Dines Bjørner's MAP-i Lecture # 0

Opening Lecture

Monday, 25 May 2015: 10:00–10:20

Domain Science & Engineering

A Prerequisite for Requirements Engineering

MAP-i: A Universities of Minho, Aveiro and Porto PhD Course

Dines Bjørner

Fredsvej 11, DK–2840 Holte, Denmark

May 23, 2015: 15:29

Summary of PhD Course

- This document takes the view that software specifications and programs are best understood as mathematical objects.
  - This is in contrast to other views,
  - notably such which are dominant in the USA,
  - that the development of software
  - is best understood as sociological and psychological objects.

- In this PhD course we cover two aspects of **software engineering:**
  - **domain engineering** (Lectures 1–6) and
  - **requirements engineering** (Lectures 7–10).
- We also cover some aspects of
  - **domain science.**
- The lectures are supported by extensive material:
  - A comprehensive set of **lecture notes:** `www.imm.dtu.dk/~dibj/portugal/Braga-MAP-i.pdf`, and
  - each lectures by **lecture slides:** `www.imm.dtu.dk/~dibj/portugal/BL0.pdf--BL11.pdf`.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

- We will be together Monday, Tuesday and Thursday 10:00–17:30
  - **'Formal Lectures'** alternate
  - with **'Workshop Sessions'.**
- In workshop sessions we shall try, You and I, to describe a domain.
  - We will select this domain right after lunch today
  - and start describing it.
  - You are supposed to think about this domain
    - mornings, before wee meet
    - and late afternoons, after we have "left".
- Wednesday I will give a Faculty Seminar:
  - **A New Foundation for Computing Science**
  - 14:00–14:45, Room DI-A2

Monday 25 May, 2015

- **L0**: **Opening Lecture** [Slides 1–8]
  Monday, 25 May 2015: 10:00–10:20
- **L1**: **An Overview of Domain Description** [Slides 9–79]
  Monday, 25 May 2015: 10:30–11:15
- **L2**: **Parts** [Slides 80–145]
  Monday, 25 May 2015: 11:30–12:15
- **1. Workshop:** An Example Domain
  Monday, 25 May 2015: 12:30–13:00
- **Lunch:** 13:00–14:30
- **L3**: **Unique Identifiers, Mereologies and Attributes** [Slides 146–202]
  Monday, 25 May 2015: 14:30–15:15
- **2. Workshop:** An Example Domain
  Monday, 25 May 2015: 15:30–16:15
- **L4**: **Components, Materials – and Discussion of Endurants** [Slides,203–241]
  Monday, 25 May 2015: 16:45–17:30

Tuesday 26 May, 2015

- **L5**: **Perdurants** [242–309]
  Tuesday, 26 May 2015: 10:00–10:45
- **3. Workshop:** An Example Domain
  Tuesday, 26 May 2015: 11:00–11:45
- **L6**: **A Summary Domain Description** [310–351]
  Tuesday, 26 May 2015: 12:00–13:00
- **Lunch:** 13:00–14:30
- **4. Workshop:** An Example Domain
  Tuesday, 26 May 2015: 14:30–15:15
- **L7**: **Requirements – An Overview, and Projection** [352–393]
  Tuesday, 26 May 2015: 15:30–16:15
- **L8**: **Domain Requirements: Instantiation and Determination** [394–423]
  Tuesday, 26 May 2015: 16:45–17:30

Wednesday 27 May:

- 14:00–14:45 **Faculty Seminar:** Room DI-A2

  Title: A New Foundation for Computing Science Paper, Slides

  Abstract: We argue that computing systems requirements must be based on precisely described domain models — and we argue that domain science & engineering offers a new dimension in computing. We review our work in this area and we outline a research and experimental engineering programme for the triptych of domain enginering, requirements engineering and software design.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

Thursday 28 May, 2015

- **L9**: **Domain Requirements: Extension and Fitting** [Slides 424–468]
  Thursday, 28 May 2015: 10:00–11:15
- **5. Workshop:** Example Domain
  Thursday, 28 May 2015: 11:30–12:00
- **L10**: **Interface Requirements** [Slides 469–556]
  Thursday, 28 May 2015: 12:15–13:00
- **Lunch:** 13:00–14:30
- **6. Workshop:** Example Domain
  Thursday, 28 May 2015: 14:30–15:15
- **L11**: **Conclusion** [Slides 557–561]
  Thursday, 28 May 2015: 15:30–16:30
- **L12**: **Discussion of Research Topics** [Slides 562–601]
  Thursday, 28 May 2015: 16:45–17:30

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

End of MAP-i Lecture # 0:
Opening Lecture

Monday, 25 May 2015: 10:00–10:20

Dines Bjørner's MAP-i Lecture # 1

An Overview Of Domain Description

Monday, 25 May 2015: 10:30–11:15

1. Domain Analysis & Description

Abstract

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

  can be precisely described:

  - narrated and
  - formalised.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

- 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 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`.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

- 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, technique and tools
  - for constructing domain descriptions.

1.1. Introduction

- The broader subject of this seminar is that of software development.
- The narrower subject is that of manifest domain engineering.
- We see software development
  in the context of the `TripTych` approach.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

- The contribution of this seminar 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,
  - that the (yet to be investigated) "calculus" is a first such "method calculus".

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

1.1.1. The `TripTych` Approach to Software Engineering

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

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

- 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.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

- 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.

1.1.2. Method and Methodology

1.1.2.1. Method

- By a **method** we shall understand
  - a "somehow structured" set of `principles`
  - for `select`ing 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.

1.1.2.2. 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.

- 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.

 © Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

- 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`.

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

    of the domain analysis & description method.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

1.1.2.3. 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.

1.1.3. 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.

 © Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

- 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.

- This paper is a computing science paper.
  - It is concerned with the construction of domain descriptions.
  - It puts forward a calculus for analysing and describing domains.
  - It does not theorize about this calculus.
  - There are no theorems about this calculus and hence no proofs.
  - We leave that to another study and paper.

1.1.4. 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 . Manifest Domain Names: Examples of suggestive names of manifest domains are:

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

- 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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

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 ■

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* ∎

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

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.* ■

- Our definitions of what a manifest domain is
  - are, to our own taste, not fully adequate;
  - they ought be so sharp that one can unequivocally distinguish such domains that are not manifest domains from those which are (!).
  - Examples of the former are:
    - the Internet,
    - language compilers,
    - operating systems,
    - data bases,

    etcetera.
- As we progress we shall sharpen our definition of 'manifest domain'.

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

1.1.5. 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 ■

Example 6 . Narrative Description of Bank System Endurants:

1 A banking system consists of a bank and collections of clients and of passbooks.

2 A bank attribute is that of a general ledger.

3 A collection of clients is a set of uniquely identified clients.

4 A collection of passbooks is a set of uniquely identified passbooks.

5 A client "possess" zero, one or more passbook identifiers.

6 Two or more clients may share the same passbook.

7 The general ledger records, for each passbook identifier, amongst others, the set of one or more client identifiers sharing that passbook, etc.

Etcetera ■

Example 7 . Formal Description of Bank System Endurants:

type
1. B, CC, CPB

value
1. **obs_part**_CC: B $\rightarrow$ CC,
1. **obs_part**_CPB: B $\rightarrow$ CPB

type
2. GL

value
2. **attr**_GL: B $\rightarrow$ GL

type
3. C, CI, CC = C-**set**,
4. PB, PBI, CPB = PB-**set**

value
5. **attr**_C: C $\rightarrow$ PBI-**set**

type
7. GL = PBI $\underset{m}{\rightarrow}$ SH $\times$ ...
7. SH = PBI-**set**

Etcetera ■

Example 8 . Narrative Description of Bank System Perdurants:

8 Clients and the bank possess cash (i.e., monies).

9 Clients can open a bank account and receive in return a passbook.

10 Clients may deposit monies into an account in response to which the passbook and the general ledger are updated.

11 Clients may withdraw monies from an account: if the balance of monies in the designated account is not less than the requested amount the client is given the (natural number) designated monies and the passbook and the general ledger are updated.

Etcetera ■

Example 9 . Formal Description of Bank System Perdurants:

type

8. M

value

8. **attr**_M: (B|C) $\rightarrow$ M
9. open: B $\rightarrow$ B $\times$ PB
10. deposit: PB $\rightarrow$ M $\rightarrow$ B $\rightarrow$ B $\times$ PB
11. withdraw: PB $\rightarrow$ B $\rightarrow$ **Nat** $\stackrel{\sim}{\rightarrow}$ B $\times$ PB $\times$ M

Etcetera ■

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

- 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
      - **action**s,
      - **event**s or
      - **behaviour**s.
  - 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.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

1.1.6. Towards a Methodology of Domain Analysis & Description

1.1.6.0.1 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 – May 23, 2015: 15:29

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

1.1.6.0.2 The Four Domain Analysis & Description "Players"

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

- The ***domain*** is there.
  - The domain analyser & describer cannot change the domain.
  - Analysing & describing the domain does not change it[5].
  - In a meta-physical sense it is inert.
  - 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 changes what is being observed.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

- 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.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

- 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.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

- 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.
  - So they are corrected.

1.1.6.0.3 An Interactive Domain Analysis & Description Dialogue

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

1.1.6.0.4 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.

- 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: `describe_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.

1.1.6.0.5 A Domain Analysis & Description Language

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

1.1.6.0.6 The Domain Description Language

- The 'Domain Description Language' is RSL [39], the RAISE Specification Language [40].
- With suitable, simple adjustments it could also be either of
  - Alloy [45],
  - Event B [1],
  - VDM-SL [30, 31, 37] or
  - Z [55].
- We have chosen RSL because of its simple provision for
  - defining sorts,
  - expressing axioms, and
  - postulating observers over sorts.

1.1.6.0.7 Domain Descriptions: Narration & Formalisation

- Descriptions
  - ***must*** be readable and
  - ***should*** 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.

1.1.7. 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, and vice versa.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

- 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 model of "standard" domains.

 © Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

- 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.

1.1.8. 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 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 [38] addresses this issue.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

1.1.8.1. A Formalisation

Some Notation:

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

- 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".

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

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

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".

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

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 $\mathbb{K}$. ■

1.1.8.2. 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 !*

1.1.8.3. 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.

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

1.1.8.4. Formal Concepts: A Wider Implication

- The formal concepts of a domain form Galois Connections [38].
  - 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's MAP-i Lecture # 1

End of MAP-i Lecture # 1: An Overview Of Domain Description

Monday, 25 May 2015: 10:30–11:15

Dines Bjørner's MAP-i Lecture # 2

Parts

Monday, 25 May 2015: 11:30–12:15

1.2. Endurant Entities

- In the rest of this seminar we shall consider entities in the context of their being manifest (i.e., spatio-temporal).

1.2.1. 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* ■[8]

---

[8] **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* [9]
- `is_entity` is said to be a **prerequisite prompt** for all other prompts.

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

Whither Entities:

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

  raises some philosophical questions.
- Are sentiments, like feelings, emotions or "hunches" observable ?
- 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.

1.2.2. 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.

 © Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

Example 10 . 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 ■

Definition 3 . Perdurant:

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

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

 © Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

Example 11 . 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) ■

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** ■

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

- In the words of Whitehead(1920) — as communicated by Sowa(2000) —
  - 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.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

1.2.3. 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 12 . Discrete Endurants:

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

Definition 5 . Continuous Endurant:

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

Example 13 . Continuous Endurants:

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

Analysis Prompt 4 . *is_discrete:*

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

Analysis Prompt 5 . *is_continuous:*

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

1.2.4. Parts, Components and Materials
1.2.4.1. 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 define the terms 'unique identification', 'mereology', and 'attributes' shortly.

Example 14 . Parts: Example

- 10 on Slide 84 illustrated,

and examples

- 18 on Slide 109 and
- 19 on Slide 111 illustrate

parts ■

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 15 . 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 ■

"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.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

- (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.

Definition 8 . Material:

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

Example 16 . 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 17 . Parts Containing Materials:

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

1.2.4.2. Part, Component and Material 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 99.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

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 99.

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 99.
-

1.2.5. 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.

- 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 part**s *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 ■

Example 18 . Atomic Parts: Examples of atomic parts of the above mentioned domains are:

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

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 ■

Example 19 . Composite Parts: Examples of atomic parts of the above mentioned domains are:

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

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`.

Whither Atomic or Composite:

- If we are analysing & describing vehicles
  in the context of a road net, cf. the Traffic System Example Slide 84,
  - 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.

 © Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

1.2.6. On Observing Part Sorts

1.2.6.1. Types and Sorts

- We use the term 'sort'
  - when we wish to speak of an abstract type,
  - that is, a type for which we do not wish to express a model[10].
  - We shall use the term 'type' to cover both
    - abstract types and
    - concrete types.

[10]

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

  or other.

1.2.6.2. 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.

 © Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

- 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, or is, a formal concept.*
- Thus there is, in this analysis, a "eureka",
  - a step where we shift focus
  - from the concrete to the abstract,
  - from observing specific part instances
  - to postulating a sort:
    - from one to the many.

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.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

1.2.6.3. Part Sort Observer Functions

- The above analysis amounts to the analyser
  - first "applying" the domain analysis prompt
  - `is_composite`(p) to a discrete endurant,
  - where we now assume that the obtained truth value is **true**.
  - Let us assume that parts p:P consists of 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 ($[1 \leq i \leq \mathrm{m}]$)
    - 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 description language observer 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:*

 © Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

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_i → **Bool** [$1 \leq i \leq m$]

proof obligation [Disjointness of part sorts]
[p] $\forall\ p{:}(P_1|P_2|...|P_m) \cdot$
[p] $\bigwedge \{\mathbf{is\_}\mathsf{P}_i(\mathsf{p}) \equiv \bigvee \sim \{\mathbf{is\_}\mathsf{P}_j(\mathsf{p}) \mid \mathsf{j} \in \{1..\mathsf{m}\} \setminus \{\mathsf{i}\}\} \mid \mathsf{i} \in \{1..\mathsf{m}\}\}$

Example 20 . Composite and Atomic Part Sorts of Transportation:

- The following example illustrates the multiple use of the `observe_part_sor` function:
  - first to δ, a specific transport domain, Item 12,
  - then to an $n : N$, the net of that domain, Item 13, and
  - then to an $f : F$, the fleet of that domain, Item 14.

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

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

14 A fleet is a collection of vehicles.

- The monitor is considered an atomic part.

type
12. N, F, M
value
12. **obs_part**_N:Δ→N, **obs_part**_F:Δ→F, **obs_part**_M:Δ→M
type
13. HC, LC
value
13. **obs_part**_HC:N→HC, **obs_part**_LC:N→LC
type
14. VC
value
14. **obs_part**_VC:F→VC

- *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* ■

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

1.2.6.4. 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.

Domain Description Prompt 2 . *observe_part_type*:

- *Then the domain analyser applies the* **domain description prompt***:*
  - ◈ ***observe_part_type****(p)*[11]
- *to parts p:P which then yield the* part type and part type observers *domain description text according to the following schema:*

[11] 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: $P \rightarrow T$

■

- The type names,
  - 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.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

Example 21 . Concrete Part Types of Transportation:

We continue Example 20 on Slide 123:

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

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

17 A collection of vehicles is a set of vehicles.

18 Vehicles are, until further analysis, part sorts.

type
15. Hs = H-**set**, Ls = L-**set**
16. H, L
17. Vs = V-**set**
18. V

value
15. **obs_part**_Hs:HC→Hs, **obs_part**_Ls:LC→Ls
17. **obs_part**_Vs:VC→Vs ■

1.2.6.5. 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
  - $\mathsf{T=A}^*$ or
  - $\mathsf{T=ID} \xrightarrow[m]{} \mathsf{A}$ or
  - $\mathsf{T=A}_t|\mathsf{B}_t|...|\mathsf{C}_t$

  where

  - ID is a sort of unique identifiers,
  - $\mathsf{T=A}_t|\mathsf{B}_t|...|\mathsf{C}_t$ defines the disjoint types
    - $\mathsf{A}_t\mathsf{==mkA}_s(\mathsf{s:A}_s)$,
    - $\mathsf{B}_t\mathsf{==mkB}_s(\mathsf{s:B}_s)$, ...,
    - $\mathsf{C}_t\mathsf{==mkC}_s(\mathsf{s:C}_s)$,

    and where
  - A, A_s, B_s, ..., C_s are sorts.
  - Instead of $\mathsf{A}_t\mathsf{==mkA(a:A}_s)$, etc., we may write $\mathsf{A}_t\mathsf{::A}_s$ etc.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

1.2.6.6. Part Sort and Type Derivation Chains

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

1.2.6.6.1 No Recursive Derivations

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

1.2.6.7. Names of Part Sorts and Types

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

  — as well as the
  - attribute_names,
  - observe_material_sorts,
  - observe_unique_identifier,
  - observe_mereology and
  - observe_attributes

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

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

Example 22 . Container Line Sorts:

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

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

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

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

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

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

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

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

26 Containers are further undefined.

- After a some slight editing we get:

type
CL
VS, VI, V, Vs = VI $\overrightarrow{m}$ V,
PS, PI, P, Ps = PI $\overrightarrow{m}$ 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 $\overrightarrow{m}$ BAY
ROWS, RI, ROW, Rows=RI $\overrightarrow{m}$ ROW
STKS, SI, STK, Stks=SI $\overrightarrow{m}$ 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 ■

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

1.2.6.8. More On Part Sorts and Types

- The above "experimental example" motivates the below.
  - We can always assume that composite parts p:P abstractly consists of a definite number of sub-parts.
    - **Example 23**. We comment on Example 20 on Slide 123: 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 24**. We comment on Example 20 on Slide 123: parts of type N, F, HC, LC and VC are all composite ■

- There are, pragmatically speaking, two cases for such compositionality.
  - Either the part, p_{i_z}, of type t_{i_z}, is is composed from a definite number of abstract or concrete sub-parts of distinct types.
    - **Example 25.** We comment on Example 20 on Slide 123: 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 26.** We comment on Example 20 on Slide 123: parts of type HC, LC and VC are composed from an indefinite numbers of hubs, links and vehicles, respectively ■

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

Example 27 . Pipeline Parts:

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

28 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.

