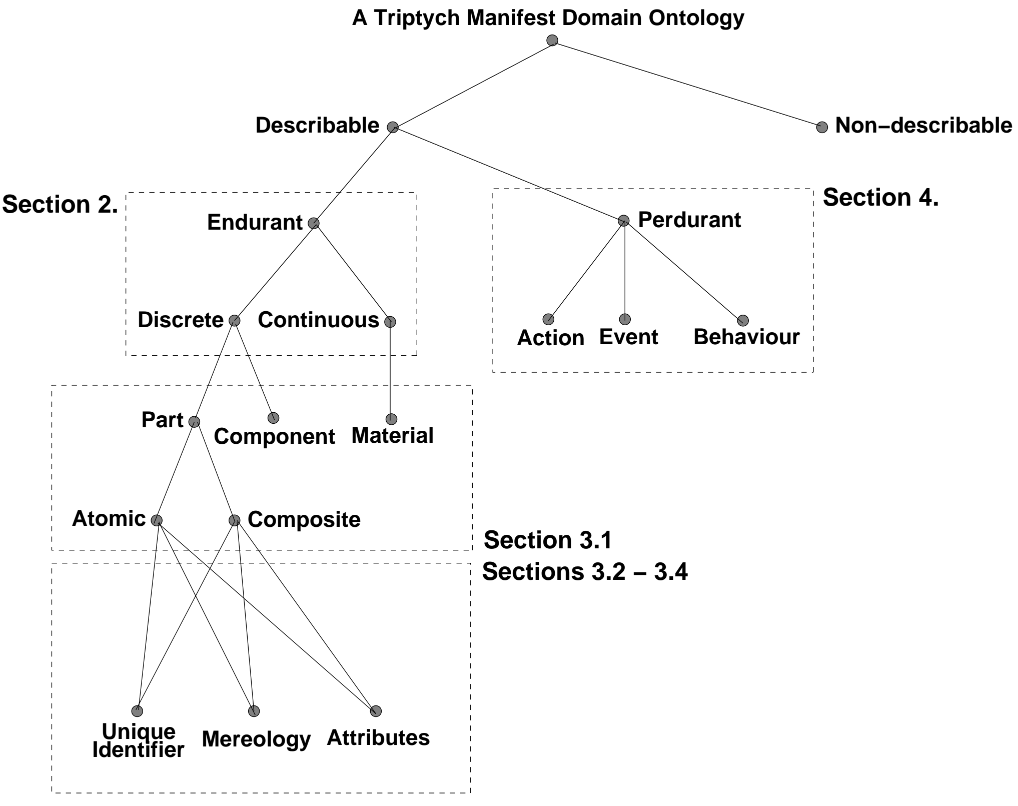


Figure 1: An Upper Ontology for Domains

3. Endurants

- This section brings a comprehensive treatment of the analysis and description of endurants.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

3.1. Parts, Components and Materials

3.1.1. General

Definition 6 Part:

- *By a* **part** *we shall understand*
  - *a discrete endurant*
  - *which the domain engineer chooses*
  - *to endow with* **internal qualities** *such as*
    - *unique identification,*
    - *mereology, and*
    - *one or more attributes* ■

We shall soon define the terms
‘unique identification’, ‘mereology’, and ‘attributes’.

Example 11 Parts: Example

- 7 on Slide 87 illustrated,

and examples

- 15 on Slide 116 and
- 16 on Slide 118

shall illustrate parts

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

Definition 7 Component:

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

Example 12 Components:

- Examples of components are:
  - chairs, tables, sofas and book cases in a living room,
  - letters, newspapers, and small packages in a mail box,
  - machine assembly units on a conveyor belt,
  - boxes in containers of a container vessel,
  - etcetera

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

"At the Discretion of the Domain Engineer":

- We emphasise the following analysis and description aspects:
  - (a) The domain is full of observable phenomena.
    - It is the decision of the domain analyser cum describer
    - whether to analyse and describe some such phenomena,
    - that is, whether to include them in a domain model.
  - (b) The borderline between an endurant
    - being (considered) discrete or
    - being (considered) continuous
    - is fuzzy.
    - It is the decision of the domain analyser cum describer
    - whether to model an endurant as discrete or continuous.

- (c) The borderline between a discrete endurant
  - being (considered) a part or
  - being (considered) a component
  - is fuzzy.
  - It is the decision of the domain analyser cum describer
  - whether to model a discrete endurant as a part or as a component.
- (d) We shall later show how to “compile” parts into processes.
  - A factor, therefore, in determining whether
  - to model a discrete endurant as a part or as a component
  - is whether we may consider a discrete endurant as also representing a process.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

Definition 8 Material:

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

Example 13 Materials: Examples of material endurants are:

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

Example 14 Parts Containing Materials:

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

3.1.2. Part, Component and Material Analysis Prompts

Analysis Prompt 6 *is_part:*

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

- We remind the reader that the outcome of `is_part`(*e*)
- is very much dependent on the domain engineer's intention
- with the domain description, cf. Slide 106.

Analysis Prompt 7 *is_component:*

- *The domain analyser analyse endurants e into component entities as prompted by the* **domain analysis prompt***:*
  - ***is_component*** *— e is a component if* ***is_component(e)*** *holds*

- We remind the reader that the outcome of `is_component`(e)
- is very much dependent on the domain engineer's intention
- with the domain description, cf. Slide 106.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

Analysis Prompt 8 *is_material:*

- *The domain analyser analyse endurants e into material entities as prompted by the* **domain analysis prompt***:*
  - **is_material** — *e is a material if* **is_material**(*e*) *holds*

- We remind the reader that the outcome of `is_material`(e)
- is very much dependent on the domain engineer's intention
- with the domain description, cf. Slide 106.

3.1.3. Atomic and Composite Parts

- A distinguishing quality
  - of 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.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

- That is,
  - parts become the domain endurants of main interest,
  - whereas components 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.
  - If the domain engineer wishes to investigate
    - the details of a discrete endurant
    - then the domain engineer choose to model
    - the discrete endurant as a part
    - otherwise as a component.

Definition 9 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 ■

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

Example 15 Atomic Parts: Examples of atomic parts of the above mentioned domains are:

- aircraft[11] (of air traffic),
- demand/deposit accounts (of banks),
- containers (of container lines),
- documents (of document systems),
- hubs, links and vehicles (of road traffic),
- patients, medical staff and beds (of hospitals),
- pipes, valves and pumps (of pipeline systems), and
- rail units and locomotives (of railway systems) ■

[11] An aircraft from the point of view of airport management are atomic. From the point of view of aircraft manufacturers they are composite.

Definition 10 Composite Part:

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

Example 16 Composite Parts: Examples of composite parts of the above mentioned domains are:

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

Analysis Prompt 9 *is_atomic:*

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

Analysis Prompt 10 *is_composite:*

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

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

Whither Atomic or Composite:

- If we are analysing & describing vehicles in the context of a road net, cf. the Traffic System Example Slide 87,
  - then we have chosen to abstract vehicles
  - as atomic;
- if, on the other hand, we are analysing & describing vehicles in the context of an automobile maintenance garage
  - then we might very well choose to abstract vehicles
  - as composite —
  - the sub-parts being the object of diagnosis
  - by the auto mechanics.

3.1.4. On Observing Part Sorts and Types

- We use the term 'sort'
  - when we wish to speak of an abstract type,
  - that is, a type for which we do not wish to express a model[12].
  - We shall use the term 'type' to cover both
    - abstract types and
    - concrete types.

[12]

- for example, in terms of the concrete types:
  - sets,
  - Cartesians,
  - lists,
  - maps,

  or other.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

3.1.5. On Discovering Part Sorts

- Recall from the section on *Types Are Formal Concepts* (Slide 76) that we "equate" a formal concept with a type (i.e., a sort).
  - Thus, to us, a part sort is a set of all those entities
  - which all have exactly the same qualities.
- Our aim now
  - is to present the basic principles that let
  - the domain analyser decide on part sorts.

- 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 11 *observe_parts:*

- *The* **domain analysis prompt***:*
  - *observe_parts*(p)
- *directs the domain analyser to observe the sub-parts of* p ■

Let us say the sub-parts of p are: $\{p_1,p_2,\ldots,p_m\}$.

- *(β) The analyser analyses, for each of these parts,* p_{i_k}*,*
  - *which formal concept, i.e., sort, it belongs to;*
  - *let us say that it is of sort* P_k*;*
  - *thus the sub-parts of* p *are of sorts* $\{P_1,P_2,\ldots,P_m\}$.
- *Some* P_k *may be atomic sorts, some may be composite sorts.*

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

    of the same, respective, part sorts.

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

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

3.1.6. Part Sort Observer Functions

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

  we must prove it.

Domain Description Prompt 1 *observe_part_sorts*:

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

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

1. `observe_part_sorts` schema

Narration:

[s] … narrative text on sorts …

[o] … narrative text on sort observers …

[i] … narrative text on sort recognisers …

[p] … narrative text on proof obligations …

Formalisation:

type

[s] P,

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

value

[o] **obs_part**_P_i: P → P_i $[1 \leq i \leq m]$

[i] **is**_P_i: $(P_1|P_2|...|P_m) \rightarrow \mathbf{Bool}$ $[1 \leq i \leq m]$

proof obligation [Disjointness of part sorts]

[p] $\forall\ p{:}(P_1|P_2|...|P_m) \cdot$

[p] $\bigwedge$ {**is**_P_i(p) ≡ $\bigwedge$ {∼**is**_P_j(p) | j ∈ {1..m} \ {i}} | i ∈ {1..m}}

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

Example 17 Composite and Atomic Part Sorts of Transportation:

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

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

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

3 A fleet is a collection of vehicles.

- The monitor is considered an atomic part.

type
1. Δ, N, F, M

value
1. **obs_part**_N: $\Delta \to$ N,
1. **obs_part**_F: $\Delta \to$ F,
1. **obs_part**_M: $\Delta \to$ M

type
2. HS, LS

value
2. **obs_part**_HS: N $\to$ HS,
2. **obs_part**_LS: N $\to$ LS

type
3. VS

value
3. **obs_part**_VS: F $\to$ VS

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

3.1.7. On Discovering Concrete Part Types

Analysis Prompt 12 *has_concrete_type:*

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

- 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.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

Domain Description Prompt 2 *observe_part_type*:

- *Then the domain analyser applies the* **domain description prompt***:*
  - ⟐ ***observe_part_type***(p)[13]
- *to parts p:P which then yield the* **part type and part type observers** *domain description text according to the following schema:*

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

2. observe_part_type schema

Narration:

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

Formalisation:

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

■

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

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

Example 18 Concrete Part Types of Transportation:
We continue Example 17 on Slide 130:

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

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

6 A collection of vehicles is a set of vehicles.

7 Vehicles are, until further analysis, part sorts.

type
4. Hs = H-**set**, Ls = L-**set**
5. H, L
6. Vs = V-**set**
7. V
value
4. **obs_part**_Hs: HS → Hs, **obs_part**_Ls: LS → Ls
6. **obs_part**_Vs: VS → Vs ■

3.1.8. Forms of Part Types

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

  where

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

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

3.1.9. Part Sort and Type Derivation Chains

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

3.1.9.1 No Recursive Derivations

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

3.1.10. Names of Part Sorts and Types

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

— as well as the

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

prompts introduced below — "yield" type names.

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

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

Example 19 Container Line Sorts:

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

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

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

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

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

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

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

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

15 Containers are further undefined.

- After a some slight editing we get:

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

```
type
  CL
  VS, VI, V, Vs = VI →m V,
  PS, PI, P, Ps = PI →m P
value
  obs_part_VS: CL → VS
  obs_part_Vs: VS → Vs
  obs_part_PS: CL → PS
  obs_part_Ps: CTPS → CTPs
type
  CSA
value
  obs_part_CSA: V → CSA
  obs_part_CSA: P → CSA
```

```
type
  BAYS, BI, BAY, Bays=BI →m BAY
  ROWS, RI, ROW, Rows=RI →m ROW
  STKS, SI, STK, Stks=SI →m STK
  C
value
  obs_part_BAYS: CSA → BAYS,
  obs_part_Bays: BAYS → Bays
  obs_part_ROWS: BAY → ROWS,
  obs_part_Rows: ROWS → Rows
  obs_part_STKS: ROW → STKS,
  obs_part_Stks: STKS → Stks
  obs_part_Stk: STK → C*
```

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

3.1.11. More On Part Sorts and Types

- The above "experimental example" motivates the below.
  - We can always assume that composite parts p:P abstractly consists of a definite number of sub-parts.
    - **Example 20**. We comment on Example 17, Page 130: Parts of type Δ and N are composed from three, respectively two abstract sub-parts of distinct types ■
  - Some of the parts, say p_{i_z} of $\{p_{i_1}, p_{i_2}, \ldots, p_{i_m}\}$, of p:P, may themselves be composite.
    - **Example 21**. We comment on Example 17: Parts of type N, F, HS, LS and VS are all composite ■

- There are, pragmatically speaking, two cases for such compositionality.
  - Either the part, p_{i_z}, of type t_{i_z}, is is composed from a definite number of abstract or concrete sub-parts of distinct types.
    - **Example 22.** We comment on Example 17: Parts of type N are composed from three sub-parts ■
  - Or it is composed from an indefinite number of sub-parts of the same sort.
    - **Example 23.** We comment on Example 17: Parts of type HS, LS and VS are composed from an indefinite numbers of hubs, links and vehicles, respectively ■

Example 24 Pipeline Parts:

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

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

18 All these unit sorts are atomic and disjoint.

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

3.1.12. External and Internal Qualities of Parts

- By an **external part quality** we shall understand the
  - is_atomic,
  - is_composite,
  - is_discrete and
  - is_continuous

  qualities ■

- By an **internal part quality** we shall understand the part qualities to be outlined in the next sections:
  - unique ids,
  - mereology and
  - attributes

- By **part qualities** we mean the sum total of
  - external endurant and
  - internal endurant

  qualities ■

3.1.13. Three Categories of Internal Qualities

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

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

3.2. Unique Part Identifiers

- We introduce a notion of unique identification of parts.
- We assume
  - (i) that all parts, p, of any domain P, have **unique identifiers**,
  - (ii) that **unique identifiers** (of parts p:P) are **abstract values** (of the **unique identifier** sort PI of parts p:P),
  - (iii) such that distinct part 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.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

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 3 *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:*

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

3. `observe_unique_identifier` schema

Narration:

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

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

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

Formalisation:

type

[s] PI

value

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

axiom

[a] $\mathcal{U}$

Example 25 Unique Transportation Net Part Identifiers:
We continue Example 17 on Slide 130.

19 Links and hubs have unique identifiers

20 and unique identifier observers.

type
19. LI, HI
value
20. **uid**_LI: L → LI
20. **uid**_HI: H → HI
axiom [Well−formedness of Links, L, and Hubs, H]
19. $\forall$ l,l′:L · l≠l′⇒**uid**_LI(l)≠**uid**_LI(l′),
19. $\forall$ h,h′:H · h≠h′⇒**uid**_HI(h)≠**uid**_HI(h′) ■

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

3.3. 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, Bjø14a].

3.3.1. Part Relations

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

Example 26 Shared Timetable Mereology (I):

- Two or more distinct public transport busses
  - may "run" according to the (identically) same,
  - thus "shared", bus time table
  - (cf. Example 37 on Slide 201) ■

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

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

Example 27 Topological Connectedness Mereology:

- (i) two rail units may be connected (i.e., adjacent);
- (ii) a road link may be connected to two road hubs;
- (iii) a road hub may be connected to zero or more road links;
- (iv) distinct vehicles of a road net may be monitored by one and the same road pricing sub-system ■

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

3.3.2. Part Mereology: Types and Functions

Analysis Prompt 13 *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).

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

- We can define a **mereology type** as a type $\mathcal{E}$xpression over unique [part] identifier types.
  - We generalise to unique [part] identifiers over a definite collection of part sorts, P1, P2, ..., Pn,
  - where the parts p1:P1, p2:P2, ..., pn:Pn are not necessarily (immediate) sub-parts of some part p:P.

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

Domain Description Prompt 4 *observe_mereology*:

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

4. `observe_mereology` schema

Narration:

[t] ... narrative text on mereology type ...

[m] ... narrative text on mereology observer ...

[a] ... narrative text on mereology type constraints ...

Formalisation:

type

[t] MT[14]= $\mathcal{E}$(PI1,PI2,...,PIm)

value

[m] **obs_mereo**_P: P → MT

axiom [Well−formedness of Domain Mereologies]

[a] $\mathcal{A}$(MT)

[14] MT will be used several times in Sect. .

- *Here* $\mathcal{E}$(PI1,PI2,...,PIm) *is a type expression over possibly all unique identifier types of the domain description,*
- *and* $\mathcal{A}$(MT) *is a predicate over possibly all unique identifier types of the domain description.*
- *To write down the concrete type definition for* MT *requires a bit of analysis and thinking.*
- `has_mereology` *is a* **prerequisite prompt** *for* `observe_mereology` ■

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

Example 28 Road Net Part Mereologies:

We continue Example 17 on Slide 130 and Example 25 on Slide 155.

21 Links are connected to exactly two distinct hubs.

22 Hubs are connected to zero or more links.

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

```
type
21.  LM′ = HI-set, LM = {|his:HI-set · card(his)=2|}
22.  HM = LI-set
value
21.  obs_mereo_L: L → LM
22.  obs_mereo_H: H → HM
axiom [Well−formedness of Road Nets, N]
23.  ∀ n:N,l:L,h:H·
23.      l ∈ obs_part_Ls(obs_part_LS(n))
23.    ∧ h ∈ obs_part_Hs(obs_part_HS(n))
23.      let his=mereology_H(l), lis=mereology_H(h) in
23.      his⊆∪{uid_H(h) | h ∈ obs_part_Hs(obs_part_HS(n))}
23.    ∧ lis⊆∪{uid_H(l) | l ∈ obs_part_Ls(obs_part_LS(n))} end ■
```

Example 29 Pipeline Parts Mereology:

- We continue Example 24 on Slide 147.
- Pipeline units serve to conduct fluid or gaseous material.
- The flow of these occur in only one direction: from so-called input to so-called output.

24 Wells have exactly one connection to an output unit.

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

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

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

28 Sinks have exactly one connection from an input unit.

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

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

3.3.3. Formulation of Mereologies

- The `observe_mereology` domain descriptor, Slide 162,
  - 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.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

3.4. Part 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.

3.4.1. Inseparability of Attributes from Parts

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

3.4.2. Attribute Quality and Attribute Value

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

Example 30 Attribute Propositions and Other Values:

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

3.4.3. Endurant Attributes: Types and Functions

- Let us recall that attributes cover qualities other than unique identifiers and mereology.
- Let us then consider that parts have one or more attributes.
  - These attributes are qualities
  - which help characterise "what it means" to be a part.
- Note that we expect every part to have at least one attribute.

Example 31 Atomic Part Attributes:

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

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

  etc.

Example 32 Composite Part Attributes:

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

  etc.

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

  etc.

- We now assume that all parts have attributes.
- The question is now, in general, how many and, particularly, which.

Analysis Prompt 14 *attribute_names:*

- *The* **domain analysis prompt** **attribute_names**
  - *when applied to a part p*
  - *yields the set of names of its attribute types:*
  - **attribute_names**(p)*:* $\{\eta A_1, \eta A_2, ..., \eta A_n\}$.
- η *is a type operator. Applied to a type A it yields is name*[15] ■

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

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

  Therefore we must prove it.

3.4.3.1 The Attribute Value Observer

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

Domain Description Prompt 5 *observe_attributes*:

- *The domain analyser experiments, thinks and reflects about part attributes.*
- *That process is initated by the* **domain description prompt**:
  - *observe_attributes*.
- *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:*

5. observe_attributes schema

Narration:

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

Formalisation:

type
[t] A_i $[1 \leq i \leq n]$
value
[o] **attr**_A_i:P→A_i $[1 \leq i \leq n]$
[i] **is**_A_i:(A_1|A_2|...|A_n)→**Bool** $[1 \leq i \leq n]$
proof obligation [Disjointness of Attribute Types]
[p] $\forall$ δ:Δ
[p] **let** P be any part sort **in** [the Δ domain description]
[p] **let** a:(A_1|A_2|...|A_n) **in** **is**_A_i(a) $\neq$ **is**_A_j(a) **end end** $[i \neq j, 1 \leq i,j \leq n]$

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

- *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:$P \rightarrow A_1$,
  - **attr**_A_2:$P \rightarrow A_2$,
  - ...,
  - **attr**_A_n:$P \rightarrow A_n$

  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:$P \rightarrow A_i$ *etcetera* ■

- The fact that, for example, A_1, A_2, ..., A_n, are attributes of p:P, means that the propositions
  - has_attribute_$A_1(p)$,
    has_attribute_$A_2(p)$,
    ..., and
    has_attribute_$A_n(p)$

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

Example 33 **Road Hub Attributes**: After some analysis a domain analyser may arrive at some interesting hub attributes:

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

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

32 such that

a. the links referred to in the state are links of the hub mereology

b. and the state is in the state space.

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

type
30 $\mathsf{H\Sigma} = (\mathsf{LI}\times\mathsf{LI})$**-set**
31 $\mathsf{H\Omega} = \mathsf{H\Sigma}$**-set**
value
30 **attr**_$\mathsf{H\Sigma}$:H$\to$HΣ
31 **attr**_$\mathsf{H\Omega}$:H$\to$HΩ
axiom [Well−formedness of Hub States, HΣ]
32 $\forall$ h:H $\cdot$ **let** lis = **obs_mereo**_H(h) **in**
32 **let** hσ = **attr**_HΣ(h) **in**
32a. {li,li′|li,li′:LI·(li,li′)$\in$ hσ}$\subseteq$lis
32b. $\wedge$ hσ $\in$ **attr**_HΩ(h)
32 **end end** ■

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

3.4.4. Attribute Categories

- One can suggest a hierarchy of part 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 [Jac95b, M.A. Jackson].

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

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

Dynamic attributes are either inert, reactive or active attributes.

- 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 properties of these new values.
- By a **reactive attribute**, a:A, `is_reactive_attribute`(a), we shall understand a dynamic attribute whose values,
  - if they vary, change value in response to
  - the change of other attribute values.
- By an **active attribute**, 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.

- 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".[16]
- By a **biddable attribute**, a:A, `is_biddable_attribute`(a), (of a part) we shall understand a dynamic active attribute whose values
  - are prescribed
  - but may fail to be observed as such.
- By a **programmable attribute**, a:A, `is_programmable_attribute`(a), we shall understand a dynamic active attribute whose values
  - can be prescribed.

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

Example 34 Static and Dynamic Attributes:

- Link lengths can be considered **static.**
- Buses (i.e., vehicles) have a *timetable* attribute which is **inert**, i.e., can change, only when the bus company decides so.
- The weather can be considered **autonomous.**
- Pipeline valve units include the two attributes of *valve opening* (`open`, `close`) and *internal flow* (measured, say `gallons` per `second`).
  - The valve opening attribute is of the **biddable** attribute category.
  - The flow attribute is **reactive** (flow changes with valve opening/closing).
- Hub states (red, yellow, green) can be considered **biddable:** one can "try" set the signals but the electro-mechanics may fail.
- Bus companies **program** their own timetables, i.e., bus company timetables are **programmable** — are computers ■

- **External Attributes:** By an **external attribute** we shall understand
  - a dynamic attributes
  - which is not a programmable attribute ■
- Thus we can define the domain analysis prompt:
  - `is_external_attribute`,
  - as:

value
  is_external_attribute: P → **Bool**
  is_external_attribute(p) ≡
    is_dynamic_attribute(p) ∧ ∼is_programmable_attribute(p)
    pre: is_endurant(p) ∧ is_discrete(p)

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

- The idea of external attributes is this:
  - They are the attributes whose (deterministic or non-deterministic) values are determined by factors
    "outside" the part of which they are an attribute.
  - In contrast, the programmable attributes have their values determined
    by the part of which they are an attribute.

- Figure 2 captures an attribute value ontology.

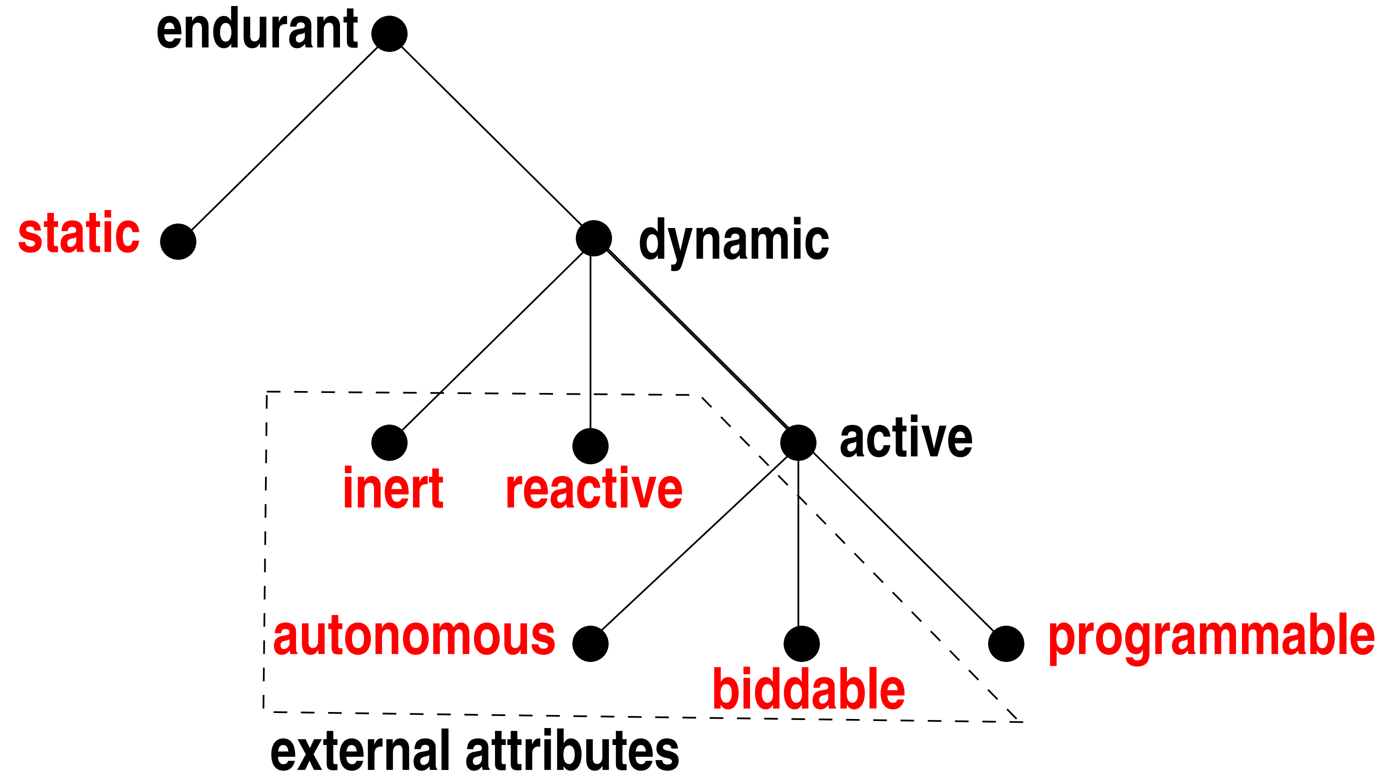


Figure 2: Attribute Value Ontology

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

3.4.5. Access to Attribute Values

- In an action, event or a behaviour description
  - static values of parts, p,
  - (say of type A)
  - can be "copied", **attr**_A(p),
  - and still retain their (static) value.
- But, for action, event or behaviour descriptions,
  - external dynamic values of parts, p,
  - cannot be "copied",
  - but **attr**_A(p) must be "performed"
  - every time they are needed.

- That is:
  - **static values** require at most one domain access,
  - whereas **external attribute values** require repeated domain accesses.
- We shall return to the issue of **attribute value access** in Sect. 1.3.8.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

3.4.6. Event Values

- Among the external attribute values we observe a new kind of value: the **event values.**
  - We may optionally ascribe ordinarily typed, say A, values, a:A, with event attributes.
  - By an **event attribute** we shall understand
    - an attribute whose values are
      - either "nil" ([f]or "absent"),
      - or are some more definite value (a:A) ■
  - Event values *occur* instantaneously.
    - They can be thought of as the raising of a signal
    - followed immediately by the lowering of that signal.

Example 35 Event Attributes:

- (i) The passing of a vehicle past a tollgate is an event.
  - It occurs at a usually unpredictable time.
  - It otherwise “carries” no specific value.
- (ii) The identification of a vehicle by a tollgate sensor is an event.
  - It occurs at a usually unpredictable time.
  - It specifically “carries” a vehicle identifier value ■

- Event attributes are not to be confused with event perdurants.
- External attributes are either event attributes or are not.
- More on access to event attribute values in Sect. 4.7.4 [as from Slide 269].

3.4.7. Shared Attributes

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

Attribute Naming:

- Thus the domain describer has to exert great care when naming attribute types.
  - If P_i and P_j are two distinct types of a domain
  - then if and only if an attribute of P_i is to be shared with an attribute of P_j
  - must that attribute be identically named in the description of P_i and P_j and
  - otherwise the attribute names of P_i and P_j must be distinct.

Example 36. Shared Attributes. Examples of shared attributes:

- Bus **timetable attributes** have the same value as the fleet **timetable attribute** – cf. Example 37 below.
- A link incident upon or emanating from a hub shares the **connection** between that link and the hub as an attribute.
- Two pipeline units[17], p_i with unique identifier π_i, and p_j with unique identifier π_j, that are **connected**, such that an outlet marked π_j of p_i "feeds into" inlet marked π_i of p_j, are said to share the connection (modeled by, e.g., $\{(\pi_i, \pi_j)\}$) ■

[17] See Example 29 on Slide 166

Example 37 Shared Timetables:

- The fleet and vehicles of Example 17 on Slide 130 and Example 18 on Slide 137 is that of a bus company.

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

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

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

37 A bus timetable is a shared fleet and bus attribute.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

```
type
34.  FI, VI, BT
value
34.  uid_F: F → FI
34.  uid_V: V → VI
35.  obs_mereo_F: F → VI-set
36.  obs_mereo_V: V → FI
37.  attr_BT: (F|V) → BT
axiom
   □ ∀ f:F ⇒
     ∀ v:V • v ∈ obs_part_Vs(obs_part_VC(f)) • attr_BT(f) = attr_BT(v)
```

3.4.8. Master/Slave Shared Attribute Relations

- We can assume, without loss of generality, that there exists a uniquely decidable master/slave relationship between shared attributes.
  - Let A be a shared attribute of part sorts P, Q, . . . , R (i.e., for at least two sorts).
  - We can now assume that the A attributes of sorts Q, . . . , R are all *inert*, and that that of sort P is *programmable.*
  - We shall therefore refer to attribute A of sort P as the **master attribute** and to A of sorts Q, . . . , R as **inert slave attribute**s.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

- We therefore introduce the
  - ⟐ is_shared_attribute_A,
  - ⟐ is_master_attribute_A and
  - ⟐ is_slave_attribute_A

  domain analysis prompts.
- is_shared_attribute_A(p) holds if (p) has an attribute named A and this attribute of p is shared with parts (q:Q).
- is_master_attribute_A(p) holds if (p) has an attribute named A and this attribute of p is the master (with respect to attribute A).
- is_slave_attribute_A(q) holds if (q) has an attribute named A and this attribute of q is a slave (with respect to attribute A).

Example 38 Shared Timetable Mereology (II):

- Example 26 on Slide 157 illustrated this point:
  - The fleet, f:F, of our "running" road transport example
  - is now assumed to model the management of its bus vehicles.
  - The fleet has unique identifier, fi:FI,
  - and a timetable attribute
  - which is shared with all vehicles (i.e., buses) of the fleet.
  - The fleet timetable attribute is considered programmable
  - whereas the bus timetable attributes are all considered inert.
  - The bus timetables are, so-to-speak, "inherited" from the fleet.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

- This "inheritance" is reflected in a combination of facets:
  - in the vehicle mereology – which contains **fi:FI**,
  - in the fleet mereology – which contains the set of all vehicle (cum bus) unique identifiers **vis:VI-set**, and
  - in the bus accesses to "their" timetables –
    - which, conceptually is modeled, as we shall see, in Sect. ,
    - as occurring over channels
    - between the fleet administrator and each individual bus,
  - with the fleet administrator occasionally "updating" the global timetable,
  - "broadcasting" the "new" bus timetable,
  - and the buses "reading" the updated timetable

  all over the "sharing" channels ■

3.5. Components

- Components are discrete endurants
- which the domain analyser & describer has chosen to **not** endow
- with **internal qualities**.

Example 39 Parts and Components:

- We observe components as associated with atomic parts:
  - The contents, that is, the collection of zero, one or more boxes, of a container are the components of the container part.
  - Conveyor belts transport machine assembly units and these are thus considered the components of the conveyor belt ■

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

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

Analysis Prompt 15 *has_components:*

- *The* **domain analysis prompt***:*
  - ⟐ ***has_components***(p)
- *yields* **true** *if atomic part p may contain zero, one or more components otherwise false* ■

- Let us assume that parts $p{:}P$ embody components of sorts $\{K_1,K_2,\ldots,K_n\}$.
- Since we cannot automatically guarantee that our domain descriptions secure that
  - each K_i ($[1{\leq}i{\leq}n]$)
  - denotes disjoint sets of entities

  we must prove it.

Domain Description Prompt 6 *observe_component_sorts*:

- *The* **domain description prompt***:*
  - *observe_component_sorts_P(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:*

6. `observe_component_sorts_P` schema

Narration:

[s] ... narrative text on component sorts ...
[o] ... narrative text on component observers ...
[i] ... narrative text on component sort recognisers ...
[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] **components**: P → KS
[i] **is**_K_i: $(K_1|K_2|...|K_n) \to$ **Bool** $[1 \le i \le n]$

Proof Obligation: [Disjointness of Component Sorts]

[p] $\forall\ k_i{:}(K_1|K_2|...|K_n) \cdot$
[p] $\bigwedge \{\textbf{is_}K_i(k_i) \equiv \bigwedge\{\sim\textbf{is_}K_j(k_j)|j \in \{1..m\}\setminus\{i\}\}|i \in \{1..m\}\}$

Example 40 **Container Components**: We continue Example 19 on Slide 142.

38 When we apply `obs_component_sorts_C` to any container c:C we obtain

a. a type clause stating the sorts of the various components, ck:CK, of a container,

b. a union type clause over these component sorts, and

c. the component observer function signature.

type
38a. CK1, CK2, ..., CKn
38b. CKS = (CK1|CK2|...|CKn)-**set**
value
38c. **obs_comp**_CKS: C → CKS

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

- 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.

3.6. Materials

- Continuous endurants (i.e., **materials**) are entities, m, which satisfy:
  - is_material$(m) \equiv$ is_endurant$(m) \wedge$ is_continuous(m)

Example 41 Parts and Materials:

- We observe materials as associated with atomic parts:
  - Thus liquid or gaseous materials are observed in pipeline units

- We shall in this seminar not cover
  the case of parts being immersed in materials.

- We assume, without loss of generality, that only atomic parts may contain materials.
- Let $p{:}P$ be some atomic part.

Analysis Prompt 16 *has_materials:*

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

- 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.

Domain Description Prompt 7 *observe_material_sorts_P*:

- *The* **domain description prompt***:*
  - *observe_material_sorts_P(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:*

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

7. `observe_material_sorts_P` 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**
value
[o] **obs_mat**_M_i: P → M [$1 \leq i \leq n$]
[o] **materials**: P → MS
[i] **is**_M_i: M → **Bool** [$1 \leq i \leq n$]
proof obligation [Disjointness of Material Sorts]
[p] $\forall\, m_i$:M · $\bigwedge$ {**is**_$M_i(m_i) \equiv \bigwedge\{\sim$**is**_$M_j(m_j)$|j ∈ {1..m}\{i}}|i ∈ {1..m}}

Example 42 Pipeline Material: We continue Example 24 on Slide 147 and Example 29 on Slide 166.

39 When we apply obs_material_sorts_U to any unit u:U we obtain

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

b. a material observer function signature.

```
    type
39a.     LoG
    value
39b.     obs_mat_LoG: U → LoG
```

has_materials(u) is a prerequisite for **obs_mat**_LoG(u)

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

3.6.1. Materials-related Part Attributes

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

Example 43 **Pipeline Material Flow**: We continue Examples 24, 29 and 42.