29 All these unit sorts are atomic and disjoint.

type
27. PL, U, We, Pi, Pu, Va, Fo, Jo, Si
27. Well, Pipe, Pump, Valv, Fork, Join, Sink

value
27. **obs_part**_Us: PL → U-**set**

type
28. U == We | Pi | Pu | Va | Fo | Jo | Si
29. We::Well, Pi::Pipe, Pu::Pump, Va::Valv, Fo:Fork, Jo::Join, Si::Sink ■

1.2.6.8.1 Derivation Lattices

- Derivation chains
  - start with the domain name, say Δ, and
  - (definitively) end with the name of an atomic sort.
- Sets of derivation chains form **join lattices** [3].

Example 28 . Derivation Chains:

- Figure 1 on the following slide illustrates
  - two part sort and type derivation chains.
  - based on Examples 20 on Slide 123 and 22 on Slide 135, respectively.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

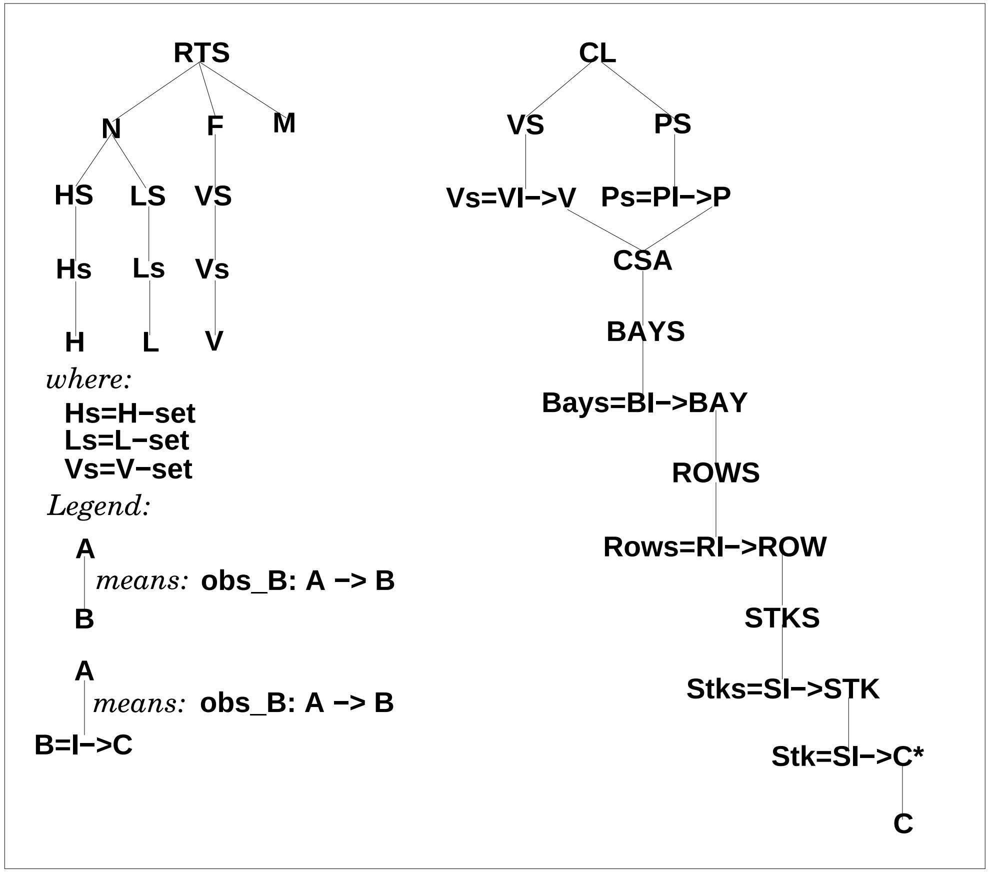

Figure 1: Two Domain Lattices: Examples 20 on Slide 123 and 22 on Slide 135

- The "−>" of Fig. 1 stands for $\xrightarrow[m]{}$

1.2.6.9. 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.

1.2.6.10. 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.

- 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).

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

Dines Bjørner's MAP-i Lecture # 2

End of MAP-i Lecture # 2: Parts

Monday, 25 May 2015: 11:30–12:15

Dines Bjørner's MAP-i Lecture # 3

Unique Identifiers, Mereologies and Attributes

Monday, 25 May 2015: 14:30–15:15

1.2.7. Unique Part Identifiers

- Two parts are either identical or a distinct, i.e., unique.
  - Two parts are identical
    - if all their respective qualities
    - have the same values.

    That is, their location in space/time are one and the same.
  - Two parts are distinct
    - even if all the attribute qualities of the two parts,
    - that we have chosen to consider have the same values,
    - if, in that case, their space/time locations are distinct.

- We can assume, without any loss of generality,
  - (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 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.

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:*

3. `observe_unique_identifier` schema

Narration:

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

Formalisation:

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

Example 29 . Unique Transportation Net Part Identifiers:
We continue Example 20 on Slide 123.

30 Links and hubs have unique identifiers

31 and unique identifier observers.

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

1.2.8. 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 [32, 21].

1.2.8.1. 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 30 . Shared Attribute Mereology:

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

etcetera ■

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

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

  Example 31 . 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;

  etcetera. ■

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

1.2.8.2. 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*

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

- 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 (i.e., part relation) observers.
- We can define a **mereology type** as a type $\mathcal{E}$xpression over unique [part] identifier types.
  - We generalise to unique [part] 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 observers *domain description text according to the following schema:*

4. `observe_mereology` schema

Narration:

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

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

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

Formalisation:

type

[t] MT[12]= $\mathcal{E}$(PI1,PI2,...,PIm)

value

[m] **obs_mereo**_P: P → MT

axiom [Well−formedness of Domain Mereologies]

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

[12] 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 – May 23, 2015: 15:29

Example 32 . Road Net Part Mereologies: We continue Example 20 on Slide 123 and Example 29 on Slide 151.

32 Links are connected to exactly two distinct hubs.

33 Hubs are connected to zero or more links.

34 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

32. $\mathsf{LM'} = \mathsf{HI}\textbf{-set}$, $\mathsf{LM} = \{|\mathsf{his}{:}\mathsf{HI}\textbf{-set} \cdot \textbf{card}(\mathsf{his}){=}2|\}$

33. $\mathsf{HM} = \mathsf{LI}\textbf{-set}$

value

32. $\textbf{obs\_mereo}\_\mathsf{L}: \mathsf{L} \to \mathsf{LM}$

33. $\textbf{obs\_mereo}\_\mathsf{H}: \mathsf{H} \to \mathsf{HM}$

axiom [Well−formedness of Road Nets, N]

34. $\forall$ n:N,l:L,h:H· l $\in$ **obs_part**_Ls(**obs_part**_LC(n))$\wedge$h $\in$ **obs_part**_Hs(**ob**

34. **let** his=mereology_H(l), lis=mereology_H(h) **in**

34. his$\subseteq\cup${**uid**_H(h) | h $\in$ **obs_part**_Hs(**obs_part**_HC(n))}

34. $\wedge$ lis$\subseteq\cup${**uid**_H(l) | l $\in$ **obs_part**_Ls(**obs_part**_LC(n))} **end**

Example 33 . Pipeline Parts Mereology:

- We continue Example 27 on Slide 140.
- 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.

35 Wells have exactly one connection to an output unit.

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

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

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

39 Sinks have exactly one connection from an input unit.

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

```
type
40.  UM′=(UI-set×UI-set)
40.  UM={|(iuis,ouis):UI-set×UI-set•iuis ∩ ouis={}|}
value
40.  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
35.              (0,1) → is_We(u),
36.              (1,1) → is_Pi(u)∨is_Pu(u)∨is_Va(u),
37.              (1,2) → is_Fo(u),
38.              (2,1) → is_Jo(u),
39.              (1,0) → is_Si(u)
       end end
```


© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

1.2.8.3. Update of Mereologies

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

Mereology Update Function

$\diamond$ **upd_mereology**: $P \rightarrow M \rightarrow P$

for all relevant M and P.

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

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

 © Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

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

Example 34 . Mereology Update:

- The example is that of updating the mereology of a hub.
- Cf. Example 32 on Slide 160.

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

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

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

value

41. update_hub_mereology: H → LI → H
42. update_hub_mereology(h)(li) ≡
43. **let** m = {li} ∪ **obs_mereo**_H(h) **in** **upd_mereology**(h)(m) **end**

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

1.2.8.4. Formulation of Mereologies

- The `observe_mereology` domain descriptor, Slide 158,
  - 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, in general, depend on any part sort
  - the mereo type MT can, for some domains,
  - "first" be described when all part sorts have been dealt with.
- In *Domain Analysis: Endurants – An Analysis & Description Process Model* we we present a model of one form of evaluation of the `TripTych` analysis and description prompts.

1.2.9. Part Attributes

1.2.9.1. Inseparability of Attributes from Endurants

- 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 – May 23, 2015: 15:29

1.2.9.2. 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 35 . 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 ■

1.2.9.3. 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.

 © Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

Example 36 . 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 37 . 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*[13] ■

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

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

  Therefore we must prove it.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

1.2.9.3.1 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:*

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

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_i→**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** [$i \neq j$, $1 \leq i,j \leq n$]

- *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 (type* 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.

Example 38 . Road Hub Attributes: After some analysis a domain analyser may arrive at some interesting hub attributes:

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

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

46 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.

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

type
44. $\mathsf{H}\Sigma = (\mathsf{LI}\times\mathsf{LI})$-**set**
45. $\mathsf{H}\Omega = \mathsf{H}\Sigma$-**set**
value
44. **attr**_$\mathsf{H}\Sigma$:$\mathsf{H}\rightarrow\mathsf{H}\Sigma$
45. **attr**_$\mathsf{H}\Omega$:$\mathsf{H}\rightarrow\mathsf{H}\Omega$
axiom [Well−formedness of Hub States, $\mathsf{H}\Sigma$]
46. $\forall$ h:H $\cdot$ **let** lis = **obs_mereo**_H(h) **in**
46. **let** hσ = **attr**_$\mathsf{H}\Sigma$(h) **in**
46a.. $\{$li,li$'$|li,li$'$:LI$\cdot$(li,li$'$)$\in$ h$\sigma\}\subseteq$lis
46b.. $\wedge$ hσ $\in$ **attr**_$\mathsf{H}\Omega$(h)
46. **end end**
type
47. ..., ...
value
47. **attr**_..., ...

1.2.9.4. 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.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

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

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

Dynamic attributes are either inert, reactive or active attributes.

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

Example 39 . Inert and Reactive Attributes:

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

Active attributes are either autonomous, biddable or programmable attributes.

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

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

 © Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

Example 40 . Static, Programmable and Inert Link Attributes:

48 Some link attributes

a. length, b. name,

can be considered static,

49 whereas other link attributes

a. state, b. state space

can be considered programmable,

50 Finally link attributes

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

can be considered inert.

type
48a.. LEN
value
48a.. **obs_part**_LEN: L → LEN
type
48b.. Name
value
48b.. **obs_part**_Name: L → Name
type
49a.. $\mathsf{L}\Sigma'$=(HI×HI)-**set**
49a.. $\mathsf{L}\Sigma = \{|\mathsf{l}\sigma{:}\mathsf{L}\Sigma \cdot \textbf{card}\ \mathsf{l}\sigma \leq 2|\}$
value
49a.. **obs_part**_$\mathsf{L}\Sigma$: L → $\mathsf{L}\Sigma$
type
49b.. $\mathsf{L}\Omega'$=$\mathsf{L}\Sigma$-**set**
49b.. $\mathsf{L}\Omega = \{|\mathsf{l}\omega{:}\mathsf{L}\Omega \cdot \textbf{card}\ \mathsf{l}\omega = 1|\}$
value
49b.. **obs_part**_$\mathsf{L}\Omega$: L → $\mathsf{L}\Omega$
type
50a.. LSoR
50b.. DLM
value
50a.. **obs_part**_LSoR: L → LSoR
50b.. **obs_part**_DLM: L → DLM

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

Example 41 . Autonomous and Programmable Hub Attributes: We continue Example **??**.

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

- **External Attributes:** By an **external attribute** we shall understand
  - either a inert,
  - or an autonomous,
  - or a reactive,
  - or a biddable

  attribute

- Thus we can define the domain analysis prompt:
  - `is_external_attribute`,
  - as:

value

$$\begin{array}{l} \mathsf{is\_external\_attribute}\colon\ \mathsf{P} \to \mathbf{Bool} \\ \mathsf{is\_external\_attribute(p)} \equiv \\ \quad \mathsf{is\_dynamic\_attribute(p)} \wedge \sim\!\mathsf{is\_programmable\_attribute(p)} \\ \mathbf{pre}\colon\ \mathsf{is\_endurant(p)} \wedge \mathsf{is\_discrete(p)} \end{array}$$

 © Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

- Figure 2 captures the attribute value ontology.

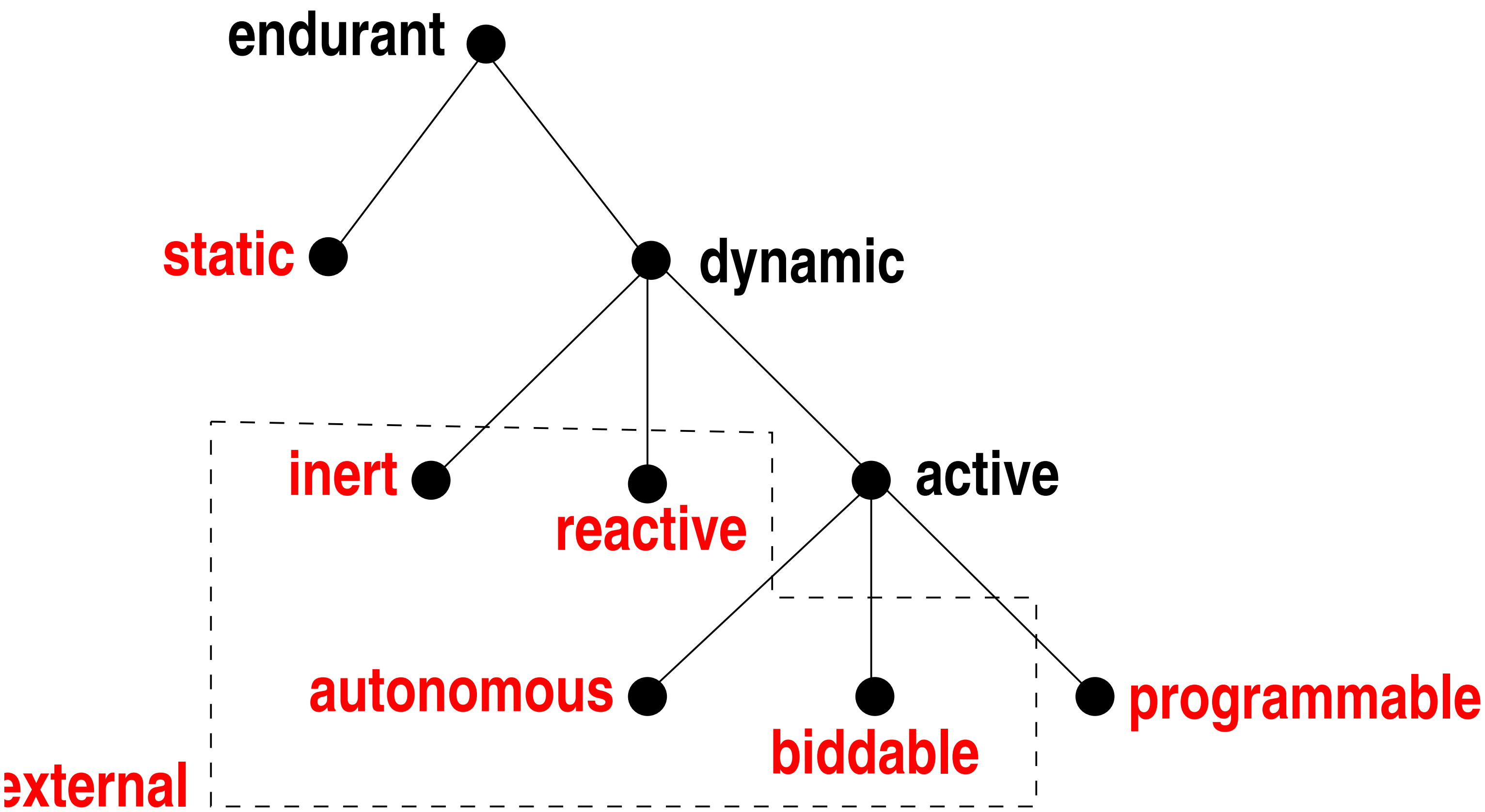

Figure 2: Attribute Value Ontology

1.2.9.5. 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,
  - dynamic values of parts, p,
  - cannot be "copied",
  - but **attr**_A(p) must be "performed"
  - every time they are needed.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

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

1.2.9.6. Shared Attributes

- Normally part attributes of different part sorts are distinctly named.
- If, however, observe_attributes(p_{ik}:P_i) and observe_attributes($p_{j\ell}$:
  - 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.

Example 42. Shared Attributes. Examples of shared attributes:

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

[15] See upcoming Example 33 on Slide 162

Example 43 . Shared Timetables:

- The fleet and vehicles of Example 20 on Slide 123 and Example 21 on Slide 130 is that of a bus company.

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

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

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

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

type

51. FI, VI, BT

value

51. **uid**_F: F $\rightarrow$ FI
51. **uid**_V: V $\rightarrow$ VI
52. **obs_mereo**_F: F $\rightarrow$ VI-**set**
53. **obs_mereo**_V: V $\rightarrow$ FI
54. **attr**_BT: (F|V) $\rightarrow$ BT

axiom

$\Box$ $\forall$ f:F $\Rightarrow$

$\forall$ v:V $\cdot$ v $\in$ **obs_part**_Vs(**obs_part**_VC(f)) $\cdot$ **attr**_BT(f) $=$ **attr**_BT(v)

[which is the same as]

$\Box$ $\forall$ f:F $\Rightarrow$

{**attr**_BT(f)}={**attr**_BT(v):v:V$\cdot$v $\in$ **obs_part**_Vs(**obs_part**_VC(f))}

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

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

    are shared.

Example 44 . Shared Passbooks:

55 A banking system contains

- an administration and
- a set of customers.

56 The administration contains a general ledger.

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

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

59 Passbooks are uniquely identified by unique customer identifiers.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

type

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

58. [attributes] PB

value

55. **obs_part**_AD: BS $\rightarrow$ AD

56. **obs_part**_GL: AD $\rightarrow$ GL

57. **attr**_PBs: GL $\rightarrow$ PB-**set**

55. **obs_part**_CS: BS $\rightarrow$ CS

55. **obs_part**_Cs: BS $\rightarrow$ Cs

58. **attr**_PB: C $\rightarrow$ PB

59. **uid**_PB: PB $\rightarrow$ PBI

axiom

$\square$ $\forall$ bs:BS ·

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

= {**attr**_PB(c)|c:C·c $\in$ **obs_part**_Cs(**obs_part**_CS(bs))} ■

Dines Bjørner's MAP-i Lecture # 3

End of MAP-i Lecture # 3: Unique Identifiers, Mereologies and Attributes

Monday, 25 May 2015: 14:30–15:15

Dines Bjørner's MAP-i Lecture # 4

Components, Materials – and Discussion of Endurants

Monday, 25 May 2015: 16:45–17:30

1.2.10. Components

- Components are discrete endurants which are not considered parts.
  - $\texttt{is\_component}(k) \equiv \texttt{is\_endurant}(k) \wedge \sim \texttt{is\_part}(k)$

Example 45 . 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 is the components of the container part.
  - Conveyor belts transport machine assembly units and are thus considered the components of the conveyor belt.

- 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 potentially contains components otherwise false* ■

- Let us assume that parts p:P embodies 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***(e)

  yields the ***component sorts and component sort observers*** *domain description text according to the following schema:*

6. `observe_component_sorts` schema

Narration:

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

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

[i] ... narrative text on component sort recognisers ...

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

Formalisation:

type

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

[s] KS = (K1|K2|...|Kn)-**set**

value

[o] **components**: P → KS

[i] **is**_K$_i$: K → **Bool** $[1 \leq i \leq n]$

Proof Obligation:

[Disjointness of Component Sorts]

[p] $\forall\ m_i$:(K$_1$|K$_2$|...|K$_n$) ·

[p] $\bigwedge$ {**is**_K$_i$(m_i) $\equiv$ $\bigvee$∼{**is**_K$_j$(m_i)|j ∈ {1..m}\{i}}|i ∈ {1..m}}

Example 46 . Container Components: We continue Example 22 on Slide 135.

60 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 of a container,

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

c. the component observer function signature.

type
60a. K1, K2, ..., Kn
60b. KS = (K1|K2|...|Kn)-**set**
value
60c. **obs_comp**_KS: C → KS ■

- 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 describing qualities of components.
  - We suggest that conventional abstraction of modeling techniques and tools be applied.

1.2.11. Materials

- Continuous endurants (i.e., **materials**) are entities, m, which satisfy:
  - $\mathtt{is\_material}(m) \equiv \mathtt{is\_endurant}(m) \wedge \mathtt{is\_continuous}(m)$

Example 47 . 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.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

- 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 contains materials otherwise false* ■

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

  we must prove it.

Domain Description Prompt 7. *observe_material_sorts*:

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

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

7. `observe_material_sorts` schema

Narration:

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

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

[i] ... narrative text on material sort recognisers ...

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

Formalisation:

type

[s] M_i [$1 \le i \le n$]

[s] MS = M1 M2 ... Mn

value

[o] **obs_mat**_M_i: P → M_i [$1 \le i \le n$]

[o] **materials**: P → MS

[i] **is**_M_i: M → **Bool** [$1 \le i \le n$]

proof obligation [Disjointness of Material Sorts]

[p] $\forall\ m_i$:($\mathsf{M}_1|\mathsf{M}_2|...|\mathsf{M}_n$) •

[p] $\bigwedge$ {**is**_$\mathsf{M}_i(m_i) \equiv \bigvee\!\sim${**is**_$\mathsf{M}_j(m_i)$|j ∈ {1..m}\{i}}|i ∈ {1..m}}

- *The M_i are all distinct* ■

Example 48 . Pipeline Material: We continue Example 27 on Slide 140 and Example 33 on Slide 162.

61 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

61a. LoG

value

61b. **obs_mat**_LoG: U $\rightarrow$ LoG ■

1.2.11.1. Materials-related Part Attributes

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

Example 49 . Pipeline Material Flow: We continue Examples 27, 33 and 48.

- 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

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

63 current flow of material into a unit input connector,

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

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

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

67 current leak of material at a unit input connector,

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

69 current leak of material at a unit input connector,

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

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

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

type
62. F, L
value
63. **attr**_cur_iF: U → UI → F
64. **attr**_max_iF: U → UI → F
65. **attr**_cur_oF: U → UI → F
66. **attr**_max_oF: U → UI → F
67. **attr**_cur_iL: U → UI → L
68. **attr**_max_iL: U → UI → L
69. **attr**_cur_oL: U → UI → L
70. **attr**_max_oL: U → UI → L
71. **attr**_cur_L: U → L
72. **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 are dynamic attributes ■

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

1.2.11.2. 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 50 . Pipelines: Intra Unit Flow and Leak Law:

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

74 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 – May 23, 2015: 15:29

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

73. ∀ pls:PLS,b:B\We\Si,u:U ·
73. b ∈ **obs_part**_Bs(pls)∧u=**obs_part**_U(b)⇒
73. **let** (iuis,ouis) = **obs_mereo**_U(u) **in**
74. sum_cur_iF(iuis)(u) =
74a.. sum_cur_iL(iuis)(u)
74b.. ⊕ **attr**_cur_L(u)
74c.. ⊕ sum_cur_oF(ouis)(u)
74d.. ⊕ sum_cur_oL(ouis)(u)
73. **end**

75 The **sum_cur_iF** (cf. Item 74) sums current input flows over all input connectors.

76 The **sum_cur_iL** (cf. Item 74a.) sums current input leaks over all input connectors.

77 The **sum_cur_oF** (cf. Item 74c.) sums current output flows over all output connectors.

78 The **sum_cur_oL** (cf. Item 74d.) sums current output leaks over all output connectors.

75. sum_cur_iF: UI-**set** → U → F
75. sum_cur_iF(iuis)(u) ≡ ⊕ {**attr**_cur_iF(ui)(u)|ui:UI•ui ∈ iuis}
76. sum_cur_iL: UI-**set** → U → L
76. sum_cur_iL(iuis)(u) ≡ ⊕ {**attr**_cur_iL(ui)(u)|ui:UI•ui ∈ iuis}
77. sum_cur_oF: UI-**set** → U → F
77. sum_cur_oF(ouis)(u) ≡ ⊕ {**attr**_cur_iF(ui)(u)|ui:UI•ui ∈ ouis}
78. sum_cur_oL: UI-**set** → U → L
78. sum_cur_oL(ouis)(u) ≡ ⊕ {**attr**_cur_iL(ui)(u)|ui:UI•ui ∈ ouis}
⊕: (F|L) × (F|L) → F

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

Example 51 . Pipelines: Inter Unit Flow and Leak Law:

79 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)]

79. $\forall$ pls:PLS,b,b′:B,u,u′:U•
79. {b,b′}⊆**obs_part**_Bs(pls)∧b≠b′∧u′=**obs_part**_U(b′)
79. ∧ **let** (iuis,ouis)=**obs_mereo**_U(u),(iuis′,ouis′)=**obs_mereo**_U(u′),
79. ui=**uid**_U(u),ui′=**uid**_U(u′) **in**
79. ui ∈ iuis ∧ ui′ ∈ ouis′ ⇒
79a.. **attr**_cur_oF(u′)(ui′) − **attr**_leak_oF(u′)(ui′)
79b.. = **attr**_cur_iF(u)(ui) + **attr**_leak_iF(u)(ui)
79. **end**
79. **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.
- We need formalising the flow and leak summation functions.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

1.2.12. "No Junk, No Confusion"

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

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

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

- So, since one naturally wishes "no junk, no confusion" what does one do ?
- The answer to that is
  - *one proceeds with great care !*
- To avoid junk we have stated a number of sort well-formedness axioms, for example:
  - Slide 151 for *Well-formedness of Links, L, and Hubs, H*,
  - Slide 158 for *Well-formedness of Domain Mereologies*,
  - Slide 161 for *Well-formedness of Road Nets, N*,
  - Slide 163 for *Well-formedness of Pipeline Systems, PLS (0)*,
  - Slide 182 for *Well-formedness of Hub States, $H\Sigma$*,
  - Slide 220 for *Well-formedness of Pipeline Systems, PLS (1)*,
  - Slide 222 for *Well-formedness of Pipeline Systems, PLS (2)*,
  - Slide 229 for *Well-formedness of Pipeline Route Descriptors* and
  - Slide 233 for *Well-formedness of Pipeline Systems, PLS (3)*.

- To avoid **confusion** we have stated a number of **proof obligation**s:
  - Slide 122 for *Disjointness of Part Sorts*,
  - Slide 178 for *Disjointness of Attribute Types* and
  - Slide 212 for *Disjointness of Material Sorts*.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

Example 52 . No Pipeline Junk:

- We continue Example 27 on Slide 140 and Example 33 on Slide 162.

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

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

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

To formalise the above we describe some auxiliary notions.

1.2.12.0.1 Pipe Routes

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

type

80. $\mathsf{R}' = \mathsf{U}^\omega$

80. $\mathsf{R} = \{|\ \mathsf{r{:}Route}' \cdot \mathsf{wf\_Route(r)}\ |\}$

81. $\mathsf{RD} = \mathsf{UI}^\omega$

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

81. $\forall\ \mathsf{rd{:}RD} \cdot \exists\ \mathsf{r{:}R} \cdot \mathsf{rd{=}descriptor(r)}$

value

81. $\mathsf{descriptor{:}}\ \mathsf{R} \rightarrow \mathsf{RD}$

81. $\mathsf{descriptor(r)} \equiv \langle \mathbf{uid}\_\mathsf{UI}(\mathsf{r}[\,\mathsf{i}\,])|\mathsf{i}{:}\mathbf{Nat}\cdot 1 \leq \mathsf{i} \leq \mathbf{len}\ \mathsf{r}\rangle$

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

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

value

82. adjacent: U × U → **Bool**
82. adjacent(u,u′) ≡
82. **let** (,ouis)=**obs_mereo**_U(u),
82. (iuis,)=**obs_mereo**_U(u′) **in**
82. ouis ∩ iuis ≠ {} **end**

83 Given a pipeline system, pls, one can identify the (possibly infinite) set of (possibly infinite) routes of that pipeline system.

a. The empty sequence, $\langle\rangle$, is a route of pls.

b. Let u be a unit of pls, then $\langle u\rangle$ is a route of pls.

c. Let u, u' be adjacent units of pls then $\langle u, u'\rangle$ is a route of pls.

d. If r and r' are routes of pls such that the last element of r is the same as the first element of r', then $r\widehat{\ }\mathbf{tl}\,r'$ is a route of pls.

e. No sequence of units is a route unless it follows from a finite number of applications of the basis and induction clauses of Items 83a.–83d..

value

83. Routes: PLS $\rightarrow$ R-**infset**

83. Routes(pls) $\equiv$

83a.. **let** rs = $\langle\rangle$

83b.. $\cup$ $\{\langle$u$\rangle$|u:U•u $\in$ **obs_part**_Us(pls)$\}$

83c.. $\cup$ $\{\langle$u,u$'\rangle$|u,u$'$:U•$\{$u,u$'\}\subseteq$**obs_part**_Us(pls) $\wedge$ adjacent(u,u$'$)$\}$

83d.. $\cup$ $\{$r$\widehat{\ }$**tl** r$'$|r,r$'$:R•$\{$r,r$'\}\subseteq$rs$\wedge$r[**len** r]=**hd** r$'\}$

83e.. **in** rs **end**

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

1.2.12.0.2 Well-formed Routes

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

value

84. acyclic_Route: R $\rightarrow$ **Bool**
84. acyclic_Route(r) $\equiv$ $\sim\exists$ i,j:**Nat**$\cdot\{$i,j$\}\subseteq$**inds** r $\wedge$ i$\neq$j $\wedge$ r[i]=r[j]

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

1.2.12.0.3 Well-formed Pipeline Systems

85 A pipeline system is well-formed if

a. none of its routes are circular and

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

axiom [Well−formedness of Pipeline Systems, PLS (3)]
85. $\forall$ pls:PLS $\cdot$
85a.. non_circular(pls)
85b.. $\wedge$ are_embedded_in_well_to_sink_Routes(pls)
value
85. non_circular_PLS: PLS $\rightarrow$ **Bool**
85. non_circular_PLS(pls) $\equiv$
85. $\forall$ r:R$\cdot$r $\in$ routes(p)$\wedge$acyclic_Route(r)

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

86 We define well-formedness in terms of well-to-sink routes, i.e., routes which start with a well unit and end with a sink unit.

value

86. $\mathsf{well\_to\_sink\_Routes}$: PLS $\rightarrow$ R-**set**
86. $\mathsf{well\_to\_sink\_Routes(pls)} \equiv$
86. **let** rs = Routes(pls) **in**
86. $\{\mathsf{r|r{:}R{\cdot}r} \in \mathsf{rs} \wedge \textbf{is}\_\mathsf{We(r[1])} \wedge \textbf{is}\_\mathsf{Si(r[\,\mathbf{len}\ r\,])}\}$ **end**

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

87. are_embedded_in_well_to_sink_Routes: PLS $\rightarrow$ **Bool**
87. are_embedded_in_well_to_sink_Routes(pls) $\equiv$
87. **let** wsrs = well_to_sink_Routes(pls) **in**
87. $\forall$ r:R $\cdot$ r $\in$ Routes(pls) $\Rightarrow$
87. $\exists$ r′:R,i,j:**Nat** $\cdot$
87. r′ $\in$ wsrs
87. $\wedge$ $\{$i,j$\}\subseteq$**inds** r′$\wedge$i$\leq$j
87. $\wedge$ r = $\langle$r′[k]$|$k:**Nat**$\cdot$i$\leq$k$\leq$j$\rangle$ **end**

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

1.2.12.0.4 Embedded Routes

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

value

88. embedded_Routes: R $\rightarrow$ R-**set**
88. embedded_Routes(r) $\equiv$
88. $\{\langle r[\,k\,] | k{:}\mathbf{Nat}{\cdot}i \leq k \leq j \rangle \mid i,j{:}\mathbf{Nat}{\cdot}\ i\ \{i,j\} \subseteq \mathbf{inds}(r) \wedge i \leq j\}$

1.2.12.0.5 A Theorem

89 The following theorem is conjectured:

a. the set of all routes (of the pipeline system)

b. is the set of all well-to-sink routes (of a pipeline system) and

c. all their embedded routes

theorem:

89. $\forall$ pls:PLS $\cdot$
89. **let** rs = Routes(pls),
89. wsrs = well_to_sink_Routes(pls) **in**
89a.. rs =
89b.. wsrs $\cup$
89c.. $\cup$ $\{\{$r$'|$r$'$:R $\cdot$ r$'$ $\in$ embedded_Routes(r$''$)$\}$ $|$ r$''$:R $\cdot$ r$''$ $\in$ wsrs$\}$
88. **end** ■

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

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

1.2.13. Discussion of Endurants

- In Sect. 4.2.2 a “depth-first” search for part sorts was hinted at.
- It essentially expressed
  - that we discover domains epistemologically[16]
  - but understand them ontologically.[17]
- 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.