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

40 . **type** F, L.

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

41 current flow of material into a unit input connector,

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

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

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

45 current leak of material at a unit input connector,

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

47 current leak of material at a unit input connector,

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

49 current leak of material from “within” a unit, and

50 maximum guaranteed leak of material from “within” a unit.

type
40. F, L
value
41. **attr**_cur_iF: U → UI → F
42. **attr**_max_iF: U → UI → F
43. **attr**_cur_oF: U → UI → F
44. **attr**_max_oF: U → UI → F
45. **attr**_cur_iL: U → UI → L
46. **attr**_max_iL: U → UI → L
47. **attr**_cur_oL: U → UI → L
48. **attr**_max_oL: U → UI → L
49. **attr**_cur_L: U → L
50. **attr**_max_L: U → L

- The maximum flow attributes are static attributes and are typically provided by the manufacturer as indicators of flows below which laminar flow can be expected.
- The current flow attributes may be considered either reactive or biddable attributes ∎

3.6.2. Laws of Material Flows and Leaks

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

Example 44 Pipelines: Intra Unit Flow and Leak Law:

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

52 The flows into a unit equal

a. the leak at the inputs

b. plus the leak within the unit

c. plus the flows out of the unit

d. plus the leaks at the outputs.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

```
axiom [Well−formedness of Pipeline Systems, PLS (1)]
51.  ∀ pls:PLS,b:B\We\Si,u:U ·
51.       b ∈ obs_part_Bs(pls)∧u=obs_part_U(b)⇒
51.       let (iuis,ouis) = obs_mereo_U(u) in
52.       sum_cur_iF(u)(iuis) =
52a..         sum_cur_iL(u)(iuis)
52b..       ⊕ attr_cur_L(u)
52c..       ⊕ sum_cur_oF(u)(ouis)
52d..       ⊕ sum_cur_oL(u)(ouis)
51.       end
```

53 The **sum_cur_iF** (cf. Item 52) sums current input flows over all input connectors.

54 The **sum_cur_iL** (cf. Item 52a.) sums current input leaks over all input connectors.

55 The **sum_cur_oF** (cf. Item 52c.) sums current output flows over all output connectors.

56 The **sum_cur_oL** (cf. Item 52d.) sums current output leaks over all output connectors.

```
53.  sum_cur_iF:  U → UI-set →F
53.  sum_cur_iF(u)(iuis) ≡ ⊕ {attr_cur_iF(u)(ui)|ui:UI•ui ∈ iuis}
54.  sum_cur_iL: U → UI-set → L
54.  sum_cur_iL(u)(iuis) ≡ ⊕ {attr_cur_iL(u)(ui)|ui:UI•ui ∈ iuis}
55.  sum_cur_oF: U → UI-set → F
55.  sum_cur_oF(u)(ouis) ≡ ⊕ {attr_cur_iF(u)(ui)|ui:UI•ui ∈ ouis}
56.  sum_cur_oL: U → UI-set → L
56.  sum_cur_oL(u)(ouis) ≡ ⊕ {attr_cur_iL(u)(ui)|ui:UI•ui ∈ ouis}
     ⊕: (F|L) × (F|L) → F
```


© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

Example 45 Pipelines: Inter Unit Flow and Leak Law:

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

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

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

```
axiom [ Well−formedness of Pipeline Systems, PLS (2) ]
57.     ∀ pls:PLS,b,b′:B,u,u′:U•
57.        {b,b′}⊆obs_part_Bs(pls)∧b≠b′∧u′=obs_part_U(b′)
57.         ∧ let (iuis,ouis)=obs_mereo_U(u),(iuis′,ouis′)=obs_mereo_U(u′),
57.                ui=uid_U(u),ui′=uid_U(u′) in
57.             ui ∈ iuis ∧ ui′ ∈ ouis′ ⇒
57a..              attr_cur_oF(u′)(ui′) − attr_leak_oF(u′)(ui′)
57b..             = attr_cur_iF(u)(ui) + attr_leak_iF(u)(ui)
57.             end
57.     comment: b′ precedes b ■
```

- From the above two laws one can prove the **theorem:**
  - what is pumped from the wells equals
  - what is leaked from the systems plus what is output to the sinks.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

3.7. “No Junk, No Confusion”

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

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

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

- So, since one naturally wishes "no junk, no confusion" what does one do ?
- The answer to that is
  - *one proceeds with great care !*
- To avoid **junk** we have stated a number of **sort well-formedness axioms**, for example:[18]
  - Slide 155 for *wf* links and hubs,
  - Slide 162 for *wf* road net mereologies,
  - Slide 165 for *wf* pipeline mereologies,
  - Slide 185 for *wf* hub states,
  - Slide 224 for *wf* pipeline systems,
  - Slide 226 for *wf* pipeline systems,

[18] Let *wf* abbreviate *well-formed.*

- To avoid **confusion** we have stated a number of **proof obligations**:
  - Slide 129 for *Disjointness of Part Sorts*,
  - Slide 181 for *Disjointness of Attribute Types* and
  - Slide 216 for *Disjointness of Material Sorts*.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

3.8. Discussion of Endurants

- In Sect. 4.2.2 a "depth-first" search for part sorts was hinted at.
- It essentially expressed
  - that we discover domains epistemologically[19]
  - but understand them ontologically.[20]
- The Danish philosopher Søren Kirkegaard (1813–1855) expressed it this way:
  - *Life is lived forwards,*
  - *but is understood backwards.*
- The presentation of the of the **domain analysis prompt**s and the **domain description prompt**s results in domain descriptions which are ontological.
- The "depth-first" search recognizes the epistemological nature of bringing about understanding.

[19] **Epistemology**: the theory of knowledge, especially with regard to its methods, validity, and scope. Epistemology is the investigation of what distinguishes justified belief from opinion.

[20] **Ontology**: the branch of metaphysics dealing with the nature of being.

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

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

4. Perdurants

- We shall not present
  - a set of **domain analysis prompts** and
  - a set of **domain description prompts**

  leading to description language,
  i.e., RSL texts describing perdurant entities.
- The reason for giving this albeit cursory overview of perdurants
  - is that we can justify our detailed study of endurants,
    - their part and subparts,
    - their unique identifiers, mereology and attributes.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

- This justification is manifested
  - (i) in expressing the types of **signatures**,
  - (ii) in basing behaviours on parts,
  - (iii) in basing the for need for
    CSP-oriented inter-behaviour communications
    on shared part attributes,
  - (iv) in indexing behaviours as are parts, i.e., on unique identifiers,

  and

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

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

4.1. States

Definition 11 State: *By a* **state** *we shall understand*

- *any collection of* **parts**
- *each of which has*
- *at least one* **dynamic attribute**
- *or* `has_components` *or* `has_material` *s* ■

Example 46 States:

- A road hub can be a state,
  cf. Hub State, $\mathsf{H}\Sigma$, Example 33 on Slide 184.
- A road net can be a state – since its hubs can be.
- Container stowage areas, CSA, Example 19 on Slide 142,
  of container vessels and container terminal ports
  can be states as containers can be removed from
  and put on top of container stacks.
- Pipeline pipes can be states as they potentially carry material.
- Conveyor belts can be states as they may carry components

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

4.2. Actions, Events and Behaviours

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

  behaviours.

4.2.1. Time Considerations

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

- Instead we shall refer to two seminal monographs:
  - Specifying Systems [Leslie Lamport, 2002] and
  - Duration Calculus: A Formal Approach to Real-Time Systems [Zhou ChaoChen and Michael Reichhardt Hansen, 2004] (and [Bjø06, Chapter 15]).

- For a seminal book on "time in computing" we refer to the eclectic [FMMR12, Mandrioli et al., 2012].
- And for seminal book on time at the epistemology level we refer to [van91, J. van Benthem, 1991].

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

4.2.2. Actors

Definition 12 Actor: *By an* **actor** *we shall understand*

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

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

Example 47 Actors: We refer to the road transport and the pipeline systems examples of earlier.

- The fleet, each vehicle and the road management of the *Transportation System* of Examples 17 on Slide 130 and 37 on Slide 201 can be considered actors;
- so can the net and its links and hubs.
- The pipeline monitor and each pipeline unit of the *Pipeline System*, Example 24 on Slide 147 and Examples 24 on Slide 147 and 29 on Slide 166 will be considered actors ■

4.2.3. Parts, Attributes and Behaviours

- Example 47 on the preceding slide focused on what shall soon become a major relation within domains:
  - that of parts being also considered actors,
  - or more specifically, being also considered to be behaviours.

Example 48 Parts, Attributes and Behaviours:

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

4.3. Discrete Actions

Definition 13 Discrete Action: *By a* **discrete action** *[WS12, Wilson and Shpall] we shall understand*

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

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

Example 49 Road Net Actions:

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

 © Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

4.4. Discrete Events

Definition 14 Event: *By an* **event** *we shall understand*

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

- Events can be characterised by
  - a pair of (before and after) states,
  - a predicate over these
  - and, optionally, a **time** or **time interval**.
- The notion of event continues to puzzle philosophers [Dre67, Qui79, Mel80, Dav80, Hac82, Bad05, Kim93, CV96, Pi99, CV10].

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

Example 50 Road Net and Road Traffic Events:

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

4.5. Discrete Behaviours

Definition 15 **Discrete Behaviour:** *By a* **discrete behaviour** *we shall understand*

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

Example 51 Behaviours:

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

4.5.1. Channels and Communication

- Behaviours
  - 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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

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

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

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

4.5.2. Relations Between Attribute Sharing and Channels

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

  as we shall soon see

  - part attributes,
  - part components and
  - part materials

  with behaviour states.

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

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

        channel $\{\mathsf{ch}[\,\{\pi_i,\pi_j\}\,]|\pi_i{=}\mathbf{uid_}\mathsf{P}_i(\mathsf{p}_i)\wedge\pi_j \in \mathsf{uids}\}$:A.

Example 52 Bus System Channels:

- We extend Examples 17 on Slide 130 and 37 on Slide 201.
- We consider the **fleet** and the **vehicle**s to be behaviours.

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

59 the **fleet** and

60 the **vehicles**.

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

value

58. δ:Δ,
59. f:F = **obs_part**_F(δ),
60. vs:V-**set** = **obs_part**_Vs(**obs_part**_VC((**obs_part**_F(δ))))

channel

61. {fch[{**uid**_F(f),**uid**_V(v)}]|v:V•v ∈ vs}:BT ■

4.6. Continuous Behaviours

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

Example 53 Flow in Pipelines:

- We refer to Examples 29, 42, 43, 44 and 45.
- Let us assume that oil is the (only) material of the pipeline units.
- Let us assume that there is a sufficient volume of oil in the pipeline units leading up to a pump.
- Let us assume that the pipeline units leading from the pump (especially valves and pumps) are all open for oil flow.
- Whether or not that oil is flowing, if the pump is pumping (with a sufficient **head**) then there will be oil flowing from the pump outlet into adjacent pipeline units ■

- To describe the flow of material (say in pipelines) requires knowledge about a number of material attributes — not all of which have been covered in the above-mentioned examples.
- To express flows one resorts to the mathematics of fluid-dynamics using such second order differential equations as first derived by Bernoulli (1700–1782) and Navier–Stokes (1785–1836 and 1819–1903).
- There is, as yet, no notation that can serve to integrate formal descriptions (like those of `Alloy, B, The B Method,RSL,VDM` or `Z`) with first, let alone second order differential equations. But some progress has been made [LWZ13, ZWZ13] since [WYZ94].

4.7. Attribute Value Access

- We distinguish between four kinds of attributes:
  - the **static attribute**s which are those whose values are fixed, i.e., does not change,
  - the **programmable**s and **biddable attribute**s which are those dynamic values are exclusively set by part processes, and
  - the remaining **dynamic attribute**s
    which here, technically speaking, are seen as separate **external processes**s.
  - The **event attribute**s are those external attributes whose value occur for an instant of time.

4.7.1. Access to Static Attribute Values

- The **static attributes** can be "copied", **attr**_A(p), and retain their values.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

4.7.2. Access to External Attribute Values

- By the **external attributes**, to repeat,
  - we shall understand the
    - inert, the
    - reactive, the
    - autonomous and the
    - biddable,

    i.e., the attributes ■

62 Let ξA be the set of names, ηA, of all **external attributes**.

63 Each **external attribute**, A, is seen as an individual behaviour, each "accessible" by means of unique channel, attr_A_ch.

64 External attribute values are then accessed by the input, attr_A_ch ?.

65 The **type** of attr_A_ch is considered to be **Unit**$\tilde{\rightarrow}$A, abbreviated $\mathbb{U}$ A.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

62. **value**
62. ξA: $\{\eta$A$|$A is any external attribute name$\}$
63. **channel**
63. $\{$attr_A_ch$|\eta$A $\in \xi$A$\}$
64. **value**
64. attr_A_ch ?
64. **type**
64. attr_A_ch: **Unit** $\stackrel{\sim}{\rightarrow}$ A [abbrev.:$\mathbb{U}$ A]

- We shall omit the η prefix in actual descriptions.
- The choice of representing **external attribute values** as CSP processes[21] is a technical one.

[21]— not to be confused with domain behaviours

4.7.3. Access to Biddable and Programmable Attribute Values

- The **programmable attributes** are treated as function arguments.
- This is a technical choice. It is motivated as follows.
  - We find that programmable attribute values are set (i.e., updated) by part behaviours.
  - That is, to each part, whether atomic or composite, we associate a behaviour.
  - That behaviour is (to be) described as we describe functions.
  - These functions (normally) *"go on forever"*.
  - Therefore these functions are described basically by a "tail" recursive definition:

$$\textbf{value}\ \ \mathsf{f}\colon \mathsf{Arg} \to \mathsf{Arg};\ \ \mathsf{f(a)} \equiv (\ldots\ \textbf{let}\ \mathsf{a}' = \mathcal{F}(\ldots)(\mathsf{a})\ \textbf{in}\ \mathsf{f(a')}\ \textbf{end})$$

  - where $\mathcal{F}$ is some expression based on values defined within the function definition body of f and on f's "input" argument a, and
  - where a can be seen as a programmable attribute.

4.7.4. Access to Event Values

- Event values reflect a stage change in a part behaviour.
  - We therefore model events as messages
  - communicated over a channel, attr_A_ch,
  - that is, attr_A_ch ! a,
  - where A is the event attribute, i.e., message type.
  - Thus fulfillment of attr_A_ch ? expresses
    - both that the event has taken place
    - and its value, if relevant.
  - Example 58 on Slide 300 illustrates the concept of event attributes and event values.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

4.8. Perdurant Signatures and Definitions

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

Definition 16 **Function Signature:** *By a* **function signature** *we shall understand*

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

Definition 17 Function Type Expression: *By a* **function type expression** *we shall understand*

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

4.9. 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: $\mathcal{P}$(v,σ)
    post: $\mathcal{Q}$(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 $\mathcal{P}$(v,σ).
- The post condition $\mathcal{Q}$(v,σ,σ') characterises the "after" state, σ':Σ, with respect to the "before" state, σ:Σ, and possible arguments (v:VAL).

Example 54 Insert Hub Action Formalisation: We formalise aspects of the above-mentioned hub action:

66 Insertion of a hub requires

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

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

value

66. $\mathsf{insert_H}: \mathsf{H} \to \mathsf{N} \xrightarrow{\sim} \mathsf{N}$
66. $\mathsf{insert_H(h)(n)}$ **as** n'
67. **pre**: $\sim\exists\ \mathsf{h}':\mathsf{H}\cdot\mathsf{h}' \in \mathbf{obs_part}_\mathsf{Hs}(\mathbf{obs_part}_\mathsf{HS(n)})\cdot\mathbf{uid}_\mathsf{H(h)}=\mathbf{uid}_\mathsf{H(h')}$
68. **post**: $\mathbf{obs_part}_\mathsf{Hs}(\mathbf{obs_part}_\mathsf{HS(n')})=\mathbf{obs_part}_\mathsf{Hs}(\mathbf{obs_part}_\mathsf{HS(n)})\cup\{\mathsf{h}\}$ ∎

- 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 !

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

- **Perdurant (action) analysis thus proceeds as follows:**
  - identifying relevant actions,
  - assigning names to these,
  - delineating the “smallest” relevant state[22],
  - ascribing signatures to action functions, and
  - determining
    - action pre-conditions and
    - action post-conditions.
  - 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”.

[22] 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.

- Example 54 showed example of a signature with only a part argument.
- Example 55 shows examples of signatures whose arguments are
  - parts and unique identifiers, or
  - parts, unique identifiers and attribute values.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

Example 55 Some Function Signatures:

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

  value insert_L: $\mathsf{L} \times (\mathsf{HI} \times \mathsf{HI}) \rightarrow \mathsf{N} \stackrel{\sim}{\rightarrow} \mathsf{N}$

- Removing a hub and removing a link:

  value remove_H: $\mathsf{HI} \rightarrow \mathsf{N} \stackrel{\sim}{\rightarrow} \mathsf{N}$
  remove_L: $\mathsf{LI} \rightarrow \mathsf{N} \stackrel{\sim}{\rightarrow} \mathsf{N}$

- Changing a hub state.

  value change_HΣ: $\mathsf{HI} \times \mathsf{H}\Sigma \rightarrow \mathsf{N} \stackrel{\sim}{\rightarrow} \mathsf{N}$ ■

4.10. 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 $= Q(\sigma,\sigma')$

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

- 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.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

Example 56 **Link Disappearence Formalisation**: We formalise aspects of the above-mentioned link disappearance event:

69 The result net is not well-formed.

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

71 and such a link may disappear such that

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

value

69. link_diss_event: $\mathsf{N} \times \mathsf{N}' \stackrel{\sim}{\to}$ **Bool**
69. link_diss_event(n,n′) **as** tf
70. **pre**: **obs_part**_Ls(**obs_part**_LS(n))≠{}
71. **post**: tf = ∃ l:L·l ∈ **obs_part**_Ls(**obs_part**_LS(n)) ⇒
72. l ∉ **obs_part**_Ls(**obs_part**_LS(n′))
72. ∧ n′ ∪ {l} = **obs_part**_Ls(**obs_part**_LS(n)) ■

4.11. Discrete Behaviour Signatures and Definitions

4.11.1. Behaviour Signatures

- We shall only cover behaviour signatures when expressed in RSL/CSP [GHH+92].
- The behaviour functions are now called processes.
- That a behaviour function is a never-ending function, i.e., a process, is "revealed" in the function signature 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" in the function signature as follows:

  behaviour: ... → **in** ch ...

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

- That a process offers channel, viz.: ch, outputs is “revealed” in the function signature as follows:

  behaviour: ... → **out** ch ...

- That a process accepts other arguments is “revealed” in the function signature as follows:

  behaviour: ARG → ...

- where ARG can be any type expression:

  T, T→T, T→T→T, etcetera

  where T is any type expression.

- As shown in [Bjø14a] we can, without loss of generality, associate with each part a behaviour;
  - parts which share attributes
  - (and are therefore referred to in some parts' mereology),
  - can communicate (their "sharing") via channels.
- The process evolves around a state, or, rather, a set of values:
  - its unique identity, $\pi : \Pi$,[23]
  - its possibly changing mereology, mt:MT[24],
  - the possible components and materials of the part, and
  - the constant, the external and the programmable attributes of the part.

[23] Unique identifiers of parts are like static attributes and hence (not really) contributing to the part state.

[24] For MT see footnote 14 on Slide 162.

- A behaviour signature is therefore:

  behaviour: π:Π $\times$ me:MT $\times$ sa:SA $\times$ ea:EA $\rightarrow$ pa:PA $\rightarrow$ **out** ochs **in** ichns **Unit**

  where

  - (i) π:Π is the unique identifier of part p, i.e., π=**uid**_P(p),
  - (ii) me:ME is the mereology of part p, me = **obs_mereo**_P(p),
  - (iii) sa:SA lists the static attribute values of the part,
  - (iv) ea:EA lists the external attribute channels of the part,
  - (v) pa:PA lists the programmable attribute values of the part, and where
  - (vi) ochs and ichns refer to otherwise mereology-determined output/input channels of the behaviours.

- We focus, for a little while, on the expression of
  - sa:SA,
  - ea:EA and
  - pa:PA,
- that is, on the concrete types of SA, EA and PA.
  - $\mathcal{S}_\mathcal{A}(p)$: SA simply lists the static value types: $svT_1, svT_2, ..., svT_s$ where s is the number of static attributes of parts p:P.
  - $\mathcal{E}_\mathcal{A}(p)$; EA simply lists the external attribute value channels: $(\mathbb{U}A_1$, $\mathbb{U}$A$_2$, ..., $\mathbb{U}$A$_x$)[25] where x is the number, 0 or more, of external attributes of parts p:P.
  - $\mathcal{P}_\mathcal{A}(p)$; PA simply lists the biddable and programmable value expression types: $(pvT_1, pvT_2, ..., pvT_q)$ where q is the number of programmable attributes of parts p:P.

[25] See paragraph ***Access to External Attribute Values*** on Slide 266.

4.11.2. Behaviour Definitions

- Let P be a composite sort defined in terms of sub-sorts PA, PB, . . . , PC.
  - The process definition compiled from p:P, is composed from
    - a process description, $\mathcal{M}_{cP_{\text{CORE}}}$..., relying on and handling the unique identifier, mereology and attributes of part p
    - operating in parallel with processes $p_a, p_b, \ldots, p_c$ where
      - p_a is "derived" from pa:PA,
      - p_b is "derived" from pb:PB,
      - ..., and
      - p_c is "derived" from pc:PC.
- The domain description "compilation" schematic below "formalises" the above.

Process Schema I: Abstract `is_composite(p)`

value

compile_process: P → RSL-**Text**

compile_process(p) ≡

$\mathcal{M}_{cP_{\rm CORE}}$(**uid**_P(p),**obs_mereo**_P(p),$\mathcal{S}_{\mathcal{A}}$(p),$\mathcal{E}_{\mathcal{A}}$(p))($\mathcal{P}_{\mathcal{A}}$(p))

‖ compile_process(**obs_part**_PA(p))

‖ compile_process(**obs_part**_PB(p))

‖ ...

‖ compile_process(**obs_part**_PC(p))

- The text macros: $\mathcal{S}_{\mathcal{A}}$, $\mathcal{E}_{\mathcal{A}}$ and $\mathcal{P}_{\mathcal{A}}$ were informally explained above.
- Part sorts PA, PB, ..., PC are obtained from the `observe_part_sorts` prompt, Slide 129.

- 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, $\mathcal{M}_{cP_{\text{CORE}}}$, 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 II: Concrete `is_composite(p)`

type
  Qs = Q-**set**
value
  qs:Q-**set** = **obs_part**_Qs(p)
  compile_process: P → RSL-**Text**
  compile_process(p) ≡
    $\mathcal{M}_{cP_{\mathrm{CORE}}}$(**uid**_P(p),**obs_mereo**_P(p),$\mathcal{S}_{\mathcal{A}}$(p),$\mathcal{E}_{\mathcal{A}}$(p))($\mathcal{P}_{\mathcal{A}}$(p))
   ∥ ∥{compile_process(q)|q:Q•q ∈ qs}

Process Schema III: `is_atomic(p)`

value
  compile_process: P → RSL-**Text**
  compile_process(p) ≡
    $\mathcal{M}_{aP_{\mathrm{CORE}}}$(**uid**_P(p),**obs_mereo**_P(p),$\mathcal{S}_{\mathcal{A}}$(p),$\mathcal{E}_{\mathcal{A}}$(p))($\mathcal{P}_{\mathcal{A}}$(p))

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

Example 57 Bus Timetable Coordination:

- We refer to Examples 17 on Slide 130, 18 on Slide 137, 37 on Slide 201 and 52 on Slide 257.

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

74 The fleet process is compiled as per Process Schema II (Slide 291).

- The definitions of the fleet and vehicle processes
  - are simplified
  - so as to emphasize the master/slave, programmable/inert
  - relations between these processes.

type
Δ, F, VS [Example 17 on Slide 130]
V, Vs=V-**set** [Example 18 on Slide 137]
FI, VI, BT [Example 37 on Slide 201]
value
73. δ:Δ,
73. f:F = **obs_part**_F(δ),
73. vs:V-**set** = **obs_part**_Vs(**obs_part**_VS(f))
axiom
73. $\forall$ v:V•v $\in$ vs $\Rightarrow$ $\Box$ **attr**_BT(f) = **attr**_BT(v) [Example 37 on Slide 201]
value
74. fleet: fi:FI $\rightarrow$ BT $\rightarrow$ **Unit**
74. fleet(fi)(bt) $\equiv$
74. $\mathcal{M}_F$(fi,attr_BT_ch)(bt)
74. $\|$ $\|$ {vehicle(**uid**_V(v),attr_BT_ch)|v:V•v $\in$ vs}

74. vehicle: vi:VI $\times$ $\mathbb{U}$BT $\rightarrow$ **Unit**
74. vehicle(vi,attr_BT_ch) $\equiv$ $\mathcal{M}_V$(fi,attr_BT_ch) ? ... ; vehicle(vi,attr_BT_ch)

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

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

value
  $\mathcal{M}_F$: fi:FI$\times\mathbb{U}\,BT \to$ BT $\to$ **Unit**
  $\mathcal{M}_F$(fi,attr_BT_ch)(bt) $\equiv$
    let bt′ = $\mathcal{F}$(fi,$\mathbb{U}\,BT$)(bt) **in** $\mathcal{M}_F$(fi,$\mathbb{U}\,BT$)(bt′) **end**

- Function $\mathcal{F}$ is a simple action.

- The expression of actual synchronisation and communication between the **fleet** and the **vehicle** processes
- is contained in $\mathcal{F}$.

```
value
    𝓕: fi:FI → bt:BT → BT
    𝓕(fi,attr_BT_ch)(bt) ≡
        let bt′ = f(bt)(...) in bt′ end ⌈⌉ attr_VT_ch ! bt ; bt
    f: BT → ... → BT
```

- The auxiliary function **f** "embodies" the programmable nature of the timetable attribute ■

Process Schema IV: Core Process (I)

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

$\mathcal{M}_{cP_{\text{CORE}}}$: π:Π×me:MT×sa:SA×ea:EA→pa:PA→**in** inchs **out** ochs **Unit**
$\mathcal{M}_{cP_{\text{CORE}}}$($\pi$,me,sa,ea)(pa) ≡
    let (me′,pa′) = $\mathcal{F}$(π,me,sa,ea)(pa) **in**
    $\mathcal{M}_{cP_{\text{CORE}}}$($\pi$,me′,sa,ea)(pa′) **end**

$\mathcal{F}$: π:Π×me:MT×sa:SA×ea:EA→PA→**in** inchs **out** ochs → MT×PA

- $\mathcal{F}$
  - potentially communicates with all those part processes (of the whole domain)
  - with which it shares attributes, that is, has connectors.
  - $\mathcal{F}$ is expected to contain input/output clauses referencing the channels of the **in** ... **out** ... part of their signatures.
  - These clauses enable the sharing of attributes.
  - $\mathcal{F}$ also contains expressions, attr_A_ch ?, to external attributes.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

- The $\mathcal{F}$ action non-deterministically internal choice chooses between
  - either [1,2,3,4]
    - [1] accepting input from
    - [4] a suitable (“offering”) part process,
    - [2] then optionally offering a reply to that other process, and
    - [3] finally delivering an updated state;
  - or [5,6,7,8]
    - [5] finding a suitable “order” (val)
    - [8] to a suitable (“inquiring”) behaviour (π'),
    - [6] offering that value (on channel ch[π']
    - [7] and then delivering an updated state;
  - or [9] doing own work resulting in an updated state.

Process Schema V: Core Process (II)

```
value
    𝓕: π:Π × me:MT × sa:SA × ea:EA → pa:PA → in,out 𝓔(π,me)  MT × PA
    𝓕(π,me,sa,ea)(pa) ≡
[1]         ⨆⨅{ let val = ch[π′] ? in
[2]               ( ch[π′] ! in_reply(val)(me,sa,ea)(pa) ⨅ skip ) ;
[3]               in_update(val)(me,sa,ea)(pa) end
[4]             | π′: Π • π′ ∈ 𝓔(π,me)}
[5]       ⨅⨆⨅{ let val = await_reply(π′)(me,sa,ea)(pa) in
[6]               ch[π′] ! val ;
[7]               out_update(val)(me,sa,ea)(pa) end
[8]             | π′: Π • π′ ∈ 𝓔(π,me)}
[9]       ⨅       (me,own_work(sa,ea)(pa))

    in_reply:  VAL → SA×EA → PA → VAL
    in_update: VAL → MT×SA×EA → PA → MT×PA
    await_reply: Π → MT×SA×EA → PA → VAL
    out_update: VAL → MT×SA×EA → PA → MT×PA
    own_work: SA×EA → PA → PA
```


© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

Example 58 Tollgates: Part and Behaviour:

- Our example is disconnected from that of a larger example of road pricing.
  - Figure 3 abstracts essential features of a tollgate.

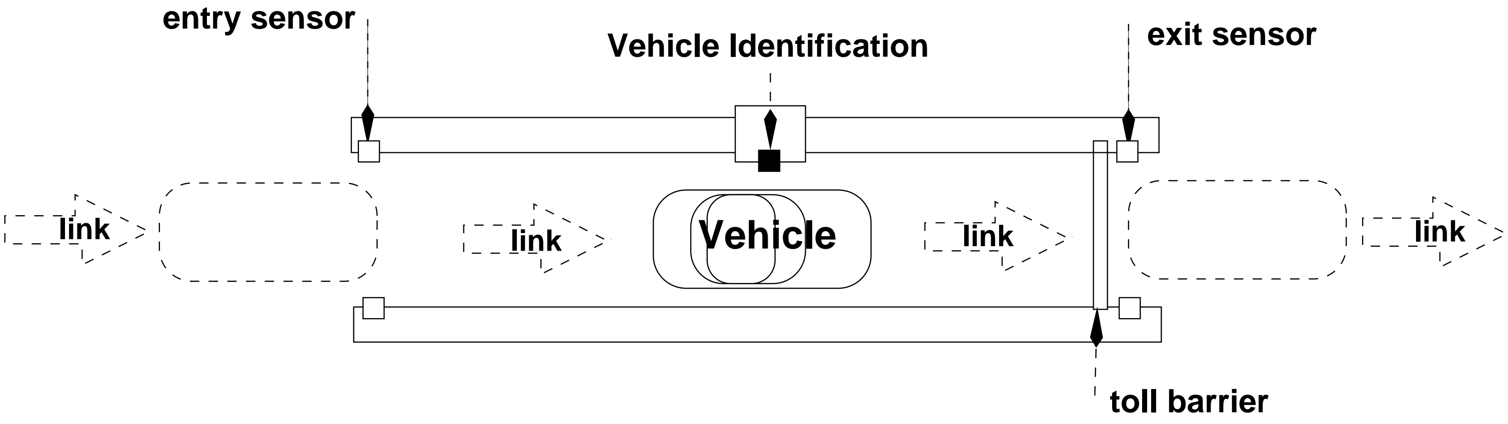


Figure 3: A tollgate

75 A tollgate is a composite part.
It consists of

76 an entry sensor (ES),
a vehicle identity sensor (IS),
a barrier (B), and
an exit sensor (XS).

77 The sensors function as follows:

a. When a vehicle first starts passing the entry sensor
then it sends an appropriate (event) message to the tollgate.

b. When a vehicle's identity is recognised by the identity sensor
then it sends an appropriate (event) message to the tollgate.

c. When a vehicle ends passing the exit sensor
then it sends an appropriate (event) message to the tollgate.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

78 We therefore model these sensors as shared dynamic event attributes.

a. For the sensors these are master attributes.

b. For the tollgate they are slave attributes.

c. In all three cases they are therefore modeled as channels.

79 A vehicle passing the gate

a. first "triggers" the entry sensor ("Enter"),

b. which results in the lowering ("Lower") of the barrier,

c. then the vehicle identity sensor ("vi:VI"),

d. with the tollgate "mysteriously"[26] handling that identity, and, simultaneously

e. raising ("Raise") the barrier, and

f. finally the output sensor ("Exit") is triggered as the vehicle leaves the tollgate,

g. and the barrier is lowered.

80 whereupon the tollgate resumes being a tollgate.

81 Chs is the type of a quadruple of channels.

82 TGI is the type unique tollgate identifiers.

[26]... that is, passes vi on to the road pricing monitor — where we omit showing relevant channels.

- Instead of one tollgate we may think of a number of tollgates:
  - Each with their unique identifier — together with a finite set of two or more such identifiers, **tgis:TGI-set**.

```
type
75.    TG
76.    ES, IS, B, XS
79a..  En = {|"Enter"|}
79b..  Ba = {|"Lower","Raise"|}
79c..  Id = VI
79e..  Ex = {|"Exit"|}
81.    Chs = 𝕌En×𝕌Id×𝕌Ba×𝕌Ex
82.    TGI
value
76.    obs_part_ES: TG → ES
76.    obs_part_IS: TG → IS
76.    obs_part_B: TG → B
76.    obs_part_XS: TG → XS
82.    uid_TGI: TG → TGI
79a..  attr_Enter: TG|ES → {|"Enter"|} event
79c..  attr_Identity: TG|IS → VI event
79e..  attr_Exit: TG|XS → {|"Exit"|} event
81.    chs:Chs
channel
79.    {attr_En_ch[tgi]|tgi:TGI•tgi∈tgis}
79.        {|"Enter"|} event
79.    {attr_Id_ch[tgi]|tgi:TGI•tgi∈tgis}
79.        VI event
79.    {attr_Ba_ch[tgi]|tgi:TGI•tgi∈tgis}
79.        {|"Lower","Raise"|}
79.    {attr_Ex_ch[tgi]|tgi:TGI•tgi∈tgis}
79.        {|"Exit"|} event
value
79.    gate: tgi:TGI × Chs → Unit
79.    gate(tgi,chs) ≡
79a..      attr_En_ch[tgi] ? ;
79b..      attr_Ba_ch[tgi] ! "Lower" ;
79c..      let vi = attr_Id_ch[tgi] ? in
79d..      ( handle(vi) ||
79e..      attr_Ba_ch[tgi] ! "Raise" ) ;
79f..      attr_Ex_ch[tgi] ? ;
79g..      attr_Ba[tgi] ! "Lower" ;
80.        gate(tgi,chs) end
```


© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

- The enter, identity and exit events are
  - slave attributes of the tollgate part and
  - master attributes of respectively
    - the entry sensor,
    - the vehicle identity sensor, and
    - the exit sensor sub-parts.
- We do not define the behaviours of these sub-parts.
  - We only assume that they each issue appropriate
  - attr_A_ch ! **out**put messages
  - where A is either Enter, Identity, or Exit and where event values en:Enter and ex:Exit are ignored ∎

4.12. Concurrency: Communication and Synchronisation

- Process Schemas I, II and IV (Slides 289, 291 and 296), reveal
  - that two or more parts, which temporally coexist (i.e., at the same time),
  - imply a notion of concurrency.
  - Process Schema IV, 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.

4.13. 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.

4.13.1. 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.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

4.13.2. Discussion

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

5. Closing

- It is time to conclude.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

5.1. Analysis & Description Calculi for Other Domains

- The analysis and description calculus of this paper appears suitable for manifest domains.
- For other domains other calculi may be necessary.
  - There is the introvert, composite domain(s) of systems software:
    - operating systems, compilers, database management systems, Internet-related software, etcetera.
    - The classical computer science and software engineering disciplines related to these components of systems software appears to have provided the necessary analysis and description “calculi.”

- There is the domain of financial systems software
  - accounting & bookkeeping,
  - banking systems,
  - insurance,
  - financial instruments handling (stocks, etc.),
  - etcetera.
- Etcetera.
- For each domain characterisable by a distinct set of analysis & description calculus prompts such calculi must be identified.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

5.2. On Domain Description Languages

- We have in this seminar expressed the domain descriptions in the RAISE [GHH+95] specification language RSL [GHH+92].
- With what is thought of as minor changes, one can reformulate these domain description texts in either of
  - Alloy [Jac06] or
  - The B-Method [Abr09] or
  - VDM [BJ78, BJ82, FL98] or
  - Z [WD96].