[16]**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.

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

- 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 – May 23, 2015: 15:29

- 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.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

Dines Bjørner's MAP-i Lecture # 4

End of MAP-i Lecture # 4: Components, Materials – and Discussion of Endurants

Monday, 25 May 2015: 16:45–17:30

Dines Bjørner's MAP-i Lecture # 5

Perdurants: Actions, Events and Behaviours

Tuesday, 26 May 2015: 10:00–10:45

1.3. Perdurant Entities

- We shall give only a cursory overview of perdurants.
- That is, 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, through this cursory overview, we can justify our detailed study of endurants,
    - their part and subparts,
    - their unique identifiers, mereology and attributes.

- This justification is manifested
  - (i) in expressing the types of **signature**s,
  - (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.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

- 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**.

1.3.1. 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 53 . States: Some examples of states are:

- A road hub can be a state, cf. Hub State, HΣ, Example 38 on Slide 181.
- A road net can be a state – since its hubs can be.
- Container stowage areas, CSA, Example 22 on Slide 135, 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 potentially carry components

1.3.2. Actions, Events and Behaviours

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

  behaviours.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

On Action, Event and Behaviour Distinctions:

- The distinction into action, event and behaviour perdurants is pragmatic.

1.3.2.1. 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 – May 23, 2015: 15:29

- 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].

- For a seminal book on "time in computing" we refer to the eclectic *Modeling Time in Computing, Springer 2012*.

- And for seminal book on time at the epistemology level we refer to *The Logic of Time, Kluwer 1991*.

1.3.2.2. 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.

 © Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

Example 54 . 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 20 on Slide 123 and 43 on Slide 198 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 27 on Slide 140 and Examples 27 on Slide 140 and 33 on Slide 162 will be considered actors.
- The bank general ledger and each bank customer of the *Shared Passbooks* example, Example 44 on Slide 201, will be considered actors ■

1.3.2.3. Parts, Attributes and Behaviours

- Example 54 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 55 . 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 ■

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

1.3.3. Discrete Actions

Definition 13 . Discrete Action: *By a* **discrete action** *[54] 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 56 . 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 – May 23, 2015: 15:29

1.3.4. 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 [36, 51, 49, 35] [41, 2, 47, 34] [50, 33].
- We note, in particular, [35, 2, 47].

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

Example 57 . 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 ■

1.3.5. 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 – May 23, 2015: 15:29

Example 58 . 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 ■

1.3.5.1. Channels and Communication

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

type M
channel ch M

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

- 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.

1.3.5.2. 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.

 © Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

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

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

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

Example 59 . Bus System Channels:

- We extend Examples 20 on Slide 123 and 43 on Slide 198.
- We consider the **fleet** and the **vehicle**s to be behaviours.

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

91 the **fleet** and

92 the **vehicles**.

93 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 – May 23, 2015: 15:29

value

90. δ:Δ,

91. f:F = **obs_part**_F(δ),

92. vs:V-**set** = **obs_part**_Vs(**obs_part**_VC((**obs_part**_F(δ))))

channel

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

Example 60 . Bank System Channels:

- We extend Example 44 on Slide 201.
- We consider the **general ledger** and the **customers** to be behaviours.

94 We assume some **bank system**. From the **bank system**

95 we observe the **general ledger**.

96 and the set of **customers**.

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

value
94. bs:BS
95. gl=**obs_part**_GL(**obs_part**_AD(bs)):GL
96. cs=**obs_part**_Cs(**obs_part**_CS(bs)):C-**set**
channel
97. {bsch[{**uid**_GL(gl),**uid**_C(c)}]|c:C•c ∈ cs}:PB ■

1.3.6. Continuous Behaviours

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

Example 61 . Flow in Pipelines:

- We refer to Examples 33, 48, 49, 50 and 51.
- 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 ■

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

- 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).

1.3.7. Attribute Value Access

- We can distinguish between three kinds of attributes:
  - the **constant attribute**s which are those whose values are **static**;
  - the **programmable attribute**s which are those dynamic values are exclusively set by part processes; and
  - the remaining **dynamic attribute**s

  are here seen as **individual behaviour**s.

1.3.7.1. Access to Static Attribute Values

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

1.3.7.2. Access to External Attribute Values

- By the **external behaviour attributes**
  - we shall thus understand the
    - inert,
    - reactive,
    - autonomous and the
    - biddable

    attributes

98 Let ξA be the set of names, ηA, of all external behaviour attributes.

99 Let $\Pi_{\xi A}$ be the set of indexes into the external attribute channel, say attr_A_ch, one for each distinct attribute name, A, in ξA.

100 Each external behaviour attribute is seen as an individual behaviour, each "accessible" by means of a channel, attr_A_ch.

101 External attribute values are then accessed by the input, from channel attr_A_ch[π]-accessible external attribute behaviours.

102 The **type** of attr_A_ch[π] is considered to be **Unit**$\stackrel{\sim}{\rightarrow}$A.

 © Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

98. **value**
98. ξA: $\{\eta$A|A is any external attribute name$\}$
99. $\Pi_{\xi A}$: Π**-set**
100. **channel**
100. $\{$attr_A_ch$[\,\pi\,]|\pi \in \Pi_{\xi A}\}$
101. **value**
101. attr_A_ch$[\,\pi\,]$?
101. **type**
101. attr_A_ch$[\,\pi\,]$: **Unit**$\stackrel{\sim}{\rightarrow}$A $[$abbrv.:$\mathbb{U}$A$]$

- We shall omit the η prefix in actual descriptions.
- The choice of representing **external behaviour attributes** as behaviours is a technical one.
- See Items 187c. and 187a. Slide 426 for a use of the concept of **external behaviour attribute** channels.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

1.3.7.3. Access to 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 processes.
  - 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:

$$\textbf{value}\ \ \mathsf{f}\text{: } \mathsf{Arg} \rightarrow \mathsf{Arg};\ \ \mathsf{f(a)} \equiv (\ldots\ \textbf{let}\ \mathsf{a}' = \mathsf{F}(\ldots)(\mathsf{a})\ \textbf{in}\ \mathsf{f}(\mathsf{a}')\ \textbf{end})$$

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

1.3.8. Perdurant Signatures and Definitions

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

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

1.3.9. 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 62 . Insert Hub Action Formalisation: We formalise aspects of the above-mentioned hub and link actions:

103 Insertion of a hub requires

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

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

value

103. $\mathsf{insert\_H}\colon \mathsf{H} \rightarrow \mathsf{N} \stackrel{\sim}{\rightarrow} \mathsf{N}$

103. $\mathsf{insert\_H(h)(n)}$ **as** $\mathsf{n'}$

104. **pre**: $\sim\exists\ \mathsf{h'{:}H}\bullet\mathsf{h'} \in$ **obs_part**_Hs(**obs_part**_HS(n))•**uid**_H(h)=**uid**_H(h′)

105. **post**: **obs_part**_Hs(**obs_part**_HS(n′))=**obs_part**_Hs(**obs_part**_HS(n))$\cup\{\mathsf{h}\}$

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

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

    but they are rare!

- **Perdurant (action) analysis thus proceeds as follows:**
  - identifying relevant actions,
  - assigning names to these,
  - delineating the "smallest" relevant state[18],
  - ascribing signatures to action functions, and
  - determining
    - action pre-conditions and
    - action post-conditions.
  - Of these, ascribing signatures is, perhaps, the most crucial:
    - In the process of determining the action signature
    - one oftentimes discovers
    - that part or material attributes have been left "undiscovered".

[18] 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 63 shows examples of signatures whose arguments are
  - either parts,
  - or parts and unique identifiers,
  - or parts and unique identifiers and attributes.

Example 63 . Some Function Signatures:

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

  $$\textbf{value}\ \mathsf{insert\_L}: \mathsf{L} \times (\mathsf{HI} \times \mathsf{HI}) \to \mathsf{N} \xrightarrow{\sim} \mathsf{N}$$

- Removing a hub and removing a link:

  $$\textbf{value}\ \mathsf{remove\_H}: \mathsf{HI} \to \mathsf{N} \xrightarrow{\sim} \mathsf{N}$$
  $$\mathsf{remove\_L}: \mathsf{LI} \to \mathsf{N} \xrightarrow{\sim} \mathsf{N}$$

- Changing a hub state.

  $$\textbf{value}\ \mathsf{change\_H\Sigma}: \mathsf{HI} \times \mathsf{H\Sigma} \to \mathsf{N} \xrightarrow{\sim} \mathsf{N}$$ ■

1.3.10. 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: $\Sigma \times \Sigma \to \textbf{Bool}$
event(σ,σ') **as true**$\lceil\rceil$**false**
pre: $P(\sigma)$
post: $Q(\sigma,\sigma')$

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

- 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 $\mathsf{event}(\sigma,\sigma')$
  - may not be defined for some states σ.
- The resulting state may, or may not, satisfy axioms and well-formedness conditions over Σ — as expressed by the 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.

Example 64 . Link Disappearence Formalisation: We formalise aspects of the above-mentioned link disappearance event:

106 The result net is not well-formed.

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

108 and such a link may disappear such that

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

value

106. $\mathsf{link\_diss\_event}$: $\mathsf{N} \times \mathsf{N}' \times \mathbf{Bool}$
106. $\mathsf{link\_diss\_event(n,n')}$ **as tf**
107. **pre**: **obs_part**_Ls(**obs_part**_$\mathsf{LS(n)}$)$\neq\{\}$
108. **post**: $\exists$ $\mathsf{l{:}L{\cdot}l}$ $\in$ **obs_part**_Ls(**obs_part**_$\mathsf{LS(n)}$) $\Rightarrow$
109. l $\notin$ **obs_part**_Ls(**obs_part**_$\mathsf{LS(n')}$)
109. $\wedge$ $\mathsf{n'} \cup \{\mathsf{l}\}$ $=$ **obs_part**_Ls(**obs_part**_$\mathsf{LS(n)}$) ■

1.3.11. Discrete Behaviour Signatures and Definitions

- We shall only cover behaviour signatures when expressed in RSL/CSP [39].
- 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: ... $\rightarrow$ ... **Unit**

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

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

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

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

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

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

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

  behaviour: ARG $\rightarrow$ ...

- where ARG can be any type expression:

  T, T$\rightarrow$T, T$\rightarrow$T$\rightarrow$T, etcetera

- As shown in [21] 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:
  - its unique identity, $\pi : \Pi$,,
  - its possibly changing mereology, mt:MT[19],
  - the possible components and materials of the part[20], and
  - the constant, the external and the programmable attributes of the part.

[19] For MT see footnote 12 on Slide 158.

[20] — we shall neither treat components nor materials further in this document

- 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 behaviour,
  - (iv) ea:EA lists the external attribute channels of the part behaviour,
  - (v) ps:PA lists the programmable attribute values of the part behaviour, and where
  - (vi) ochs and ichns refer to the shared attributes 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.
  - $\mathcal{S}_\mathcal{A}$: 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.
  - $\mathcal{E}_\mathcal{A}$ EA simply lists the channel indexes to the external attribute values: $((eA_1, \pi_{eA_1}), (eA_2, \pi_{eA_2}), ..., (eA_x, \pi_{eA_x}))$[21] where x is the number, 0 or more, of external attributes of parts p:P.
  - $\mathcal{P}_\mathcal{A}$ PA simply lists appropriate programmable value expression type:
    $(pvT_1, pvT_2, ..., pvT_q)$
    where q is the number of programmable attributes of parts p:P

[21] See paragraph *Access to External Attribute Values* on Slide 274.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

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

Process Schema I: Abstract `is_composite(p)`

value
  compile_process: P $\rightarrow$ RSL-**Text**
  compile_process(p) $\equiv$
    $\mathcal{M}_{cP_{\mathrm{CORE}}}$(**uid**_P(p),**obs_mereo**_P(p),$\mathcal{S}_{\mathcal{A}}$(p),$\mathcal{E}_{\mathcal{A}}$(p))($\mathcal{P}_{\mathcal{A}}$(p))
    $\|$ compile_process(**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 122.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

- Let P be a composite sort defined in terms of the concrete type **Q-set**.
  - The process compiled from p:P, is composed from
    - a process, $\mathcal{M}_{cP_{\mathrm{CORE}}}$, 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 $\rightarrow$ RSL-**Text**

compile_process(p) $\equiv$

$\mathcal{M}_{cP_{\mathrm{CORE}}}$(**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 $\in$ qs}

Process Schema III: is_atomic(p)

value

compile_process: P $\rightarrow$ RSL-**Text**

compile_process(p) $\equiv$

$\mathcal{M}_{aP_{\mathrm{CORE}}}$(**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 – May 23, 2015: 15:29

Example 65 . Bus Timetable Coordination:

- We refer to Examples 20 on Slide 123, 21 on Slide 130, 43 on Slide 198 and 59 on Slide 265.

110 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.

111 The **fleet** process is compiled as per Process Schema II (Slide 297)

type

Δ, F, VC [Example 20 on Slide 123]

V, Vs=V-**set** [Example 21 on Slide 130]

FI, VI, BT [Example 43 on Slide 198]

channel

{fch...} [Example 59 on Slide 265]

value

110. δ:Δ,
110. f:F = **obs_part**_F(δ),
110. vs:V-**set** = **obs_part**_Vs(**obs_part**_VC(f)),
110. bt:BT = **attr**_BT(f)

axiom

110. $\forall$ v:V•v $\in$ vs $\Rightarrow$ bt = **attr**_BT(v) [Example 43 on Slide 198]

value

111. fleet: fi:FI$\times$BT $\rightarrow$ **in**,**out** {fch[{fi,**uid**_V(v)}]|v:V•v $\in$ vs} **process**
111. fleet(fi,bt) $\equiv$
111. $\mathcal{M}_F$(fi,bt)
111. $\|$ $\|$ {vehicle(**uid**_V(v),fi:FI,bt)|v:V•v $\in$ vs}
111. vehicle: vi:VI$\times$fi:FI$\times$bt:BT $\rightarrow$ **in**,**out** fch[{fi,vi}] **process**
111. vehicle(vi,fi,bt) $\equiv$ $\mathcal{M}_V$(vi,fi,bt)

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

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

value

$\mathcal{M}_F$: fi:FI$\times$bt:BT $\rightarrow$ **in**,**out** $\{$fch$[\,\{$fi,**uid**_V(v)$\}\,]|$v:V$\cdot$v $\in$ vs$\}$ **process**
$\mathcal{M}_F$(fi,bt) $\equiv$ **let** bt$'$ = $\mathcal{F}$(fi,bt) **in** $\mathcal{M}_F$(fi,bt$'$) **end**

$\mathcal{M}_V$: vi:VI$\times$fi:FI$\times$bt:BT $\rightarrow$ **in**,**out** fch$[\,\{$fi,vi$\}\,]$ **process**
$\mathcal{M}_V$(vi,fi,bt) $\equiv$ **let** bt$'$ = $\mathcal{V}$(vi,bt) **in** $\mathcal{M}_V$(vi,fi,bt$'$) **end**

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

are simple actions.

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

value

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

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

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

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:$\Pi\times$me:MT$\times$sa:SA$\times$ea:EA$\rightarrow$pa:PA$\rightarrow$**in** inchs **out** ochs **Unit**
$\mathcal{M}_{cP_{\text{CORE}}}$($\pi$,me,sa,ea)(pa) $\equiv$
  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:$\Pi\times$me:MT$\times$sa:SA$\times$ea:EA$\rightarrow$PA$\rightarrow$**in** inchs **out** ochs $\rightarrow$ MT$\times$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_ch**$[(\mathsf{A},\pi)]$?, to external attributes.
- An example of the update of programmable attributes is shown in the **veh**icle definitions in Sect. 6.2.3, Slides 344 and 346.

- The $\mathcal{F}$ action non-deterministically internal choice chooses between
  - either [1,2,3,4]
    - [1] accepting input from
    - [4] another part process,
    - [2] then optionally offering a reply to that other process, and
    - [3] finally delivering an updated state;
  - or [5,6,7,8] offering
    - [5] an output,
    - [6] val,
    - [8] to another part process,
    - [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) ≡
          ⊔⊓{ let val = ch[π′] ? in
               ch[π′] ! in_reply(sa,ea,pa)(val) ;
               in_update(me,sa,ea,pa)(π′,sa,ea,pa) end
             | π′ ∈ ℰ(π,me)}
       ⌈⌉⊔⊓{ let (π′,val) = await_reply(me,sa,ea,pa) in
               ch[π′] ! out_reply(val,sa,ea,pa) ;
               out_update(me,sa,ea,pa) end
             | π′ ∈ ℰ(π,me)}
       ⌈⌉      (me,own_work(sa,ea,pa))

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


© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

1.3.12. Concurrency: Communication and Synchronisation

- Process Schemas I, II and IV (Slides 295, 297 and 305), 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.

1.3.13. 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.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

1.3.13.1. 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.

1.3.13.2. Discussion

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

Dines Bjørner's MAP-i Lecture # 5

End of MAP-i Lecture # 5: Perdurants: Actions, Events and Behaviours

Tuesday, 26 May 2015: 10:00–10:45

Dines Bjørner's MAP-i Lecture # 6

A Domain Description

Tuesday, 26 May 2015: 12:00–13:00

6. A Domain Description

6.1. Endurants

6.1.1. Domain, Net, Fleet and Monitor

- The root domain, $\Delta_{\mathcal{D}}$,
- the step-wise unfolding of whose description is to be exemplified, 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.

112 We analyse the composite traffic system into

a. a composite road net,

b. a composite fleet (of vehicles), and

c. an atomic monitor.

113 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 – May 23, 2015: 15:29

type

112. Δ_Δ

112a.. N_Δ

112b.. F_Δ

112c.. M_Δ

value

112a.. $\textbf{obs\_part\_}\mathsf{N}_\Delta$: $\Delta_\Delta \to \mathsf{N}_\Delta$

112b.. $\textbf{obs\_part\_}\mathsf{F}_\Delta$: $\Delta_\Delta \to \mathsf{F}_\Delta$

112c.. $\textbf{obs\_part\_}\mathsf{M}_\Delta$: $\Delta_\Delta \to \mathsf{M}_\Delta$

type

113a.. HA_Δ

113b.. LA_Δ

value

113a.. $\textbf{obs\_part\_}\mathsf{HA}_\Delta$: $\mathsf{N}_\Delta \to \mathsf{HA}_\Delta$

113b.. $\textbf{obs\_part\_}\mathsf{LA}_\Delta$: $\mathsf{N}_\Delta \to \mathsf{LA}_\Delta$

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

6.1.2. Hubs and Links

114 Hub aggregates are sets of hubs.

115 Link aggregates are sets of links.

116 Fleets are set of vehicles.

117 We introduce some auxiliary functions.

a. links extracts the links of a network.

b. hubs extracts the hubs of a network.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

type

114. H_Δ, $\mathsf{HS}_\Delta = \mathsf{H}_\Delta$**-set**

115. L_Δ, $\mathsf{LS}_\Delta = \mathsf{L}_\Delta$**-set**

116. V_Δ, $\mathsf{VS}_\Delta = \mathsf{V}_\Delta$**-set**

value

114. **obs_part**_HS_Δ: $\mathsf{HA}_\Delta \rightarrow \mathsf{HS}_\Delta$

115. **obs_part**_LS_Δ: $\mathsf{LA}_\Delta \rightarrow \mathsf{LS}_\Delta$

116. **obs_part**_VS_Δ: $\mathsf{F}_\Delta \rightarrow \mathsf{VS}_\Delta$

117a.. links_Δ: $\Delta_\Delta \rightarrow$ L-**set**

117a.. $\mathsf{links}_\Delta(\delta_\Delta) \equiv$ **obs_part**_LS(**obs_part**_LA(δ_Δ))

117b.. hubs_Δ: $\Delta_\Delta \rightarrow$ H-**set**

117b.. $\mathsf{hubs}_\Delta(\delta_\Delta) \equiv$ **obs_part**_HS(**obs_part**_HA(δ_Δ))

6.1.3. Unique Identfiers

We cover the unique identifiers of all parts, whether needed or not.

118 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.

119 We introduce some auxiliary functions:

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

b. xtr_his extracts all hub identifiers of a traffic system.

c. given an appropriate link identifier and a net get_link 'retrieves' the designated link.

d. given an appropriate hub identifier and a net get_hub 'retrieves' the designated hub.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

type

118a.. NI, HAI, LAI, HI, LI, FI, VI, MI

value

118c.. **uid**_NI: $N_\Delta \to$ NI

118c.. **uid**_HAI: $HA_\Delta \to$ HAI

118c.. **uid**_LAI: $LA_\Delta \to$ LAI

118c.. **uid**_HI: $H_\Delta \to$ HI

118c.. **uid**_LI: $L_\Delta \to$ LI

118c.. **uid**_FI: $F_\Delta \to$ FI

118c.. **uid**_VI: $V_\Delta \to$ VI

118c.. **uid**_MI: $M_\Delta \to$ MI

axiom

118b.. $NI \bigcap HAI = \emptyset$, $NI \bigcap LAI = \emptyset$, $NI \bigcap HI = \emptyset$, etc.

where axiom 118b.. is expressed semi-formally, in mathematics.

value

119a.. xtr_lis: $\Delta_\Delta \rightarrow$ LI-**set**
119a.. xtr_lis(δ_Δ) $\equiv$
119a.. **let** ls = links(δ_Δ) **in** {**uid**_LI(l)|l:L•l $\in$ ls} **end**
119b.. xtr_his: $\Delta_\Delta \rightarrow$ HI-**set**
119b.. xtr_his(δ_Δ) $\equiv$
119b.. **let** hs = hubs(δ_Δ) **in** {**uid**_HI(h)|h:H•k $\in$ hs} **end**
119c.. get_link: LI $\rightarrow \Delta_\Delta \stackrel{\sim}{\rightarrow}$ L
119c.. get_link(li)(δ_Δ) $\equiv$
119c.. **let** ls = links(δ_Δ) **in**
119c.. **let** l:L • l $\in$ ls $\wedge$ li=**uid**_LI(l) **in** l **end end**
119c.. **pre**: li $\in$ xtr_lis(δ_Δ)
119d.. get_hub: HI $\rightarrow \Delta_\Delta \stackrel{\sim}{\rightarrow}$ H
119d.. get_hub(hi)(δ_Δ) $\equiv$
119d.. **let** hs = hubs(δ_Δ) **in**
119d.. **let** h:H • h $\in$ hs $\wedge$ hi=**uid**_HI(h) **in** h **end end**
119d.. **pre**: hi $\in$ xtr_his(δ_Δ)

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

6.1.4. Mereology

We cover the mereologies of all part sorts introduced so far. We decide that nets, hub aggregates, link aggregates and fleets have no mereologies of interest.

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

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

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

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

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

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

126 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

120. $\textbf{obs\_mereo\_}\mathsf{H}_\Delta$: $\mathsf{H}_\Delta \rightarrow$ LI-**set**

121. **obs_mereo**_L: L $\rightarrow$ HI-**set** **axiom** $\forall$ l:L$\cdot$**card** **obs_mereo**_L(l)=2

122. **obs_mereo**_V: V $\rightarrow$ MI

123. **obs_mereo**_M: M $\rightarrow$ VI-**set**

axiom

124. $\forall\ \delta:\Delta$, hs:$\mathsf{HS}_\Delta\cdot$hs=hubs(δ), ls:$\mathsf{LS}_\Delta\cdot$ls=links(δ) $\cdot$

124. $\forall$ h:$\mathsf{H}_\Delta\cdot$h $\in$ hs$\cdot$**obs_mereo**_H(h)$\subseteq$xtr_his(δ) $\wedge$

125. $\forall$ l:$\mathsf{L}_\Delta\cdot$l $\in$ ls$\cdot$**obs_mereo**_L(l)$\subseteq$xtr_lis(δ) $\wedge$

126a.. **let** f:$\mathsf{F}_\Delta\cdot$f=**obs_part**_F(δ) $\Rightarrow$

126a.. **let** m:$\mathsf{M}_\Delta\cdot$m=**obs_part**_M(δ),

126a.. vs:VS$\cdot$vs=**obs_part**_VS(f) **in**

126a.. $\forall$ v:$\mathsf{V}_\Delta\cdot$v $\in$ vs$\Rightarrow$**uid**_V(v) $\in$ **obs_mereo**_M(m)

126b.. $\wedge$ **obs_mereo**_M(m) = {**uid**_V(v)|v:V$\cdot$v $\in$ vs}

126b.. **end** **end**

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

6.1.5. Attributes, I

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

- **Hubs:**
  - *location*s are considered static,
  - *wear and tear* (condition of road surface) is considered inert,
  - *hub states* and *hub state space*s are considered programmable;
- **Links:**
  - *length*s and *location*s are considered static,
  - *wear and tear* (condition of road surface) is considered inert,
  - *link states* and *link state space*s are considered programmable;

- **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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

6.1.6. Attributes, II

We treat one attribute each for hubs, links, vehicles and the monitor. First we treat hubs.

127 Hubs

a. have *hub states* which are sets of pairs of identifiers of links connected to the hub[22],

b. and have *hub state spaces* which are sets of hub states[23].

128 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.

129 Hubs have geodetic and cadestral location.

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

[22]A hub state "signals" which input-to-output link connections are open for traffic.

[23]A hub state space indicates which hub states a hub may attain over time.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

type
127a.. $\mathsf{H\Sigma} = (\mathsf{LI}\times\mathsf{LI})$**-set**
127b.. $\mathsf{H\Omega} = \mathsf{H\Sigma}$**-set**
value
127a.. **attr**_$\mathsf{H\Sigma}$: $\mathsf{H} \to \mathsf{H\Sigma}$
127b.. **attr**_$\mathsf{H\Omega}$: $\mathsf{H} \to \mathsf{H\Omega}$
axiom
128. $\forall\ \delta:\Delta$,
128. **let** hs = hubs(δ) **in**
128. $\forall$ h:H • h $\in$ hs •
128a.. xtr_lis(h)$\subseteq$xtr_lis(δ)
128b.. $\wedge$ **attr**_Σ(h) $\in$ **attr**_Ω(h)
128. **end**
type
129. HGCL
value
129. **attr**_HGCL: H $\to$ HGCL
130. xtr_lis: H $\to$ LI**-set**
130. xtr_lis(h) $\equiv$
130. {li | li:LI,(li′,li″):LI$\times$LI •
130. (li′,li″) $\in$ **attr**_HΣ(h) $\wedge$ li $\in$ {li′,li″}}

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

Then links.

131 Links have lengths.

132 Links have geodetic and cadestral location.

133 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.

type
131. LEN
132. LGCL
133a.. $\mathsf{L}\Sigma$ = (HI×HI)-**set**
133b.. $\mathsf{L}\Omega$ = $\mathsf{L}\Sigma$-**set**
value
131. **attr**_LEN: L → LEN
132. **attr**_LGCL: L → LGCL
133a.. **attr**_$\mathsf{L}\Sigma$: L → $\mathsf{L}\Sigma$
133b.. **attr**_$\mathsf{L}\Omega$: L → $\mathsf{L}\Omega$
axiom
133. ∀ n:N •
133. **let** ls = xtr−links(n), hs = xtr_hubs(n) **in**
133. ∀ l:L•l ∈ ls ⇒
133a.. **let** lσ = **attr**_$\mathsf{L}\Sigma$(l) **in**
133a.. 0≤**card** lσ≤4
133a.. ∧ ∀ (hi′,hi″):(HI×HI)•(hi′,hi″) ∈ lσ ⇒
133a.. {get_H(hi′)(n),get_H(hi″)(n)}=**obs_mereo**_L(l)
133b.. ∧ **attr**_$\mathsf{L}\Sigma$(l) ∈ **attr**_$\mathsf{L}\Omega$(l)
133. **end end**

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

Then vehicles.

134 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.

type

134. $\mathsf{VPos} = \mathsf{onL} \mid \mathsf{atH}$

134a.. $\mathsf{onL} :: \mathsf{LI}\ \mathsf{HI}\ \mathsf{HI}\ \mathsf{R}$

134b.. $\mathsf{R} = \mathbf{Real}$ $\quad$ **axiom** $\forall\ \mathsf{r}{:}\mathsf{R} \cdot 0 \leq \mathsf{r} \leq 1$

134c.. $\mathsf{atH} :: \mathsf{HI}\ \mathsf{LI}\ \mathsf{LI}$

value

134. $\textbf{attr}\_\mathsf{VPos}{:}\ \mathsf{V}_\triangle \rightarrow \mathsf{VPos}$

axiom

134a.. $\forall\ \mathsf{n}_\triangle{:}\mathsf{N}_\triangle,\ \mathsf{onL(li,fhi,thi,r){:}VPos} \cdot$

134a.. $\quad \exists\ \mathsf{l}_\triangle{:}\mathsf{L}_\triangle \cdot \mathsf{l}_\triangle \in \textbf{obs\_part}\_\mathsf{LS}(\textbf{obs\_part}\_\mathsf{N}_\triangle(\mathsf{n}_\triangle))$

134a.. $\quad\quad \Rightarrow \mathsf{li} = \textbf{uid}\_\mathsf{L}_\triangle(\mathsf{l}) \wedge \{\mathsf{fhi,thi}\} = \textbf{obs\_mereo}\_\mathsf{L}_\triangle(\mathsf{l}_\triangle),$

134c.. $\forall\ \mathsf{n}_\triangle{:}\mathsf{N}_\triangle,\ \mathsf{atH(hi,fli,tli){:}VPos} \cdot$

134c.. $\quad \exists\ \mathsf{h}_\triangle{:}\mathsf{H}_\triangle \cdot \mathsf{h}_\triangle \in \textbf{obs\_part}\_\mathsf{HS}_\triangle(\textbf{obs\_part}\_\mathsf{N}(\mathsf{n}_\triangle))$

134c.. $\quad\quad \Rightarrow \mathsf{hi} = \textbf{uid}\_\mathsf{H}_\triangle(\mathsf{h}_\triangle) \wedge (\mathsf{fli,tli}) \in \textbf{attr}\_\mathsf{L\Sigma}(\mathsf{h}_\triangle)$

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

135 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!