- One could also express domain descriptions algebraically, for example in `CafeOBJ`.
  - The analysis and the description prompts remain the same.
  - The description prompts now lead to `Alloy`, `B-Method`, `VDM`, `Z` or `CafeOBJ` texts.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

- We did not go into much detail with respect to perdurants.
  - For all the very many domain descriptions, covered elsewhere, `RSL` (with its `CSP` sub-language) suffices.
  - But there are cases, not documented in this seminar, where, [BGH+in], we have conjoined our `RSL` domain descriptions with descriptions in
    - `Petri Nets` [Rei10] or
    - `MSC` [IT99] or
    - `StateCharts` [Har87].

5.3. Comparison to Other Work

5.3.1. Background: The TripTych Domain Ontology

- We shall now compare the approach of this paper to
  - a number of techniques and tools
  - that are somehow related —
  - if only by the term 'domain'!
- Common to all the "other" approaches is that
  - none of them presents a prompt calculus
  - that help the domain analyser
  - elicit a, or the, domain description.
- Figure 1 on Slide 100 shows the tree-like structuring of what modern day AI researchers cum ontologists would call *an upper ontology*.

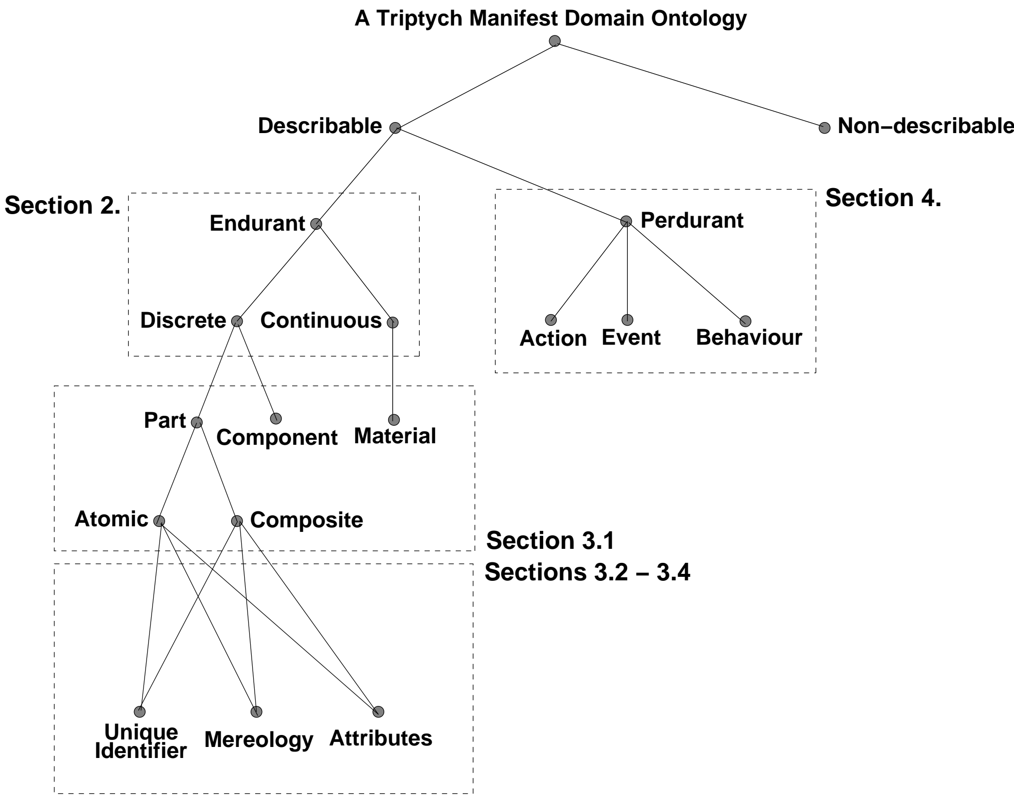


Figure 4: The Upper Ontology of TripTych Manifest Domains

5.3.2. General

Two related approaches to structuring domain understanding will be reviewed.

5.3.2.1 Ontology Science & Engineering

- Ontologies are *"formal representations of a set of concepts within a domain and the relationships between those concepts"* — expressed usually in some logic.
- **Ontology engineering** [BF98] construct ontologies.
- **Ontology science** appears to mainly study structures of ontologies, especially so-called **upper ontology** structures, and these studies "waver" between **philosophy** and **information science**[27].

[27] We take the liberty of regarding information science as part of **computer science**, cf. Slide 34.

- Internet published ontologies usually consists of thousands of logical expressions.
- These are represented in some, for example, low-level mechanisable form so that they can be interchanged between ontology research groups and processed by various tools.
- There does not seem to be a concern for “deriving” such ontologies into requirements for software.

- Usually ontology presentations
  - either start with the presentation of,
  - or makes reference to its reliance on,

  an **upper ontology**.
- The term 'ontology' has been much used in connection with automating the design of various aspects WWW applications.
- Description Logic *[The Description Logic Handbook: Theory, Implementation and Applications]* has been proposed as a language for the Semantic Web *(Baader et al., 2005)*.

- The interplay between endurants and perdurants is studied in [*Endurants and perdurants in directly depicting ontologies*].
  - That study investigates axiom systems for two ontologies.
  - One for endurants (`SPAN`), another for perdurants (`SNAP`).
  - No examples of descriptions of specific domains are, however, given, and thus no specific techniques nor tools are given, method components which could help the engineer in constructing specific domain descriptions.
  - [BDS04] is therefore only relevant to the current paper insofar as it justifies our emphasis on endurant versus perdurant entities.
- The interplay between endurant and perdurant entities and their qualities is studied in [*Qualities, Quantities, and the Endurant-Perdurant Distinction in Top-Level Ontologies*].

- In our study
  - the term **quality** is made specific and covers the ideas of
    - external and
    - internal

    qualities, cf. Sect. 2.6.10.
  - External qualities focus on
    - whether endurant or perdurant,
    - whether part, component or material,
    - whether action, event or behaviour,
    - whether atomic or composite part,
    - etcetera.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

- Internal qualities focus on
  - unique identifiers (of parts),
  - the mereology (of parts), and
  - the attributes (of parts, components and materials),

  that is, of endurants.

- In [*Johansson*]
  - the relationship between universals (types), particulars (values of types) and qualities is not "restricted"
  - as in the **TripTych** domain analysis,
  - but is axiomatically interwoven
  - in an almost "recursive" manner.

- Values [of types ('quantities' [of 'qualities'])] are,
  - for example, seen as sub-ordinated types;
  - this is an ontological distinction that we do not make.
- The concern of [*Johansson*] is also the relations
  - between qualities and both endurant and perdurant entities,
  - where we have yet to focus on "qualities",
  - other than signatures, of perdurants.

- [*Johansson*] investigates
  - the quality/quantity issue wrt. endurance/perdurance
  - and poses the questions:
    - [b] are non-persisting quality instances enduring, perduring or neither? and
    - [c] are persisting quality instances enduring, perduring or neither?
  - and arrives, after some analysis of the endurance/perdurance concepts, at the answers:
    - [b′] non-persisting quality instances are neither enduring nor perduring particulars (i.e., entities), and
    - [c′] persisting quality instances are enduring particulars.
  - Answer [b′] justifies our separating enduring and perduring entities into two disjoint, but jointly "exhaustive" ontologies.

- The more general study of [*Johansson*]
  - is therefore really not relevant to our prompt calculi,
  - in which we do not speculate on more abstract, conceptual qualities,
  - but settle
    - on external endurant qualities,
    - on the **unique identifier**, **mereology** and **attribute** qualities of endurants,
    - and the simple relations between endurants and perdurants, specifically
      - in the relations between **signature**s of actions, events and behaviours and the endurant sorts , and
      - especially the relation between parts and behaviours as outlined in Sect. 3.7.4.

- That is, the `TripTych` approach to ontology,
  - i.e., its domain concept,
  - is not only model-theoretic,
  - but, we risk to say, radically different.

5.3.2.2 Knowledge Engineering

- The concept of **knowledge** has occupied philosophers since Plato.
  - No common agreement on what 'knowledge' is has been reached.
  - From [LFCO87, Aud95, Mer04, Sta99] we may learn that
    - *knowledge is a familiarity with someone or something;*
      - *it can include facts, information, descriptions, or skills acquired through experience or education;*
      - *it can refer to the theoretical or practical understanding of a subject;*
    - *knowledge is produced by socio-cognitive aggregates*
      - *(mainly humans)*
      - *and is structured according to our understanding of how human reasoning and logic works.*

- The aim of **knowledge engineering** was formulated, in 1983, by an originator of the concept, Edward A. Feigenbaum [FM83]:
  - **knowledge engineering** is an engineering discipline
  - that involves integrating knowledge into computer systems
  - in order to solve complex problems
  - normally requiring a high level of human expertise.

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

- **Knowledge engineering** can, perhaps, best be understood in contrast to **algorithmic engineering**:
  - In the latter we seek more-or-less conventional, usually **imperative programming language** expressions of algorithms
    - whose algorithmic structure embodies the knowledge
    - required to solve the problem being solved by the algorithm.
  - The former seeks to solve problems based on an interpreter inferring possible solutions from logical data. This logical data has three parts:
    - a collection that "mimics" the semantics of, say, the imperative programming language,
    - a collection that formulates the problem, and
    - a collection that constitutes the knowledge particular to the problem.
- We refer to [BN92].

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

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

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

    etcetera.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

5.3.3. Specific

5.3.3.1 Database Analysis

- There are different, however related "schools of database analysis".
  - DSD: the Bachman (or data structure) diagram model [Bac69];
  - RDM: the relational data model [Cod70]; and
  - ER: entity set relationshp model [Che76] "schools".
  - DSD and ER aim at graphically specifying database structures.
  - Codd's RDM simplifies the data models of DSD and ER while offering two kinds of languages with which to operate on RDM databases: SQL and Relational Algebra.
- All three "schools" are focused
  - more on data modeling for databases
  - than on domain modeling both endurant and perdurant entities.

5.3.3.2 Domain Analysis

- **Domain analysis**, or **product line analysis** (see below), as it was then conceived in the early 1980s by James Neighbors [Nei84],
  - is the analysis of related software systems in a domain
  - to find their common and variable parts.
- This form of domain analysis turns matters "upside-down":
  - it is the set of software "systems" (or packages)
  - that is subject to some form of inquiry,
  - albeit having some domain in mind,
  - in order to find common features of the software
  - that can be said to represent a named domain.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

- In this section we shall mainly be comparing the TripTych approach to domain analysis to that of Reubén Prieto-Dĩaz's approach [PD87, PD90, PDA91].
- Firstly, our understanding of **domain analysis** basically coincides with Prieto-Dĩaz's.
- Secondly, in, for example, [PD87], Prieto-Dĩaz's domain analysis is focused on the very important stages that precede the kind of **domain modeling** that we have described:
  - major concerns are
    - selection of what appears to be similar, but specific entities,
    - identification of common features,
    - abstraction of entities and
    - classification.
  - Selection and identification is assumed in our approach, but we suggest to follow the ideas of Prieto-Dĩaz.
  - Abstraction (from values to types and signatures) and classification into parts, materials, actions, events and behaviours is what we have focused on.

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

  but we find that that is not an important issue.

5.3.3.3 Domain Specific Languages

- Martin Fowler[28] defines a *Domain-specific language* (DSL)
  - *as a computer programming language*
  - *of limited expressiveness*
  - *focused on a particular domain* [Fow20].
- Other references are [MHS05, Spi01].
- Common to [Spi01, MHS05, Fow20] is
  - that they define a domain in terms of classes of software packages;
  - that they never really "derive" the DSL from a description of the domain; and
  - that they certainly do not describe the domain in terms of that DSL, for example, by formalising the DSL.

[28] http://martinfowler.com/dsl.html

- In [HPK11]
  - a domain specific language for railway tracks is the basis for verification of the monitoring and control of train traffic on these tracks.
  - Specifications in that domain specific language, DSL, manifested by track layout drawings and signal interlocking tables, are then translated into **RSL** [GHH+92], and, from there into **SystemC** [GLMS02].
  - [HPK11] thus takes one very specific DSL and shows how to (informally) translate their "programs", which are not "directly executable", and hence does not satisfy Fowler's definition of **DSL**a, into executable programs.
  - [HPK11] is a great paper, but it is not solving our problem, that of systematically describing any manifest domain.
  - [HPK11] does, however, point a way to search for — say graphical — DSLs and the possible translation of their programs into executable ones.

5.3.3.4 Feature-oriented Domain Analysis (FODA)

- Feature oriented domain analysis (**FODA**)
  - is a domain analysis method
  - which introduced feature modeling to domain engineering.
  - **FODA** was developed in 1990 following several U.S. Government research projects.
  - Its concepts have been regarded as "critically advancing software engineering and software reuse."
- The US Government–supported report [KCH+90] states: "***FODA** is a necessary first step*" for software reuse.

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

  indeed encourages reuse at all levels:
  - **domain description**s and
  - **requirements prescription**,

  we can only agree.
- Another source on FODA is [CE00].
- Since FODA "leans" quite heavily on 'Software Product Line Engineering' our remarks in that section, next, apply equally well here.

5.3.3.5 Software Product Line Engineering

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

- These are not the primary concerns of TripTych domain science & engineering.
  - But they do become concerns as we move from **domain descriptions** to **requirements prescriptions**.
  - But it strongly seems that **software product line engineering** is not really focused on the concerns of **domain description** — such as is TripTych **domain engineering**.
  - It seems that **software product line engineering** is primarily based, as is, for example, `FODA: Feature-oriented Domain Analysis`, on analysing features of software systems.
  - Our [Bjø11c] puts the ideas of **software product lines** and **model-oriented software development** in the context of the TripTych approach.

5.3.3.6 Problem Frames

- The concept of **problem frames** is covered in [Jac01].
- Jackson's prescription for software development focus on the "triple development" of descriptions of
  - the **problem world**,
  - the **requirements** and
  - the **machine** (i.e., the **hardware** and **software**) to be built.
- Here **domain analysis** means the same as for us: the **problem world analysis**.

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

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

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

  one is considering a potentially smaller fragment [Jac10] of the domain than is suggested by the **TripTych** approach.
- At the same time one is, however, sure to
  - consider aspects of the domain
  - that might have been overlooked when pursuing **domain description development**
  - in the "more general" **TripTych** approach.

5.3.3.7 Domain Specific Software Architectures (DSSA)

- It seems that the concept of DSSA
  - was formulated by a group of ARPA[29] project "seekers"
  - who also performed a year long study (from around early-mid 1990s);
  - key members of the DSSA project were Will Tracz, Bob Balzer, Rick Hayes-Roth and Richard Platek [Tra94].
- The [Tra94] definition of **domain engineering** is *"the process of creating a DSSA:*
  - ***domain analysis*** *and* ***domain modeling***
  - *followed by creating a* ***software architecture***
  - *and populating it with* ***software components.****"*

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

- This definition is basically followed also by [MG92, SG96, MC04].
- Defined and pursued this way, DSSA appears,
  - notably in these latter references, to start with
  - the analysis of software components, "per domain",
  - to identify commonalities within application software,
  - and to then base the idea of **software architecture**
  - on these findings.

- Thus DSSA turns matter "upside-down" with respect to TripTych requirements development
  - by starting with software components,
  - assuming that these satisfy some requirements,
  - and then suggesting domain specific software
  - built using these components.
- This is not what we are doing:
  - we suggest, [Bjø08], that requirements
    - can be "derived" systematically from,
    - and formally related back to domain descriptionss
    - without, in principle, considering software components,
    - whether already existing, or being subsequently developed.

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

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

5.3.3.8 Domain Driven Design (DDD)

- Domain-driven design (DDD)[30]
  - *"is an approach to developing software for complex needs*
  - *by deeply connecting the implementation to an evolving model of the core business concepts;*
  - *the premise of domain-driven design is the following:*
    - *placing the project's primary focus on the core domain and domain logic;*
    - *basing complex designs on a model;*
    - *initiating a creative collaboration between technical and domain experts to iteratively cut ever closer to the conceptual heart of the problem."*[31]

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

- We have studied some of the DDD literature,
  - mostly only accessible on the Internet, but see also [Hay09],
  - and find that it really does not contribute to new insight into **domains** such as wee see them:
  - it is just "plain, good old software engineering cooked up with a new jargon.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

5.3.3.9 Unified Modeling Language (UML)

- Three books representative of UML are [BRJ98, RJB98, JBR99].
- The term **domain analysis** appears numerous times in these books,
  - yet there is no clear, definitive understanding
  - of whether it, the **domain**, stands for entities in the domain such as we understand it,
  - or whether it is wrought up, as in several of the 'approaches' treated in this section, to wit, in items [3–5, 7–9] with
    - either **software design** (as it most often is),
    - or **requirements prescription**.

- Certainly, in UML,
  - in [BRJ98, RJB98, JBR99] as well as
  - in most published papers claiming "adherence" to UML,
  - that domain analysis usually
    - is manifested in some UML text
    - which "models" some **requirements** facet.
  - Nothing is necessarily wrong with that,
  - but it is therefore not really the TripTych form of **domain analysis**
    - with its concepts of abstract representations of endurant and perdurants,
    - with its distinctions between **domain** and **requirements**, and
    - with its possibility of "deriving"
      * **requirements prescription**s from
      * **domain descriptions**.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

- The UML notion of **class diagrams** is worth relating to our structuring of the domain.
  - Class diagrams appear to be inspired by [Bac69, Bachman, 1969] and [Che76, Chen, 1976].
  - It seems that
    - (i) each part sort — as well as other than part sorts — deserves a class diagram (box); and
    - (ii) that (assignable) attributes — as well as other non-part types — are written into the diagram box.

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

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

5.3.3.10 Requirements Engineering

- There are in-numerous books and published papers on requirements engineering.
  - A seminal one is [van09].
  - I, myself, find [Lau02] full of very useful, non-trivial insight.
  - [DT97] is seminal in that it brings a number or early contributions and views on requirements engineering.

- Conventional text books, notably [Pfl01, Pre01, Som06] all have their "mandatory", yet conventional coverage of **requirements engineering**.
  - None of them "derive" requirements from domain descriptions,
    - yes, OK, from domains,
    - but since their description is not mandated
    - it is unclear what "the domain" is.
  - Most of them repeatedly refer to **domain analysis**
    - but since a written record of that **domain analysis** is not mandated
    - it is unclear what "domain analysis" really amounts to.

- Axel van Laamsweerde's book [van09] is remarkable.
  - Although also it does not mandate descriptions of domains
  - it is quite precise as to the relationships between domains and requirements.
  - Besides, it has a fine treatment of the distinction between **goals** and **requirements**,
  - also formally.
- Most of the advices given in [Lau02]
  - can beneficially be followed also in
  - **TripTych requirements development**.
- Neither [van09] nor [Lau02] preempts **TripTych requirements development**.

5.3.4. Summary of Comparisons

- We find that there are two kinds of relevant comparisons:
  - the concept of ontology, its science more than its engineering, and
  - the *Problem Frame* work of Michael A. Jackson.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

- The ontology work, as commented upon in Item **[1]** (Slides 319–328), is partly relevant to our work:
  - There are at least two issues:
    - Different classes of domains may need distinct upper ontologies.
      - Section 4.2 suggests that there may be different upper ontologies for non-manifest domains such as *financial systems*, etcetera.
      - This seems to warrant at least a comparative study.
    - We have assumed, cf. Sect. 2.9.1, that attributes cannot be separated from parts.
      - [Joh05, Johansson 2005] develops the notion that *persisting quality instances are enduring particulars.*
      - The issue need further clarification.

- Of all the other "comparison" items (**[2]**–**[12]**) basically only Jackson's *problem frames* (Item **[8]**) and [HPK11] (Item **[5]**) really take
  - the same view of **domain**s and,
  - in essence, basically maintain similar relations between
    - **requirements prescription** and
    - **domain description**.
- So potential sources of, we should claim, mutual inspiration
  - ought be found in one-another's work —
  - with, for example, [GGJZ00, Jac10, HPK11],
  - and the present document,
  - being a good starting point.

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

  as highlighted in the process schemas of Sect. 5.4.4 Slides 289–299.

5.4. Open Problems

- The present paper has outlined a great number of
  - principles,
  - techniques and
  - tools

  of domain analysis & description.
- They give rise, now, to the investigation of further
  - principles,
  - techniques and
  - tools

  as well as underlying theories.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

- We list some of these “to do” items:
  - *a mathematical model of prompts;*
  - *a sharpened definition of “what is a domain”;*
  - *laws of description prompts;*
  - *a formal understanding of domain facets* [Bjø10a];
  - *a prompt calculus for perdurants;*
  - *commensurate discrete and continuous models;*
  - *a study of the interplay between parts, materials and components;* and
  - *specific domain theories.*

5.5. 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[32]:

"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.

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

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 are covered, to some extent, in [Bjø06, Part IV].
- Tony Hoare's list pertains to a wider range that just the Manifest Domains treated in this chapter.

5.6. Beauty Is Our Business

It's life that matters, nothing but life –
the process of discovering, the everlasting and perpetual process,
not the discovery itself, at all.[33]

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

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – Wien October 21, 2015: 09:39

5.7. Acknowledgements

- This paper has been many years underway.
- Earlier versions have been the basis for ("innumerable") PhD lectures and seminars around the world — after I was retired from The Technical University of Denmark.
- I thank the many organisers of those events for their willingness to "hear me out":
  - Jin Song Dong, NUS, Singapore;
  - Kokichi Futatsgi and Kazuhiro Ogata, JAIST, Japan;
  - Dominique Méry, Univ. of Nancy, France;
  - Franz Wotawa and Bernhard K. Aichernig, Techn. Univ. of Graz, Austria;
  - Wolfgang J. Paul, Univ. of Saarland, Germany;

- Alan Bundy, University of Edinburgh, Scotland;
- Tetsuo Tamai, then at Tokyo Univ., now at Hosei Univ., Tokyo, Japan;
- Jens Knoop, Techn. Univ. of Vienna, Austria;
- Dömölki Balint and Kozma László, Eötös Loránt Univ., Budapest, Hungary;
- Lars-Henrik Ericsson, Univ. of Uppsala, Sweden;
- Peter D. Mosses, Univ. of Swansea, Wales;
- Magne Haveraaen, Univ. of Bergen, Norway;

- Sun Meng, Peking Univ., China;
- He JiFeng and Zhu HuiBiao, East China Normal Univ., Shanghai, China;
- Zhou Chaochen, Lin Huimin and Zhan Naijun, Inst. of Softw., CAS, Beijing, China;
- Victor P. Ivannikov, Inst. of Sys. Prgr., RAS, Moscow, Russia;
- Luís Soares Barbosa and Jose Nuno Oliveira, Univ. of Minho, Portugal and
- Jens Knoop, Techn. Univ. of Vienna, Austria.

- I finally thank the referees of this paper for their most helpful comments.

From Domain Descriptions to Requirement Prescriptions

A Different Approach to Requirements Engineering

Dines Bjørner

Fredsvej 11, DK–2840 Holte, Denmark

October 21, 2015: 09:39

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

- In [Manifest Domains: Analysis & Description] we introduced a method for analysing and describing manifest domains.
- In the next lectures of this PhD course
  - we show how to systematically,
  - but, of course, not automatically,
  - "derive" requirements prescriptions from
  - domain descriptions.

- There are, as we see it, three kinds of requirements:
  - domain requirements —
  - also referred to as the design assumptions, and the
  - interface requirements and
  - machine requirements —
  - these last two together referred to as the design requirements.
- The machine is the hardware and software to be developed from the requirements.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

- (i) **Domain requirements** are those requirements which can be expressed sôlely using technical terms of the domain.
- (ii) **Interface requirements** are those requirements which can be expressed using technical terms of both the domain and the machine.
- (iii) **Machine requirements** are those requirements which can be expressed sôlely using technical terms of the machine.
- Since it is the machine we wish to design we
  - refer to (ii-iii) as the design requirements
  - and to (i) as the design assumptions.

- We show principles, techniques and tools for "deriving"
  - domain requirements.
- The domain requirements development focus on
  - (i.1) projection,
  - (i.2) instantiation,
  - (i.3) determination,
  - (i.4) extension and
  - (i.5) fitting.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

- These domain-to-requirements operators can be described briefly:
  - (i.1) **projection** removes such descriptions which are to be omitted for consideration in the requirements,
  - (i.2) **instantiation** mandates specific mereologies,
  - (i.3) **determination** specifies less non–determinism,
  - (i.4) **extension** extends the evolving requirements prescription with further domain description aspects and
  - (i.5) **fitting** resolves "loose ends" as they may have emerged during the domain-to-requirements operations.
- Domain requirements will also be referred to a design assumptions.

- We briefly review principles, techniques and tools for "deriving" **interface requirements** based on sharing domain
  - (ii.1) endurants,
  - (ii.2) actions, events and behaviours

  with their machine correspondants.
- Interface and machine requirements (next) will also be referred to as **design requirements**.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

- Finally we
  - review **machine requirements**
  - while relating a number of machine requirements aspects
  - to domain phenomena.
- The "twist" here is that we introduce the machine requirements concepts of
  - (iii.1) **derived machine requirements** (or just **derived requirements**),
  - (iii.2) **technology requirements** and
  - (iii.3) **development requirements**.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

- The reason for exploring machine requirements in some detail
  - is to analyse requirements issues
  - in the light of the domain concept.
- We think that there is some clarification to be gained.
- We claim that our approach
  - contributes to a restructuring of
  - the field of requirements engineering
  - and its very many diverse concerns,
  - a structuring that is logically motivated
  - and is based on viewing
    software specifications as mathematical objects.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

1. Introduction

- In [*Manifest Domains: Analysis & Description*] we introduced a method for analysing and describing manifest domains.
  - In these lectures
    - we show how to systematically,
    - but, of course, not automatically,
    - "derive" requirements prescriptions from
    - domain descriptions.

1.1. The Triptych Dogma of Software Development

- We see software development progressing as follows:
  - *Before one can design software*
  - *one must have a firm grasp of the requirements.*
  - *Before one can prescribe requirements*
  - *one must have a reasonably firm grasp of the domain.*
- Software engineering, to us, therefore include these three phases:
  - *domain engineering,*
  - *requirements engineering* and
  - *software design.*

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

1.2. Software As Mathematical Objects

- Our base view is that computer programs are mathematical objects.
  - That is, the text that makes up a computer program can be reasoned about.
  - This view entails that computer program specifications can be reasoned about.
  - And that the requirements prescriptions upon which these specifications are based can be reasoned about.

- This base view entails, therefore,
  - that specifications,
    - whether **software design specification**s,
    - or **requirements prescription**s,
    - or **domain description**s,
  - must [also] be **formal specification**s.
- This is in contrast to considering **software design specification**s
  - being artifacts of sociological,
  - or even of psychological

  "nature".

1.3. The Contribution of These Lectures

- We claim that the present lecture content contributes to our understanding and practice of **software engineering** as follows:
  - (i) it shows how the new phase of engineering,
    - domain engineering,
    - as introduced in [Bjø16b],

    forms a prerequisite for requirements engineering;
  - (ii) it endows the "classical" form of requirements engineering with a structured set of development stages and steps:
    - (a) first a domain requirements stage,
    - (b) to be followed by an interface requirements stages, and
    - (c) to be concluded by a machine requirements stage;

- (iii) it further structures and gives a reasonably precise contents to the stage of domain requirements:
  - (1) first a projection and simplification step,
  - (2) then an instantiation step,
  - (3) then a determination step,
  - (4) then an extension step, and
  - (5) finally a fitting step —

  with these five steps possibly being iterated;

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

- (iv) it also structures and gives a reasonably precise contents to the stage of interface requirements based on a notion of shared entities; and
- (v) it finally structures and gives a reasonably precise contents to the stage of machine requirements:
  - (α) soft requirements,
  - (β) technology requirements and
  - (γ) development requirements.

- Stages (a–c) and steps (1–5, α–γ), we claim, are new.
- They reflect a serious contribution, we claim, to a logical structuring of the field of requirements engineering and its very many otherwise seemingly diverse concerns.

1.4. Some Comments on the Lecture Content

- These lectures are, perhaps, unusual in the following respects:
  - They are methodology[34] lectures, hence there are no "neat" theories, no succinctly expressed propositions, lemmas nor theorems, and hence no proofs.
  - As a consequence the lectures are borne by many, and by extensive examples.
  - The examples of these lectures are all focused on a generic road transport net.

[34] By **methodology** we understand the study and knowledge of one or more methods ■ By a **method** understand the study and knowledge of the principle, techniques and tools for constructing some artifact, here (primarily) software ■

- To reasonably fully exemplify the requirements approach,
  - illustrating how our method copes with
  - a seeming complexity of interrelated method aspects,
  - the full example of these lectures embodies
  - hundreds of concepts (types, axioms, functions).

- These methodology lectures covers a "grand" area of software engineering:
  - Many textbooks and papers are written on *Requirements Engineering*.
  - We postulate, in contrast to all such books (and papers), that requirements engineering should be founded on domain engineering.
  - Hence we must, somehow, show that our approach relates to major elements of what the *Requirements Engineering* books put forward.
- As a result these lectures are many!

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

1.5. Structure of Lectures

- The structure of the paper is as follows:
  - Section 2. provides a fair-sized, hence realistic example
  - Sections 3–6. covers our approach to requirements development.
    - Section 3. overviews the issue of 'requirements', relates our approach (Sects. 4.–6.) to
      - *Systems*,
      - *User and External Equipment* and
      - *Functional Requirements*,

      and
    - Sect. 3. also introduces the concepts of
      - the *machine* to be requirements prescribed,
      - the *domain*,
      - the *interface* and
      - the *machine requirements*.

- Section 4. covers the ***domain requirements*** stages of
  - *projection* (Sect. 4.1),
  - *instantiation* (Sect. 4.2),
  - *determination* (Sect. 4.3),
  - *extension* (Sect. 4.3),
  - *fitting* (Sect. 4.5).
- Section 5. covers key features of ***interface requirements***:
  - shared phenomena (Sect. 5.1),
  - shared endurants (Sect. 5.2),
  - shared actions,
    shared events
    shared behaviours (Sect. 5.3).

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

- Section 6. covers key features of ***machine requirements***:
  - soft requirements (Sect. 6.2),
  - technology requirements (Sect. 6.3)
  - development requirements (Sect. 6.4)
- Section 7. concludes the paper.

2. An Example Domain: Transport

- In order to exemplify the various stages and steps of requirements development we first bring a domain description example.
  - The example follows the steps of an idealised domain description.
  - First we describe the endurants,
  - then we describe the perdurants.
- Endurant description initially focus on the composite and atomic parts.
- Then on their "internal" qualities:
  - unique identifications,
  - mereologies, and
  - attributes.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

- The descriptions alternate between
  - enumerated, i.e., labeled narrative sentences and
  - correspondingly "numbered" formalisations.
- The narrative labels cum formula numbers
  - will be referred to, frequently in the
  - various steps of domain requirements development.

2.1. Endurants

- Since we have chosen a manifest domain,
  that is, a domain whose endurants can be pointed at, seen, touched,
  we shall follow the analysis & description process as
  outlined in [Bjø16b] and formalised in [Bjø14b].
  - That is, we first identify, analyse and describe (manifest) parts, composite and atomic, abstract (Sect.) or concrete (Sect.).
  - Then we identify, analyse and describe
    - their unique identifiers (Sect.),
    - mereologies (Sect.), and
    - attributes (Sects. –).

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

2.1.1. Domain, Net, Fleet and Monitor

Applying `observe_part_sorts` [Bjø16b, Sect. 3.1.6] to to a transport domain δ:Δ yields the following.

- The root domain, Δ, is that of a composite traffic system
  - with a road net,
  - with a fleet of vehicles and
  - of whose individual position on the road net we can speak, that is, monitor.

83 We analyse the composite traffic system into

a. a composite road net,

b. a composite fleet (of vehicles), and

c. an atomic monitor.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

type
83 Δ
83a. N
83b. F
83c. M
value
83a. **obs_part**_N: $\Delta \rightarrow$ N
83b. **obs_part**_F: $\Delta \rightarrow$ F
83c. **obs_part**_M: $\Delta \rightarrow$ M

Applying `observe_part_sorts` [Bjø16b, Sect. 3.1.6] to a net, n:N, yields the following.

84 The road net consists of two composite parts,

a. an aggregation of hubs and

b. an aggregation of links.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

type
84a. HA
84b. LA
value
84a. **obs_part**_HA: N → HA
84b. **obs_part**_LA: N → LA

2.1.2. Hubs and Links

Applying `observe_part_types` [Bjø16b, Sect. 3.1.7] to hub and link aggregates yields the following.

85 Hub aggregates are sets of hubs.

86 Link aggregates are sets of links.

87 Fleets are set of vehicles.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

type
85 H, HS = H-**set**
86 L, LS = L-**set**
87 V, VS = V-**set**
value
85 **obs_part**_HS: HA → HS
86 **obs_part**_LS: LA → LS
87 **obs_part**_VS: F → VS

88 We introduce some auxiliary functions.

a. links extracts the links of a network.

b. hubs extracts the hubs of a network.

value
88a. links: Δ → L-**set**
88a. links(δ) ≡ **obs_part**_LS(**obs_part**_LA(**obs_part**_N(δ)))
88b. hubs: Δ → H-**set**
88b. hubs(δ) ≡ **obs_part**_HS(**obs_part**_HA(**obs_part**_N(δ)))

2.1.3. Unique Identifiers

Applying `observe_unique_identifier` [Bjø16b, Sect. 3.2] to the observed parts yields the following.

89 Nets, hub and link aggregates, hubs and links, fleets, vehicles and the monitor all

a. have unique identifiers

b. such that all such are distinct, and

c. with corresponding observers.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

type
89a. NI, HAI, LAI, HI, LI, FI, VI, MI
value
89c. **uid**_NI: N → NI
89c. **uid**_HAI: HA → HAI
89c. **uid**_LAI: LA → LAI
89c. **uid**_HI: H → HI
89c. **uid**_LI: L → LI
89c. **uid**_FI: F → FI
89c. **uid**_VI: V → VI
89c. **uid**_MI: M → MI
axiom
89b. $\mathsf{NI} \bigcap \mathsf{HAI} = \emptyset$, $\mathsf{NI} \bigcap \mathsf{LAI} = \emptyset$, $\mathsf{NI} \bigcap \mathsf{HI} = \emptyset$, etc.

where axiom 89b.. is expressed semi-formally, in mathematics.

We introduce some auxiliary functions:

90 **xtr_lis** extracts all link identifiers of a traffic system.

91 **xtr_his** extracts all hub identifiers of a traffic system.

92 Given an appropriate link identifier and a net **get_link** 'retrieves' the designated link.

93 Given an appropriate hub identifier and a net **get_hub** 'retrieves' the designated hub.

```
value
90  xtr_lis: Δ → LI-set
90  xtr_lis(δ) ≡
90    let ls = links(δ) in {uid_LI(l)|l:L•l ∈ ls} end
91  xtr_his: Δ → HI-set
91  xtr_his(δ) ≡
91    let hs = hubs(δ) in {uid_HI(h)|h:H•k ∈ hs} end
92  get_link: LI → Δ →~ L
92  get_link(li)(δ) ≡
92     let ls = links(δ) in
92     let l:L • l ∈ ls ∧ li=uid_LI(l) in l end end
92     pre: li ∈ xtr_lis(δ)
93  get_hub: HI → Δ →~ H
93  get_hub(hi)(δ) ≡
93     let hs = hubs(δ) in
93     let h:H • h ∈ hs ∧ hi=uid_HI(h) in h end end
93     pre: hi ∈ xtr_his(δ)
```


© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

2.1.4. Mereology

Applying `observe_mereology` [Bjø16b, Sect. 3.3.2] to hubs, links, vehicles and the monitor yields the following.

94 Hub mereologies reflect that they are connected to zero, one or more links.

95 Link mereologies reflect that they are connected to exactly two distinct hubs.

96 Vehicle mereologies reflect that they are connected to the monitor.

97 The monitor mereology reflects that it is connected to all vehicles.

98 For all hubs of any net it must be the case that their mereology designates links of that net.

99 For all links of any net it must be the case that their mereologies designates hubs of that net.

100 For all transport domains it must be the case that

a. the mereology of vehicles of that system designates the monitor of that system, and that

b. the mereology of the monitor of that system designates vehicles of that system.

```
value
94   obs_mereo_H: H → LI-set
95   obs_mereo_L: L → HI-set
axiom
95   ∀ l:L·card obs_mereo_L(l)=2
value
96   obs_mereo_V: V → MI
97   obs_mereo_M: M → VI-set
axiom
98   ∀ δ:Δ, hs:HS·hs=hubs(δ), ls:LS·ls=links(δ) ·
98        ∀ h:H·h ∈ hs·obs_mereo_H(h)⊆xtr_lis(δ) ∧
99        ∀ l:L·l ∈ ls·obs_mereo_L(l)⊆xtr_his(δ) ∧
100a.         let f:F·f=obs_part_F(δ) ⇒
100a.             let m:M·m=obs_part_M(δ),
100a.                 vs:VS·vs=obs_part_VS(f) in
100a.             ∀ v:V·v ∈ vs⇒uid_V(v) ∈ obs_mereo_M(m)
100b.           ∧ obs_mereo_M(m) = {uid_V(v)|v:V·v ∈ vs}
100b.         end end
```


© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

2.1.5. Attributes, I

We may not have shown all of the attributes mentioned below — so consider them informally introduced !

- **Hubs:**
  - *location*s are considered static,
  - *hub state*s and *hub state space*s are considered programmable;

- **Links:**
  - *length*s and *location*s are considered static,
  - *link state*s and *link state space*s are considered programmable;

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

- **Vehicles:**
  - *manufacturer name*, *engine type* (whether diesel, gasoline or electric) and *engine power* (kW/horse power) are considered static;
  - *velocity* and *acceleration* may be considered reactive (i.e., a function of gas pedal position, etc.),
  - *global position* (informed via a `GNSS: Global Navigation Satellite System`) and *local position* (calculated from a global position) are considered biddable

Applying `observe_attributes` [Bjø16b, Sect. 3.4.3] to hubs, links, vehicles and the monitor yields the following.

First hubs.

101 Hubs

a. have geodetic locations, GeoH,

b. have ***hub states*** which are sets of pairs of identifiers of links connected to the hub[35],

c. and have ***hub state spaces*** which are sets of hub states[36].

102 For every net,

a. link identifiers of a hub state must designate links of that net.

b. Every hub state of a net must be in the hub state space of that hub.

103 We introduce an auxiliary function: xtr_lis extracts all link identifiers of a hub state.

[35] A hub state “signals” which input-to-output link connections are open for traffic.

[36] A hub state space indicates which hub states a hub may attain over time.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

```
type
101a.     GeoH
101b.   HΣ = (LI×LI)-set
101c.   HΩ = HΣ-set
value
101a.     attr_GeoH: H → GeoH
101b.   attr_HΣ: H → HΣ
101c.   attr_HΩ: H → HΩ
axiom
102      ∀ δ:Δ,
102        let hs = hubs(δ) in
102        ∀ h:H • h ∈ hs •
102a.           xtr_lis(h)⊆xtr_lis(δ)
102b.         ∧ attr_Σ(h) ∈ attr_Ω(h)
102        end
value
103   xtr_lis: H → LI-set
103   xtr_lis(h) ≡
103       {li | li:LI,(li′,li″):LI×LI •
103             (li′,li″) ∈ attr_HΣ(h) ∧ li ∈ {li′,li″}}
```

Then links.

104 Links have lengths.

105 Links have geodetic location.

106 Links have states and state spaces:

a. States modeled here as pairs, (hi', hi''), of identifiers the hubs with which the links are connected and indicating directions (from hub h' to hub h''.) A link state can thus have 0, 1, 2, 3 or 4 such pairs.

b. State spaces are the set of all the link states that a link may enjoy.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

```
type
104     LEN
105     GeoL
106a.   LΣ = (HI×HI)-set
106b.   LΩ = LΣ-set
value
104     attr_LEN: L → LEN
105    attr_GeoL: L → GeoL
106a.   attr_LΣ: L → LΣ
106b.   attr_LΩ: L → LΩ
axiom
106   ∀ n:N •
106      let ls = xtr−links(n), hs = xtr_hubs(n) in
106      ∀ l:L•l ∈ ls ⇒
106a.        let lσ = attr_LΣ(l) in
106a.        0≤card lσ≤4
106a.      ∧ ∀ (hi′,hi″):(HI×HI)•(hi′,hi″) ∈ lσ
106a.            ⇒ {hi′,hi″}=obs_mereo_L(l)
106b.      ∧ attr_LΣ(l) ∈ attr_LΩ(l)
106      end end
```

Then vehicles.

107 Every vehicle of a traffic system has a position which is either 'on a link' or 'at a hub'.

a. An 'on a link' position has four elements: a unique link identifier which must designate a link of that traffic system and a pair of unique hub identifiers which must be those of the mereology of that link.

b. The 'on a link' position real is the fraction, thus properly between 0 (zero) and 1 (one) of the length from the first identified hub "down the link" to the second identifier hub.

c. An 'at a hub' position has three elements: a unique hub identifier and a pair of unique link identifiers — which must be in the hub state.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

```
type
107     VPos = onL | atH
107a.    onL :: LI HI HI R
107b.    R = Real          axiom ∀ r:R • 0≤r≤1
107c.    atH :: HI LI LI
value
107    attr_VPos: V → VPos
axiom
107a.    ∀ n:N, onL(li,fhi,thi,r):VPos •
107a.        ∃ l:L•l ∈obs_part_LS(obs_part_N(n))
107a.            ⇒ li=uid_L(l)∧{fhi,thi}=obs_mereo_L(l),
107c.    ∀ n:N, atH(hi,fli,tli):VPos •
107c.        ∃ h:H•h ∈obs_part_HS(obs_part_N(n))
107c.            ⇒ hi=uid_H(h)∧(fli,tli) ∈ attr_LΣ(h)
```

108 We introduce an auxiliary function **distribute**.

a. **distribute** takes a net and a set of vehicles and

b. generates a map from vehicles to distinct vehicle positions on the net.

c. We sketch a "formal" **distribute** function, but, for simplicity we omit the technical details that secures distinctness — and leave that to an axiom!

109 We define two auxiliary functions:

a. xtr_links extracts all links of a net and

b. xtr_hub extracts all hubs of a net.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

```
type
108b.   MAP = VI →m VPos
axiom
108b.   ∀ map:MAP · card dom map = card rng map
value
108    distribute: VS → N → MAP
108    distribute(vs)(n) ≡
108a.      let (hs,ls) = (xtr_hubs(n),xtr_links(n)) in
108a.      let vps = {onL(uid_(l),fhi,thi,r) |
108a.                 l:L•l ∈ls∧{fhi,thi}
108a.                 ⊆obs_mereo_L(l)∧0≤r≤1}
108a.               ∪ {atH(uid_H(h),fli,tli)|
108a.                 h:H•h ∈hs∧{fli,tli}
108a.                 ⊆obs_mereo_H(h)} in
108b.      [uid_V(v)↦vp|v:V,vp:VPos•v ∈vs∧vp∈vps]
108      end end
```

And finally monitors. We consider only one monitor attribute.

110 The monitor has a vehicle traffic attribute.

a. For every vehicle of the road transport system the vehicle traffic attribute records a possibly empty list of time marked vehicle positions.

b. These vehicle positions are alternate sequences of ‘on link’ and ‘at hub’ positions

i such that any sub-sequence of ‘on link’ positions record the same link identifier, the same pair of ‘’to’ and ‘from’ hub identifiers and increasing fractions,

ii such that any sub-segment of ‘at hub’ positions are identical,

iii such that vehicle transition from a link to a hub is commensurate with the link and hub mereologies, and

iv such that vehicle transition from a hub to a link is commensurate with the hub and link mereologies.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

```
type
110    Traffic = VI →m (T × VPos)*
value
110    attr_Traffic: M → Traffic
axiom
110b.   ∀ δ:Δ •
110b.       let m = obs_part_M(δ) in
110b.       let tf = attr_Traffic(m) in
110b.       dom tf ⊆ xtr_vis(δ) ∧
110b.       ∀ vi:VI • vi ∈ dom tf •
110b.          let tr = tf(vi) in
110b.          ∀ i,i+1:Nat • {i,i+1}⊆dom tr •
110b.             let (t,vp)=tr(i),(t′,vp′)=tr(i+1) in
110b.             t<t′
110(b.)i           ∧ case (vp,vp′) of
110(b.)i                (onL(li,fhi,thi,r),onL(li′,fhi′,thi′,r′))
110(b.)i                  → li=li′∧fhi=fhi′∧thi=thi′∧r≤r′ ∧ li ∈ xtr_lis(δ) ∧ {fhi,thi} = obs_mereo_L(get_link(li)(δ)),
110(b.)ii               (atH(hi,fli,tli),atH(hi′,fli′,tli′))
110(b.)ii                 → hi=hi′∧fli=fli′∧tli=tli′ ∧ hi ∈ xtr_his(δ) ∧ (fli,tli) ∈ obs_mereo_H(get_hub(hi)(δ)),
110(b.)iii              (onL(li,fhi,thi,1),atH(hi,fli,tli))
110(b.)iii                → li=fli∧thi=hi ∧ {li,tli} ⊆ xtr_lis(δ) ∧ {fhi,thi}=obs_mereo_L(get_link(li)(δ))
110(b.)iii                 ∧ hi ∈ xtr_his(δ) ∧ (fli,tli) ∈ obs_mereo_H(get_hub(hi)(δ)),
110(b.)iv               (atH(hi,fli,tli),onL(li′,fhi′,thi′,0))
110(b.)iv                 →  etcetera,
110b.               _  → false
110b.       end end end end end
```

2.2. Perdurants

- Our presentation of example perdurants is not as systematic as that of example endurants.
- Give the simple basis of endurants covered above there is now a huge variety of perdurants, so we just select one example from each of the three classes of perdurants (as outline in [Bjø16b]):
  - a simple hub insertion ***action*** (Sect.),
  - a simple link disappearance ***event*** (Sect.) and
  - a not quite so simple ***behaviour***, that of road traffic (Sect.).

 © Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

2.2.1. Hub Insertion Action

111 Initially inserted hubs, h, are characterised

a. by their unique identifier which not one of any hub in the net, n, into which the hub is being inserted,

b. by a mereology, $\{\}$, of zero link identifiers, and

c. by — whatever — attributes, $attrs$, are needed.

112 The result of such a hub insertion is a net, n',

a. whose links are those of n, and

b. whose hubs are those of n augmented with h.

value

111 insert_hub: H → N → N
112 insert_hub(h)(n) **as** n′
111a. **pre**: **uid**_H(h) ∉ xtr_his(n)
111b. ∧ **obs_mereo**_H= {}
111c. ∧ ...
112a. **post**: **obs_part**_Ls(n) = **obs_part**_Ls(n′)
112b. ∧ **obs_part**_Hs(n) ∪ {h} = **obs_part**_Hs(n′)

 © Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

2.2.2. Link Disappearance Event

We formalise aspects of the link disappearance event:

113 The result net, n':N', is not well-formed.

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

115 and such a link may disappear such that

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

```
value
113  link_diss_event: N × N′ × Bool
113  link_diss_event(n,n′) as tf
114     pre:  obs_part_Ls(obs_part_LS(n))≠{}
115     post: ∃ l:L·l ∈ obs_part_Ls(obs_part_LS(n)) ⇒
116              l ∉ obs_part_Ls(obs_part_LS(n′))
116              ∧ n′ ∪ {l} = obs_part_Ls(obs_part_LS(n))
```

2.2.3. Road Traffic

- The analysis & description of the road traffic behaviour is composed
  - (i) from the description of the global values of
    - nets, links and hubs,
    - vehicles,
    - monitor,
    - a clock, and
    - an initial distribution, ***map***, of vehicles, “across” the net;
  - (ii) from the description of channels
    - between vehicles and
    - the monitor;

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

- (iii) from the description of behaviour signatures, that is, those of
  - the overall road traffic system,
  - the vehicles, and
  - the monitor; and
- (iv) from the description of the individual behaviours, that is,
  - the overall road traffic system, ***rts***,
  - the individual vehicles, ***veh***, and
  - the monitor, ***mon***.

2.2.3.1 Global Values:

- There is given some globally observable parts.

117 besides the domain, δ:Δ,

118 a net, **n:N**,

119 a set of vehicles, **vs:V-set**,

120 a monitor, **m:M**, and

121 a clock, **clock**, behaviour.

122 From the net and vehicles we generate an initial distribution of positions of vehicles.

- The **n:N**, **vs:V-set** and **m:M** are observable from any road traffic system domain δ.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

```
value
117   δ:Δ
118   n:N = obs_part_N(δ),
118   ls:L-set=links(δ),hs:H-set=hubs(δ),
118   lis:LI-set=xtr_lis(δ),his:HI-set=xtr_his(δ)
119   va:VS=obs_part_VS(obs_part_F(δ)),
119   vs:Vs-set=obs_part_Vs(va),
119   vis:VI-set = {uid_VI(v)|v:V•v ∈ vs},
120   m:obs_part_M(δ),
120   mi=uid_MI(m),
120   ma:attributes(m)
121   clock: 𝕋 → out {clk_ch[ vi|vi:VI•vi ∈ vis ]}  Unit
122   vm:MAP•vpos_map = distribute(vs)(n);
```

2.2.3.2 Channels:

123 We additionally declare a set of vehicle-to-monitor-channels indexed

a. by the unique identifiers of vehicles

b. and the (single) monitor identifier.[37]

and communicating vehicle positions.

channel

123 $\{$v_m_ch[vi,mi]|vi:VI•vi $\in$ vis$\}$:VPos

[37] Technically speaking: we could omit the monitor identifier.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

2.2.3.3 Behaviour Signatures:

124 The road traffic system behaviour, rts, takes no arguments; and "behaves", that is, continues forever.

125 The vehicle behaviour

a. is indexed by the unique identifier, uid_V(v):VI,

b. the vehicle mereology, in this case the single monitor identifier mi:MI,

c. the vehicle attributes, obs_attribs(v)

d. and — factoring out one of the vehicle attributes — the current vehicle position.

e. The vehicle behaviour offers communication to the monitor behaviour; and behaves "forever".

126 The **mon**itor behaviour takes

a. the monitor identifier,

b. the monitor mereology,

c. the monitor attributes,

d. and — factoring out one of the vehicle attributes — the discrete road traffic, drtf:dRTF;

e. the behaviour otherwise behaves forever.

```
value
124   rts: Unit → Unit
125   veh: vi:VI × mi:MI → vp:VPos → out vm_ch[ vi,mi ] Unit
126   mon: m:M × vis:VI-set → RTF → in {v_m_ch[ vi,mi ]|vi:VI·vi ∈ vis},clk
```


© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

2.2.3.4 The Road Traffic System Behaviour:

127 Thus we shall consider our **road traffic system**, rts, as

a. the concurrent behaviour of a number of vehicles and,
to "observe", or, as we shall call it, to monitor their movements,

b. the **mon**itor behaviour.

value

127 rts() $=$

127a. $\|$ $\{$veh(**uid**_VI(v),mi)(vm(**uid**_VI(v)))|v:V·v $\in$ vs$\}$

127b. $\|$ mon(mi,vis)([vi$\mapsto\langle\rangle$|vi:VI·vi $\in$ vis])

- where, wrt, the monitor, we
  - dispense with the mereology and the attribute state arguments
  - and instead just have a monitor traffic argument which
    - records the discrete road traffic, MAP,
    - initially set to "empty" traces ($\langle\rangle$, of so far "no road traffic"!).
- In order for the monitor behaviour to assess the vehicle positions
  - these vehicles communicate their positions
  - to the monitor
  - via a vehicle to monitor channel.
- In order for the monitor to time-stamp these positions
  - it must be able to "read" a clock.

128 We describe here an abstraction of the vehicle behaviour **at** a Hub (hi).

a. Either the vehicle remains at that hub informing the monitor of its position,

b. or, internally non-deterministically,

  i moves onto a link, tli, whose "next" hub, identified by thi, is obtained from the mereology of the link identified by tli;

  ii informs the monitor, on channel vm[vi,mi], that it is now at the very beginning (0) of the link identified by tli, whereupon the vehicle resumes the vehicle behaviour positioned at the very beginning of that link,

c. or, again internally non-deterministically, the vehicle "disappears — off the radar" !

```
128      veh(vi,mi)(vp:atH(hi,fli,tli)) ≡
128a.            v_m_ch[vi,mi]!vp ; veh(vi,mi)(vp)
128b.        ⌈⌉
128(b.)i         let {hi′,thi}=obs_mereo_L(get_link(tli)(n)) in
128(b.)i                                    assert: hi′=hi
128(b.)ii        v_m_ch[vi,mi]!onL(tli,hi,thi,0) ;
128(b.)ii        veh(vi,mi)(onL(tli,hi,thi,0)) end
128c.        ⌈⌉ stop
```

129 We describe here an abstraction of the vehicle behaviour **on** a Link (ii). Either

a. the vehicle remains at that link position informing the monitor of its position,

b. or, internally non-deterministically, if the vehicle's position on the link has not yet reached the hub,

   i then the vehicle moves an arbitrary increment ℓ_ϵ (less than or equal to the distance to the hub) along the link informing the monitor of this, 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 the monitor that it is now at the hub identified by thi, whereupon the vehicle resumes the vehicle behaviour positioned at that hub.

c. or, internally non-deterministically, the vehicle "disappears — off the radar" !

```
129       veh(vi,mi)(vp:onL(li,fhi,thi,r)) ≡
129a.         v_m_ch[vi,mi]!vp ; veh(vi,mi,va)(vp)
129b.         ⌈⌉ if r + ℓϵ≤1
129(b.)i           then
129(b.)i             v_m_ch[vi,mi]!onL(li,fhi,thi,r+ℓϵ) ;
129(b.)i             veh(vi,mi)(onL(li,fhi,thi,r+ℓϵ))
129(b.)ii          else
129(b.)iiA             let li′:LI•li′ ∈ obs_mereo_H(get_hub(thi)(n)) in
129(b.)iiB              v_m_ch[vi,mi]!atH(li,thi,li′);
129(b.)iiB              veh(vi,mi)(atH(li,thi,li′)) end end
129c.         ⌈⌉ stop
```

The Monitor Behaviour

130 The monitor behaviour evolves around

a. the monitor identifier,

b. the monitor mereology,

c. and the attributes, ma:ATTR

d. — where we have factored out as a separate arguments — a table of traces of time-stamped vehicle positions,

e. while accepting messages

i about time

ii and about vehicle positions

f. and otherwise progressing “in[de]finitely”.

131 Either the monitor "does own work"

132 or, internally non-deterministically accepts messages from vehicles.

a. A vehicle position message, vp, may arrive from the vehicle identified by vi.

b. That message is appended to that vehicle's movement trace – prefixed by time (obtained from the time channel),

c. whereupon the monitor resumes its behaviour —

d. where the communicating vehicles range over all identified vehicles.

```
130   mon(mi,vis)(trf) ≡
131          mon(mi,vis)(trf)
132      ⌈⌉
132a.     ⌊⌋⌈⌉{let tvp = (clk_ch?,v_m_ch[ vi,mi ]?) in
132b.          let trf′ = trf † [ vi ↦ trf(vi)⌢<tvp> ] in
132c.          mon(mi,vis)(trf′)
132d.          end end | vi:VI • vi ∈ vis}
```

- We are about to complete a long, i.e., a 50 slide example (!).
- We can now comment on the full example:
  - The domain, $\delta : \Delta$ is a manifest part.
  - The road net, $n : N$ is also a manifest part.
  - The fleet, $f : F$, of vehicles, $vs : VS$, likewise, is a manifest part.
  - But the monitor, $m : M$, is a concept.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

- One does not have to think of it as a manifest "observer".
- The vehicles are on — or off — the road (i.e., links and hubs).
- We know that from a few observations and generalise to all vehicles.
- They either move or stand still. We also, similarly, know that.
- Vehicles move. Yes, we know that.
- Based on all these repeated observations and generalisations we introduce the concept of vehicle traffic.
- Unless positioned high above a road net — and with good binoculars — a single person cannot really observe the traffic.
- There are simply too many links, hubs, vehicles, vehicle positions and times.

◈ Thus we conclude that, even in a richly manifest domain, we can also "speak of", that is, describe concepts over manifest phenomena, including time!

2.3. Domain Facets

- The example of this section does not reflect the concepts of domain facets such as
  - domain intrinsics,
  - domain support technologies,
  - domain rules, regulations & scripts,
  - organisation & management, and
  - human behaviour.
  - These facets are covered in [Bjø10a, 2008].

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

3. Requirements

- This and the next three sections,
  Sects. 3., 4., 5. and 6., are the main sections
  of these lectures' coverage of requirements.
  - Section 4. is the most detailed and systematic section.
  - It covers the ***domain requirements*** operations of
    - projection,
    - instantiation,
    - determination,
    - extension and, less detailed,
    - fitting.

- Section 5. surveys the *interface requirements* issues of *shared phenomena*:
  - shared endurants,
  - shared actions,
  - shared events and
  - shared behaviour,

  and "completes" the exemplification of the detailed *domain extension* of our requirements into a *road pricing system*.

- Section 6. relates some machine requirements issues to the overall design of the *road pricing system*:
  - derived requirements,
  - technology requirements and
  - development requirements.
- This the, initial section captures main concepts and principles of requirements.

Definition 18 Requirements (I): *By a* **requirements** *we understand (cf., IEEE Standard 610.12):*

- *"A condition or capability needed by a user to solve a problem or achieve an objective"* ■

- The objective of requirements engineering is to create a **requirements prescription**:
  - A **requirements prescription** specifies observable properties of endurants and perdurants of **the machine** such as the requirements stake-holders wish them to be ■
  - The **machine** is what is required: that is, the *hardware* and *software* that is to be designed and which are to satisfy the requirements ■

- A ***requirements prescription*** thus (***putatively***) expresses what there should be.
- A requirements prescription expresses nothing about the design of the possibly desired (required) software.
- But as the requirements prescription is presented in the form of a model, one can base the design on that model.
- We shall show how a major part of a requirements prescription can be "derived" from "its" prerequisite domain description.
- Note that requirements is about ***systems***.

Rule 1 The "Golden Rule" of Requirements Engineering: *Prescribe only those requirements that can be objectively shown to hold for the designed software* ■

- "Objectively shown" means that the designed software can
  - either be tested,
  - or be model checked,
  - or be proved (verified),

  to satisfy the requirements.
- **Caveat**[38]

[38] Will not be illustrated!

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

Rule 2 An "Ideal Rule" of Requirements Engineering: *When prescribing (including formalising) requirements, also formulate tests and properties for model checking and theorems whose proof should show adherence to the requirements* ■

- The rule is labelled "ideal" since such precautions will not be shown in this seminar.
- The rule is clear.
- It is a question for proper management to see that it is adhered to.
- See the "Caveat" above!

Rule 3 Requirements Adequacy: *Make sure that requirements cover what users expect* ■

- That is,
  - do not express a requirement for which you have no users,
  - but make sure that all users' requirements are represented or somehow accommodated.
- In other words:
  - the requirements gathering process needs to be like an extremely "fine-meshed net":
  - One must make sure that all possible stake-holders have been involved in the requirements acquisition process,
  - and that possible conflicts and other inconsistencies have been obviated.

Rule 4 Requirements Implementability: *Make sure that requirements are implementable* ■

- That is, do not express a requirement for which you have no assurance that it can be implemented.
- In other words,
  - although the requirements phase is not a design phase,
  - one must tacitly assume, perhaps even indicate, somehow, that an implementation is possible.
- But the requirements in and by themselves, may stay short of expressing such designs.
- **Caveat** !

Definition 19: **Requirements (II):**

- By **requirements** we shall understand
  - ◈ a document
  - ◈ which prescribes desired properties of
  - ◈ a machine:
    - ⊚ what endurants the machine shall "maintain", and
    - ⊚ what the machine shall (must; not should) offer of
      - $*$ functions and of
      - $*$ behaviours
    - ⊚ while also expressing which events the machine shall "handle"

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

- By a machine that "maintains" endurants we shall mean:
  - ◈ a machine which, "between" users' use of that machine,
  - ◈ "keeps" the data that represents these entities.
- From earlier we repeat:

Definition 20: **Machine:** By *machine* we shall understand a, or the, combination of hardware and software that is the target for, or result of the required computing systems development ■

- So this, then, is a main objective of requirements development:
- to start towards the design of the hardware + software for the computing system.

Definition 21: **Requirements (III):** To specify the machine

- When we express requirements and wish to "convert" such requirements to a realisation, i.e., an implementation, then we find
  - that some requirements (parts) imply certain properties to hold of the hardware on which the software to be developed is to "run",
  - and, obviously, that remaining — probably the larger parts of the — requirements imply certain properties to hold of that software.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

3.1. Four Requirements Facets

- We shall unravel requirements in two stages —
  - (i) the first stage is sketchy (and thus informal)
  - (ii) while the last stage is systematic
    and both informal and formal.
  - The sketchy stage consists of
    - a narrative **problem/objective sketch**,
    - a narrative **system requirements sketch**, and
    - a narrative **user & external equipment requirements sketch**,

- The narrative and formal stage
  - consists of:
    - **design assumptions prescription** and
    - **design requirements prescription**,
  - It is systematic, and mandates
    - both strict narrative
    - and formal

    prescriptions.
  - And it is "derivable" from the domain description.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

3.1.1. Problem, Solution and Objective Sketch

Definition 22 Problem, Solution and Objective Sketch: *By a problem, solution and objective sketch we understand*

- *a narrative which emphasises*
- *what the* **problem** *to be solved is,*
- *outlines a possible* **solution**
- *and sketches an* **objective** *of the solution* ■

Example 59 . The Problem/Objective Requirements: A Sketch:

- The *problem* is that of traffic congestion.
- The chosen *solution* is to [build and] operate a toll-road system integrated into a road net and charge toll-road users a usage fee.
- The *objective* is therefore to create a **road-pricing product.**
  - By a road-pricing product
    - we shall understand an IT-based system
    - containing C&C equipment and software
    - that enables the recording of *vehicle* movements
    - within the *toll-road*
    - and thus enables
      - the *owner* of the road net to charge
      - the *owner* of the vehicles
      - *fees* for the usage of that toll-road ■

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

3.1.2. Systems Requirements

Definition 23 **System Requirements:** *By a* **system requirements narrative** *we understand*

- *a narrative which emphasises*
- *the overall assumed and/or required*
- *hardware and software system equipment* ■

Example 60 . The Road-pricing System Requirements: A Narrative:

- The requirements are based on the following constellation of system equipment:
  - there is assumed a GNSS: a GLOBAL NAVIGATION SATELLITE SYSTEM;
  - there are *vehicles* equipped with GNSS receivers;
  - there is a well-delineated road net called a *toll-road* net with specially equipped *toll-gates* with
    - *vehicle identification sensors*,
    - *barriers* which afford (only specially equipped) vehicles to enter into and exit from the toll-road net;

    and
  - there is a *road-pricing calculator*.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

- **The system to be designed (from the requirements) is the road-pricing calculator.**
- These four system elements are required to behave and interact as follows:
  - The GNSS is assumed to continuously offer vehicles information about their global position;
  - *vehicles* shall contain a GNSS receiver which based on the global position information shall regularly calculate their timed local position and offer this to the *calculator* — while otherwise cruising the general road net as well as the toll-road net, the latter while carefully moving through toll-gates;

- *toll-gates* shall register the identity of vehicles entering and exiting the toll-road and offer this information to the calculator; and
- the *calculator* shall accept all messages from vehicles and gates and use this information to record the movements of vehicles and bill these whenever they exit the toll-road.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

- The requirements are therefore to include **assumptions about**
  - the *GNSS* satellite and telecommunications equipment,
  - the vehicle *GNSS receiver* equipment,
  - the vehicle handling of *GNSS* input and forwarding, to the road pricing system, of its interpretation of *GNSS* input,
  - the toll-gate sensor equipment,
  - the toll-gate barrier equipment,
  - the toll-gate handling of entry, vehicle identification and exit sensors and the forwarding of vehicle identification to the road pricing calculator, and
  - the communications between toll-gates and vehicles, on "one side", and the road pricing calculator, on the "other side".

- It is in this sense that the requirements are for an information technology-based system
  - of both software and
  - hardware —
    - not just hard computer and communications equipment,
    - but also movement sensors
    - and electro-mechanical "gear" ■

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

3.1.3. User and External Equipment Requirements

Definition 24: **User and External Equipment Requirements:** By a **user and external equipment requirements narrative** we understand

- a narrative which emphasises assumptions about
  - the human user and
  - external equipment

  interfaces
- to the system components ■
- The user and external equipment requirements
  - detail, and thus make explicit
  - the assumptions listed in Example 60.

Example 61 . The Road-pricing User and External Equipment Requirements: Narrative:

- The human users of the road-pricing system are:
  - *vehicle drivers,*
  - toll-gate sensor, actuator and barrier *service staff,* and
  - the road-pricing calculator *service staff.*
- The external equipment are:
  - firstly, the *GNSS* satellites and the telecommunications equipment which enables *communication* between
    - the *GNSS* satellites and vehicles,
    - vehicles and the road-pricing calculator and
    - toll-gates and the road-pricing calculator.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

- Moreover, the external *equipment* are
  - the toll-gates with their sensors:
    - entry,
    - vehicle identity, and
    - exit,

    and the barrier actuator.
- The external *equipment* are, finally, the vehicles ! ■

- That is,
  - although we do indeed exemplify domain and requirements aspects of users and external equipment,
  - we do not expect to machine, i.e., to hardware or software design these elements;
  - *they are assumed already implemented !*

3.1.4. Design Requirements

Definition 25 Assumption and Design Requirements:

- *By* **assumption and design requirements** *we understand precise prescriptions of*
  - *the endurants*
  - *and perdurants*

  of the (to be designed) system components
- *and the assumptions which that design must rely upon* ■

- The **design** will be done, extensively, in the examples of Sects. 5.–6.
- The **assumptions** upon which the design can be relied upon, that is, shall be verified ("against") are illustrated in Sect. 4.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

3.2. The Three Phases of Requirements Engineering

- There are, as we see it, three kinds of design assumptions and requirements:
  - domain requirements (the primary design assumptions) ,
  - interface requirements and
  - machine requirements.
  - The last two being the primary design requirements.

- The **domain requirements** are those requirements which can be expressed sôlely using technical terms of the domain ■
- The **interface requirements** are those requirements which can be expressed only using technical terms of both the domain and the machine ■
- The **machine requirements** are those requirements which, in principle, can be expressed sôlely using technical terms of the machine ■

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

Definition 26 Verification Paradigm:

- Some preliminary designations:
  - let $\mathcal{D}$ designate the the domain requirements;
  - let $\mathcal{R}$ designate the interface and machine requirements and
  - let $\mathcal{S}$ designate the system design.
- Now $\mathcal{D}, \mathcal{S} \models \mathcal{R}$ shall be read:
  - it must be verified that the $\mathcal{S}$ystem design
  - satisfies the interface and machine $\mathcal{R}$equirements
  - in the context of the $\mathcal{D}$omain requirements ■

- The "in the context of $\mathcal{D}$..." term means that
  - proofs of $\mathcal{S}$oftware design correctness
  - with respect to $\mathcal{R}$equirements
  - will often have to refer to $\mathcal{D}$omain requirements assumptions.
- We refer to [GGJZ00, Gunter, Jackson, Zave, 2000] for an analysis of a varieties of forms in which $\models$ relate to variants of $\mathcal{D}$, $\mathcal{R}$ and $\mathcal{S}$.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

3.3. Order of Presentation of Requirements Prescriptions

- The **domain requirements development** stage — as we shall see — can be sub-staged into:
  - projection,
  - instantiation,
  - determination,
  - extension and
  - fitting.
- The **interface requirements development** stage — can be sub-staged into *shared:*
  - *endurant,*
  - *action,*
  - *event and*
  - *behaviour*

  developments, where "sharedness" pertains to phenomena shared between, i.e., "present" in, both the domain (concretely, manifestly) and the machine (abstractly, conceptually).

- These development stages need not be pursued in the order of the three stages and their sub-stages.
- We emphasize that
  - one thing is the stages and steps of development, as for example these:
    - projection, instantiation, determination, extension, fitting,
    - shared endurants, shared actions, shared events, shared behaviours,
    - etcetera,
  - another thing is the requirements prescription that results from these development stages and steps.
    - The further software development,
    - after and on the basis of the requirements prescription
    - starts only when all stages and steps of the requirements prescription have been fully developed.

- The domain engineer is now free to rearrange the final prescription,
  - irrespective of the order in which the various sections were developed,
  - in such a way as to give a most
    - pleasing,
    - pedagogic and
    - cohesive

    reading (i.e., presentation).
- From such a requirements prescription one can therefore not necessarily see in which order the various sections of the prescription were developed.

3.4. Design Requirements and Design Assumptions

- A crucial distinction is between **design requirements** and **design assumptions**.
  - The **design requirements**
    - are those requirements for which
    - the system designer **has to** implement
    - hardware or software
    - in order satisfy system user expectations ■

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

- The **design assumptions**
  - are those requirements for which
  - the system designer
    does not have to implement hardware or software,
  - but whose properties
  - the designed hardware, respectively software relies on
    for proper functioning ∎

Example 62 . Road Pricing System — Design Requirements:

- The design requirements for the road pricing calculator of these lectures are for the design of:
  - for that part of the vehicle software which interfaces the GNSS receiver and the road pricing calculator (cf. Items 210–213),
  - for that part of the toll-gate software which interfaces the toll-gate and the road pricing calculator (cf. Items 218–220) and
  - the road pricing calculator (cf. Items 249–262) ■

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

Example 63 . Road Pricing System — Design Assumptions:

- The design assumptions for the road pricing calculator include:
  - that *vehicles* behave as prescribed in Items 209–213,
  - that the `GNSS` regularly offers vehicles correct information as to their global position (cf. Item 210),
  - that *toll-gates* behave as prescribed in Items 215–220, and
  - that the *road net* is formed and well-formed as defined in Examples 68–70 ■

Example 64 . Toll-Gate System — Design Requirements:

- The design requirements for the toll-gate system of these lectures are for the design of
  - software for the toll-gate
  - and its interfaces to the road pricing system,
  - i.e., Items 214–215 ■

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

Example 65 . Toll-Gate System — Design Assumptions:

- The design assumptions for the toll-gate system include
  - that the vehicles behave as per Items 209–213, and
  - that the road pricing calculator behave as per Items 249–262

4. Domain Requirements

- Domain requirements express the assumptions
  - that a design must rely upon
  - in order that that design can be verified.

Definition 27 Domain Requirements Prescription: *A* **domain requirements prescription**

- *is that subset of the requirements prescription*
- *which can be expressed sôlely using terms from the domain description* ■

- To determine a relevant subset all we need is collaboration with requirements, cum domain stake-holders.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

- Experimental evidence,
  - in the form of example developments
    - of requirements prescriptions
    - from domain descriptions,

    appears to show
  - that one can formulate techniques for such developments
  - around a few domain-description-to-requirements-prescription operations.
  - We suggest these:
    - projection,
    - instantiation,
    - determination,
    - extension and
    - fitting.

- In Sect. 3.3
  - we mentioned that the order in which one performs
  - these description-to-prescription operations
  - is not necessarily the order in which we have listed them here,
  - but, with notable exceptions, one is well-served in starting out requirements development
  - by following this order.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

4.1. Domain Projection & Simplification

Definition 28 Domain Projection: *By a* **domain projection & simplification** *we mean*

- *a subset of the domain description,*
- *one which* projects out *all those*
  - *endurants:*
    - *parts,*
    - *materials and*
    - *components,*

    as well as
  - *perdurants:*
    - *actions,*
    - *events and*
    - *behaviours*

  that the stake-holders do not wish represented or relied upon by the machine.
- *Simplification means that we simplify (refine) some internal qualities, like mereologies and/or attributes* ■

- The resulting document is a partial domain requirements prescription.
- In determining an appropriate subset
  - the requirements engineer must secure
  - that the final "projection prescription"
  - is complete and consistent — that is,
    - that there are no "dangling references",
    - i.e., that all entities
    - and their internal properties
    - that are referred to
    - are all properly defined.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

4.1.1. Domain Projection — Narrative

- We now start on a series of examples
- that illustrate domain requirements development.

Example 66 . Domain Requirements. Projection: A Narrative Sketch:

- We require that the road pricing system shall [at most] relate to the following domain entities – and only to these[39]:
  - the net,
    - its links and hubs,
    - and their properties (unique identifiers, mereologies and some attributes),
  - the vehicles, as endurants, and
  - the general vehicle behaviours, as perdurants.

[39] By 'relate to ... these' we mean that the required system does not rely on domain phenomena that have been "projected away".