136 We define two auxiliary functions:

a. xtr_links extracts all links of a net and

b. xtr_hub extracts all hubs of a net.

type

135b.. $\mathsf{MAP} = \mathsf{VI} \underset{m}{\rightarrow} \mathsf{VPos}$

135b.. $\forall\ \mathsf{map:MAP} \cdot \textbf{card dom}\ \mathsf{map} = \textbf{card rng}\ \mathsf{map}$

value

135. $\mathsf{distribute:}\ \mathsf{VS}_\Delta \rightarrow \mathsf{N}_\Delta \rightarrow \mathsf{MAP}$

135. $\mathsf{distribute(vs_\Delta)(n_\Delta)} \equiv$

135a.. $\quad \textbf{let}\ \mathsf{(hs,ls) = (xtr\_hubs(n_\Delta),xtr\_links(n_\Delta))}\ \textbf{in}$

135a.. $\quad \textbf{let}\ \mathsf{vps} = \{\mathsf{onL}(\textbf{uid\_}(\mathsf{l}_\Delta),\mathsf{fhi,thi,r})|\mathsf{l}_\Delta\mathsf{:L}_\Delta\cdot\mathsf{l}_\Delta\in\mathsf{ls}\wedge\{\mathsf{fhi,thi}\}\subseteq\textbf{obs\_mereo}\mathsf{\_L(l)}\wedge 0\leq\mathsf{r}\leq 1$

135a.. $\qquad\quad \cup\ \{\mathsf{atH}(\textbf{uid}\mathsf{\_H}(\mathsf{h}_\Delta),\mathsf{fli,tli})|\mathsf{h}_\Delta\mathsf{:H}_\Delta\cdot\mathsf{h}_\Delta\in\mathsf{hs}\wedge\{\mathsf{fli,tli}\}\subseteq\textbf{obs\_mereo}\mathsf{\_H}_\Delta(\mathsf{h}_\Delta)\}\ \textbf{in}$

135b.. $\quad [\,\textbf{uid}\mathsf{\_V}_\Delta(\mathsf{v})\mapsto\mathsf{vp}|\mathsf{v}_\Delta\mathsf{:V}_\Delta\mathsf{,vp:VPos}\cdot\mathsf{v}_\Delta\in\mathsf{vs}\wedge\mathsf{vp}\in\mathsf{vps}\,]$

135. $\quad \textbf{end end}$

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

136a.. $\mathsf{xtr\_links}_{\triangle}$: $\mathsf{N}_{\triangle} \rightarrow \mathsf{L}_{\triangle}$**-set**
136a.. $\mathsf{xtr\_links}_{\triangle}(\mathsf{n}_{\triangle})\equiv$**obs_part**_LS(**obs_part**_LA($\mathsf{n}_{\triangle}$))
136b.. $\mathsf{xtr\_hubs}_{\triangle}$: $\mathsf{N}_{\triangle} \rightarrow \mathsf{H}_{\triangle}$**-set**
136a.. $\mathsf{xtr\_hubs}_{\triangle}(\mathsf{n}_{\triangle})\equiv$**obs_part**_$\mathsf{HS}_{\triangle}$(**obs_part**_$\mathsf{HA}_{\triangle}$($\mathsf{n}_{\triangle}$))

And finally monitors. We consider only one monitor attribute.

137 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 – May 23, 2015: 15:29

```
type
137.   Traffic = VI →m (T × VPos)*
value
137.   attr_Traffic: M → Traffic
axiom
137b..   ∀ δ:Δ •
137b..       let m = obs_part_M_Δ(δ) in
137b..       let tf = attr_Traffic(m) in
137b..       dom tf ⊆ xtr_vis(δ) ∧
137b..       ∀ vi:VI • vi ∈ dom tf •
137b..          let tr = tf(vi) in
137b..          ∀ i,i+1:Nat • {i,i+1}⊆dom tr •
137b..             let (t,vp)=tr(i),(t′,vp′)=tr(i+1) in
137b..             t<t′
137(b.)i.           ∧ case (vp,vp′) of
137(b.)i.                (onL(li,fhi,thi,r),onL(li′,fhi′,thi′,r′))
137(b.)i.                  → li=li′∧fhi=fhi′∧thi=thi′∧r≤r′
137(b.)i.                  ∧ li ∈ xtr_lis(δ)
137(b.)i.                  ∧ {fhi,thi} = obs_mereo_L(get_link(li)(δ)),
137(b.)ii.               (atH(hi,fli,tli),atH(hi′,fli′,tli′))
137(b.)ii.                 → hi=hi′∧fli=fli′∧tli=tli′
137(b.)ii.                 ∧ hi ∈ xtr_his(δ)
137(b.)ii.                 ∧ (fli,tli) ∈ obs_mereo_H(get_hub(hi)(δ)),
137(b.)iii.              (onL(li,fhi,thi,1),atH(hi,fli,tli))
137(b.)iii.                → li=fli∧thi=hi
137(b.)iii.                ∧ {li,tli} ⊆ xtr_lis(δ)
137(b.)iii.                ∧ {fhi,thi}=obs_mereo_L(get_link(li)(δ))
137(b.)iii.                ∧ hi ∈ xtr_his(δ)
137(b.)iii.                ∧ (fli,tli) ∈ obs_mereo_H(get_hub(hi)(δ)),
137(b.)iv.               (atH(hi,fli,tli),onL(li′,fhi′,thi′,0))
137(b.)iv.                 →  etcetera,
137b..            _  → false
137b..       end end end end end
```

6.1.7. Routes

- We bring a model of routes.

TO BE WRITTEN

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

6.2. Perdurants

6.2.1. Vehicle to Monitor Channel

138 Let δ be the traffic system domain.

139 Then focus on the set of vehicles

140 and the monitor —

141 and we obtain an appropriate channel array for communication between vehicles and the traffic observing monitor.

value

139. **let** vs:VS $\cdot$ vs $=$ **obs_part**_VS(**obs_part**_F(δ)),

140. m:M $\cdot$ m $=$ **obs_part**_M(δ) **in**

channel

141. {v_m_ch[**uid**_VI(v),**uid**_MI(m)]|v:V·v $\in$ vs} **end**

6.2.2. Link Disappearance Event

We formalise aspects of the above-mentioned link disappearance event:

142 The result net, n':N', is not well-formed.

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

144 and such a link may disappear such that

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

value

142. link_diss_event: $N \times N' \times$ **Bool**
142. link_diss_event(n,n′) **as** tf
143. **pre**: **obs_part**_Ls(**obs_part**_LS(n))$\neq$\{\}
144. **post**: $\exists$ l:L·l $\in$ **obs_part**_Ls(**obs_part**_LS(n)) $\Rightarrow$
145. l $\notin$ **obs_part**_Ls(**obs_part**_LS(n′))
145. $\wedge$ n′ $\cup$ \{l\} $=$ **obs_part**_Ls(**obs_part**_LS(n))

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

6.2.3. Road Traffic

Global Values

- There is given some globally observable parts.

146 besides the domain, δ_Δ:Δ_Δ,

147 a net, n:N,

148 a set of vehicles, vs:V-**set**,

149 a monitor, m:M, and

150 a clock, clock, behaviour.

151 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.

value

146. $\delta_\triangle$:$\Delta_\triangle$

147. n:N = **obs_part**_N($\delta_\triangle$),

147. ls:L-**set**=linksLs(δ),hs:H-**set**=hubs($\delta_\triangle$),

147. lis:LI-**set**=xtr_lis(δ),his:HI-**set**=xtr_his($\delta_\triangle$)

148. vs:V-**set**=**obs_part**_Vs(**obs_part**_VS(**obs_part**_F(δ)$_\triangle$)),

148. vis:VI-**set** = {**uid**_VI(v)|v:V·v ∈ vs},

149. m:**obs_part**_M(δ), mi=**uid**_MI(m), ma:**attributes**(m)

150. clock: $\mathbb{T}$ → **out** {clk_ch[vi|vi:VI·vi ∈ vis]} **Unit**

151. vm:MAP·vpos_map = distribute(vs)(n);

Channels

152 We additionally declare a set of vehicle to monitor channels indexed

a. by the unique identifiers of vehicles
b. and the (single) monitor identifier.[24]

and communicating vehicle positions.

channel

152. $\{\mathsf{v\_m\_ch}[\,\mathsf{vi,mi}\,]|\mathsf{vi{:}VI{\cdot}vi} \in \mathsf{vis}\}\mathsf{:VPos}$

[24] Technically speaking: we could omit the monitor identifier.

Behaviour Signatures

153 The road traffic system behaviour, rts, takes no arguments; and "behaves", that is, continues forever.

154 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".

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

155 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

153. trs: **Unit** → **Unit**
154. veh$_\Delta$: vi:VI × mi:MI → vp:VPos →
154. **out** vm_ch[vi,mi] **Unit**
155. mon$_\Delta$: m:M$_\Delta$ × vis:VI-**set** → RTF →
155. **in** {v_m_ch[vi,mi]|vi:VI•vi ∈ vis},clk_ch **Unit**

The Road Traffic System Behaviour

156 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

156. $\mathsf{trs}() =$

156a.. $\quad \| \; \{\mathsf{veh}_\triangle(\mathbf{uid}\_\mathsf{VI}(\mathsf{v}),\mathsf{mi})(\mathsf{vm}(\mathbf{uid}\_\mathsf{VI}(\mathsf{v})))|\mathsf{v}:\mathsf{V}\cdot\mathsf{v} \in \mathsf{vs}\}$

156b.. $\quad \| \; \mathsf{mon}_\triangle(\mathsf{mi},\mathsf{vis})([\,\mathsf{vi}\mapsto\langle\rangle|\mathsf{vi}:\mathsf{VI}\cdot\mathsf{vi} \in \mathsf{vis}\,])$

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

- where, wrt, the monitor, we
  - dispense with the mereology and the attribute state arguments
  - and instead just have a **mon**itor 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.

157 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,

   iii whereupon the vehicle resumes the vehicle behaviour positioned at the very beginning of that link,

c. or, again internally non-deterministically,

d. the vehicle "disappears — off the radar" !

 © Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

```
157.        veh△(vi,mi)(vp:atH(hi,fli,tli)) ≡
157a..              v_m_ch[ vi,mi ]!vp ; veh△(vi,mi)(vp)
157b..          ⌈⌉
157(b.)i.           let {hi′,thi}=obs_mereo_L(get_link(tli)(n)) in
157(b.)i.                                   assert: hi′=hi
157(b.)ii.          v_m_ch[ vi,mi ]!onL(tli,hi,thi,0) ;
157(b.)iii.         veh△(vi,mi)(onL(tli,hi,thi,0)) end
157c..          ⌈⌉
157d..              stop
```

158 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,

c. 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, 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),

A the vehicle informs the monitor that it is now at the hub identified by thi,

B whereupon the vehicle resumes the vehicle behaviour positioned at that hub.

159 or, internally non-deterministically,

160 the vehicle "disappears — off the radar" !

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

158. $\mathsf{veh}_\triangle$(vi,mi)(vp:onL(li,fhi,thi,r)) $\equiv$
158a.. v_m_ch[vi,mi]!vp ; veh($_\triangle$vi,mi,va)(vp)
158b.. $\lceil\rceil$
158c.. **if** r + $\ell_\epsilon \leq 1$
158(c.)i. **then** v_m_ch[vi,mi]!onL(li,fhi,thi,r+ℓ_ϵ) ;
158(c.)i. $\mathsf{veh}_\triangle$(vi,mi)(onL(li,fhi,thi,r+ℓ_ϵ))
158(c.)ii. **else let** li′:LI•li′ $\in$ **obs_mereo**_H(get_hub(thi)(n)) **in**
158(c.)iiA. v_m_ch[vi,mi]!atH(li,thi,li′);
158(c.)iiB. $\mathsf{veh}_\triangle$(vi,mi)(atH(li,thi,li′)) **end end**
159. $\lceil\rceil$
160. **stop**

The Monitor Behaviour

161 The **mon**itor 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".

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

162 Either the monitor “does own work”

163 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.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

161. $\mathsf{mon}_\triangle(\mathsf{mi,vis})(\mathsf{trf}) \equiv$
162. $\quad\quad \mathsf{mon}_\triangle(\mathsf{mi,vis})(\mathsf{trf})$
163. $\quad \sqcap$
163a.. $\quad\quad \lfloor\!\rceil\{\mathbf{let}\ \mathsf{tvp} = (\mathsf{clk\_ch?,v\_m\_ch}[\,\mathsf{vi,mi}\,]?)\ \mathbf{in}$
163b.. $\quad\quad\quad \mathbf{let}\ \mathsf{trf}' = \mathsf{trf} \dagger [\,\mathsf{vi} \mapsto \mathsf{trf(vi)}\widehat{\ }\langle\mathsf{tvp}\rangle\,]\ \mathbf{in}$
163c.. $\quad\quad\quad \mathsf{mon}_\triangle(\mathsf{mi,vis})(\mathsf{trf}')$
163d.. $\quad\quad\quad \mathbf{end\ end} \mid \mathsf{vi:VI} \cdot \mathsf{vi} \in \mathsf{vis}\}$

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

- We are about to complete a long, i.e., a 16 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.

- 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!

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

Dines Bjørner's MAP-i Lecture # 6

End of MAP-i Lecture # 6: A Domain Description

Tuesday, 26 May 2015: 12:00–13:00

Dines Bjørner's MAP-i Lecture # 7

Requirements – An Overview and Projection

Tuesday, 26 May 2015: 15:30–16:15

7. Requirements

- In Chapter 1. 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,
  - interface requirements and
  - machine requirements.
- The **machine** is the hardware and software
  to be developed from the requirements ■

- **Domain requirements** are those requirements which can be expressed sôlely using technical terms of the domain ■
- **Interface requirements** are those requirements which can be expressed only using technical terms of both the domain and the machine ■
- **Machine requirements** are those requirements which can be expressed sôlely using technical terms of the machine ■

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

- We show principles, techniques and tools for "deriving"
  - domain requirements and
  - interface requirements.
- The domain requirements development focus on
  - projection,
  - instantiation,
  - determination,
  - extension and
  - fitting.

- These domain-to-requirements operators can be described briefly:
  - **projection** removes such descriptions which are to be omitted for consideration in the requirements,
  - **instantiation** mandates specific mereologies,
  - **determination** specifies less non–determinism,
  - **extension** extends the evolving requirements prescription with further domain description aspects and
  - **fitting** resolves “loose ends” as they may have emerged during the domain-to-requirements operations.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

7.1. Introduction

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"* ■

7.1.1. General Considerations

- The objective of requirements engineering is to create a **requirements prescription**:
  - A **requirements prescription** specifies externally observable properties of endurants and perdurants: functions, events and behaviours 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.
- We shall show how a major part of a requirements prescription can be “derived” from “its” prerequisite domain description.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

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.

Rule 2 An "Ideal Rule" of Requirements Engineering: *When prescribing (including formalising) requirements, also formulate tests and properties for model checking and theorems whose actualisation 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.

 © Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

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, stay short of expressing such designs.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

Rule 5 Requirements Verifiability and Validability: *Make sure that requirements are verifiable and can be validated* ■

- That is, do not express a requirement for which you have no assurance that it can be verified and validated.
- In other words,
  - once a first-level software design has been proposed,
  - one must show that it satisfies the requirements.
- Thus specific parts of even abstract software designs are usually provided with references to specific parts of the requirements that they are (thus) claimed to implement.

Definition 19 . Requirements (II): *By* **requirements** *we shall understand a document which prescribes desired properties of a machine:*

- *(i) what endurants the machine shall "maintain", and*
- *what the machine shall (must; not should) offer of*
  - *(ii) functions and of*
  - *(iii) behaviours*
- *(iv) while also expressing which events the machine shall "handle"* ■

- 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.
- So we find
  - that although we may believe that our job is software engineering,
- important parts of our job are to also "design the machine"!

 © Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

7.1.2. Four Stages of Requirements Development

- We shall unravel requirements in four stages — the first three stages are sketchy (and thus informal) while the last stage
  - is systematic,
  - mandates both strict narrative,
  - and formal descriptions, and
  - is "derivable" from the domain description.
- The four stages are:
  - the *problem/objective* sketch,
  - the narrative **system requirements sketch**,
  - the narrative **user requirements sketch**, and
  - the systematic narrative and formal **functional requirements prescription**.