- We treat projection together with a concept of simplification.
- The example simplifications are
  - vehicle positions and,
  - related to the simpler vehicle position,
  - vehicle behaviours.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

- To prescribe and formalise this we copy the domain description.
- From that domain description we remove all mention of
  - the hub insertion action,
  - the link disappearance event, and
  - the monitor ■
- As a result we obtain $\Delta_{\mathcal{P}}$, the projected version of the domain requirements prescription[40].

[40] Restrictions of the net to the toll road nets, hinted at earlier, will follow in the next domain requirements steps.

4.1.2. Domain Projection — Formalisation

- The requirements prescription hinges, crucially,
  - not only on a systematic narrative of all the
    - projected,
    - instantiated,
    - determinated,
    - extended and
    - fitted

    specifications,
  - but also on their formalisation.
- In the formal domain projection example we, regretfully, omit the narrative texts.
  - In bringing the formal texts we keep the item numbering from Sect. 2.,
  - where you can find the associated narrative texts.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

Example 67 . Domain Requirements — Projection:

Main Sorts

type
83 $\Delta_{\mathcal{P}}$
83a. $N_{\mathcal{P}}$
83b. $F_{\mathcal{P}}$
value
83a. **obs_part**_$N_{\mathcal{P}}$: $\Delta_{\mathcal{P}} \to N_{\mathcal{P}}$
83b. **obs_part**_$F_{\mathcal{P}}$: $\Delta_{\mathcal{P}} \to F_{\mathcal{P}}$
type
84a. $HA_{\mathcal{P}}$
84b. $LA_{\mathcal{P}}$
value
84a. **obs_part**_HA: $N_{\mathcal{P}} \to$ HA
84b. **obs_part**_LA: $N_{\mathcal{P}} \to$ LA

Concrete Types

type

85 $\mathsf{H}_{\mathcal{P}}$, $\mathsf{HS}_{\mathcal{P}} = \mathsf{H}_{\mathcal{P}}$**-set**

86 $\mathsf{L}_{\mathcal{P}}$, $\mathsf{LS}_{\mathcal{P}} = \mathsf{L}_{\mathcal{P}}$**-set**

87 $\mathsf{V}_{\mathcal{P}}$, $\mathsf{VS}_{\mathcal{P}} = \mathsf{V}_{\mathcal{P}}$**-set**

value

85 **obs_part**_$\mathsf{HS}_{\mathcal{P}}$: $\mathsf{HA}_{\mathcal{P}} \rightarrow \mathsf{HS}_{\mathcal{P}}$

86 **obs_part**_$\mathsf{LS}_{\mathcal{P}}$: $\mathsf{LA}_{\mathcal{P}} \rightarrow \mathsf{LS}_{\mathcal{P}}$

87 **obs_part**_$\mathsf{VS}_{\mathcal{P}}$: $\mathsf{F}_{\mathcal{P}} \rightarrow \mathsf{VS}_{\mathcal{P}}$

88a. links: $\Delta_{\mathcal{P}} \rightarrow$ L**-set**

88a. links($\delta_{\mathcal{P}}$) $\equiv$ **obs_part**_$\mathsf{LS}_{\mathcal{R}}$(**obs_part**_$\mathsf{LA}_{\mathcal{R}}$($\delta_{\mathcal{R}}$))

88b. hubs: $\Delta_{\mathcal{P}} \rightarrow$ H**-set**

88b. hubs($\delta_{\mathcal{P}}$) $\equiv$ **obs_part**_$\mathsf{HS}_{\mathcal{P}}$(**obs_part**_$\mathsf{HA}_{\mathcal{P}}$($\delta_{\mathcal{P}}$))

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

Unique Identifiers

type
89a. HI, LI, VI, MI
value
89c. **uid**_HI: $H_{\mathcal{P}} \to$ HI
89c. **uid**_LI: $L_{\mathcal{P}} \to$ LI
89c. **uid**_VI: $V_{\mathcal{P}} \to$ VI
89c. **uid**_MI: $M_{\mathcal{P}} \to$ MI
axiom
89b. $HI \bigcap LI = \emptyset$, $HI \bigcap VI = \emptyset$, $HI \bigcap MI = \emptyset$,
89b. $LI \bigcap VI = \emptyset$, $LI \bigcap MI = \emptyset$, $VI \bigcap MI = \emptyset$

Mereology

```
value
94   obs_mereo_H𝒫: H𝒫 → LI-set
95   obs_mereo_L𝒫: L𝒫 → HI-set
95          axiom ∀ l:L𝒫 • card obs_mereo_L𝒫(l)=2
96   obs_mereo_V𝒫: V𝒫 → MI
97   obs_mereo_M𝒫: M𝒫 → VI-set
axiom
98   ∀ δ𝒫:Δ𝒫, hs:HS•hs=hubs(δ), ls:LS•ls=links(δ𝒫) ⇒
98       ∀ h:H𝒫•h ∈ hs ⇒
98           obs_mereo_H𝒫(h)⊆xtr_his(δ𝒫) ∧
99       ∀ l:L𝒫•l ∈ ls •
98           obs_mereo_L𝒫(l)⊆xtr_lis(δ𝒫) ∧
100a.    let f:F𝒫•f=obs_part_F𝒫(δ𝒫) ⇒
100a.            vs:VS𝒫•vs=obs_part_VS𝒫(f) in
100a.        ∀ v:V𝒫•v ∈ vs ⇒
100a.            uid_V𝒫(v) ∈ obs_mereo_M𝒫(m) ∧
100b.        obs_mereo_M𝒫(m)
100b.            = {uid_V𝒫(v)|v:V•v ∈ vs}
100b.    end
```


© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

Attributes: We project attributes of hubs, links and vehicles.
First **hubs:**

type
101a. GeoH
101b. $H\Sigma_{\mathcal{P}} = (LI \times LI)$**-set**t
101c. $H\Omega_{\mathcal{P}} = H\Sigma_{\mathcal{P}}$**-set**
value
101b. **attr**_$H\Sigma_{\mathcal{P}}$: $H_{\mathcal{P}} \rightarrow H\Sigma_{\mathcal{P}}$
101c. **attr**_$H\Omega_{\mathcal{P}}$: $H_{\mathcal{P}} \rightarrow H\Omega_{\mathcal{P}}$
axiom
102 $\forall\ \delta_{\mathcal{P}}:\Delta_{\mathcal{P}}$,
102 **let** hs = hubs($\delta_{\mathcal{P}}$) **in**
102 $\forall$ h:$H_{\mathcal{P}}$ · h $\in$ hs ·
102a. xtr_lis(h)$\subseteq$xtr_lis($\delta_{\mathcal{P}}$)
102b. $\wedge$ **attr**_$\Sigma_{\mathcal{P}}$(h) $\in$ **attr**_$\Omega_{\mathcal{P}}$(h)
102 **end**

Then **links:**

type
105 GeoL
106a. $\mathsf{L}\Sigma_{\mathcal{P}} = (\mathsf{HI}\times\mathsf{HI})$**-set**
106b. $\mathsf{L}\Omega_{\mathcal{P}} = \mathsf{L}\Sigma_{\mathcal{P}}$**-set**
value
105 **attr**_GeoL: L $\to$ GeoL
106a. **attr**_$\mathsf{L}\Sigma_{\mathcal{P}}$: $\mathsf{L}_{\mathcal{P}} \to \mathsf{L}\Sigma_{\mathcal{P}}$
106b. **attr**_$\mathsf{L}\Omega_{\mathcal{P}}$: $\mathsf{L}_{\mathcal{P}} \to \mathsf{L}\Omega_{\mathcal{P}}$
axiom
106a.– 106b. on Slide 417.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

Finally **vehicles:**

- For 'road pricing' we need vehicle positions.
  - But, for "technical reasons",
  - we must abstain from the detailed description
  - given in Items 107–107c.
  - The 'technical reasons' are that we assume that the *GNSS* cannot provide us with direction of vehicle movement
  - and therefore we cannot, using only the *GNSS* provide the details of 'offset' along a link (*onL*) nor the "from/to link" at a hub (*atH*).
- We therefore simplify vehicle positions.

133 A simplified vehicle position designates

a. either a link

b. or a hub,

type

133 SVPos = SonL | SatH

133a. SonL :: LI

133b. SatH :: HI

axiom

107a.' ∀ n:N, SonL(li):SVPos ·

107a.' ∃ l:L·l ∈**obs_part**_LS(**obs_part**_N(n)) ⇒ li=**uid**_L(l)

107c.' ∀ n:N, SatH(hi):SVPos ·

107c.' ∃ h:H·h ∈**obs_part**_HS(**obs_part**_N(n)) ⇒ hi=**uid**_H(h)

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

Global Values

value

117 $\delta_{\mathcal{P}}$:$\Delta_{\mathcal{P}}$,

118 n:$N_{\mathcal{P}}$ = **obs_part**_$N_{\mathcal{P}}(\delta_{\mathcal{P}})$,

118 ls:$L_{\mathcal{P}}$-**set** = links$(\delta_{\mathcal{P}})$,

118 hs:$H_{\mathcal{P}}$-**set** = hubs$(\delta_{\mathcal{P}})$,

118 lis:LI-**set** = xtr_lis$(\delta_{\mathcal{P}})$,

118 his:HI-**set** = xtr_his$(\delta_{\mathcal{P}})$

Behaviour Signatures: We omit the monitor behaviour.

134 We leave the vehicle behaviours' attribute argument undefined.

type
134 ATTR
value
124 $\mathsf{trs}_{\mathcal{P}}$: **Unit** $\rightarrow$ **Unit**
125 $\mathsf{veh}_{\mathcal{P}}$: $\mathsf{VI}\times\mathsf{MI}\times\mathsf{ATTR} \rightarrow$ ... **Unit**

The System Behaviour: We omit the monitor behaviour.

value
127a. $\mathsf{trs}_{\mathcal{P}}()=\|\{\mathsf{veh}_{\mathcal{P}}(\textbf{uid_}\mathsf{VI}(v),\textbf{obs_mereo_}\mathsf{V}(v),_) \mid v{:}\mathsf{V}_{\mathcal{P}}\cdot v \in vs\}$

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

The Vehicle Behaviour:

- Given the simplification of vehicle positions
- we *simplify* the vehicle behaviour given in Items 128–129

```
128'      veh(vi,mi)(vp:SatH(hi)) ≡
128a.'          v_m_ch[vi,mi]!SatH(hi) ; veh(vi,mi)(SatH(hi))
128(b.)i'     ⌈⌉ let li:LI•li ∈ obs_mereo_H(get_hub(hi)(n)) in
128(b.)ii'         v_m_ch[vi,mi]!SonL(li) ; veh(vi,mi)(SonL(li)) end
128c.'        ⌈⌉ stop

129'      veh(vi,mi)(vp:SonL(li)) ≡
129a.'          v_m_ch[vi,mi]!SonL(li) ; veh(vi,mi,va)(SonL(li))
129(b.)iiA'   ⌈⌉ let hi:HI•hi ∈ obs_mereo_L(get_link(li)(n)) in
129(b.)iiB'        v_m_ch[vi,mi]!SatH(hi) ; veh(vi,mi)(atH(hi)) end
129c.'        ⌈⌉ stop
```

- We can simplify Items 128′–129c.′ further.

```
135  veh(vi,mi)(vp) ≡
136        v_m_ch[vi,mi]!vp ; veh(vi,mi,va)(vp)
137      ⌈⌉ case vp of
137          SatH(hi) →
138             let li:LI·li ∈ obs_mereo_H(get_hub(hi)(n)) in
139             v_m_ch[vi,mi]!SonL(li) ; veh(vi,mi)(SonL(li)) end,
137          SonL(li) →
140             let hi:HI·hi ∈ obs_mereo_L(get_link(li)(n)) in
141             v_m_ch[vi,mi]!SatH(hi) ; veh(vi,mi)(atH(hi)) end end
142       ⌈⌉ stop
```


© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

135 This line coalesces Items 128′ and 129′.

136 Coalescing Items 128a.′ and 129′.

137 Captures the distinct parameters of Items 128′ and 129′.

138 Item 128(b.)i′.

139 Item 128(b.)ii′.

140 Item 129(b.)iiA′.

141 Item 129(b.)iiB′.

142 Coalescing Items 128c.′ and 129c.′.

- The above **veh**icle behaviour definition
  - will be transformed (i.e., further "refined")
  - in Sect. 5.4's Example 76;
  - cf. Items 209– 213 on Slide 568 ■

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

4.2. Domain Instantiation

Definition 29 Instantiation: *By* **domain instantiation** *we mean*

- *a* ***refinement*** *of the partial domain requirements prescription*
- *(resulting from the projection step)*
- *in which the refinements aim at rendering the*
  - *endurants:*
    - *parts,*
    - *materials and*
    - *components,*

    as well as the
  - *perdurants:*
    - *actions,*
    - *events and*
    - *behaviours*

  of the domain requirements prescription
- *more concrete, more specific*

- Properties that hold of the projected domain shall also hold of the (therefrom) instantiated domain.
- Refinement of endurants can be expressed
  - either in the form of concrete types,
  - or of further "delineating" axioms over sorts,
  - or of a combination of concretisation and axioms.
- We shall exemplify the third possibility.
- Example 68 express requirements that the road net
  - (on which the road-pricing system is to be based)

  must satisfy.
- Refinement of perdurants will not be illustrated (other than the simplification of the *vehicle* projected behaviour).

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

4.2.1. Domain Instantiation

Example 68 . Domain Requirements. Instantiation Road Net:

- We now require that there is, as before, a road net, $n_{\mathcal{I}}$:$N_{\mathcal{I}}$, which can be understood as consisting of two, "connected sub-nets".
  - A toll-road net, $trn_{\mathcal{I}}$:$TRN_{\mathcal{I}}$, cf. Fig. 5 on the facing slide,
  - and an ordinary road net, $n_{\mathcal{P}'}$.
  - The two are connected as follows:
    - The toll-road net, $trn_{\mathcal{I}}$, borders some toll-road plazas, in Fig. 5 on the next slide shown by white filled circles.
    - These toll-road plaza hubs are proper hubs of the 'ordinary' road net, $n'_{\mathcal{P}}$.

.

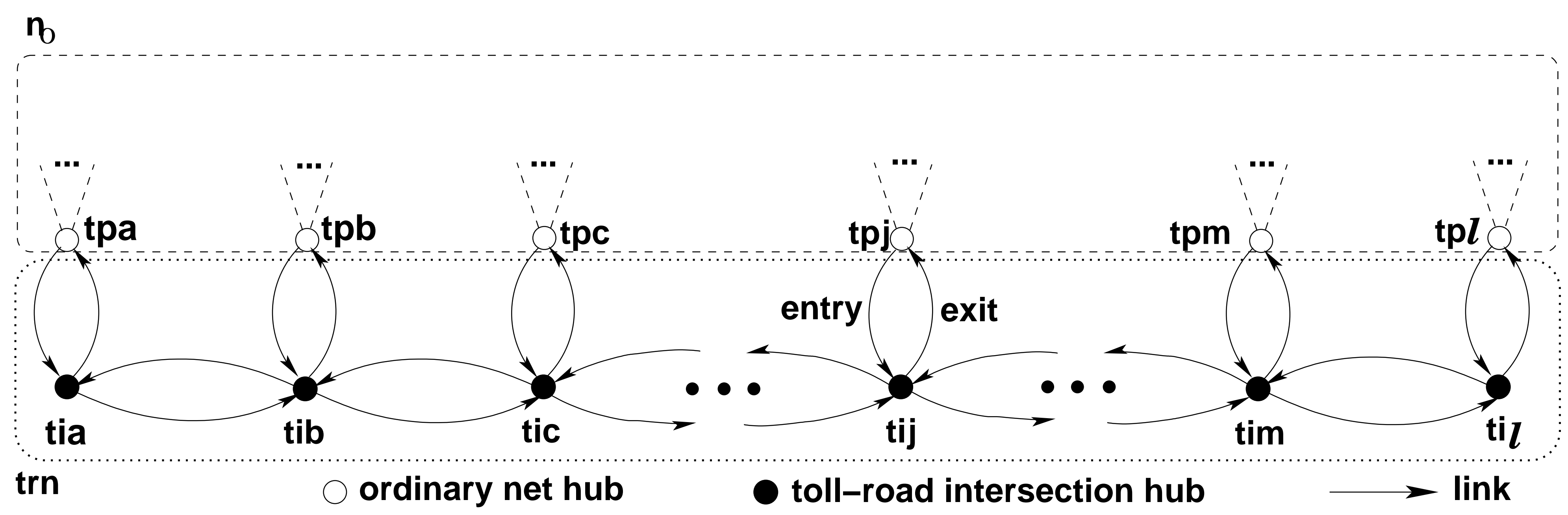


Figure 5: A simple, linear toll-road net

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

143 The instantiated domain, $\delta_{\mathcal{I}}$:$\Delta_{\mathcal{I}}$ has just the net, $\mathsf{n}_{\mathcal{I}}$:$\mathsf{N}_{\mathcal{I}}$ being instantiated.

144 The road net consists of two "sub-nets"

a. an "ordinary" road net, n_{O}:$\mathsf{N}_{\mathcal{P}'}$ and

b. a toll-road net proper, trn:$\mathsf{TRN}_{\mathcal{I}}$ —

c. “connected” by an interface hil:HIL:

i That interface consists of a number of toll-road plazas (i.e., hubs), modeled as a list of hub identifiers, hil:HI*.

ii The toll-road plaza interface to the toll-road net, trn:$\text{TRN}_{\mathcal{I}}$[41], has each plaza, hil[i], connected to a pair of toll-road links: an entry and an exit link: $(l_e{:}L, l_x{:}L)$.

iii The toll-road plaza interface to the ‘ordinary’ net, n_o:$\text{N}_{\mathcal{P}'}$, has each plaza, i.e., the hub designated by the hub identifier hil[i], connected to one or more ordinary net links, $\{l_{i_1}, l_{i_2}, \cdots, l_{i_k}\}$.

[41] We (sometimes) omit the subscript $_{\mathcal{I}}$ when it should be clear from the context what we mean.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

144b. The toll-road net, trn:TRN$_{\mathcal{I}}$, consists of three collections (modeled as lists) of links and hubs:

i a list of pairs of toll-road entry/exit links: $\langle(l_{e_1}, l_{x_1}), \cdots, (l_{e_\ell}, l_{x_\ell})\rangle$,

ii a list of toll-road intersection hubs: $\langle h_{i_1}, h_{i_2}, \cdots, h_{i_\ell}\rangle$, and

iii a list of pairs of main toll-road ("up" and "$d$own") links: $\langle(ml_{i_{1u}}$,-$ml_{i_{1d}}), (m_{i_{2u}}, m_{i_{2d}}), \cdots, (m_{i_{\ell u}}, m_{i_{\ell d}})\rangle$.

d. The three lists have commensurate lengths (ℓ).

type
143 $\Delta_{\mathcal{I}}$
144 $\mathsf{N}_{\mathcal{I}} = \mathsf{N}_{\mathcal{P}'} \times \mathsf{HIL} \times \mathsf{TRN}$
144a. $\mathsf{N}_{\mathcal{P}'}$
144b. $\mathsf{TRN}_{\mathcal{I}} = (\mathsf{L}\times\mathsf{L})^*\times\mathsf{H}^*\times(\mathsf{L}\times\mathsf{L})^*$
144c. $\mathsf{HIL} = \mathsf{HI}^*$

axiom
144d. $\forall\ \mathsf{n}_{\mathcal{I}}:\mathsf{N}_{\mathcal{I}}\ \cdot$
144d. **let** $(\mathsf{n}_{\Delta},\mathsf{hil},(\mathsf{exll},\mathsf{hl},\mathsf{lll})) = \mathsf{n}_{\mathcal{I}}$ **in**
144d. **len** hil = **len** exll = **len** hl = **len** lll + 1
144d. **end**

[Lecturer explains $\mathsf{N}_{\mathcal{P}'}$]

- The partial concretisation of the net sorts, $\mathsf{N}_{\mathcal{P}}$, into $\mathsf{N}_{\mathcal{I}}$ requires some additional well-formedness conditions to be satisfied.

145 The toll-road intersection hubs all[42] have distinct identifiers.

145 wf_dist_toll_road_isect_hub_ids: $\mathsf{H}^*\rightarrow$**Bool**
145 wf_dist_toll_road_isect_hub_ids(hl) $\equiv$
145 **len** hl = **card** xtr_his(hl)

[42] A 'must' can be inserted in front of all 'all's,

146 The toll-road links all have distinct identifiers.

```
146  wf_dist_toll_road_u_d_link_ids: (L×L)*→Bool
146  wf_dist_toll_road_u_d_link_ids(lll) ≡
146       2 × len lll = card xtr_lis(lll)
```

147 The toll-road entry/exit links all have distinct identifiers.

```
147  wf_dist_e_x_link_ids: (L×L)*→Bool
147  wf_dist_e_x_link_ids(exll) ≡
147       2 × len exll = card xtr_lis(exll)
```

148 Proper net links must not designate toll-road intersection hubs.

```
148  wf_isoltd_toll_road_isect_hubs: HI*×H*→N_I→Bool
148  wf_isoltd_toll_road_isect_hubs(hil,hl)(n_I) ≡
148       let ls=xtr_links(n_I) in
148       let his = ∪ {obs_mereo_L(l)|l:L·l ∈ ls} in
148       his ∩ xtr_his(hl) = {} end end
```

149 The plaza hub identifiers must designate hubs of the 'ordinary' net.

```
149 wf_p_hubs_pt_of_ord_net: HI*→N'_Δ→Bool
149 wf_p_hubs_pt_of_ord_net(hil)(n'_Δ) ≡
149     elems hil ⊆ xtr_his(n'_Δ)
```

150 The plaza hub mereologies must each,

a. besides identifying at least one hub of the ordinary net,

b. also identify the two entry/exit links with which they are supposed to be connected.

```
150 wf_p_hub_interf: N'_Δ→Bool
150 wf_p_hub_interf(n_o,hil,(exll,_,_)) ≡
150     ∀ i:Nat • i ∈ inds exll ⇒
150         let h = get_H(hil(i))(n'_Δ) in
150         let lis = obs_mereo_H(h) in
150         let lis' = lis \ xtr_lis(n') in
150         lis' = xtr_lis(exll(i)) end end end
```


© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

151 The mereology of each toll-road intersection hub must identify

a. the entry/exit links

b. and exactly the toll-road ‘up’ and ‘down’ links

c. with which they are supposed to be connected.

```
151    wf_toll_road_isect_hub_iface: N_I→Bool
151    wf_toll_road_isect_hub_iface(_,_,(exll,hl,lll)) ≡
151        ∀ i:Nat • i ∈ inds hl ⇒
151            obs_mereo_H(hl(i)) =
151a.            xtr_lis(exll(i)) ∪
151            case i of
151b.                1 → xtr_lis(lll(1)),
151b.                len hl → xtr_lis(lll(len hl−1))
151b.                _ → xtr_lis(lll(i)) ∪ xtr_lis(lll(i−1))
151            end
```

152 The mereology of the entry/exit links must identify exactly the

a. interface hubs and the

b. toll-road intersection hubs

c. with which they are supposed to be connected.

```
152  wf_exll: (L×L)*×HI*×H*→Bool
152  wf_exll(exll,hil,hl) ≡
152      ∀ i:Nat • i ∈ len exll
152          let (hi,(el,xl),h) = (hil(i),exll(i),hl(i)) in
152          obs_mereo_L(el) = obs_mereo_L(xl)
152          = {hi} ∪ {uid_H(h)} end
152      pre: len eell = len hil = len hl
```


© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

153 The mereology of the toll-road ‘up’ and ‘down’ links must

a. identify exactly the toll-road intersection hubs

b. with which they are supposed to be connected.

```
153   wf_u_d_links: (L×L)*×H*→Bool
153   wf_u_d_links(lll,hl) ≡
153       ∀ i:Nat · i ∈ inds lll ⇒
153           let (ul,dl) = lll(i) in
153           obs_mereo_L(ul) = obs_mereo_L(dl) =
153a.         uid_H(hl(i)) ∪ uid_H(hl(i+1)) end
153       pre: len lll = len hl+1
```

- We have used some additional auxiliary functions:

```
xtr_his: H*→HI-set
xtr_his(hl) ≡ {uid_HI(h)|h:H•h ∈ elems hl}
xtr_lis: (L×L)→LI-set
xtr_lis(l′,l″) ≡ {uid_LI(l′)}∪{uid_LI(l″)}
xtr_lis: (L×L)*− LI-set
xtr_lis(lll) ≡
 ∪{xtr_lis(l′,l″)|(l′,l″):(L×L)•(l′,l″)∈ elems lll}
```

 © Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

154 The well-formedness of instantiated nets is now the conjunction of the individual well-formedness predicates above.

```
154 wf_instantiated_net: N_I → Bool
154 wf_instantiated_net(n′_Δ,hil,(exll,hl,lll))
145     wf_dist_toll_road_isect_hub_ids(hl)
146   ∧ wf_dist_toll_road_u_d_link_ids(lll)
147   ∧ wf_dist_e_e_link_ids(exll)
148   ∧ wf_isolated_toll_road_isect_hubs(hil,hl)(n′)
149   ∧ wf_p_hubs_pt_of_ord_net(hil)(n′)
150   ∧ wf_p_hub_interf(n′_Δ,hil,(exll,_,_))
151   ∧ wf_toll_road_isect_hub_iface(_,_,(exll,hl,lll))
152   ∧ wf_exll(exll,hil,hl)
153   ∧ wf_u_d_links(lll,hl)
```

4.2.2. Domain Instantiation — Abstraction

Example 69 . Domain Requirements. Instantiation Road Net, Abstraction:

- Domain instantiation has refined
  - an abstract definition of net sorts, $n_{\mathcal{P}}$:$N_{\mathcal{P}}$,
  - into a partially concrete definition of nets, $n_{\mathcal{I}}$:$N_{\mathcal{I}}$.
- We need to show the refinement relation:
  - abstraction($n_{\mathcal{I}}$) $=$ $n_{\mathcal{P}}$.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

value

155 abstraction: $N_{\mathcal{I}} \rightarrow N_{\mathcal{P}}$
156 abstraction(n'_{Δ},hil,(exll,hl,lll)) $\equiv$
157 **let** $n_{\mathcal{P}}$:$N_{\mathcal{P}}$ •
157 **let** hs = **obs_part**_HS$_{\mathcal{P}}$(**obs_part**_HA$_{\mathcal{P}}$($n'_{\mathcal{P}}$)),
157 ls = **obs_part**_LS$_{\mathcal{P}}$(**obs_part**_LA$_{\mathcal{P}}$($n'_{\mathcal{P}}$)),
157 ths = **elems** hl,
157 eells = xtr_links(eell), llls = xtr_links(lll) **in**
158 hs∪ths=**obs_part**_HS$_{\mathcal{P}}$(**obs_part**_HA$_{\mathcal{P}}$($n_{\mathcal{P}}$))
159 ∧ ls∪eells∪llls=**obs_part**_LS$_{\mathcal{P}}$(**obs_part**_LA$_{\mathcal{P}}$($n_{\mathcal{P}}$))
160 $n_{\mathcal{P}}$ **end end**

155 The abstraction function takes a concrete net, $\mathsf{n}_{\mathcal{I}}$:$\mathsf{N}_{\mathcal{I}}$, and yields an abstract net, $\mathsf{n}_{\mathcal{P}}$:$\mathsf{N}_{\mathcal{P}}$.

156 The abstraction function doubly decomposes its argument into constituent lists and sub-lists.

157 There is postulated an abstract net, $\mathsf{n}_{\mathcal{P}}$:$\mathsf{N}_{\mathcal{P}}$, such that

158 the hubs of the concrete net and toll-road equals those of the abstract net, and

159 the links of the concrete net and toll-road equals those of the abstract net.

160 And that abstract net, $\mathsf{n}_{\mathcal{P}}$:$\mathsf{N}_{\mathcal{P}}$, is postulated to be an abstraction of the concrete net.

4.3. Domain Determination

Definition 30 Determination: *By* **domain determination** *we mean*

- *a refinement of the partial domain requirements prescription,*
- *resulting from the instantiation step,*
- *in which the refinements aim at rendering the*
  - *endurants:*
    - *parts,*
    - *materials and*
    - *components, as well as the*
  - *perdurants:*
    - *functions,*
    - *events and*
    - *behaviours*

  of the partial domain requirements prescription
- *less non-determinate, more determinate* ■

- Determinations usually render these concepts less general.
  - That is, the value space
    - of endurants that are made more determinate
    - is "smaller", contains fewer values,
    - as compared to the endurants
      before determination has been "applied".

4.3.1. Domain Determination: Example

- We show an example of 'domain determination'.
  - It is expressed sôlely in terms of
  - axioms over the concrete toll-road net type.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

Example 70 . **Domain Requirements. Determination Toll-roads**:

- We focus only on the toll-road net.
- We single out only two 'determinations':

All Toll-road Links are One-way Links

161 *The entry/exit and toll-road links*

a. are always all one way links,

b. as indicated by the arrows of Fig. 2,

c. such that each pair allows traffic in opposite directions.

161 opposite_traffics: $(L \times L)^* \times (L \times L)^* \to$ **Bool**
161 opposite_traffics(exll,lll) $\equiv$
161 $\forall$ (lt,lf):$(L \times L)$ $\cdot$ (lt,lf) $\in$ **elems** exll$\widehat{\ }$lll $\Rightarrow$
161a. **let** (ltσ,lfσ) = (**attr**_LΣ(lt),**attr**_LΣ(lf)) **in**
161a.$'$. **attr**_LΩ(lt)=$\{$lt$\sigma\}$$\wedge$**attr**_L$\Omega$(ft)=$\{ft\sigma\}$
161a.$''$. $\wedge$ **card** ltσ = 1 = **card** lfσ
161 $\wedge$ **let** ($\{$(hi,hi$'$)$\}$,$\{$(hi$''$,hi$'''$)$\}$) = (ltσ,lfσ) **in**
161c. hi=hi$'''$ $\wedge$ hi$'$=hi$''$
161 **end end**

All Toll-road Hubs are Free-flow

162 *The hub state spaces* are singleton sets of the toll-road hub states which always allow exactly these (and only these) crossings:

a. from $e$ntry links back to the paired e$x$it $l$ink$s$,

b. from $e$ntry links to emanating $t$oll-road $l$ink$s$,

c. from incident $t$oll-road links to e$x$it $l$ink$s$, and

d. from incident $t$oll-road link to emanating $t$oll-road $l$ink$s$.

```
162    free_flow_toll_road_hubs: (L×L)*×(L×L)*→Bool
162    free_flow_toll_road_hubs(exl,ll) ≡
162        ∀ i:Nat•i ∈ inds hl ⇒
162            attr_HΣ(hl(i)) =
162a.              hσ_ex_ls(exl(i))
162b.            ∪ hσ_et_ls(exl(i),(i,ll))
162c.            ∪ hσ_tx_ls(exl(i),(i,ll))
162d.            ∪ hσ_tt_ls(i,ll)
```

162a.: from $e$ntry links back to the paired e$x$it $l$ink$s$:

162a. hσ_ex_ls: (L$\times$L)$\rightarrow$LΣ
162a. hσ_ex_ls(e,x) $\equiv$ {(**uid**_LI(e),**uid**_LI(x))}

162b.: from *e*ntry links to emanating *t*oll-road *l*ink*s*:

```
162b.  hσ_et_ls: (L×L)×(Nat×(em:L×in:L)*)→LΣ
162b.  hσ_et_ls((e,_),(i,ll)) ≡
162b.      case i of
162b.          2          → {(uid_Ll(e),uid_Ll(em(ll(1))))},
162b.          len ll+1 → {(uid_Ll(e),uid_Ll(em(ll(len ll))))},
162b.          _          → {(uid_Ll(e),uid_Ll(em(ll(i−1)))),
162b.                          (uid_Ll(e),uid_Ll(em(ll(i))))}
162b.      end
```

- The *em* and *in* in the toll-road link list (em:L×in:L)* designate selectors for *em*anating, respectively *in*cident links.

162c.: from incident $t$oll-road links to e$x$it $l$ink$s$:

162c. hσ_tx_ls: (L×L)×(**Nat**×(em:L×in:L)*)→LΣ
162c. hσ_tx_ls((_,x),(i,ll)) ≡
162c. **case** i **of**
162c. 2 → {(**uid**_Ll(in(ll(1))),**uid**_Ll(x))},
162c. **len** ll+1 → {(**uid**_Ll(in(ll(**len** ll))),**uid**_Ll(x))},
162c. _ → {(**uid**_Ll(in(ll(i−1))),**uid**_Ll(x)),
162c. (**uid**_Ll(in(ll(i))),**uid**_Ll(x))}
162c. **end**

162d.: from incident *t*oll-road link to emanating *t*oll-road *l*ink*s*:

```
162d.  hσ_tt_ls: Nat×(em:L×in:L)*→LΣ
162d.  hσ_tt_ls(i,ll) ≡
162d.      case i of
162d.          2          → {(uid_LI(in(ll(1))),uid_LI(em(ll(1))))},
162d.          len ll+1 → {(uid_LI(in(ll(len ll))),uid_LI(em(ll(len ll))))},
162d.          _           → {(uid_LI(in(ll(i−1))),uid_LI(em(ll(i−1)))),
162d.                           (uid_LI(in(ll(i))),uid_LI(em(ll(i))))}
162d.      end
```

- The example above illustrated 'domain determination' with respect to endurants.
  - Typically "endurant determination" is expressed in terms of axioms that limit state spaces —
  - where "endurant instantiation" typically "limited" the mereology of endurants: how parts are related to one another.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

- We shall not exemplify domain determination with respect to perdurants.
  - Typically perdurants are expressed in terms of expressions and statements.
  - And, typically, perdurant non-determinism is expressed in terms of the choice or parallelism operators: $\mathsf{C}_{i_a} \sqcap \mathsf{C}_{i_b}$ or $\mathsf{C}_{x_a} \sqcup\!\!\!\sqcap \mathsf{C}_{x_b}$ or $\mathsf{C}_{p_a} \| \mathsf{C}_{p_b}$.
  - 'Perdurant determination' is then, typically, a matter of making the choice conditional (i,x) or of "sequentializing" parallelism (p).

    (i) $\mathsf{C}_{i_a} \sqcap \mathsf{C}_{i_b} \Rightarrow$ **if** B_i **then** C_{i_a} **else** C_{i_b} **end**
    (x) $\mathsf{C}_{x_a} \sqcup\!\!\!\sqcap \mathsf{C}_{x_b} \Rightarrow$ **if** B_x **then** C_{x_a} **else** C_{x_b} **end**
    (p) $\mathsf{C}_{p_a} \| \mathsf{C}_{p_b} \Rightarrow \mathsf{C}_{p_a} \mathsf{;} \mathsf{C}_{p_b}$

4.4. Domain Extension

Definition 31 Extension: *By* **domain extension** *we understand the*

- *introduction of endurants and perdurants that were not feasible in the original domain,*
- *but for which, with computing and communication,*
- *and with new, emerging technologies,*
- *for example, sensors, actuators and satellites,*
- *there is the possibility of feasible implementations,*
- *hence the requirements,*
- *that what is introduced becomes part of the unfolding requirements prescription* ■

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

- Usually extensions involving one of the main sorts entails extensions involving several of the main sorts.
- In our example we introduce (i.e., “extend”)
  - vehicles with GPSS-like sensors,
  - and introduce toll-gates with
    - entry sensors,
    - vehicle identification sensors,
    - gate actuators and
    - exit sensors.
  - Finally road pricing calculators are introduced.

4.4.1. The Requirements Example: Domain Extension

Example 71 . Domain Requirements — Extension:

- We present the extensions in several steps.
  - Some of them will be developed in this section.
  - Development of the remaining will be deferred to Sect. .
  - The reason for this deferment is that those last steps are examples of interface requirements.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

- The initial extension-development steps are:
  - [a] vehicle extension,
  - [b] sort and unique identifiers of road price calculators,
  - [c] vehicle to road pricing calculator channel,
  - [d] sorts and dynamic attributes of toll-gates,
  - [e] road pricing calculator attributes,
  - [f] “total” system state, and
  - [g] the overall system behaviour.
- This decomposition establishes system interfaces in “small, easy steps”.

4.4.1.1 [a] Vehicle Extension:

163 There is a domain, $\delta_{\mathcal{E}}$:$\Delta_{\mathcal{E}}$, which contains

164 a fleet, $\mathsf{f}_{\mathcal{E}}$:$\mathsf{F}_{\mathcal{E}}$, that is,

165 a set, $\mathsf{vs}_{\mathcal{E}}$:$\mathsf{VS}_{\mathcal{E}}$, of

166 extended vehicles, $\mathsf{v}_{\mathcal{E}}$:$\mathsf{V}_{\mathcal{E}}$ — their extension amounting to

167 a dynamic reactive attribute, whose value, ti-gpos:TiGpos, at any time, reflects that vehicle's ***time-stamped global position.***

168 The vehicle's GNSS receiver calculates, *loc_pos*, its local position, lpos:LPos, based on these signals.

169 Vehicles access these **external attribute**s via the **external attribute** channel, attr_TiGPos_ch.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

type
163 $\Delta_{\mathcal{E}}$
164 $F_{\mathcal{E}}$
165 $VS_{\mathcal{E}} = V_{\mathcal{E}}$**-set**
166 $V_{\mathcal{E}}$
167 TiGPos = $\mathbb{T} \times$ GPos
168 GPos, LPos
value
163 $\delta_{\mathcal{E}}:\Delta_{\mathcal{E}}$
164 **obs_part**_$F_{\mathcal{E}}$: $\Delta_{\mathcal{E}} \to F_{\mathcal{E}}$
164 f = **obs_part**_$F_{\mathcal{E}}(\delta_{\mathcal{E}})$
165 **obs_part**_$VS_{\mathcal{E}}$: $F_{\mathcal{E}} \to VS_{\mathcal{E}}$
165 vs = **obs_part**_$VS_{\mathcal{E}}$(f)
165 vis = xtr_vis(vs)
167 attr_TiGPos_ch[vi]?
168 loc_pos: GPos $\to$ LPos
channel
168 {attr_TiGPos_ch[vi]|vi:VI•vi $\in$ vis}:TiGPos

We define two auxiliary functions,

170 **xtr_vs**, which given a domain, or a fleet, extracts its set of vehicles, and

171 **xtr_vis** which given a set of vehicles generates their unique identifiers.

value

170 $\quad$ xtr_vs: $(\Delta_{\mathcal{E}}|\mathsf{F}_{\mathcal{E}}|\mathsf{VS}_{\mathcal{E}}) \rightarrow \mathsf{V}_{\mathcal{E}}$**-set**
170 $\quad$ xtr_vs(arg) $\equiv$
170 $\quad\quad$ **is**_$\Delta_{\mathcal{E}}$(arg) $\rightarrow$ **obs_part**_$\mathsf{VS}_{\mathcal{E}}$(**obs_part**_$\mathsf{F}_{\mathcal{E}}$(arg)),
170 $\quad\quad$ **is**_$\mathsf{F}_{\mathcal{E}}$(arg) $\rightarrow$ **obs_part**_$\mathsf{VS}_{\mathcal{E}}$(arg),
170 $\quad\quad$ **is**_$\mathsf{VS}_{\mathcal{E}}$(arg) $\rightarrow$ arg
171 $\quad$ xtr_vis: $(\Delta_{\mathcal{E}}|\mathsf{F}_{\mathcal{E}}|\mathsf{VS}_{\mathcal{E}}) \rightarrow$ VI**-set**
171 $\quad$ xtr_vis(arg) $\equiv$ {**uid**_VI(v)|v $\in$ xtr_vs(arg)}

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

4.4.1.2 [b] Road Pricing Calculator: Basic Sort and Unique Identifier:

172 The domain $\delta_{\mathcal{E}}$:$\Delta_{\mathcal{E}}$, also contains a pricing calculator, c:$\mathsf{C}_{\delta_{\mathcal{E}}}$, with unique identifier ci:CI.

```
type
172     C, CI
value
172     obs_part_C: Δ_ℰ → C
172     uid_CI: C → CI
172     c = obs_part_C(δ_ℰ)
172     ci = uid_CI(c)
```

4.4.1.3 [c] Vehicle to Road Pricing Calculator Channel:

173 Vehicles can, on their own volition, offer the timed local position, viti-lpos:VITiLPos

174 to the pricing calculator, c:$C_{\mathcal{E}}$ along a vehicles-to-calculator channel, v_c_ch.

type
173 VITiLPos = VI × ($\mathbb{T}$ × LPos)
channel
174 {v_c_ch[vi,ci]|vi:VI,ci:CI•vi∈vis∧ci=**uid**_C(c)}:VITiLPos

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

4.4.1.4 [d] Toll-gate Sorts and Dynamic Types:

- We extend the domain with toll-gates for vehicles entering and exiting the toll-road entry and exit links.
- Figure 6 illustrates the idea of gates.

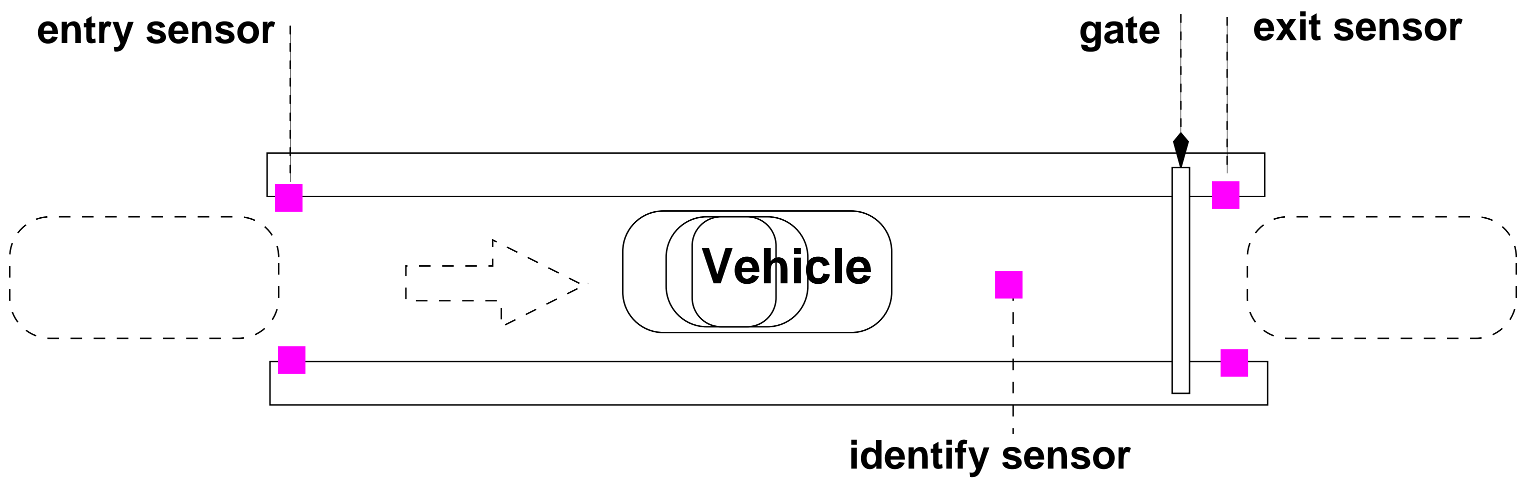


Figure 6: A toll plaza gate

- Figure 6 on the preceding slide is intended to illustrate a vehicle entering (or exiting) a toll-road entry link.
  - The toll-gate is equipped with three sensors:
    an entry sensor, a vehicle identification sensor and an exit sensor.
  - The entry sensor serves to prepare
    the vehicle identification sensor.
  - The exit sensor serves to prepare
    the gate for closing when a vehicle has passed.
  - The vehicle identify sensor identifies the vehicle and "delivers" a pair: the current time and the vehicle identifier.
  - Once the vehicle identification sensor has identified a vehicle
    - the gate opens and
    - a message is sent to the road pricing calculator as to the passing vehicle's identity and the identity of the link associated with the toll-gate (see Items 190- 191 on Slide 557).

175 The domain contains the extended net, n:$N_{\mathcal{E}}$,

176 with the net extension amounting to the toll-road net, $TRN_{\mathcal{E}}$, that is, the instantiated toll-road net, trn:$TRN_{\mathcal{I}}$, is extended, into trn:$TRN_{\mathcal{E}}$, with entry, eg:EG, and exit, xg:XG, toll-gates.

From entry- and exit-gates we can observe

177 their unique identifier and

178 their mereology: pairs of entry-, respectively exit link and calculator unique identifiers; further

179 a pair of gate entry and exit sensors modeled as **external attribute** channels, (ges:ES,gls:XS), and

180 a time-stamped vehicle identity sensor modeled as **external attribute** channels.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

type

175 $\mathsf{N}_{\mathcal{E}}$

176 $\mathsf{TRN}_{\mathcal{E}} = (\mathsf{EG}\times\mathsf{XG})^* \times \mathsf{TRN}_{\mathcal{I}}$

177 GI

value

175 **obs_part**_$\mathsf{N}_{\mathcal{E}}$: $\Delta_{\mathcal{E}} \rightarrow \mathsf{N}_{\mathcal{E}}$

176 **obs_part**_$\mathsf{TRN}_{\mathcal{E}}$: $\mathsf{N}_{\mathcal{E}} \rightarrow \mathsf{TRN}_{\mathcal{E}}$

177 **uid**_G: (EG|XG) → GI

178 **obs_mereo**_G: (EG|XG) → (LI×CI)

176 trn:$\mathsf{TRN}_{\mathcal{E}}$ = **obs_part**_$\mathsf{TRN}_{\mathcal{E}}(\delta_{\mathcal{E}})$

channel

179 {attr_entry_ch[gi]|gi:GI•xtr_eGIds(trn)} ″enter″

179 {attr_exit_ch[gi]|gi:GI•xtr_xGIds(trn)} ″exit″

180 {attr_identity_ch[gi]|gi:GI•xtr_GIds(trn)} TIVI

type

180 TIVI = $\mathbb{T}$ × VI

We define some **auxiliary functions** over toll-road nets, trn:$\mathsf{TRN}_{\mathcal{E}}$:

181 xtr_eGℓ extracts the ℓist of entry gates,

182 xtr_xGℓ extracts the ℓist of exit gates,

183 xtr_eGIds extracts the $s$et of entry gate identifiers,

184 xtr_xGIds extracts the $s$et of exit gate identifiers,

185 xtr_Gs extracts the $s$et of all gates, and

186 xtr_GIds extracts the $s$et of all gate identifiers.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

value

181 $\mathsf{xtr_eG\ell}$: $\mathsf{TRN}_{\mathcal{E}} \rightarrow \mathsf{EG}^*$

181 $\mathsf{xtr_eG\ell(pgl,_)} \equiv \{\mathsf{eg}|(\mathsf{eg,xg}):(\mathsf{EG,XG})\cdot(\mathsf{eg,xg})\in$ **elems** $\mathsf{pgl}\}$

182 $\mathsf{xtr_xG\ell}$: $\mathsf{TRN}_{\mathcal{E}} \rightarrow \mathsf{XG}^*$

182 $\mathsf{xtr_xG\ell(pgl,_)} \equiv \{\mathsf{xg}|(\mathsf{eg,xg}):(\mathsf{EG,XG})\cdot(\mathsf{eg,xg})\in$ **elems** $\mathsf{pgl}\}$

183 $\mathsf{xtr_eGIds}$: $\mathsf{TRN}_{\mathcal{E}} \rightarrow$ GI-**set**

183 $\mathsf{xtr_eGIds(pgl,_)} \equiv \{$**uid**_GI(g)|g:EG·g $\in$ xtr_eGs(pgl,_)$\}$

184 $\mathsf{xtr_xGIds}$: $\mathsf{TRN}_{\mathcal{E}} \rightarrow$ GI-**set**

184 $\mathsf{xtr_xGIds(pgl,_)} \equiv \{$**uid**_GI(g)|g:EG·g $\in$ xtr_xGs(pgl,_)$\}$

185 $\mathsf{xtr_Gs}$: $\mathsf{TRN}_{\mathcal{E}} \rightarrow$ G-**set**

185 $\mathsf{xtr_Gs(pgl,_)} \equiv \mathsf{xtr_eGs(pgl,_)} \cup \mathsf{xtr_xGs(pgl,_)}$

186 $\mathsf{xtr_GIds}$: $\mathsf{TRN}_{\mathcal{E}} \rightarrow$ GI-**set**

186 $\mathsf{xtr_GIds(pgl,_)} \equiv \mathsf{xtr_eGIds(pgl,_)} \cup \mathsf{xtr_xGIds(pgl,_)}$

187 A **well-formedness condition** expresses

a. that there are as many entry end exit gate pairs as there are toll-plazas,

b. that all gates are uniquely identified, and

c. that each entry [exit] gate is paired with an entry [exit] link and has that link's unique identifier as one element of its mereology, the other elements being the calculator identifier and the vehicle identifiers.

The well-formedness relies on awareness of

188 the unique identifier, ci:CI, of the road pricing calculator, c:C, and

189 the unique identifiers, vis:VI-**set**, of the fleet vehicles.

```
axiom
187    ∀ n:N_R3, trn:TRN_R3 ·
187        let (exgl,(exl,hl,lll)) = obs_part_TRN_R3(n) in
187a.          len exgl = len exl = len hl = len lll + 1
187b.      ∧ card xtr_GIds(exgl) = 2 ∗ len exgl
187c.      ∧ ∀ i:Nat·i ∈ inds exgl·
187c.              let ((eg,xg),(el,xl)) = (exgl(i),exl(i)) in
187c.              obs_mereo_G(eg) = (uid_U(el),ci,vis)
187c.            ∧ obs_mereo_G(xg) = (uid_U(xl),ci,vis)
187        end end
```

4.4.1.5 [e] Toll-gate to Calculator Channels:

190 Toll-gate entry and exit gates offer passing a pair: whether it is an entry or an exit gates, and pair of the vehicle's identity and the time-stamped identity of the link associated with the toll-gate

191 to the road pricing calculator via channel.

type

190 $\mathsf{EEVITiLI} = (''\mathtt{Entry}''|''\mathtt{Exit}'')\times(\mathsf{VI}\times(\mathbb{T}\times\mathsf{SonL}))$

channel

191 $\{\mathsf{g_c_ch}[\,\mathsf{gi,ci}\,]|\mathsf{gi:GI}\cdot\mathsf{gi} \in \mathsf{gis}\}:\mathsf{EEVITiLI}$

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

4.4.1.6 [f] Road Pricing Calculator Attributes:

192 The road pricing attributes include a programmable traffic map, trm:TRM, which, for each vehicle inside the toll-road net, records a chronologically ordered list of each vehicle's timed position, (τ,lpos), and

193 a static (total) road location function, vplf:VPLF. The *v*ehicle *p*osition *l*ocation *f*unction, vplf:VPLF, which, given a local position, lpos:LPos, yields *either* the simple vehicle position, svpos:SVPos, designated by the GNSS-provided position, *or* yields the response that the provided position is off the toll-road net The vplf:VPLF function is constructed, construct_vplf,

194 from awareness, of a geodetic road map, GRM, of the topology of the extended net, $\mathsf{n}_{\mathcal{E}}$:$\mathsf{N}_{\mathcal{E}}$, including the mereology and the geodetic attributes of links and hubs.

type

192 $\mathsf{TRM} = \mathsf{VI} \underset{m}{\rightarrow} (\mathbb{T} \times \mathsf{SVPos})^*$

193 $\mathsf{VPLF} = \mathsf{GRM} \rightarrow \mathsf{LPos} \rightarrow (\mathsf{SVPos} \mid \text{"}\mathtt{off_N}\text{"})$

194 GRM

value

192 **attr**_TRM: $\mathsf{C}_{\mathcal{E}} \rightarrow \mathsf{TRM}$

193 **attr**_VPLF: $\mathsf{C}_{\mathcal{E}} \rightarrow \mathsf{VPLF}$

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

- The geodetic road map maps geodetic locations into hub and link identifiers.

105 Geodetic link locations represent the set of point locations of a link.

101a. Geodetic hub locations represent the set of point locations of a hub.

195 A geodetic road map maps geodetic link locations into link identifiers and geodetic hub locations into hub identifiers.

196 We sketch the construction, *geo_GRM*, of geodetic road maps.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

```
type
195   GRM = (GeoL →m LI) ∪ (GeoH →m HI)
value
196   geo_GRM: N → GRM
196   geo_GRM(n) ≡
196      let ls = xtr_links(n), hs = xtr_hubs(n) in
196      [ attr_GeoL(l)↦uid_LI(l)|l:L•l ∈ ls ]
196      ∪
196      [ attr_GeoH(h)↦uid_HI(h)|h:H•h ∈ hs ] end
```

197 The vplf:VPLF function obtains a simple vehicle position, svpos, from a geodetic road map, grm:GRM, and a local position , lpos:

```
value
197  obtain_SVPos: GRM → LPos → SVPos
197  obtain_SVPos(grm)(lpos) as svpos
197  post: case svpos of
197             SatH(hi)    → within(lpos,grm(hi)),
197             SonL(li)    → within(lpos,grm(li)),
197             "off_N"  → true end
```

4.4.1.7 [g] "Total" System State:

Global values:

198 There is a given domain, $\delta_{\mathcal{E}}$:$\Delta_{\mathcal{E}}$;

199 there is the net, $\mathsf{n}_{\mathcal{E}}$:$\mathsf{N}_{\mathcal{E}}$, of that domain;

200 there is toll-road net, $\mathsf{trn}_{\mathcal{E}}$:$\mathsf{TRN}_{\mathcal{E}}$, of that net;

201 there is a set, $\mathsf{egs}_{\mathcal{E}}$:$\mathsf{EG}_{\mathcal{E}}$-**set**, of entry gates;

202 there is a set, $\mathsf{xgs}_{\mathcal{E}}$:$\mathsf{XG}_{\mathcal{E}}$-**set**, of exit gates;

203 there is a set, $\mathsf{gis}_{\mathcal{E}}$:$\mathsf{GI}_{\mathcal{E}}$-**set**, ofgate identifiers;

204 there is a set, $\mathsf{vs}_{\mathcal{E}}$:$\mathsf{V}_{\mathcal{E}}$-**set**, of vehicles;

205 there is a set, $\mathsf{vis}_{\mathcal{E}}$:$\mathsf{VI}_{\mathcal{E}}$-**set**, of vehicle identifiers;

206 there is the road-pricing calculator, $\mathsf{c}_{\mathcal{E}}$:$\mathsf{C}_{\mathcal{E}}$ and

207 there is its unique identifier, $\mathsf{ci}_{\mathcal{E}}$:CI.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

value

198 $\delta_{\mathcal{E}}:\Delta_{\mathcal{E}}$
199 $\mathsf{n}_{\mathcal{E}}:\mathsf{N}_{\mathcal{E}} = \textbf{obs_part}_\mathsf{N}_{\mathcal{E}}(\delta_{\mathcal{E}})$
200 $\mathsf{trn}_{\mathcal{E}}:\mathsf{TRN}_{\mathcal{E}} = \textbf{obs_part}_\mathsf{TRN}_{\mathcal{E}}(\mathsf{n}_{\mathcal{E}})$
201 $\mathsf{egs}_{\mathcal{E}}:\mathsf{EG}\textbf{-set} = \mathsf{xtr_egs}(\mathsf{trn}_{\mathcal{E}})$
202 $\mathsf{xgs}_{\mathcal{E}}:\mathsf{XG}\textbf{-set} = \mathsf{xtr_xgs}(\mathsf{trn}_{\mathcal{E}})$
203 $\mathsf{gis}_{\mathcal{E}}:\mathsf{XG}\textbf{-set} = \mathsf{xtr_gis}(\mathsf{trn}_{\mathcal{E}})$
204 $\mathsf{vs}_{\mathcal{E}}:\mathsf{V}_{\mathcal{E}}\textbf{-set} = \textbf{obs_part}_\mathsf{VS}(\textbf{obs_part}_\mathsf{F}_{\mathcal{E}}(\delta_{\mathcal{E}}))$
205 $\mathsf{vis}_{\mathcal{E}}:\mathsf{VI}\textbf{-set} = \{\textbf{uid}_\mathsf{VI}(\mathsf{v}_{\mathcal{E}})|\mathsf{v}_{\mathcal{E}}:\mathsf{V}_{\mathcal{E}}\cdot\mathsf{v}_{\mathcal{E}} \in \mathsf{vs}_{\mathcal{E}}\}$
206 $\mathsf{c}_{\mathcal{E}}:\mathsf{C}_{\mathcal{E}} = \textbf{obs_part}_\mathsf{C}_{\mathcal{E}}(\delta_{\mathcal{E}})$
207 $\mathsf{ci}_{\mathcal{E}}:\mathsf{CI}_{\mathcal{E}} = \textbf{uid}_\mathsf{CI}(\mathsf{c}_{\mathcal{E}})$

4.4.1.8 [h] "Total" System Behaviour:

The signature and definition of the system behaviour is sketched as are the signatures of the vehicle, toll-gate and road pricing calculator.

- We shall model the behaviour of the road pricing system as follows:
  - we shall not model behaviours nets, nhubs and links;
  - thus we shall model only
    - the behaviour of vehicles, veh,
    - the behaviour of toll-gates, gate, and
    - the behaviour of the road-pricing calculator, calc,
  - The behaviours of vehicles and toll-gates are presented here.
  - But the behaviour of the road-pricing calculator is "deferred" till Sect. 5.4 since it reflects an interface requirements.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

208 The road pricing system behaviour, **sys**, is expressed as

a. the parallel, ∥, (distributed) composition of the behaviours of all vehicles, with the parallel composition of

b. the parallel (likewise distributed) composition of the behaviours of all entry gates, with the parallel composition of

c. the parallel (likewise distributed) composition of the behaviours of all exit gates, with the parallel composition of

d. the behaviour of the road-pricing calculator,

```
value
208   sys: Unit → Unit
208   sys() ≡
208a.       ∥ {veh(uid_V(v),(ci,gis),attr_TiGPos_ch)|v:V•v ∈ vs_ℰ}
208b.     ∥ ∥ {gate(`Entry″)(gi,obs_mereo_G(eg),
208b.              (attr_entry_ch[gi],attr_identify_ch[gi],attr_exit_ch[gi]))
208b.              |eg:EG•eg ∈ egs_ℰ∧gi=uid_EG(eg)}
208c.     ∥ ∥ {gate(`Exit″)(gi,obs_mereo_G(xg),
208c.              (attr_entry_ch[gi],attr_identify_ch[gi],attr_exit_ch[gi]))
208c.              |xg:XG•xg ∈ xgs_ℰ∧gi=uid_XG(xg)}
208d.       ∥ calc(ci_ℰ,(vis_ℰ,gis_ℰ))(rlf)(trm)

209   veh: vi:VI×(ci:CI×gis:GI-set)×𝕌TiGPos → out v_c_ch[vi,ci] Unit
215   gate: ee:EE×gi:GI×(ci:CI×VI-set×LI)×
215           (𝕌entry×𝕌identify×𝕌exit) → out g_c_ch[gi,ci]  Unit
249   calc: ci:CI×(vis:VI-set×gis:GI-set)×VPLF→TRM →
249              in {v_c_ch[vi,ci]|vi:VI•vi ∈ vis},{g_c_ch[gi,ci]|gi:GI•gi ∈ gis}  Unit
```


© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

Vehicle Behaviour:

209 Instead of moving around by explicitly expressed internal non-determinism[43] vehicles move around by unstated internal non-determinism and instead receive their current position from the global positioning subsystem.

210 At each moment the vehicle receives its time-stamped global position, (τ,gpos):TiGPos,

211 from which it calculates the local position, lpos:VPos

212 which it then communicates, with its vehicle identification, (vi,(τ,lpos)), to the road pricing subsystem —

213 whereupon it resumes its vehicle behaviour.

---

[43] We refer to Items 128b., 128c. on Slide 438 and 129b., 129(b.)ii, 129c. on Slide 440

```
value
209    veh: vi:VI×(ci:CI×gis:GI-set)×𝕌TiGPos →
209           out v_c_ch[vi,ci] Unit
209    veh(vi,(ci,gis),attr_TiGPos_ch[vi]) ≡
210        let (τ,gpos) = attr_TiGPos_ch[vi]? in
211        let lpos = loc_pos(gpos) in
212        v_c_ch[vi,ci] ! (vi,(τ,lpos)) ;
213        veh(vi,(ci,gis),attr_TiGPos_ch[vi]) end end
209        pre vi ∈ vis_ℰ ∧ ci = ci_ℰ ∧ gis = gis_ℰ
```


© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

- The above behaviour represents an assumption about the behaviour of vehicles.
  - If we were to design software for the monitoring and control of vehicles
  - then the above vehicle behaviour would have to be refined in order to serve as a proper interface requirements.
  - The refinement would include handling concerns
    - about the drivers' behaviour when entering, passing and exiting toll-gates,
    - about the proper function of the `GNSS` equipment, and
    - about the safe communication with the road price calculator.

- The above concerns would already have been addressed
  - in a model of ***domain facets*** such as
    - *human behaviour*,
    - *technology support*,
    - proper tele-communications *scripts*,
    - etcetera.
  - We refer to [Bjø10a].

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

Gate Behaviour:

- The entry and the exit gates have “vehicle enter”, “vehicle exit” and “timed vehicle identification” sensors.
  - The following assumption can now be made:
    - during the time interval between
    - a gate’s vehicle “entry” sensor having first sensed a vehicle entering that gate
    - and that gate’s “exit” sensor having last sensed that vehicle leaving that gate
    - that gate’s vehicle time and “identify” sensor registers the time when the vehicle is entering the gate and that vehicle’s unique identification.

- We sketch the toll-gate behaviour:

214 We parameterise the toll-gate behaviour as either an entry or an exit gate.

215 Toll-gates operate autonomously and cyclically.

216 The **attr_enter_ch** event "triggers" the behaviour specified in formula line Item 217–219.

217 The time-of-passing and the identity of the passing vehicle is sensed by **attr_passing_ch** channel events.

218 Then the road pricing calculator is informed of time-of-passing and of the vehicle identity **vi** and the link **li** associated with the gate.

219 And finally, after that vehicle has left the entry or exit gate

220 that toll-gate's behaviour is resumed.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

```
type
214   EE = "Enter" | "Exit"
value
215   gate: ee:EE×gi:GI×(ci:CI×VI-set×LI)×
215             (𝕌enter×𝕌passing×𝕌leave) →
215               out g_c_ch[gi,ci] Unit
215   gate(ee,gi,(ci,vis,li),
215             ea:(attr_enter_ch[gi],attr_passing_ch[gi],attr_leave_ch[gi])) ≡
216     attr_enter_ch[gi] ? ;
217     let (τ,vi) = attr_passing_ch[gi] ? in assert vi ∈ vis
218     g_c_ch[gi,ci] ! (ee,(vi,(τ,SonL(li)))) ;
219     attr_leave_ch[gi] ? ;
220     gate(ee,gi,(ci,vis,li),ea)
215       end
215       pre ci = ci_ℰ ∧ vis = vis_ℰ ∧ li ∈ lis_ℰ
```

- The above behaviour represents an assumption about the behaviour of toll-gates.
  - If we were to design software for the monitoring and control of toll-gates
  - then the above gate behaviour would have to be refined in order to serve as a proper interface requirements.
  - The refinement would include handling concerns
    - about the drivers' behaviour when entering, passing and exiting toll-gates,
    - about the proper function of the entry, passing and exit sensors,
    - about the proper function of the gate barrier (opening and closing), and
    - about the safe communication with the road price calculator.

The above concerns would already have been addressed

- in a model of *domain facets* such as
  - *human behaviour*,
  - *technology support*,
  - proper tele-communications *scripts*,
  - etcetera.
- We refer to [Bjø10a]

4.5. Requirements Fitting

- Often a domain being described
- "fits" onto, is "adjacent" to, "interacts" in some areas with,
- another domain:
  - *transportation* with *logistics*,
  - *health-care* with *insurance*,
  - *banking* with *securities trading* and/or *insurance*,
  - and so on.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

- The issue of requirements fitting arises
  - ◈ when two or more software development projects
  - ◈ are based on what appears to be the same domain.
- The problem then is
  - ◈ to harmonise the two or more software development projects
  - ◈ by harmonising, if not too late, their requirements developments.

- We thus assume
  - that there are n domain requirements developments, $d_{r_1}, d_{r_2}, \ldots, d_{r_n}$, being considered, and
  - that these pertain to the same domain — and can hence be assumed covered by a same domain description.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

Definition 32 Requirements Fitting:

- *By* **requirements fitting** *we mean*
  - *a* **harmonisation** *of* $n > 1$ *domain requirements*
  - *that have overlapping (shared) not always consistent parts and*
  - *which results in*
    - n **partial domain requirements**', $p_{d_{r_1}}, p_{d_{r_2}}, \ldots, p_{d_{r_n}}$, *and*
    - m **shared domain requirements**, $s_{d_{r_1}}, s_{d_{r_2}}, \ldots, s_{d_{r_m}}$,
    - *that "fit into" two or more of the partial domain requirements* ■
- *The above definition pertains to the result of 'fitting'.*
- *The next definition pertains to the act, or process, of 'fitting'.*

Definition 33 Requirements Harmonisation:

- *By* **requirements harmonisation** *we mean*
  - *a number of alternative and/or co-ordinated prescription actions,*
  - *one set for each of the domain requirements actions:*
    - *Projection,*
    - *Instantiation,*
    - *Determination and*
    - *Extension.*

- *They are – we assume n separate software product requirements:*
  - Projection:
    - *If the n product requirements do not have the same projections,*
    - *then identify a common projection which they all share,*
    - *and refer to it as the* common projection.
    - *Then develop, for each of the n product requirements,*
    - *if required,*
    - *a* specific projection *of the common one.*
    - *Let there be m such specific projections, $m \leq n$.*

- Instantiation:
  - *First instantiate the common projection, if any instantiation is needed.*
  - *Then for each of the m specific projections*
  - *instantiate these, if required.*
- Determination:
  - *Likewise, if required, "perform" "determination" of the possibly instantiated common projection,*
  - *and, similarly, if required,*
  - *"perform" "determination" of the up to m possibly instantiated projections.*

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

- *Extension:*
  - *Finally "perform extension" likewise:*
  - *First, if required, of the common projection (etc.),*
  - *then, if required, on the up m specific projections (etc.).*
- *These harmonization developments may possibly interact and may need to be iterated* ■

- By a **partial domain requirements** we mean a domain requirements which is short of (that is, is missing) some prescription parts: text and formula ■
- By a **shared domain requirements** we mean a domain requirements ■

- By **requirements fitting** m *s*hared *d*omain *r*equirements texts, $sdrs$, into n partial domain requirements we mean that
  - there is for each *p*artial *d*omain *r*equirements, pdr_i,
  - an identified, non-empty *s*ubset of $sdrs$ (could be all of $sdrs$), $ssdrs_i$,
  - such that textually conjoining $ssdrs_i$ to pdr_i,
  - i.e., $ssdrs_i \oplus pdr_i$
  - can be claimed to yield the "original" d_{r_i},
  - that is, $\mathcal{M}(ssdrs_i \oplus pdr_i) \subseteq \mathcal{M}(d_{r_i})$,
  - where $\mathcal{M}$ is a suitable meaning function over prescriptions ■

4.6. Discussion

- **Facet-oriented Fittings:**
  - An altogether different way of looking at domain requirements
    - may be achieved when also considering domain facets
    - not covered in neither the example of Sect. nor in this section (i.e., Sect.)

    - nor in the following two sections.
  - We refer to [Bjø10a].

Example 72 . Domain Requirements — Fitting:

- Example 71 hints at three possible sets of interface requirements:
  - (i) for a road pricing [sub-]system, as will be illustrated in Sect. ;
  - (ii) for a vehicle monitoring and control [sub-]system, and
  - (iii) for a toll-gate monitoring and control [sub-]system.
- The vehicle monitoring and control [sub-]system would focus on implementing the vehicle behaviour, see Items 209- 213 on Slide 568.
- The toll-gate monitoring and control [sub-]system would focus on implementing the calculator behaviour, see Items 215- 220 on Slide 573.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

The fitting amounts to

- (a) making precise the (narrative and formal) texts that are specific to each of of the three (i–iii) separate sub-system requirements are kept separate;
- (b) ensuring that (meaning-wise) shared texts that have different names for (meaning-wise) identical entities have these names renamed appropriately;
- (c) that these texts are subject to commensurate and ameliorated further requirements development;
- etcetera

5. Interface Requirements

- We remind the listener that
  - **interface requirements**
    - can be expressed only using terms from
    - both the domain
    - and the machine
- Users are not part of the machine.
  - So no reference can be made to users, such as
  - *"the system must be user friendly"*,
  - and the like!

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

- By an **interface requirements** we [also] mean
  - *a requirements prescription which refines and extends the domain requirements*
  - *by considering those requirements of the domain requirements whose*
    - *endurants (parts, materials) and*
    - *perdurants (actions, events and behaviours)*
  - *are* **"shared"**
  - *between the domain and the machine (being requirements prescribed)* ■
  - The two **interface requirements** definitions above go hand–in–hand, i.e., complement one-another.

5.1. Shared Phenomena

- By **sharing** we mean
  - that *some or all properties* of an **endurant** is represented both
    - in the domain and
    - "inside" the machine, and
    - that their machine representation
    - must at suitable times
    - reflect their state in the domain;

  and/or
  - that an **action**
    - requires a sequence of several "on-line" interactions
    - between the machine (being requirements prescribed) and
    - the domain, usually a person or another machine;

  and/or

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

- that an **event**
  - arises either in the domain,
    that is, in the environment of the machine,
  - or in the machine,
  - and need be communicated to the machine, respectively to the environment;

  and/or
- that a **behaviour** is manifested both
  - by actions and events of the domain and
  - by actions and events of the machine ■

- So a systematic reading of the domain requirements shall
  - result in an identification of all shared
    - endurants,
      - parts,
      - materials and
      - components;

      and
    - perdurants
      - actions,
      - events and
      - behaviours.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

- Each such shared phenomenon shall then be individually dealt with:
  - **endurant sharing** shall lead to interface requirements for data initialisation and refreshment as well as for access to endurant attributes;
  - **action sharing** shall lead to interface requirements for interactive dialogues between the machine and its environment;
  - **event sharing** shall lead to interface requirements for how such event are communicated between the environment of the machine and the machine; and
  - **behaviour sharing** shall lead to interface requirements for action and event dialogues between the machine and its environment.

5.1.0.1 Environment–Machine Interface:

- Domain requirements extension, Sect. 4.4,
  - usually introduce new endurants into (i.e., 'extend' the) domain.
  - Some of these endurants may become elements of the domain requirements.
  - Others are to be projected "away".
  - Those that are let into the domain requirements
    - either have their endurants represented, somehow, also in the machine,
    - or have (some of) their properties, usually some attributes, accessed by the machine.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

- Similarly for perdurants.
  - Usually the machine representation of shared perdurants access (some of) their properties, usually some attributes.
- The interface requirements must spell out which domain extensions are shared.
- Thus domain extensions may necessitate a review of domain
  - projection,
  - instantiations and
  - determination.

- In general, there may be several of the projection–eliminated parts (etc.) whose dynamic attributes need be accessed
  in the usual way, i.e., by means of attr_XYZ_ch channel communications (where XYZ is a projection–eliminated part attribute).

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

Example 73 . Interface Requirements — Projected Extensions: We refer to Fig. 3 on Slide 300.

- We do not represent the GNSS system in the machine:
  - only its "effect": the ability to record global positions
  - by accessing the GNSS attribute (channel):

channel

169 $\{$attr_TiGPos_ch[vi]|vi:VI·vi $\in$ xtr_VIs(vs)$\}$: TiGPos

- And we do not really represent the gate nor its sensors and actuator in the machine.
- But we do give an idealised description of the gate behaviour, see Items 215–220
- Instead we represent their dynamic gate attributes:
  - (179) the vehicle entry sensors (leftmost ■s),
  - (179) the vehicle identity sensor (center ■), and
  - (180) the vehicle exit sensors (rightmost ■s)
- by channels — we refer to Example 71 (Sect. , Slide 552):

channel

179 {attr_entry_ch[gi]|gi:GI·xtr_eGIds(trn)} ″enter″
179 {attr_exit_ch[gi]|gi:GI·xtr_xGIds(trn)} ″exit″
180 {attr_identity_ch[gi]|gi:GI·xtr_GIds(trn)} TIVI ■

5.2. Shared Endurants

Example 74 . Interface Requirements. Shared Endurants:

- The main shared endurants are
  - the vehicles,
  - the net (hubs, links, toll-gates) and
  - the price calculator.
- As domain endurants hubs and links undergo changes,
  - all the time,
  - with respect to the values of several attributes:
    - *length, geodetic information, names,*
    - *wear and tear* (where-ever applicable),
    - *last/next scheduled maintenance* (where-ever applicable),
    - *state* and *state space*,
    - and many others.