7.1.2.1. Problem and/or Objective Sketch

Definition 22 . Problem/Objective Sketch: *By a* **problem/objective sketch** *we understand*

- *a narrative which emphasises*
- *what the problem or objectie is*
- *and thereby names its main concepts* ■

 © Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

Example 66 . The Problem/Objective Requirements: A Sketch:

- The objective is to create a **road-pricing product.**
  - By a road-pricing product
    - we shall understand an information technology-based system
    - containing computers and communications equipment and software
    - that enables the recording of *vehicle* movements
    - within a well-delineated *road net*
    - and thus enables
      - the *owner* of the road net
      - to charge
      - the *owner* of the vehciles
      - *fees* for the usage of that road net

7.1.2.2. Systems Requirements

Definition 23 . System Requirements: *By a* **system requirements narrative** *we understand*

- *a narrative which emphasises*
- *the overall hardware and software*
- *system components* ■

 © Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

Example 67 . The Road-pricing System Requirements: A Narrative:

- The requirements are based on the following a-priori given constellation of system components:
  - there is assumed a GNSS: a `Global Navigation Satellite System`;
  - there are specially equipped *vehicles*;
  - there is a well-delineated road net called a *toll-road* net with specially equipped toll-gates with *barrier*s which afford (only the specially equipped) vehicles to enter into and exit from the toll-road net; and
  - there is a [road-pricing] *calculator*.

- These four system components are required to behave and interact as follows:
  - The `GNSS` is assumed to continuously offer vehicles timed information about their global positions;
  - *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-gate barriers;
  - *toll-road 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 – May 23, 2015: 15:29

- The requirements are therefore to include requirements to
  - the `GNSS` radio telecommunications equipment,
  - the vehicle `GNSS receiver` equipment,
  - the vehicle software,
  - the toll-gate in and out sensor equipment,
  - the electro-mechanical toll-gate barrier equipment,
  - the toll-gate barrier actuator equipment,
  - the toll-gate software,
  - the actuator software, and
  - the communications

- 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 – May 23, 2015: 15:29

7.1.2.3. 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*
  - *the human user and*
  - *external equipment*

  interfaces
- *to the system components* ■

Example 68 . 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 service calculator staff.
- The external equipment are
  - the GNSS satellites
  - and the telecommunications equipment
    - which enables communication between
    - the GNSS satellite sand vehicles ,
    - vehicles and the road-pricing calculator,
    - toll-gates and the road-pricing calculator and
    - the road-pricing calculator and vehicles (for billing),
  - We defer expression of
    - human user and
    - external equipment requirements

    till our treatment of relevant functional requirements

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

7.1.2.4. Functional Requirements

Definition 25 . Functional Requirements: *By* **functional requirements** *we understand precise prescriptions of*

- *the endurants*
- *and perdurants*

of the system components ∎

- There are, as we see it, three kinds of requirements:
  - domain requirements,
  - interface requirements and
  - machine requirements

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

- **Domain requirements** are those requirements which can be expressed sôlely using technical terms of the domain ■
- **Interface requirements** are those requirements which can be expressed only using technical terms of both the domain and the machine ■
- **Machine requirements** are those requirements which can be expressed sôlely using technical terms of the machine ■

7.2. Domain Requirements

Definition 26 . 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 stake-holders.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

- 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**,
    - **fitting**

  and, perhaps, other domain description to requirements prescription operations.

7.2.1. Domain Projection

Definition 27 . Domain Projection: *By a* **domain projection** *we mean*

- *a subset of the domain description,*
- *one which leaves out all those*

  - *endurants:*
    - *parts,*
    - *materials and*
    - *components,*

    as well as
  - *perdurants:*
    - *functions,*
    - *events and*
    - *behaviours*

  that the stake-holders do not wish represented by the machine.
- *The resulting document is a* partial domain requirements prescription ■

- In determining an appropriate subset
  - the requirements engineer must secure
  - that the final prescription
  - is complete and consistent — that is,
    - that there are no “dangling references”,
    - i.e., that all entities that are referred to
    - are all properly defined.

7.2.1.1. Domain Projection — Narrative

- We now start on a series of examples
- that illustrate domain requirements development.

Example 69 . Domain Requirements. Projection A Narrative Sketch:

- We require that the Road-pricing IT, computing & communications system shall embody the following domain entities, in one form or another:
  - the net,
    - its links and hubs,
    - and their properties (unique identifiers, mereologies and attributes),
  - the vehicles, as endurants,
    - as endurants,
    - and the general vehicle behaviour, i.e., the vehicle signature.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

- To formalise this we copy the domain description, $\Delta_\triangle$,
- From that domain description we remove all mention of
  - the link insertion and removal functions,
  - the link disappearance event,
  - the vehicle behaviour, and
  - the monitor
- to obtain the $\Delta_{\mathcal{P}}$ version of the domain requirements prescription.[25]

[25] Restrictions of the net to the toll road nets, hinted at earlier, will follow in the next domain requirements steps.

7.2.1.2. 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 series of domain projection examples following below 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 – May 23, 2015: 15:29

Example 70 . Domain Requirements. Projection Root Sorts:

type

112. $\Delta_{\mathcal{P}}$

112a.. $\mathsf{N}_{\mathcal{P}}$

112b.. $\mathsf{F}_{\mathcal{P}}$

value

112a.. **obs_part**_$\mathsf{N}_{\mathcal{P}}$: $\Delta_{\mathcal{P}} \rightarrow \mathsf{N}_{\mathcal{P}}$

112b.. **obs_part**_$\mathsf{F}_{\mathcal{P}}$: $\Delta_{\mathcal{P}} \rightarrow \mathsf{F}_{\mathcal{P}}$

type

113a.. $\mathsf{HA}_{\mathcal{P}}$

113b.. $\mathsf{LA}_{\mathcal{P}}$

value

113a.. **obs_part**_HA: $\mathsf{N}_{\mathcal{P}} \rightarrow \mathsf{HA}$

113b.. **obs_part**_LA: $\mathsf{N}_{\mathcal{P}} \rightarrow \mathsf{LA}$

Example 71 . Domain Requirements. Projection Sub-domain Sorts and Types:

type
114. $\mathsf{H}_\mathcal{P}$, $\mathsf{HS}_\mathcal{P} = \mathsf{H}_\mathcal{P}$-**set**
115. $\mathsf{L}_\mathcal{P}$, $\mathsf{LS}_\mathcal{P} = \mathsf{L}_\mathcal{P}$-**set**
116. $\mathsf{V}_\mathcal{P}$, $\mathsf{VS}_\mathcal{P} = \mathsf{V}_\mathcal{P}$-**set**
value
114. **obs_part**_$\mathsf{HS}_\mathcal{P}$: $\mathsf{HA}_\mathcal{P} \rightarrow \mathsf{HS}_\mathcal{P}$
115. **obs_part**_$\mathsf{LS}_\mathcal{P}$: $\mathsf{LA}_\mathcal{P} \rightarrow \mathsf{LS}_\mathcal{P}$
116. **obs_part**_$\mathsf{VS}_\mathcal{P}$: $\mathsf{F}_\mathcal{P} \rightarrow \mathsf{VS}_\mathcal{P}$
117a.. links: $\Delta_\mathcal{P} \rightarrow$ L-**set**
117a.. links($\delta_\mathcal{P}$) $\equiv$ **obs_part**_$\mathsf{LS}_\mathcal{R}$(**obs_part**_$\mathsf{LA}_\mathcal{R}$($\delta_\mathcal{R}$))
117b.. hubs: $\Delta_\mathcal{P} \rightarrow$ H-**set**
117b.. 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 – May 23, 2015: 15:29

Example 72 . Domain Requirements. Projection Unique Identifications:

type

118a.. HI, LI, VI, MI

value

118c.. **uid**_HI: $H_{\mathcal{P}} \rightarrow$ HI

118c.. **uid**_LI: $L_{\mathcal{P}} \rightarrow$ LI

118c.. **uid**_VI: $V_{\mathcal{P}} \rightarrow$ VI

118c.. **uid**_MI: $M_{\mathcal{P}} \rightarrow$ MI

axiom

118b.. $HI \bigcap LI = \emptyset$, $HI \bigcap VI = \emptyset$, $HI \bigcap MI = \emptyset$,

118b.. $LI \bigcap VI = \emptyset$, $LI \bigcap MI = \emptyset$, $VI \bigcap MI = \emptyset$

Example 73 . Domain Requirements. Projection Road Net Mereology:

value

120. $\textbf{obs\_mereo}\_H_{\mathcal{P}}$: $H_{\mathcal{P}} \rightarrow$ LI-**set**

121. $\textbf{obs\_mereo}\_L_{\mathcal{P}}$: $L_{\mathcal{P}} \rightarrow$ HI-**set**

121. **axiom** $\forall$ $l{:}L_{\mathcal{P}}$ $\cdot$ **card** $\textbf{obs\_mereo}\_L_{\mathcal{P}}(l)=2$

122. $\textbf{obs\_mereo}\_V_{\mathcal{P}}$: $V_{\mathcal{P}} \rightarrow$ MI

123. $\textbf{obs\_mereo}\_M_{\mathcal{P}}$: $M_{\mathcal{P}} \rightarrow$ VI-**set**

axiom

124. $\forall\ \delta_{\mathcal{P}}{:}\Delta_{\mathcal{P}}$, hs:HS•hs=hubs($\delta$), ls:LS•ls=links($\delta_{\mathcal{P}}$) $\Rightarrow$

124. $\forall$ h:$H_{\mathcal{P}}$•h $\in$ hs $\Rightarrow$

124. $\textbf{obs\_mereo}\_H_{\mathcal{P}}(h) \subseteq$ xtr_his($\delta_{\mathcal{P}}$) $\wedge$

125. $\forall$ l:$L_{\mathcal{P}}$•l $\in$ ls $\cdot$

124. $\textbf{obs\_mereo}\_L_{\mathcal{P}}(l) \subseteq$ xtr_lis($\delta_{\mathcal{P}}$) $\wedge$

126a.. **let** f:$F_{\mathcal{P}}$•f=$\textbf{obs\_part}\_F_{\mathcal{P}}(\delta_{\mathcal{P}})$ $\Rightarrow$

126a.. vs:$VS_{\mathcal{P}}$•vs=$\textbf{obs\_part}\_VS_{\mathcal{P}}(f)$ **in**

126a.. $\forall$ v:$V_{\mathcal{P}}$•v $\in$ vs $\Rightarrow$

126a.. $\textbf{uid}\_V_{\mathcal{P}}(v) \in \textbf{obs\_mereo}\_M_{\mathcal{P}}(m)$ $\wedge$

126b.. $\textbf{obs\_mereo}\_M_{\mathcal{P}}(m)$

126b.. $= \{\textbf{uid}\_V_{\mathcal{P}}(v) | v{:}V{\cdot}v \in vs\}$

126b.. **end**

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

Example 74 . Domain Requirements. Projection Attributes of Hubs:

type
127a.. $\mathsf{H}\Sigma_{\mathcal{P}} = (\mathsf{LI}\times\mathsf{LI})$**-sett**
127b.. $\mathsf{H}\Omega_{\mathcal{P}} = \mathsf{H}\Sigma_{\mathcal{P}}$**-set**
value
127a.. **attr**_$\mathsf{H}\Sigma_{\mathcal{P}}$: $\mathsf{H}_{\mathcal{P}} \rightarrow \mathsf{H}\Sigma_{\mathcal{P}}$
127b.. **attr**_$\mathsf{H}\Omega_{\mathcal{P}}$: $\mathsf{H}_{\mathcal{P}} \rightarrow \mathsf{H}\Omega_{\mathcal{P}}$
type
129. HGCL
value
129. **attr**_HGCL: H $\rightarrow$ HGCL
axiom
128. $\forall\ \delta_{\mathcal{P}}{:}\Delta_{\mathcal{P}}$,
128. **let** hs = hubs($\delta_{\mathcal{P}}$) **in**
128. $\forall$ h:$\mathsf{H}_{\mathcal{P}}$ $\cdot$ h $\in$ hs $\cdot$
128a.. xtr_lis(h)$\subseteq$xtr_lis($\delta_{\mathcal{P}}$)
128b.. $\wedge$ **attr**_$\Sigma_{\mathcal{P}}$(h) $\in$ **attr**_$\Omega_{\mathcal{P}}$(h)
128. **end**

Example 75 . Domain Requirements. Projection Attributes of Links:

type
131. LEN
132. LGCL
133a.. $\mathsf{L}\Sigma_{\mathcal{P}} = (\mathsf{HI}\times\mathsf{HI})$-**set**
133b.. $\mathsf{L}\Omega_{\mathcal{P}} = \mathsf{L}\Sigma_{\mathcal{P}}$-**set**
value
131. **attr**_LEN: $\mathsf{L}_{\mathcal{P}} \rightarrow$ LEN
132. **attr**_LGCL: $\mathsf{L}\ _{\mathcal{P}}\rightarrow$ LGCL
133a.. **attr**_$\mathsf{L}\Sigma_{\mathcal{P}}$: $\mathsf{L}_{\mathcal{P}} \rightarrow \mathsf{L}\Sigma_{\mathcal{P}}$
133b.. **attr**_$\mathsf{L}\Omega_{\mathcal{P}}$: $\mathsf{L}_{\mathcal{P}} \rightarrow \mathsf{L}\Omega_{\mathcal{P}}$
axiom
133a..– 133b. on Slide 324.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

Example 76 . Domain Requirements. Projection Behaviour:

Global Values

value

146. $\delta_{\mathcal{P}}$:$\Delta_{\mathcal{P}}$,
147. n:$\mathsf{N}_{\mathcal{P}}$ = **obs_part**_$\mathsf{N}_{\mathcal{P}}(\delta_{\mathcal{P}})$,
147. ls:$\mathsf{L}_{\mathcal{P}}$-**set** = links($\delta_{\mathcal{P}}$),
147. hs:$\mathsf{H}_{\mathcal{P}}$-**set** = hubs($\delta_{\mathcal{P}}$),
147. lis:LI-**set** = xtr_lis($\delta_{\mathcal{P}}$),
147. his:HI-**set** = xtr_his($\delta_{\mathcal{P}}$)

Behaviour Signatures

value

153. trs$_{\mathcal{P}}$: **Unit** → **Unit**
154. veh$_{\mathcal{P}}$: VI×MI×ATTR → ... **Unit**

The System Behaviour

value

156a.. trs$_{\mathcal{P}}$()=‖{veh$_{\mathcal{P}}$(**uid**_VI(v),**obs_mereo**_V(v),**attr**_ATTRS(v)) | v:V$_{\mathcal{P}}$•v ∈ vs}

7.2.1.3. A Projection Operator

- Domain projection thus take a domain description, $\mathcal{D}$, and yields a projected requirements prescription, ,$\mathcal{R}_{\mathcal{P}}$.
- ⟐ **type** projection: $\mathcal{D} \rightarrow \mathcal{R}_{\mathcal{P}}$.
- Semantically
  - ⟐ $\mathcal{D}$ denotes a possibly infinite set of meanings, say $\mathbb{D}$ and
  - ⟐ $\mathcal{R}_{\mathcal{P}}$ denotes a possibly infinite set of meanings, say $\mathbb{R}_{\mathbb{P}}$,
  - ⟐ such that some relation $\mathbb{R}_{\mathbb{P}} \sqsubseteq \mathbb{D}$ is satisfied.

Dines Bjørner's MAP-i Lecture # 7

End of MAP-i Lecture # 7: Requirements – An Overview and Projection

Tuesday, 26 May 2015: 15:30–16:15

Dines Bjørner's MAP-i Lecture # 8

Domain Requirements: Instantiation and Determination

Tuesday, 26 May 2015: 16:45–17:30

7.2.2. Domain Instantiation

Definition 28 . 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* ■

- 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.
- Examples 77–78 express requirements that the road net on which the road-pricing system is to be based must satisfy.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

7.2.2.1. Domain Instantiation — Narrative

Example 77 . Domain Requirements. Instantiation Road Net, Narrative:

- 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. 3 on the facing slide,
  - and an ordinary road net, n'_{Δ}.
  - The two are connected as follows:
    - The toll-road net, $trn_{\mathcal{I}}$, borders some toll-road plazas, in Fig. 3 on the next slide shown by white filled circles (i.e., hubs).
    - These toll-road plaza hubs are proper hubs of the 'ordinary' road net, n'_{Δ}.