- Similarly for vehicles:
  - their position,
  - velocity and acceleration, and
  - many other attributes.
- We then come up with something like
  - hubs and links are to be represented as tuples of relations;
  - each net will be represented by a pair of relations
    - a hubs relation and a links relation;
    - each hub and each link may or will be represented by several tuples;
  - etcetera.
- In this database modeling effort it must be secured that "standard" operations on nets, hubs and links can be supported by the chosen relational database system

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

5.2.1. Data Initialisation

- In general, one must prescribe data initialisation, that is provision for
  - an interactive user interface dialogue with a set of proper display screens,
    - one for establishing net, hub or link attributes names and their types, and, for example,
    - two for the input of hub and link attribute values.
  - Interaction prompts may be prescribed:
    - next input,
    - on-line vetting and
    - display of evolving net, etc.
  - These and many other aspects may therefore need prescriptions.

Example 75 . Interface Requirements. Shared Endurant Initialisation:

- The domain is that of the road net, n:N.
- By 'shared road net initialisation'
  we mean the "ab initio" establishment, "from scratch",
  of a data base recording the properties of all links, l:L, and hubs, h:H,
  - their unique identifications, **uid**_L(l) and **uid**_H(h),
  - their mereologies, **obs_mereo**_L(l) and **obs_mereo**_H(h),
  - the initial values of all their static and programmable attributes and
  - the access values, that is, channel designations for all other attribute categories.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

221 There are r_l and r_h "recorders" recording link, respectively hub properties – with each recorder having a unique identity.

222 Each recorder is charged with the recording of a set of links or a set of hubs according to some partitioning of all such.

223 The recorders inform a central data base, net_db, of their recordings (ri,hol,(u$_j$,m$_j$,attrs$_j$)) where

224 ri is the identity of the recorder,

225 hol is either a `hub` or a `link` literal,

226 u$_j$ = **uid**_L(l) or **uid**_H(h) for some link or hub,

227 m$_j$ = **obs_mereo**_L(l) or **obs_mereo**_H(h) for that link or hub and

228 attrs$_j$ are *attributes* for that link or hub — where *attributes* is a function which "records" all respective static and dynamic attributes (left undefined).

type

221 RI

value

221 rl,rh:NAT **axiom** rl>0 ∧ rh>0

type

223 M = RI×"link"×LNK | RI×"hub"×HUB

223 LNK = LI × HI-**set** × LATTRS

223 HUB = HI × LI-**set** × HATTRS

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

value

222 partitioning: L-**set** → **Nat** → (L-**set**)* | H-**set** → **Nat** → (H-**set**)*
222 partitioning(s)(r) **as** sl
222 **post**: **len** sl = r ∧ ∪ **elems** sl = s
222 ∧ ∀ si,sj:(L-**set**|H-**set**) ·
222 si≠{}∧sj≠{}∧{si,sj}⊆**elems** ss⇒si ∩ sj={}

229 The $r_l + r_h$ recorder behaviours interact with the one net_db behaviour

channel
229 r_db: RI×(LNK|HUB)
value
229 link_rec: RI → L-**set** → **out** r_db **Unit**
229 hub_rec: RI → H-**set** → **out** r_db **Unit**
229 net_db: **Unit** → **in** r_db **Unit**

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

230 The data base behaviour, net_db, offers to receive messages from the link and hub recorders.

231 The data base behaviour, net_db, deposits these messages in respective variables.

232 Initially there is a net, $n : N$,

233 from which is observed its links and hubs.

234 These sets are partitioned into r_l, respectively r_h length lists of non-empty links and hubs.

235 The ab-initio data initialisation behaviour, ab_initio_data, is then the parallel composition of link recorder, hub recorder and data base behaviours with link and hub recorder being allotted appropriate link, respectively hub sets.

236 We construct, for technical reasons, as the listener will soon see, disjoint lists of link, respectively hub recorder identities.

```
value
230   net_db:
variable
231   lnk_db: (RI×LNK)-set
231   hub_db: (RI×HUB)-set
value
232   n:N
233   ls:L-set = obs_Ls(obs_LS(n))
233   hs:H-set = obs_Hs(obs_HS(n))
234   lsl:(L-set)* = partitioning(ls)(rl)
234   lhl:(H-set)* = partitioning(hs)(rh)
236   rill:RI* axiom len rill = rl = card elems rill
236   rihl:RI* axiom len rihl = rh = card elems rihl
235   ab_initio_data: Unit → Unit
235   ab_initio_data() ≡
235      ‖ {lnk_rec(rill[ i ])(lsl[ i ])|i:Nat•1≤i≤rl} ‖
235      ‖ {hub_rec(rihl[ i ])(lhl[ i ])|i:Nat•1≤i≤rh}
235      ‖ net_db()
```

237 The link and the hub recorders are near-identical behaviours.

238 They both revolve around an imperatively stated **for all ... do ... end**.
The selected link (or hub) is inspected and the "data" for the data base is prepared from

239 the unique identifier,

240 the mereology, and

241 the attributes.

242 These "data" are sent, as a message, prefixed the senders identity, to the data base behaviour.

243 We presently leave the ... unexplained.

```
value
229  link_rec: RI → L-set → Unit
237  link_rec(ri,ls) ≡
238     for ∀ l:L·l ∈ ls do uid_L(l)
239        let lnk = (uid_L(l),
240                   obs_mereo_L(l),
241                   attributes(l)) in
242        rdb ! (ri,"link",lnk);
243        ... end
238     end
```


© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

```
229  hub_rec: RI × H-set → Unit
237  hub_rec(ri,hs) ≡
238    for ∀ h:H•h ∈ hs do uid_H(h)
239      let hub = (uid_L(h),
240                 obs_mereo_H(h),
241                 attributes(h)) in
242      rdb ! (ri,"hub",hub);
243      ... end
238   end
```

244 The **net_db** data base behaviour revolves around a seemingly "never-ending" cyclic process.

245 Each cycle "starts" with acceptance of some,

246 either link or hub data.

247 If link data then it is deposited in the link data base,

248 if hub data then it is deposited in the hub data base.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

```
value
244   net_db() ≡
245      let (ri,hol,data) = r_db ? in
246      case hol of
247         "link" → ... ; lnk_db := lnk_db ∪ (ri,data),
248         "hub"  → ... ; hub_db := hub_db ∪ (ri,data)
246      end end ;
244′      ... ;
244      net_db()
```

- The above model is an idealisation.
  - It assumes that the link and hub data represent a well-formed net.
  - Included in this well-formedness are the following issues:
    - (a) that all link or hub identifiers are communicated exactly once,
    - (b) that all mereologies refer to defined parts, and
    - (c) that all attribute values lie within an appropriate value range.
  - If we were to cope with possible recording errors then we could, for example, extend the model as follows:
    - (i) when a link or a hub recorder has completed its recording then it increments an initially zero counter (say at formula Item 243);
    - (ii) before the net data base recycles it tests whether all recording sessions has ended and then proceeds to check the data base for well-formedness issues (a–b–c) (say at formula Item 244′) ■

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

- The above example illustrates the 'interface' phenomenon:
  - In the formulas, for example, we show both
    - manifest domain entities, viz., n, l, h etc., and
    - abstract (required) software objects, viz., $(ui, me, attrs)$.

5.2.2. Data Refreshment

- One must also prescribe data refreshment:
  - an interactive user interface dialogue with a set of proper display screens
    - one for selecting the updating of net, of hub or of link attribute names and their types and, for example,
    - two for the respective update of hub and link attribute values.
  - Interaction-prompts may be prescribed:
    - next update,
    - on-line vetting and
    - display of revised net, etc.
  - These and many other aspects may therefore need prescriptions.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

5.3. Shared Actions, Events and Behaviours

- We now illustrate the concept of shared perdurants
  - via the domain requirements extension example
  - of Sect. 4.4, i.e. Example 71 Slides 541–576.

Example 76 . Interface Requirements — Shared Behaviours: Road Pricing Calculator Behaviour:

249 The road-pricing calculator alternates between offering to accept communication from

250 either any vehicle

251 or any toll-gate.

```
249   calc: ci:CI×(vis:VI-set×gis:GI-set)→RLF→TRM→
250         in {v_c_ch[ ci,vi ]|vi:VI•vi ∈ vis},
251            {g_c_ch[ ci,gi ]|gi:GI•gi ∈ gis}  Unit
249   calc(ci,(vis,gis))(rlf)(trm) ≡
250       react_to_vehicles(ci,(vis,gis))(rlf)(trm)
249   ⌈⌉⌊⌋
251       react_to_gates(ci,(vis,gis))(rlf)(trm)
249       pre ci = ci_ℰ ∧ vis = vis_ℰ ∧ gis = gis_ℰ
```

252 If the communication is from a vehicle inside the toll-road net

253 then its toll-road net position, vp, is found from the road location function, rlf,

254 and the calculator resumes its work with the traffic map, trm, suitably updated,

255 otherwise the calculator resumes its work with no changes.

```
250    react_to_vehicles(ci,(vis,gis),vplf)(trm) ≡
250        let (vi,(τ,lpos)) =⌈⌉⌈{v_c_ch[ ci,vi ]|vi:VI·vi∈ vis} in
252          if vi ∈ dom trm
253            then let vp = vplf(lpos) in
254                   calc(ci,(vis,gis),vplf)(trm†[ vi↦trm⌢⟨(τ,vp)⟩ ]) end
255            else   calc(ci,(vis,gis),vplf)(trm) end end
```

256 If the communication is from a gate,

257 then that gate is either an entry gate or an exit gate;

258 if it is an entry gate

259 then the calculator resumes its work with the vehicle (that passed the entry gate) now recorded, afresh, in the traffic map, trm.

260 Else it is an exit gate and

261 the calculator concludes that the vehicle has ended its to-be-paid-for journey inside the toll-road net, and hence to be billed;

262 then the calculator resumes its work with the vehicle now removed from the traffic map, trm.

```
251  react_to_gates(ci,(vis,gis),vplf)(trm) ≡
251    let (ee,(τ,(vi,li))) =
251      ⨅{g_c_ch[ ci,gi ]|gi:GI•gi∈ gis} in
257        case ee of
258          ″Enter″ →
259            calc(ci,(vis,gis),vplf)(trm∪[ vi↦⟨(τ,SonL(li))⟩ ]),
260          ″Exit″  →
261            billing(vi,trm(vi)⁀⟨(τ,SonL(li))⟩);
262            calc(ci,(vis,gis),vplf)(trm\{vi}) end end
```

- The above behaviour is the one for which we are to design software

5.4. Discussion

- ⟐ TO BE WRITTEN

 © Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

6. Machine Requirements

Definition 34 Machine Requirements: *By* **machine requirements** *we shall understand*

- *such requirements*
- *which can be expressed "sôlely" using terms*
- *from, or of* ***the machine*** ■

Definition 35 The Machine: *By the* **machine** *we shall understand*

- *the hardware*
- *and software*
- *to be built from the requirements* ■

- The expression
  - *which can be expressed*
  - *“sôlely” using terms*
  - *from, or of the machine*

  shall be understood with “a grain of salt”.
  - Let us explain.
    - The **machine requirements** statements
    - may contain references to domain entities
    - but these are meant to be generic references,
    - that is, references to certain classes of entities in general.

  We shall illustrate this “genericitiy” in some of the examples below.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

6.1. Varieties of Machine Requirements

- Analysis of different kinds of requirements,
  - such as exemplified
  - but not so classified
  - in seminal textbooks [Lau02, van09]
  - suggests the following categories of machine requirements:
    - (i) derived requirements,
    - (ii) technology requirements and
    - (iii) development requirements.

6.2. Derived Requirements

Definition 36 Derived Requirements: *By* **derived requirements** *we shall understand*

- *such machine requirements*
- *which focus on*
- *exploiting facilities*
- *of the software or hardware*
- *of the machine* ■

 © Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

- We use the term 'derived' for the following reason:
  - "exploiting facilities of the software", to us,
  - means that **design requirements**,
  - have resulted in requirements that imply the presence, in the machine, of concepts (i.e., hardware and/or software),
  - and that it is these concepts
  - that the **derived requirements** "rely" on.

- We illustrate two forms of derived requirements:
  - actions and
  - events.
- Derived behaviours could be illustrated
  - but we rely on the **interface requirements** Example 76:
  - *The Road Pricing Calculator* shared behaviour.
- There are other kinds of **derived requirements**.
  - Some are query-like.
  - We leave that for the listener to ponder about.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

6.2.1. Derived Actions

Definition 37 Derived Action:

- *By a* **derived action** *we shall understand*
  - *(a) a conceptual action*
  - *(b) that calculates a property or some non-Boolean value*
  - *(c) from a machine behaviour state*
  - *(d) as instigated by some actor* ■

Example 77 . Machine Requirements. Derived Action: Tracing Vehicles:

- The example is based on the *Road Pricing Calculator Behaviour* of Example 76 on Slide 619.
  - The "external" actor, i.e., a user of the *Road Pricing Calculator* system
  - wishes to trace specific vehicles "cruising" the toll-road.
  - That user (a *Road Pricing Calculator* staff),
    - issues a command to the *Road Pricing Calculator* system,
    - with the identity of a vehicle
    - not already being traced.
  - As a result the *Road Pricing Calculator* system
    - augments a possibly void trace of
    - the timed toll-road positions of vehicles.

 © Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

- We augment the definition of the *calc*ulator definition Items 249–262, Slides 619–622.

263 Traces are modeled by a pair of dynamic attributes:

a. as a programmable attribute, *tra:TRA*, of the set of identifiers of vehicles being traced, and

b. as a reactive attribute, *vdu:VDU*[44], that maps vehicle identifiers into time-stamped sequences of simple vehicle positions, i.e., as a subset of the *trm:TRM* programmable attribute.

264 The actor-to-calculator *begin* or *end* trace command, *cmd:Cmd*, is modeled as an autonomous dynamic attribute of the *calc*ulator.

265 The *calc*ulator signature is furthermore augmented with the three attributes mentioned above.

[44] VDU: visual display unit

266 The occurrence and handling of an actor trace command is modeled as a non-deterministic external choice and a *react_to_trace_cmd* behaviour.

267 The reactive attribute value (`attr_vdu_ch ?`) is that subset of the traffic map (*trm*) which records just the time-stamped sequences of simple vehicle positions being traced (*tra*).

type

263a. TRA = VI-**set**

263b. VDU = TRM

264 Cmd = BTr | ETr

264 BTr :: VI

264 ETr :: VI

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

```
value
265     calc: ci:CI×(vis:VI-set×gis:GI-set)×(𝕌Cmd,𝕌Trace,𝕌Trace)→RLF→TRM→TRA
250,251     in {v_c_ch[ci,vi]|vi:VI•vi ∈ vis}, {g_c_ch[ci,gi]|gi:GI•gi ∈ gis}  Unit
249     calc(ci,(vis,gis))(rlf)(trm) ≡
250         react_to_vehicles(ci,(vis,gis),(attr_cmd_ch,attr_vdu_ch))(rlf)(trm)(tra)
249     ⌈⌉⌊⌋
251         react_to_gates(ci,(vis,gis),(attr_cmd_ch,attr_vdu_ch))(rlf)(trm)(tra)
249     ⌈⌉⌊⌋
266         react_to_trace_cmd(ci,(vis,gis),(attr_cmd_ch,attr_vdu_ch))(rlf)(trm)(tra)
249         pre ci = ci_ℰ ∧ vis = vis_ℰ ∧ gis = gis_ℰ
267         axiom □ attr_vdu_ch[ci]? = trm|tra
```

```
250  react_to_vehicles(ci,(vis,gis),(attr_cmd_ch,attr_vdu_ch))(vplf)(trm)(tra) ≡
250     let (vi,(τ,lpos)) =⌈⌉⌈{v_c_ch[ci,vi]|vi:VI•vi∈ vis} in
252     if vi ∈ dom trm
253       then
253          let vp = vplf(lpos) in
253          calc(ci,(vis,gis),(attr_cmd_ch,attr_vdu_ch))(vplf)(trm†[vi↦trm⌢⟨(τ,vp)⟩])(tra)
255       else   calc(ci,(vis,gis),(attr_cmd_ch,attr_vdu_ch))(vplf)(trm)(tra) end end

251  react_to_gates(ci,(vis,gis),(attr_cmd_ch,attr_vdu_ch))(vplf)(trm)(tra) ≡
251     let (ee,(τ,(vi,li))) =⌈⌉⌈{g_c_ch[ci,gi]|gi:GI•gi∈ gis} in
257    case ee of
258      "Enter" →
258        calc(ci,(vis,gis),(attr_cmd_ch,attr_vdu_ch))(vplf)(trm∪[vi↦⟨(τ,SonL(li))⟩])(tra),
260      "Exit" →
260        billing(vi,trm(vi)⌢⟨(τ,SonL(li))⟩);
260        calc(ci,(vis,gis),(attr_cmd_ch,attr_vdu_ch))(vplf)(trm\{vi})(tra)
251     end end
```

268 The *react_to_trace_cmd* alternative behaviour is either a "Begin" or an "End" request which identifies the affected vehicle.

269 If it is a "Begin" request and the identified vehicle is already being traced then we do not prescribe what to do!

270 Else we resume the calculator behaviour, now recording that vehicle as being traced.

271 If it is an "End" request and the identified vehicle is already being traced then we do not prescribe what to do!

272 Else we resume the calculator behaviour, now recording that vehicle as no longer being traced.

```
268  react_to_trace_cmd(ci,(vis,gis),(attr_cmd_ch,attr_vdu_ch))(vplf)(trm)(tra) ≡
268    let cmd = attr_cmd_ch[ ci ]? in
268    case cmd of
268      mkBTr(vi) →
269        if vi ∈ tra then chaos
270        else calc(ci,(vis,gis),(attr_cmd_ch,attr_vdu_ch))(vplf)(trm)(tra ∪ {vi})
268      mkETr(vi) →
271        if vi ∉ tra then chaos
272        else calc(ci,(vis,gis),(attr_cmd_ch,attr_vdu_ch))(vplf)(trm)(tra\{vi})
268    end end
```

- The above behaviour, Items 249–272, is the one for which we are to design software

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

- Example 77 exemplifies a derived action requirement as per definition 37:
  - (a) the action is conceptual,
    it has no physical counterpart in the domain;
  - (b) it calculates (267) a visual display (**vdu**);
  - (c) the **vdu** value is based on a conceptual notion of traffic road maps (**trm**), an element of the **calc**ulator state;
  - (d) the calculation is triggered by an actor (`attr_cmd_ch`).

6.2.2. Derived Events

Definition 38 Derived Event: *By a* **derived event** *we shall understand*

- *(a) a conceptual event,*
- *(b) that calculates a property or some non-Boolean value*
- *(c) from a machine behaviour state change* ■

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

Example 78 . Machine Requirements. Derived Event: Current Maximum Flow:

- The example is based on the *Road Pricing Calculator Behaviour* of Examples 76 and 77.
  - By "the current maximum flow" we understand
    - a time-stamped natural number, the number
    - representing the highest number of vehicles which
      - at the time-stamped moment cruised
      - or now cruises

    around the toll-road net.
- We augment the definition of the *cal*culator definition Items 249–272, Slides 619–637.

273 We augment *calc*ulator signature with

274 a time-stamped natural number valued dynamic programmable attribute, *(t:*$\mathbb{T}$*,max:Max)*.

275 Whenever a vehicle enters the toll-road net, through one of its gates,

a. it is checked whether the resulting number of vehicles recorded in the *road traffic map* is higher than the hitherto *max*imum recorded number.

b. If so, that programmable attribute has its number element "upped" by one.

c. Otherwise not.

```
type
274     MAX = 𝕋 × NAT
value
265,273   calc: ci:CI×(vis:VI-set×gis:GI-set)×(𝕌Cmd,𝕌Trace,𝕌Trace)→RLF→TRM→TRA→MAX
250,251        in {v_c_ch[ci,vi]|vi:VI•vi ∈ vis},{g_c_ch[ci,gi]|gi:GI•gi ∈ gis}  Unit
...
251     react_to_gates(ci,(vis,gis),(attr_cmd_ch,attr_vdu_ch))(vplf)(trm)(tra)(t,m) ≡
251        let (ee,(τ,(vi,li))) =⨅{g_c_ch[ci,gi]|gi:GI•gi∈ gis} in
257        case ee of
258          "Enter" →
275a.              if card dom trm = m
275a.                then
258,275b.          calc(ci,(vis,gis),(attr_cmd_ch,attr_vdu_ch))(vplf)(trm∪[vi↦⟨(τ,SonL(li))⟩])(tra)(τ,m+1),
275c.              else
258,275c.          calc(ci,(vis,gis),(attr_cmd_ch,attr_vdu_ch))(vplf)(trm∪[vi↦⟨(τ,SonL(li))⟩])(tra)(t,m) end end
               ...
          end
```

- The above behaviour, Items 249 on Slide 619 through 275c. on the previous slide, is the one for which we are to design software ■

- Example 78 exemplifies a derived event requirement as per definition 38:
  - (a) the event is conceptual, it has no physical counterpart in the domain;
  - (b) it calculates (275b.) the max value based on a conceptual notion of traffic road maps (trm),
  - (c) an element of the calculator state.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

6.3. Technology Requirements

Definition 39 Technology Requirements: *By* **technology requirements** *we shall understand*

- *such machine requirements*
- *which primarily focus on alleviating*
  - *physical deficiencies of the hardware or*
  - *inefficiencies of the software*

  of the machine —
- *cf. Items (i–ii), i.e., Sects. – below.*

- We shall, in particular, consider the following kinds of technology requirements:
  - (i) performance requirements and
  - (ii) dependability requirements
- with dependability requirements being concerned with either
  - (a) accessibility,
  - (b) availability,
  - (c) integrity,
  - (d) reliability,
  - (e) safety,
  - (f) security and/or
  - (g) robustness.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

6.3.1. Performance Requirements

Definition 40 Performance Requirements: *By performance requirements we mean machine requirements that prescribe*

- *storage consumption,*
- *(execution, access, etc.) time consumption,*
- *as well as consumption of any other machine resource:*
  - *number of CPU units (incl. their quantitative characteristics such as cost, etc.),*
  - *number of printers, displays, etc., terminals (incl. their quantitative characteristics),*
  - *number of "other", ancillary software packages (incl. their quantitative characteristics),*
  - *of data communication bandwidth,*
  - *etcetera* ■

Example 79 . Machine Requirements. Technology: Performance:

- The road pricing system shall be able
  - to keep records of up to 50.000 vehicles at any time,
  - to record up to 10.000 vehicle positions per second, and
  - to bill up to 1000 (distinct) vehicles per second.
- A vehicle is assumed to access the road pricing calculator with a mean time between accesses of 5 seconds.
- A toll-gate is assumed to access the road pricing calculator with a mean time between accesses of 5 seconds ■

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

6.3.2. Dependability Requirements

- Dependability is a complex notion.

6.3.2.1 Failures, Errors and Faults

- To properly define the concept of *dependability* we need first introduce and define the concepts of
  - *failure*,
  - *error*, and
  - *fault*.

Definition 41 Failure:

- *A machine failure occurs*
- *when the delivered service*
- *deviates from fulfilling the machine function,*
- *the latter being what the machine is aimed at* ■

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

Definition 42 Error:

- *An error*
- *is that part of a machine state*
- *which is* ***liable to lead to subsequent failure.***
- *An error affecting the service*
- *is an indication that a failure occurs or has occurred* ■

Definition 43 Fault:

- *The adjudged (i.e., the 'so-judged') or hypothesised cause of an error*
- *is a fault* ■
- The term hazard is here taken to mean the same as the term fault.
- One should read the phrase: "adjudged or hypothesised cause" carefully:
- In order to avoid an unending trace backward as to the cause,
- we stop at *the cause which is intended to be prevented or tolerated.*

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

Definition 44 Machine Service: *The service delivered by a machine*

- *is its behaviour*
- **as it is perceptible** *by its user(s),*
- *where a user is a human, another machine or a(nother) system*
- *which interacts with it* ■

Definition 45 Dependability: *Dependability is defined*

- *as the property of a machine*
- *such that **reliance can justifiably be placed on the service** it delivers*

■

- We continue, less formally, by characterising the above defined concepts.
- "A given machine, operating in some particular environment (a wider system), may fail in the sense that some other machine (or system) makes, or could in principle have made, a *judgement* that the activity or inactivity of the given machine constitutes a *failure*".
- The concept of *dependability* can be simply defined as "the quality or the characteristic of being dependable", where the adjective 'dependable' is attributed to a machine whose failures are judged sufficiently rare or insignificant.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

- *Impairments* to dependability are the unavoidably expectable circumstances causing or resulting from "undependability": faults, errors and failures.
- *Means* for dependability are the techniques enabling one
  - to provide the ability to deliver a service on which reliance can be placed,
  - and to reach confidence in this ability.
- *Attributes* of dependability enable
  - the properties which are expected from the system to be expressed,
  - and allow the machine quality resulting from the impairments and the means opposing them to be assessed.

- Having already discussed the "threats" aspect,
- we shall therefore discuss the "means" aspect of the *dependability tree*.

- Attributes:
  - Accessibility
  - Availability
  - Integrity
  - Reliability
  - Safety
  - Security
- Means:
  - Procurement
    - Fault prevention
    - Fault tolerance
  - Validation
    - Fault removal
    - Fault forecasting
- Threats:
  - Faults
  - Errors
  - Failures

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

- Despite all the principles, techniques and tools aimed at *fault prevention*,
- *faults* are created.
- Hence the need for *fault removal.*
- *Fault removal* is itself imperfect.
- Hence the need for *fault forecasting.*
- Our increasing dependence on computing systems in the end brings in the need for *fault tolerance.*

Definition 46 Dependability Attribute: *By a dependability attribute we shall mean either one of the following:*

- *accessibility,*
- *availability,*
- *integrity,*
- *reliability,*
- *robustness,*
- *safety and*
- *security.*

That is, a machine is dependable if it satisfies some degree of "mixture" of being accessible, available, having integrity, and being reliable, safe and secure

- The crucial term above is "satisfies".
- The issue is: To what "degree"?
- As we shall see — in a later later lecture — to cope properly
  - with dependability requirements and
  - their resolution

  requires that we deploy
  - mathematical formulation techniques,
  - including analysis and simulation,

  from statistics (stochastics, etc.).

6.3.2.2 Accessibility

- Usually a desired, i.e., the required, computing system, i.e., the machine, will be used by many users — over "near-identical" time intervals.
- Their being granted access to computing time is usually specified, at an abstract level, as being determined by some internal nondeterministic choice, that is: essentially by *"tossing a coin"!*
- If such internal nondeterminism was carried over, into an implementation, some *"coin tossers"* might not get access to the machine "for a long- long time".

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

Definition 47 Accessibility: *A system being accessible — in the context of a machine being dependable —*

- *means that some form of "fairness"*
- *is achieved in guaranteeing users "equal" access*
- *to machine resources, notably computing time (and what derives from that).*

Example 80 . Machine Requirements. Technology: Accessibility:

- No vehicle access to the road pricing calculator shall wait more than 2 seconds.
- No toll-gate access to the road pricing calculator shall wait more than 2 seconds.

6.3.2.3 Availability

- Usually a desired, i.e., the required, computing system, i.e., the machine, will be used by many users — over "near-identical" time intervals.
- Once a user has been granted access to machine resources, usually computing time, that user's computation may effectively make the machine unavailable to other users —
- by "going on and on and on"!

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

Definition 48 Availability: *By availability — in the context of a machine being dependable — we mean*

- *its readiness for usage.*
- *That is, that some form of "guaranteed percentage of computing time" per time interval (or percentage of some other computing resource consumption)*
- *is achieved, for example, in the form of "time slicing"* ■

Example 81 . Machine Requirements. Technology: Availability:

- We simplify the availability requirements due to the apparent simplicity of the vehicle movement records and billings.
  - The complete handling of the recording or billing of a vehicle movement shall be done without interference from other recordings or billings ■

6.3.2.4 Integrity

Definition 49 Integrity: *A system has integrity — in the context of a machine being dependable — if*

- *it is and remains unimpaired,*
- *i.e., has no faults, errors and failures,*
- *and remains so, without these,*
- *even in the situations where the environment of the machine has faults, errors and failures* ■

- Integrity seems to be a highest form of dependability,
- i.e., a machine having integrity is 100% **dependable**!
- The machine is **sound** and is **incorruptible**.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

Example 82 . Machine Requirements. Technology: Integrity:

- We do not require an explicit formulation of integrity.
- We instead refer to the
  - reliability,
  - safety,
  - security and
  - robustness

  measures (below)

6.3.2.5 Reliability

Definition 50 Reliability: *A system being reliable — in the context of a machine being dependable — means*

- *some measure of continuous correct service,*
- *that is, measure of (mean) time to failure (MTTF)*

Example 83 . Machine Requirements. Technology: Reliability:

- A road pricing calculator shall have a MTTF of least 10^8 seconds or approx. 40 months.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

6.3.2.6 Safety

Definition 51 Safety: *By safety — in the context of a machine being dependable — we mean*

- *some measure of continuous delivery of service of*
  - *either correct service, or incorrect service after benign failure,*
- *that is: Measure of time to catastrophic failure* ■

Example 84 . Machine Requirements. Technology: Safety:

- The road pricing system, now including the
  - vehicle global position system and the
  - toll-gate sensors and barrier actuator
- shall have
  - a mean time to catastrophic failure
  - equal to the MTTF, 10^8 seconds ■

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

6.3.2.7 Security

We shall take a rather limited view of security. We are not including any consideration of security against brute-force terrorist attacks. We consider that an issue properly outside the realm of software engineering.

- Security, then, in our limited view, requires a notion of *authorised user*,
- with authorised users being fine-grained authorised to access only a well-defined subset of system resources (data, functions, etc.).
- An *unauthorised user* (for a resource) is anyone who is not authorised access to that resource.

Definition 52 Security: *A system being secure — in the context of a machine being dependable —*

- *means that an unauthorised user, after believing that he or she has had access to a requested system resource:*
  - *cannot find out what the system resource is doing,*
  - *cannot find out how the system resource is working*
  - *and does not know that he/she does not know!*
- *That is, prevention of unauthorised access to computing and/or handling of information (i.e., data)* ■

Example 85 . Machine Requirements. Technology: Security:

- We omit exemplifying road pricing system security ■

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

6.3.2.8 Robustness

Definition 53 Robustness: *A system is robust — in the context of dependability —*

- *if it retains its attributes*
  - *after failure, and*
  - *after maintenance*

- Thus a robust system is “stable”
  - across failures
  - and “across” possibly intervening “repairs”
  - and “across” other forms of maintenance.

Example 86 . Machine Requirements. Technology: Robustness:

- We restrict ourselves to consider only the software of the road pricing system.
  - For every instance of
    - restart after failure
    - it shall be verified
    - that all attributes have retained their appropriate values;
  - and for every instance of
    - software maintenance, see Sect. ,
    - the whole system shall be verified, i.e.,
      - tested,
      - model checked and
      - proven correct,
    - to the same and full extent that the original system delivery was verified ■

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

6.3.2.9 Discussion:

◇ TO BE WRITTEN

◇

6.4. Development Requirements

Definition 54 Development Requirement:

- *By* **development requirements** *we shall understand*
  - *(i) process requirements (Sect. 6.4.1),*
  - *(ii) maintenance requirements (Sect. 6.4.2),*
  - *(iii) platform requirements (Sect. 6.4.3),*
  - *(iv) management requirements (Sect. 6.4.4) and*
  - *(v) documentation requirements (Sect. 6.4.5)* ■

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

6.4.1. Process Requirements

Definition 55 Process Requirement:

- *By a* **development process requirements** *we shall understand requirements which are concerned with the development process to be followed by the development engineers:*
  - *whether pursuing formal methods, and to which degree, and (compatibly)/or*
  - *whether pursuing best practices, and possible details thereof, and (compatibly)/or*
  - *whether adhering otherwise to established, e.g., IEEE standards,*
  - *etcetera* ■

Example 87 . Machine Requirements. Development: Road Pricing System:

- The road pricing system is to be developed
  - according to the triptych approach;
  - based on, or developing itself, a generic transport domain description,
    as per the approach outlined in [Bjø16b],
    expressing suitable predicates about that description,
    and testing, model checking and proving satisfaction of these verifications;
  - accurately detailing the requirements prescriptions as per the approach outlined in [Bjø16a, this paper (!)];
  - etcetera;
  - finally testing, model checking and proving satisfaction of $\mathcal{D}, \mathcal{S} \models \mathcal{R}$.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

6.4.2. Maintenance Requirements

Definition 56 Maintenance Requirements: *By maintenance requirements we understand a combination of requirements with respect to:*

- *adaptive maintenance,*
- *corrective maintenance,*
- *perfective maintenance,*
- *preventive maintenance and*
- *extensional maintenance* ■

- Maintenance of building, mechanical, electrotechnical and electronic artifacts — i.e., of artifacts based on the natural sciences — is based both on documents and on the presence of the physical artifacts.
- Maintenance of software is based just on software, that is, on all the documents (including tests) entailed by software — see Definition 69 on Slide 693.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

6.4.2.1 Adaptive Maintenance

Definition 57 Adaptive Maintenance: *By adaptive maintenance we understand such maintenance*

- *that changes a part of that software so as to also, or instead, fit to*
  - *some other software, or*
  - *some other hardware equipment*

  (i.e., other software or hardware which provides new, respectively replacement, functions) ■

Example 88 . Machine Requirements. Development: Adaptive Maintenance:

- Road pricing system adaptive maintenance shall conclude with a full set of successful
  - formal software tests,
  - model checks, and
  - correctness proofs ■

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

6.4.2.2 Corrective Maintenance

Definition 58 Corrective Maintenance: *By corrective maintenance we understand such maintenance which*

- *corrects a software error* ■

Example 89 . Machine Requirements. Development: Corrective Maintenance:

- Road pricing system corrective maintenance shall conclude with a full set of successful
  - formal software tests,
  - model checks, and
  - correctness proofs ■

6.4.2.3 Perfective Maintenance

Definition 59 Perfective Maintenance: *By perfective maintenance we understand such maintenance which*

- *helps improve (i.e., lower) the need for*
- *hardware storage, time and (hard) equipment* ■

Example 90 . Machine Requirements. Development: Perfective Maintenance:

- Road pricing system perfective maintenance shall conclude with a full set of successful
  - formal software tests,
  - model checks, and
  - correctness proofs ■

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

6.4.2.4 Preventive Maintenance

Definition 60 Preventive Maintenance: *By preventive maintenance we understand such maintenance which*

- *helps detect, i.e., forestall, future occurrence*
- *of software or hardware failures* ■

Example 91 . Machine Requirements. Development: Preventive Maintenance:

- Road pricing system preventive maintenance shall conclude with a full set of successful
  - formal software tests,
  - model checks, and
  - correctness proofs ■

6.4.2.5 Extensional Maintenance

Definition 61 Extensional Maintenance: *By extensional maintenance we understand such maintenance which adds new functionalities to the software, i.e., which implements additional requirements* ■

Example 92 . **Machine Requirements. Development: Extensional Maintenance**:

- Road pricing system extensional maintenance shall conclude with a full set of successful
  - formal software tests,
  - model checks, and
  - correctness proofs ■

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

6.4.3. Platform Requirements

6.4.3.1 Delineation and Facets of Platform Requirements

Definition 62 Platform: *By a [computing] platform is here understood*

- *a combination of hardware and systems software*
- *so equipped as to be able to develop and execute software,*
- *in one form or another* ■

- What the “in one form or another” is
- transpires from the next characterisation.

Definition 63 Platform Requirements: *By platform requirements we mean a combination of the following:*

- *execution platform requirements,*
- *demonstration platform requirements,*
- *development platform requirements and*
- *maintenance platform requirements*

6.4.3.2 Execution Platform

Definition 64 **Execution Platform Requirements:** *By execution platform requirements we shall understand such machine requirements which*

- *detail the specific (other) software and hardware*
- *for the platform on which the software*
- *is to be executed* ∎

6.4.3.3 Demonstration Platform

Definition 65 Demonstration Platform Requirements: *By demonstration platform requirements we shall understand such machine requirements which*

- *detail the specific (other) software and hardware*
- *for the platform on which the software*
- *is to be demonstrated to the customer — say for acceptance tests, or for management demos, or for user training* ■

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

6.4.3.4 Development Platform

Definition 66 Development Platform Requirements: *By development platform requirements we shall understand such machine requirements which*

- *detail the specific software and hardware*
- *for the platform on which the software*
- *is to be developed* ■

6.4.3.5 Maintenance Platform

Definition 67 Maintenance Platform Requirements: *By maintenance platform requirements we shall understand such machine requirements which*

- *detail the specific (other) software and hardware*
- *for the platform on which the software*
- *is to be maintained* ■

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

Example 93 . Machine Requirements. Development: Platform Requirements:

- The road pricing system platform requirements are: the system shall
  - executed and demonstrated on [to be detailed] and
  - developed and maintained on [to be detailed] ■
  -

6.4.4. Management Requirements

Definition 68 Management Requirements:

- *By* **management requirements** *we shall understand requirements that express*
  - ⟐
  - ⟐
  - ⟐ *[Bjø11a, Believable Software Management]*

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

Example 94 . Machine Requirements. Development: Management:

- ⟐ TO BE WRITTEN

6.4.5. Documentation Requirements

Definition 69 Software: *By* **software** *we shall understand*

- *not only* **code** *that may be the basis for executions by a computer,*
- *but also its full* **development documentation***:*
  - *the stages and steps of* **application domain description,**
  - *the stages and steps of* **requirements prescription,** *and*
  - *the stages and steps of* **software design** *prior to code,*

with all of the above including all validation *and* verification *(incl.,* formal test *[test model, test suite, test result, etc.],* model-checking *and* proof*) documents.*

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

- *In addition, as part of our wider concept of software, we also include a comprehensive collection of supporting documents:*
  - **training manuals,**
  - **installation manuals,**
  - **user manuals,**
  - **maintenance manuals,** *and*
  - **development and maintenance logbooks.**

Definition 70 Documentation Requirements: *By documentation requirements*

- *we mean requirements*
- *of any of the software documents*
- *that together make up*
  - *software and*
  - *hardware*[45] ■

Example 95 . Machine Requirements. Development: Documentation:

- The road pricing system documentation requirements shall include
  - all of the software documents
  - implied by Definition 69 on Slide 693 above ■

[45]— we omit a definition of what we mean by hardware such as the one we gave for software, cf. Definition 69 on Slide 693.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

6.5. Discussion

TO BE TYPED

7. Conclusion

- We conclude by reviewing
  - what has been achieved,
  - present shortcomings,
  - a few words on relations to "classical requirements engineering", and
  - possible research challenges.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

7.1. What has been Achieved ?

- We have put forward a "new approach" to requirements engineering.
  - The "newness" comes from its reliance on there being a reasonably "complete" domain description already at hand.
- We refer to the introductory section, Sect. 1.2 for a "repeat" of what we think is our contribution.
- We will, in this section of the lecture examine some issues of requirements engineering.

- (i) The field of software requirements specification has yet to find a stable form
  - around which different contributors (book, paper and Web page authors)
    - structure and within which they
    - express

    their contributions.

- It seems, to this author, that there is a bewildering "culture" of different "schools".
  - The various elements of a prevailing such "school" are named:
    - (a) user requirements,
    - (b) system requirements,
    - (c) functional requirements and
    - (d) non-functional requirements.
  - But, to this author, it is hard to see the relations between any pair of these four "classes".
  - Our decomposition into domain, machine and interface requirements is clearly related to either the domain, or the machine or both.

- (ii) We have shown how **requirements engineering** structured into:
    - * *Problem, Solution and Objective Sketch*
    - * *System Requirements*
    - * *User and External Equipment Requirements*
  - ⟐ *Domain Requirements*
    - ⚭ *Projection & Simplification*
    - ⚭ *Instantiation*
    - ⚭ *Determination*
    - ⚭ *Extension*
    - ⚭ *Fitting*

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

- *Interface Requirements*
  - *Shared Endurants*
    - *Intialisation*
    - *Refreshment*
  - *Shared Actions*
  - *Shared Events*
  - *Shared Behaviours*

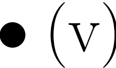
- *Machine Requirements*
  - *Derived Requirements*
  - *Technology Requirements*
    - *Performance*
    - *Dependability*[46]
  - *Development Requirements*
    - *Process Reqs.*
    - *Maintenance Reqs.*[47]
    - *Platform Reqs.*[48]
    - *Management Reqs.*
    - *Documentation Reqs.*

[46] Accessibility, Availability, Integrity, Reliability, Safety, Security, Robustness
[47] Adaptive, Corrective, Perfective,Preventive, Extensional
[48] Execution, Development, Demonstration, Development, Maintenance

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

- (iii) The above-hinted structuring, ((ii)), is logically motivated
  - and is, we claim, a definite contribution to the field of requirements engineering,
  - providing it, we claim, with a stable form.

- (iv) The rôle of
  - **domain engineering** in **software engineering**,
  - together with **domain requirements** and **interface requirements**,
  - is to ensure that the software meets client expectations.
  - By avoiding requirements that express machine concepts the software does not exhibit such properties that make it "user-unfriendly".
  - 'User-friendly' software reflects only concepts that relate to domain phenomena.



- (vi)
  - ◈
  - ◈

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

7.2. Present Shortcomings

- ◈
  - ◈
  - ◈
  - ◈

7.3. Comparison to "Classical" Requirements Engineering

- ◈ [van09]
  ◈ [Lau02]
  ◈
  ◈

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

7.4. Future Work: Research Challenges

- We have outlined three major stages of requirements development, and, within these, a number of steps.
- They can be used, we claim, to advantage, already now —
  as they have indeed been used over the years
  in projects with which we have been associated.
- But more experimental research and path-finder projects has to be absolved.

- Section 3.3 of the *Manifest Domains* “chapter” of these lectures covered ‘open problems’ with respect to domain science & engineering. So we shall only mention ‘open problems’ with respect to requirements engineering.
- A number of research issues need be studied. We list a few.
  - **Domain Facets:**
    - Here we are interested in how various domain facets may give rise to special domain-to-requirements “derivation” principles, techniques and tools.
    - We refer to [Bjø10a, 2008].

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

⟐ Formal Aspects of Domain Requirements Operations:

ↂ

⟐

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

Thanks

-
-
-

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

7.5. Acknowledgments

-

8. Bibliography

8.1. Bibliographical Notes

8.1.1. Published Papers

- Web page **www.imm.dtu.dk/~dibj/domains/** lists the published papers and reports mentioned below.
- I have thought about domain engineering for more than 25 years.
- But serious, focused writing only started to appear since [Bjø06, Part IV] — with [Bjø03, Bjø97] being exceptions:
  - [Bjø07, 2007] suggests a number of domain science and engineering research topics;
  - [Bjø10a, 2008] covers the concept of domain facets;
  - [BE10, 2008] explores compositionality and Galois connections.
  - [Bjø08, Bjø10c, 2008,2009] show how to systematically, but, of course, not automatically, "derive" requirements prescriptions from domain descriptions;

- [Bjø11a, 2008] takes the triptych software development as a basis for outlining principles for believable software management;
- [Bjø09, Bjø14a, 2009,2013] presents a model for Stanisław Leśniewski's [CV99] concept of mereology;
- [Bjø10b, Bjø11b] present an extensive example and is otherwise a precursor for the present paper;
- [Bjø11c, 2010] presents, based on the `TripTych` view of software development as ideally proceeding from domain description via requirements prescription to software design, concepts such as software demos and simulators;

- [Bjø13, 2012] analyses the **TripTych**, especially its domain engineering approach, with respect to Maslow's [49] and Peterson's and Seligman's [50] notions of humanity: how can computing relate to notions of humanity;
- the first part of [Bjø14b, 2014] is a precursor for the present paper with its second part presenting a first formal model of the elicitation process of analysis and description based on the prompts more definitively presented in the current paper; and
- [Bjø14c, 2014] focus on domain safety criticality.

The present paper basically replaces the domain analysis and description section of all of the above reference — including [Bjø06, Part IV, 2006].

[49] *Theory of Human Motivation.* Psychological Review 50(4) (1943):370-96; and *Motivation and Personality*, Third Edition, Harper and Row Publishers, 1954.
[50] *Character strengths and virtues: A handbook and classification.* Oxford University Press, 2004

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

8.1.2. Reports

We list a number of reports all of which document descriptions of domains. These descriptions were carried out in order to research and develop the domain analysis and description concepts now summarised in the present paper. These reports ought now be revised, some slightly, others less so, so as to follow all of the prescriptions of the current paper. Except where a URL is given in full, please prefix the web reference with: `http://www2.compute.dtu.dk/~dibj/`.

1 *A Railway Systems Domain:* `http://euler.fd.cvut.cz/railwaydomain/` (2003)

2 *Models of IT Security. Security Rules & Regulations:* `it-security.pdf` (2006)

3 *A Container Line Industry Domain:* `container-paper.pdf` (2007)

4 *The "Market": Consumers, Retailers, Wholesalers, Producers:* `themarket.pdf` (2007)

5 *What is Logistics ?:* `logistics.pdf` (2009)

6 *A Domain Model of Oil Pipelines:* `pipeline.pdf` (2009)

7 *Transport Systems:* `comet/comet1.pdf` (2010)

8 *The Tokyo Stock Exchange:* `todai/tse-1.pdf` and `todai/tse-2.pdf` (2010)

9 *On Development of Web-based Software. A Divertimento:* `wfdftp.pdf` (2010)

10 *Documents (incomplete draft):* `doc-p.pdf` (2013)

8.2. References

[Abr09] Jean-Raymond Abrial. The B Book: Assigning Programs to Meanings *and* Modeling in Event-B: System and Software Engineering. Cambridge University Press, Cambridge, England, 1996 and 2009.

[Aud95] Rober Audi. *The Cambridge Dictionary of Philosophy*. Cambridge University Press, The Pitt Building, Trumpington Street, Cambridge CB2 1RP, England, 1995.

[Bac69] C. Bachman. Data structure diagrams. *Data Base, Journal of ACM SIGBDP*, 1(2), 1969.

[Bad05] Alain Badiou. *Being and Event*. Continuum, 2005. (Lêtre et l'événements, Edition du Seuil, 1988).

[BDS04] Thomas Bittner, Maureen Donnelly, and Barry Smith. Endurants and Perdurants in Directly Depicting Ontologies. *AI Communications*, 17(4):247–258, December 2004. IOS Press, in [RG04].

[BE10] Dines Bjørner and Asger Eir. Compositionality: Ontology and Mereology of Domains. Some Clarifying Observations in the Context of Software Engineering in July 2008, eds. Martin Steffen, Dennis Dams and Ulrich Hannemann. In *Festschrift for Prof. Willem Paul de Roever Concurrency, Compositionality, and Correctness*, volume 5930 of *Lecture Notes in Computer Science*, pages 22–59, Heidelberg, July 2010. Springer.

[BF98] V. Richard Benjamins and Dieter Fensel. The Ontological Engineering Initiative (KA)2. Internet publication + Formal Ontology in Information Systems, University of Amsterdam, SWI, Roetersstraat 15, 1018 WB Amsterdam, The Netherlands and University of Karlsruhe, AIFB, 76128 Karlsruhe, Germany, 1998. http://www.aifb.uni-karlsruhe.de/WBS/broker/KA2.htm.

[BGH+in] Dines Bjørner, Chris W. George, Anne Eliabeth Haxthausen, Christian Krog Madsen, Steffen Holmslykke, and Martin Pěnička. "UML"–ising Formal Techniques. In *INT 2004: Third International Workshop on Integration of Specification Techniques for Applications in Engineering*, volume 3147 of *Lecture Notes in Computer Science*, pages 423–450. Springer–Verlag, 28 March 2004, ETAPS, Barcelona, Spain. Final Version.

[BJ78] Dines Bjørner and Cliff B. Jones, editors. *The Vienna Development Method: The Meta-Language*, volume 61 of *LNCS*. Springer, 1978.

[BJ82] Dines Bjørner and Cliff B. Jones, editors. *Formal Specification and Software Development*. Prentice-Hall, 1982.

[Bjø97] Dines Bjørner. Michael Jackson's Problem Frames: Domains, Requirements and Design. In Li ShaoYang and Michael Hinchley, editors, *ICFEM'97: International Conference on Formal Engineering Methods*, Los Alamitos, November 12–14 1997. IEEE Computer Society. Final Version.

[Bjø03] Dines Bjørner. Domain Engineering: A "Radical Innovation" for Systems and Software Engineering ? In *Verification: Theory and Practice*, volume 2772 of *Lecture Notes in Computer Science*, Heidelberg, October 7–11 2003. Springer–Verlag. The Zohar Manna International Conference, Taormina, Sicily 29 June – 4 July 2003. .

[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. Domain Theory: Practice and Theories, Discussion of Possible Research Topics. In *ICTAC'2007*, volume 4701 of *Lecture Notes in Computer Science (eds. J.C.P. Woodcock et al.)*, pages 1–17, Heidelberg, September 2007. Springer.

[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.

[Bjø09] 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.

[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. Domain Science & Engineering – *From Computer Science to The Sciences of Informatics, Part I of II: The Engineering Part*. *Kibernetika i sistemny analiz*, (4):100–116, May 2010.

 © Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

[Bjø10c] Dines Bjørner. The Rôle of Domain Engineering in Software Development. Why Current Requirements Engineering Seems Flawed! In *Perspectives of Systems Informatics*, volume 5947 of *Lecture Notes in Computer Science*, pages 2–34, Heidelberg, Wednesday, January 27, 2010. Springer.

[Bjø11a] Dines Bjørner. Believable Software Management. *Encyclopedia of Software Engineering*, 1(1):1–32, 2011.

[Bjø11b] Dines Bjørner. Domain Science & Engineering – *From Computer Science to The Sciences of Informatics Part II of II: The Science Part*. *Kibernetika i sistemny analiz*, (2):100–120, May 2011.

[Bjø11c] 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.

[Bjø13] Dines Bjørner. *Domain Science and Engineering as a Foundation for Computation for Humanity*, chapter 7, pages 159–177. Computational Analysis, Synthesis, and Design of Dynamic Systems. CRC [Francis & Taylor], 2013. (eds.: Justyna Zander and Pieter J. Mosterman).

[Bjø14a] 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.

[Bjø14b] Dines Bjørner. Domain Analysis: Endurants – An Analysis & Description Process Model. In Shusaku Iida, José Meseguer, and Kazuhiro Ogata, editors, *Specification, Algebra, and Software: A Festschrift Symposium in Honor of Kokichi Futatsugi*. Springer, May 2014.

[Bjø14c] Dines Bjørner. Domain Engineering – A Basis for Safety Critical Software. Invited Keynote, ASSC2014: Australian System Safety Conference, Melbourne, 26–28 May, December 2014.

[Bjø16a] Dines Bjørner. From Domain Descriptions to Requirements Prescriptions – A Different Approach to Requirements Engineering. *Submitted for consideration by Formal Aspects of Computing*, 2016.

[Bjø16b] Dines Bjørner. Manifest Domains: Analysis & Description. *Expected published by Formal Aspects of Computing*, 2016.

[BN92] Dines Bjørner and Jørgen Fischer Nilsson. Algorithmic & Knowledge Based Methods — Do they "Unify" ? In *International Conference on Fifth Generation Computer Systems: FGCS'92*, pages 191–198. ICOT, June 1–5 1992.

[BRJ98] Grady Booch, Jim Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley, 1998.

[CE00] Krzysztof Czarnecki and Ulrich W. Eisenecker. *Generative Programming: Methods, Tools, and Applications*. Addison Wesley, 2000.

[Che76] Peter P. Chen. The Entity-Relationship Model - Toward a Unified View of Data. *ACM Trans. Database Syst*, 1(1):9–36, 1976.

[Cod70] E. F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, June 1970.

[CV96] Roberto Casati and Achille C. Varzi, editors. *Events*. Ashgate Publishing Group – Dartmouth Publishing Co. Ltd., Wey Court East, Union Road, Farnham, Surrey, GU9 7PT, United Kingdom, 23 March 1996.

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

[CV10] Roberto Casati and Achille Varzi. Events. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Spring 2010 edition, 2010.

[Dav80] Donald Davidson. *Essays on Actions and Events*. Oxford University Press, 1980.

[Dre67] F. Dretske. Can Events Move? *Mind*, 76(479-492), 1967. Reprinted in [CV96, 1996], pp. 415-428.

[DT97] Merlin Dorfman and Richard H. Thayer, editors. *Software Requirements Engineering*. IEEE Computer Society Press, 1997.

[FL98] John Fitzgerald and Peter Gorm Larsen. *Modelling Systems – Practical Tools and Techniques in Software Development*. Cambridge University Press, The Edinburgh Building, Cambridge CB2 2RU, UK, 1998. ISBN 0-521-62348-0.

[FM83] Edward A. Feigenbaum and Pamela McCorduck. *The fifth generation*. Addison-Wesley, Reading, MA, USA, 1st ed. edition, 1983.

[FMMR12] Carlo A. Furia, Dino Mandrioli, Angelo Morzenti, and Matteo Rossi. *Modeling Time in Computing*. Monographs in Theoretical Computer Science. Springer, 2012.

[Fow20] Martin Fowler. *Domain Specific Languages*. Signature Series. Addison Wesley, October 20120.

[GGJZ00] Carl A. Gunter, Elsa L. Gunter, Michael A. Jackson, and Pamela Zave. A Reference Model for Requirements and Specifications. *IEEE Software*, 17(3):37–43, May–June 2000.

[GHH+92] Chris W. George, Peter Haff, Klaus Havelund, Anne Elisabeth Haxthausen, Robert Milne, Claus Bendix Nielsen, Søren Prehn, and Kim Ritter Wagner. *The RAISE Specification Language*. The BCS Practitioner Series. Prentice-Hall, Hemel Hampstead, England, 1992.

[GHH+95] Chris W. George, Anne Elisabeth Haxthausen, Steven Hughes, Robert Milne, Søren Prehn, and Jan Storbank Pedersen. *The RAISE Development Method*. The BCS Practitioner Series. Prentice-Hall, Hemel Hampstead, England, 1995.

[GLMS02] T. Grötker, S. Liao, G. Martin, and S. Swan. *System Design with SystemC*. Kluwer, Dordrecht, 2002.

[GW99] Bernhard Ganter and Rudolf Wille. *Formal Concept Analysis — Mathematical Foundations*. Springer-Verlag, January 1999.

[Hac82] P.M.S. Hacker. Events and Objects in Space and Time. *Mind*, 91:1–19, 1982. reprinted in [CV96], pp. 429-447.

[Har87] David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, 1987.

[Hay09] Dan Haywood. *Domain-Driven Design Using Naked Objects*. The Pragmatic Bookshelf (an imprint of 'The Pragmatic Programmers, LLC.'), http://pragprog.com/, 2009.

[Hoa85] Charles Anthony Richard 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).

[HPK11] A.E. Haxthausen, J. Peleska, and S. Kinder. A formal approach for the construction and and verification of railway control systems. *Formal Aspects of Computing*, 23:191–219, 2011.

[IT99] ITU-T. CCITT Recommendation Z.120: Message Sequence Chart (MSC), 1992, 1996, 1999.

[Jac95a] Michael A. Jackson. *Software Requirements & Specifications: a lexicon of practice, principles and prejudices*. ACM Press. Addison-Wesley, Reading, England;, 1995.

[Jac95b] Michael A. Jackson. *Software Requirements & Specifications: a lexicon of practice, principles and prejudices*. ACM Press. Addison-Wesley, Reading, England, 1995.

[Jac01] Michael A. Jackson. *Problem Frames — Analyzing and Structuring Software Development Problems*. ACM Press, Pearson Education. Addison-Wesley, England, 2001.

[Jac06] Daniel Jackson. *Software Abstractions: Logic, Language, and Analysis*. The MIT Press, Cambridge, Mass., USA, April 2006. ISBN 0-262-10114-9.

[Jac10] Michael A. Jackson. Program Verification and System Dependability. In Paul Boca and Jonathan Bowen, editors, *Formal Methods: State of the Art and New Directions*, pages 43–78, London, UK, 2010. Springer.

[JBR99] Ivar Jacobson, Grady Booch, and Jim Rumbaugh. *The Unified Software Development Process*. Addison-Wesley, 1999.

[Joh05] Ingvar Johansson. Qualities, Quantities, and the Endurant-Perdurant Distinction in Top-Level Ontologies. In Dengel A. Bergmann R. Nick M. Roth-Berghofer Th. Althoff, K.-D., editor, *Professional Knowledge Management WM 2005*, volume 3782 of *Lecture Notes in Artificial Intelligence*, pages 543–550. Springer, 2005. 3rd Biennial Conference, Kaiserslautern, Germany, April 10-13, 2005, Revised Selected Papers.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

[KCH+90] K.C. Kang, S.G. Cohen, J.A. Hess, W.E. Novak, and A.S Peterson. FODA: Feature-Oriented Domain Analysis. Feasibility Study CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie Mellon University, November 1990. http://www.sei.cmu.edu/library/abstracts/reports/90tr021.cfm.

[Kim93] Jaegwon Kim. *Supervenience and Mind*. Cambridge University Press, 1993.

[Lau02] Søren Lauesen. *Software Requirements - Styles and Techniques*. Addison-Wesley, UK, 2002.

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

[LWZ13] Zhiming Liu, J. C. P. Woodcock, and Huibiao Zhu, editors. *Unifying Theories of Programming and Formal Engineering Methods - International Training School on Software Engineering, Held at ICTAC 2013, Shanghai, China, August 26-30, 2013, Advanced Lectures*, volume 8050 of *Lecture Notes in Computer Science*. Springer, 2013.

[MC04] Neno Medvidovic and Edward Colbert. Domain-Specific Software Architectures (DSSA). Power Point Presentation, found on The Internet, Absolute Software Corp., Inc.: `Abs[S/W]`, 5 March 2004.

[Mel80] D.H. Mellor. Things and Causes in Spacetime. *British Journal for the Philosophy of Science*, 31:282–288, 1980.

[Mer04] Merriam Webster Staff. Online Dictionary: `http://www.m-w.com/home.htm`, 2004. Merriam–Webster, Inc., 47 Federal Street, P.O. Box 281, Springfield, MA 01102, USA.

[MG92] Erik Mettala and Marc H. Graham. The Domain Specific Software Architecture Program. Project Report CMU/SEI-92-SR-009, Software Engineering Institute Carnegie Mellon University Pittsburgh, Pennsylvania 15213, June 1992.

[MHS05] Marjan Mernik, Jan Heering, and Anthony M. Sloane. When and how to develop domain-specific languages. *ACM Computing Surveys*, 37(4):316–344, December 2005.

[Nei84] James M. Neighbors. The Draco Approach to Constructing Software from Reusable Components. *IEEE Transactions of Software Engineering*, SE-10(5), September 1984.

[PD87] Rubén Prieto-Díaz. Domain Analysis for Reusability. In *COMPSAC 87*. ACM Press, 1987.

[PD90] Rubén Prieto-Díaz. Domain analysis: an introduction. *Software Engineering Notes*, 15(2):47–54, 1990.

[PDA91] Rubén Prieto-Díaz and Guillermo Arrango. *Domain Analysis and Software Systems Modelling*. IEEE Computer Society Press, 1991.

[Pfl01] Shari Lawrence Pfleeger. *Software Engineering, Theory and Practice*. Prentice–Hall, 2nd edition, 2001.

[Pi99] Chia-Yi Tony Pi. *Mereology in Event Semantics*. Phd, McGill University, Montreal, Canada, August 1999.

[Pre01] Roger S. Pressman. *Software Engineering, A Practitioner's Approach*. International Edition, Computer Science Series. McGraw–Hill, 5th edition, 1981–2001.

[Qui79] A. Quinton. Objects and Events. *Mind*, 88:197–214, 1979.

[Rei10] Wolfang Reisig. *Petrinetze: Modellierungstechnik, Analysemethoden, Fallstudien*. Leitfäden der Informatik. Vieweg+Teubner, 1st edition, 15 June 2010. 248 pages; ISBN 978-3-8348-1290-2.

[RG04] Jochen Renz and Hans W. Guesgen, editors. *Spatial and Temporal Reasoning*, volume 14, vol. 4, Journal: AI Communications, Amsterdam, The Netherlands, Special Issue. IOS Press, December 2004.

[RJB98] Jim Rumbaugh, Ivar Jacobson, and Grady Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1998.

[SG96] Mary Shaw and David Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, 1996.

[Som06] Ian Sommerville. *Software Engineering*. Pearson, 8th edition, 2006.

[Spi01] Diomidis Spinellis. Notable design patterns for domain specific languages. *Journal of Systems and Software*, 56(1):91–99, February 2001.

[Sta99] Staff of Encyclopœdia Brittanica. Encyclopœdia Brittanica. Merriam Webster/Brittanica: Access over the Web: http://www.eb.com:180/, 1999.

[Tra94] Will Tracz. Domain-specific software architecture (DSSA) frequently asked questions (FAQ). *Software Engineering Notes*, 19(2):52–56, 1994.

[van91] Johan van Benthem. *The Logic of Time*, volume 156 of *Synthese Library: Studies in Epistemology, Logic, Methhodology, and Philosophy of Science (Editor: Jaakko Hintika)*. Kluwer Academic Publishers, P.O.Box 17, NL 3300 AA Dordrecht, The Netherlands, second edition, 1983, 1991.

[van09] Axel van Lamsweerde. *Requirements Engineering: From System Goals to UML Models to Software Specifications*. Wiley, 2009.

[WD96] J. C. P. Woodcock and J. Davies. *Using Z: Specification, Proof and Refinement.* Prentice Hall International Series in Computer Science, 1996.

[WS12] George Wilson and Samuel Shpall. Action. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Summer 2012 edition, 2012.

[WYZ94] Ji Wang, XinYao Yu, and Chao Chen Zhou. Hybrid Refinement. Research Report 20, UNU/IIST, P.O.Box 3058, Macau, 1. April 1994.

[ZWZ13] Naijun Zhan, Shuling Wang, and Hengjun Zhao. Formal modelling, analysis and verification of hybrid systems. In *ICTAC Training School on Software Engineering*, pages 207–281, 2013.

 © Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

Contents

MANIFEST DOMAINS 9
1. Introduction 17
1.1. The TripTych Approach to Software Engineering 20
1.2. Method and Methodology 26
1.2.1. Method 26
1.2.2. Discussion 28
1.2.3. Methodology 33
1.3. Computer and Computing Science 34
1.4. What Is a Manifest Domain ? 37
1.5. What Is a Domain Description ? 44
1.6. Towards a Methodology of Domain Analysis & Description 48
1.6.1. Practicalities of Domain Analysis & Description. 48
1.6.2. The Four Domain Analysis & Description "Players". 51
1.6.3. An Interactive Domain Analysis & Description Dialogue. 58
1.6.4. Prompts 59
1.6.5. A Domain Analysis & Description Language. 61
1.6.6. The Domain Description Language. 62
1.6.7. Domain Descriptions: Narration & Formalisation 63
1.7. One Domain – Many Models ? 64
1.8. Formal Concept Analysis 69
1.8.1. A Formalisation 70
1.8.2. Types Are Formal Concepts 76
1.8.3. Practicalities 77
1.8.4. Formal Concepts: A Wider Implication 79
1.9. Structure of Seminar 80
2. Entities 83
2.1. General 83
a: Analysis Prompt: is-entity 84
2.2. Endurants and Perdurants 86
b: Analysis Prompt: is-endurant 90
c: Analysis Prompt: is-perdurant 90

2.3. Discrete and Continuous Endurants . . . 93
d: Analysis Prompt: is discrete . . . 98
e: Analysis Prompt: is continuous . . . 98
2.4. An Upper Ontology Diagram of Domains . . . 99
3. Endurants . . . 101
3.1. Parts, Components and Materials . . . 102
3.1.1. General . . . 102
3.1.2. Part, Component and Material Analysis Prompts . . . 110
f: Analysis Prompt: is part . . . 110
g: Analysis Prompt: is component . . . 111
h: Analysis Prompt: is material . . . 112
3.1.3. Atomic and Composite Parts . . . 113
i: Analysis Prompt: is-atomic . . . 119
j: Analysis Prompt: is-composite . . . 119
3.1.4. On Observing Part Sorts and Types . . . 121
3.1.5. On Discovering Part Sorts . . . 122
k: Analysis Prompt: observe-parts . . . 124
3.1.6. Part Sort Observer Functions . . . 127
1: Description Prompt: observe-part-sorts . . . 128
3.1.7. On Discovering Concrete Part Types . . . 133
l: Analysis Prompt: has-concrete-type . . . 133
2: Description Prompt: observe-part-type . . . 134
3.1.8. Forms of Part Types . . . 138
3.1.9. Part Sort and Type Derivation Chains . . . 139
3.1.10. Names of Part Sorts and Types . . . 141
3.1.11. More On Part Sorts and Types . . . 145
3.1.12. External and Internal Qualities of Parts . . . 148
3.1.13. Three Categories of Internal Qualities . . . 149
3.2. Unique Part Identifiers . . . 151
3: Description Prompt: observe-unique-identifier . . . 153
3.3. Mereology . . . 156
3.3.1. Part Relations . . . 157
3.3.2. Part Mereology: Types and Functions . . . 159

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

m: Analysis Prompt: has-mereology . . . 159
4: Description Prompt: observe-mereology . . . 161
3.3.3. Formulation of Mereologies . . . 169
3.4. Part Attributes . . . 170
3.4.1. Inseparability of Attributes from Parts . . . 171
3.4.2. Attribute Quality and Attribute Value . . . 172
3.4.3. Endurant Attributes: Types and Functions . . . 174
n: Analysis Prompt: attribute-names . . . 177
5: Description Prompt: observe-attributes . . . 180
3.4.4. Attribute Categories . . . 186
3.4.5. Access to Attribute Values . . . 194
3.4.6. Event Values . . . 196
3.4.7. Shared Attributes . . . 198
Example 36: Shared Attributes . . . 200
3.4.8. Master/Slave Shared Attribute Relations . . . 203
3.5. Components . . . 207
o: Analysis Prompt: has-components . . . 208
6: Description Prompt: observe-component-sorts . . . 209
3.6. Materials . . . 213
p: Analysis Prompt: has-materials . . . 214
7: Description Prompt: observe-material-sorts . . . 215
3.6.1. Materials-related Part Attributes . . . 218
3.6.2. Laws of Material Flows and Leaks . . . 222
3.7. "No Junk, No Confusion" . . . 228
3.8. Discussion of Endurants . . . 232
4. Perdurants . . . 235
4.1. States . . . 238
4.2. Actions, Events and Behaviours . . . 240
4.2.1. Time Considerations . . . 241
4.2.2. Actors . . . 243
4.2.3. Parts, Attributes and Behaviours . . . 245
4.3. Discrete Actions . . . 246
4.4. Discrete Events . . . 248

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

4.5. Discrete Behaviours . . . 251
4.5.1. Channels and Communication . . . 253
4.5.2. Relations Between Attribute Sharing and Channels . . . 255
4.6. Continuous Behaviours . . . 259
4.7. Attribute Value Access . . . 262
4.7.1. Access to Static Attribute Values . . . 263
4.7.2. Access to External Attribute Values . . . 264
4.7.3. Access to Biddable and Programmable Attribute Values . . . 268
4.7.4. Access to Event Values . . . 269
4.8. Perdurant Signatures and Definitions . . . 270
4.9. Action Signatures and Definitions . . . 272
4.10. Event Signatures and Definitions . . . 279
4.11. Discrete Behaviour Signatures and Definitions . . . 283
4.11.1. Behaviour Signatures . . . 283
4.11.2. Behaviour Definitions . . . 288
Process Schema I: Abstract `is_composite(p)` . . . 289
Process Schema II: Concrete `is_composite(p)` . . . 291
Process Schema III: `is_atomic(p)` . . . 291
Process Schema IV: Core Process (I) . . . 296
Process Schema V: Core Process (II) . . . 299
4.12. Concurrency: Communication and Synchronisation . . . 307
4.13. Summary and Discussion of Perdurants . . . 308
4.13.1. Summary . . . 309
4.13.2. Discussion . . . 310
5. Closing . . . 311
5.1. Analysis & Description Calculi for Other Domains . . . 312
5.2. On Domain Description Languages . . . 314
5.3. Comparison to Other Work . . . 317
5.3.1. Background: The TripTych Domain Ontology . . . 317
5.3.2. General . . . 319
5.3.2.1. Ontology Science & Engineering . . . 319
5.3.2.2. Knowledge Engineering . . . 329
5.3.3. Specific . . . 336

 © Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

5.3.3.1. Database Analysis . . . 336
5.3.3.2. Domain Analysis . . . 337
5.3.3.3. Domain Specific Languages . . . 340
5.3.3.4. Feature-oriented Domain Analysis (FODA) . . . 342
5.3.3.5. Software Product Line Engineering . . . 344
5.3.3.6. Problem Frames . . . 346
5.3.3.7. Domain Specific Software Architectures (DSSA) . . . 349
5.3.3.8. Domain Driven Design (DDD) . . . 354
5.3.3.9. Unified Modeling Language (UML) . . . 356
5.3.3.10. Requirements Engineering . . . 360
5.3.4. Summary of Comparisons . . . 363
5.4. Open Problems . . . 367
5.5. Tony Hoare's Summary on 'Domain Modeling' . . . 369
5.6. Beauty Is Our Business . . . 371
5.7. Acknowledgements . . . 372

FROM DOMAINS TO REQUIREMENTS 375
Summary . . . 376
1. Introduction . . . 384
1.1. The Triptych Dogma of Software Development . . . 385
1.2. Software As Mathematical Objects . . . 386
1.3. The Contribution of These Lectures . . . 388
1.4. Some Comments on the Lecture Content . . . 391
1.5. Structure of Lectures . . . 394
2. An Example Domain: Transport . . . 397
2.1. Endurants . . . 399
2.1.1. Domain, Net, Fleet and Monitor . . . 400
2.1.2. Hubs and Links . . . 405
2.1.3. Unique Identifiers . . . 407
2.1.4. Mereology . . . 410
2.1.5. Attributes, I . . . 412
2.2. Perdurants . . . 425
2.2.1. Hub Insertion Action . . . 426

2.2.2. Link Disappearance Event 428
2.2.3. Road Traffic 429
2.3. Domain Facets 447
3. Requirements 448
3.1. Four Requirements Facets 460
3.1.1. Problem, Solution and Objective Sketch 462
3.1.2. Systems Requirements 464
3.1.3. User and External Equipment Requirements 470
3.1.4. Design Requirements 473
3.2. The Three Phases of Requirements Engineering 474
3.3. Order of Presentation of Requirements Prescriptions 478
3.4. Design Requirements and Design Assumptions 481
4. Domain Requirements 487
4.1. Domain Projection & Simplification 490
4.1.1. Domain Projection — Narrative 492
4.1.2. Domain Projection — Formalisation 495
4.2. Domain Instantiation 510
4.2.1. Domain Instantiation 512
4.2.2. Domain Instantiation — Abstraction 525
4.3. Domain Determination 528
4.3.1. Domain Determination: Example 529
4.4. Domain Extension 539
4.4.1. The Requirements Example: Domain Extension 541
4.5. Requirements Fitting 577
4.6. Discussion 586
5. Interface Requirements 589
5.1. Shared Phenomena 591
5.2. Shared Endurants 600
5.2.1. Data Initialisation 602
5.2.2. Data Refreshment 617
5.3. Shared Actions, Events and Behaviours 618
5.4. Discussion 623
6. Machine Requirements 624

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – October 21, 2015: 09:39

6.1. Varieties of Machine Requirements . . . 626
6.2. Derived Requirements . . . 627
6.2.1. Derived Actions . . . 630
6.2.2. Derived Events . . . 639
6.3. Technology Requirements . . . 644
6.3.1. Performance Requirements . . . 646
6.3.2. Dependability Requirements . . . 648
6.4. Development Requirements . . . 673
6.4.1. Process Requirements . . . 674
6.4.2. Maintenance Requirements . . . 676
6.4.3. Platform Requirements . . . 684
6.4.4. Management Requirements . . . 691
6.4.5. Documentation Requirements . . . 693
6.5. Discussion . . . 696
7. Conclusion . . . 697
7.1. What has been Achieved ? . . . 698
7.2. Present Shortcomings . . . 708
7.3. Comparison to “Classical” Requirements Engineering . . . 709
7.4. Future Work: Research Challenges . . . 710
7.5. Acknowledgments . . . 716
8. Bibliography . . . 717
8.1. Bibliographical Notes . . . 717
8.1.1. Published Papers . . . 717
8.1.2. Reports . . . 720
8.2. References . . . 721