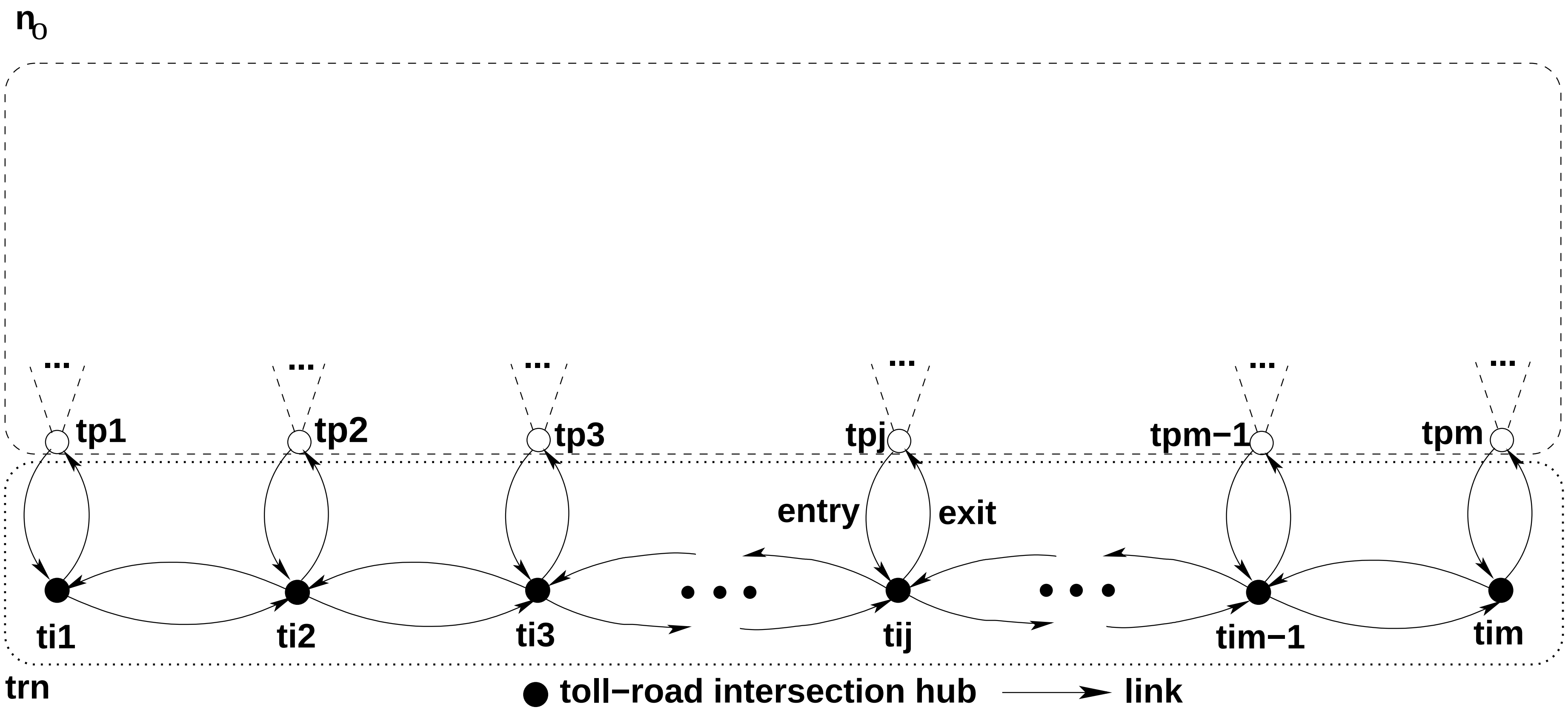

Figure 3: A simple, linear toll-road net

164 The instantiated domain, $\delta_{\mathcal{I}}$:$\Delta_{\mathcal{I}}$ has just the net, $\mathsf{n}_{\mathcal{I}}$:$\mathsf{N}_{\mathcal{I}}$ being instantiated.

165 The road net consists of two "sub-nets"

a. an "ordinary" road net, n'_{Δ}:N'_{Δ} and

b. a toll-road net proper, $\mathsf{trn}_{\mathcal{I}}$:$\mathsf{TRN}_{\mathcal{I}}$ —

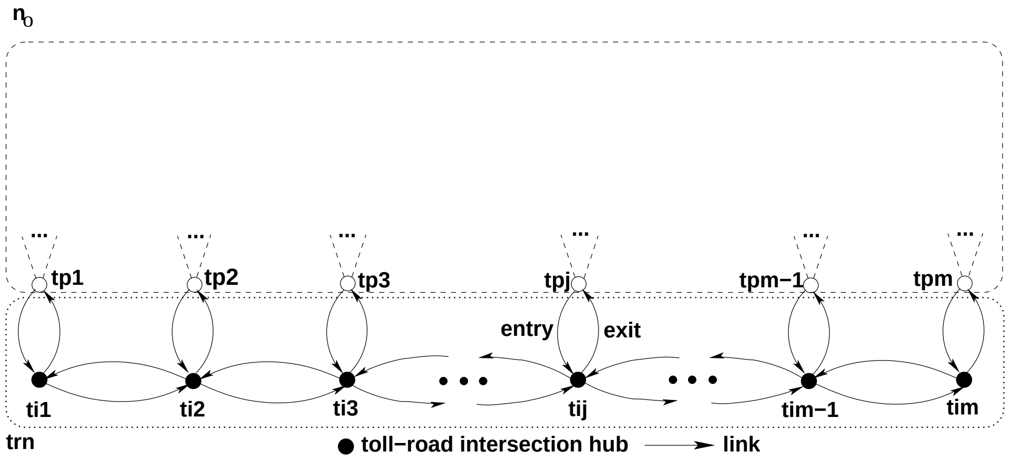

Figure 4: The Instantiated Road Net

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:$\mathsf{TRN}_{\mathcal{I}}$[26], 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, $\mathsf{n}'_\Delta{:}\mathsf{N}'_\Delta$, has each plaza, i.e., the hub designated by the hub identifier hil[i], connected to one or more ordinary net links, $\{l_{i_1}, l_{i_2}, \cdots, l_{i_\ell}\}$.

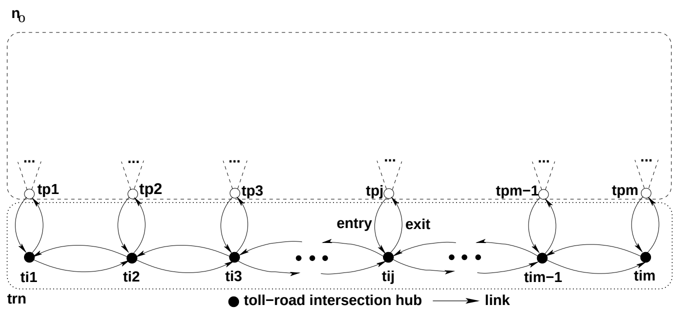

Figure 5: The Instantiated Road Net

[26]We (sometimes) omit the subscript $_{\mathcal{I}}$ when it should be clear from the context what we mean.

165b. The toll-road net, trn:$\text{TRN}_{\mathcal{I}}$, 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.

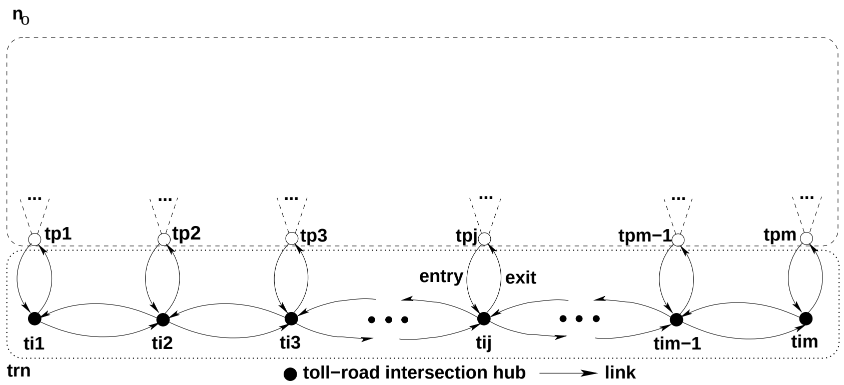

Figure 6: The Instantiated Road Net

7.2.2.2. Domain Instantiation — Formalisation

Example 78 . Domain Requirements. Instantiation Road Net, Formal Types:

type

164 $\Delta_{\mathcal{I}}$

165 $\mathsf{N}_{\mathcal{I}} = \mathsf{N}'_{\Delta} \times \mathsf{HIL} \times \mathsf{TRN}$

165a. N'_{Δ}

165b. $\mathsf{TRN}_{\mathcal{I}} = (\mathsf{L}\times\mathsf{L})^* \times \mathsf{H}^* \times (\mathsf{L}\times\mathsf{L})^*$

165c. $\mathsf{HIL} = \mathsf{HI}^*$

axiom

165d. $\forall\ \mathsf{n}_{\mathcal{I}}:\mathsf{N}_{\mathcal{I}}\ \bullet$

165d. **let** $(\mathsf{n}_{\Delta},\mathsf{hil},(\mathsf{exll},\mathsf{hl},\mathsf{lll})) = \mathsf{n}_{\mathcal{I}}$ **in**

165d. **len** hil = **len** exll = **len** hl = **len** lll + 1

165d. **end**

[Lecturer explains N'_{Δ}]

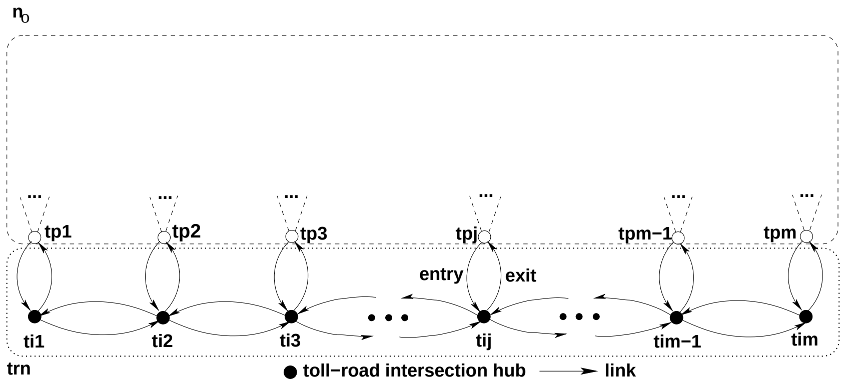

Figure 7: The Instantiated Road Net

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

7.2.2.3. Domain Instantiation — Formalisation: Well-formedness

Example 79 . Domain Requirements. Instantiation Road Net, Well-formedness:

- The partial concretisation of the net sorts, N, into $N_{\mathcal{R}_1}$ requires some well-formedness conditions to be satisfied.

166 The toll-road intersection hubs must all have distinct hub identifiers.

value
166. wf_dist_toll_road_isect_hub_ids: $H^* \rightarrow \mathbf{Bool}$
166. wf_dist_toll_road_isect_hub_ids(hl) $\equiv$
166. **len** hl = **card** xtr_his(hl)

167 The toll-road 'up' and 'down' links must all have distinct link identifiers.

value
167. wf_dist_toll_road_u_d_link_ids: $(L \times L)^* \rightarrow \mathbf{Bool}$
167. wf_dist_toll_road_u_d_link_ids(lll) $\equiv$
167. $2 \times$ **len** lll = **card** xtr_lis(lll)

168 The toll-road entry/exit links must all have distinct link identifiers.

value

168. $\mathsf{wf\_dist\_e\_x\_link\_ids}$: $(\mathsf{L}\times\mathsf{L})^*\rightarrow\mathbf{Bool}$
168. $\mathsf{wf\_dist\_e\_x\_link\_ids(exll)} \equiv$
168. $\quad 2 \times \mathbf{len}\ \mathsf{exll} = \mathbf{card}\ \mathsf{xtr\_lis(exll)}$

169 Proper net links must not designate toll-road intersection hubs.

value

169. $\mathsf{wf\_isoltd\_toll\_road\_isect\_hubs}$: $\mathsf{HI}^*\times\mathsf{H}^*\rightarrow\mathsf{N}_{\mathcal{I}}\rightarrow\mathbf{Bool}$
169. $\mathsf{wf\_isoltd\_toll\_road\_isect\_hubs(hil,hl)(n_{\mathcal{I}})} \equiv$
169. $\quad \mathbf{let}\ \mathsf{ls{=}xtr\_links(n_{\mathcal{I}})}\ \mathbf{in}$
169. $\quad \mathbf{let}\ \mathsf{his} = \cup\ \{\mathbf{obs\_mereo}\_\mathsf{L(l)}|\mathsf{l{:}L{\cdot}l} \in \mathsf{ls}\}\ \mathbf{in}$
169. $\quad \mathsf{his} \cap \mathsf{xtr\_his(hl)} = \{\}\ \mathbf{end}\ \mathbf{end}$

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

170 The plaza hub identifiers must designate hubs of the ‘ordinary’ net.

value

170. wf_p_hubs_pt_of_ord_net: $\mathsf{HI}^*\rightarrow\mathsf{N}'_\Delta\rightarrow\mathbf{Bool}$

170. wf_p_hubs_pt_of_ord_net(hil)($\mathsf{n'}_\Delta$) $\equiv$

170. **elems** hil $\subseteq$ xtr_his(n'_Δ)

171 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.

value

171. wf_p_hub_interf: $\mathsf{N}'_\Delta\rightarrow\mathbf{Bool}$

171. wf_p_hub_interf(n_o,hil,(exll,_,_)) $\equiv$

171. $\forall$ i:**Nat** $\cdot$ i $\in$ **inds** exll $\Rightarrow$

171. **let** h = get_H(hil(i))(n'_Δ) **in**

171. **let** lis = **obs_mereo**_H(h) **in**

171. **let** lis$'$ = lis $\setminus$ xtr_lis(n$'$) **in**

171. lis$'$ = xtr_lis(exll(i)) **end end end**

172 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.

value

172. wf_toll_road_isect_hub_iface: $\mathsf{N}_{\mathcal{I}} \rightarrow$ **Bool**
172. wf_toll_road_isect_hub_iface(_,_,(exll,hl,lll)) $\equiv$
172. $\forall$ i:**Nat** $\cdot$ i $\in$ **inds** hl $\Rightarrow$
172.   **obs_mereo**_H(hl(i)) =
172a..   xtr_lis(exll(i)) $\cup$
172.   **case** i **of**
172b..   1 $\rightarrow$ xtr_lis(lll(1)),
172b..   **len** hl $\rightarrow$ xtr_lis(lll(**len** hl$-$1))
172b..   _ $\rightarrow$ xtr_lis(lll(i)) $\cup$ xtr_lis(lll(i$-$1))
172.   **end**

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

173 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.

value

173. wf_exll: $(\mathsf{L}\times\mathsf{L})^*\times\mathsf{HI}^*\times\mathsf{H}^*\rightarrow\mathbf{Bool}$
173. wf_exll(exll,hil,hl) $\equiv$
173. $\forall$ i:**Nat** $\cdot$ i $\in$ **len** exll
173. **let** (hi,(el,xl),h) = (hil(i),exll(i),hl(i)) **in**
173. **obs_mereo**_L(el) = **obs_mereo**_L(xl)
173. = {hi} $\cup$ {**uid**_H(h)} **end**
173. **pre**: **len** eell = **len** hil = **len** hl

174 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.

value

174. $\mathsf{wf\_u\_d\_links}$: $(\mathsf{L}\times\mathsf{L})^*\times\mathsf{H}^*\rightarrow\mathbf{Bool}$

174. $\mathsf{wf\_u\_d\_links(lll,hl)} \equiv$

174. $\quad\forall\ \mathsf{i}{:}\mathbf{Nat}\cdot \mathsf{i} \in \mathbf{inds}\ \mathsf{lll} \Rightarrow$

174. $\qquad\mathbf{let}\ (\mathsf{ul,dl})\ = \mathsf{lll(i)}\ \mathbf{in}$

174. $\qquad\mathbf{obs\_mereo}\_\mathsf{L(ul)} = \mathbf{obs\_mereo}\_\mathsf{L(dl)} =$

174a.. $\qquad\mathbf{uid}\_\mathsf{H(hl(i))} \cup \mathbf{uid}\_\mathsf{H(hl(i+1))}\ \mathbf{end}$

174. $\quad\mathbf{pre}$: $\mathbf{len}\ \mathsf{lll} = \mathbf{len}\ \mathsf{hl}+1$

- We have used additional auxiliary functions:

value

$$
\begin{array}{l}
\mathsf{xtr\_his}\text{: } \mathsf{H}^* \to \mathsf{HI}\textbf{-set} \\
\mathsf{xtr\_his(hl)} \equiv \{\textbf{uid}\_\mathsf{HI(h)} | \mathsf{h{:}H} \bullet \mathsf{h} \in \textbf{elems}\ \mathsf{hl}\} \\
\mathsf{xtr\_lis}\text{: } (\mathsf{L} \times \mathsf{L}) \to \mathsf{LI}\textbf{-set} \\
\mathsf{xtr\_lis(l',l'')} \equiv \{\textbf{uid}\_\mathsf{LI(l')}\} \cup \{\textbf{uid}\_\mathsf{LI(l'')}\} \\
\mathsf{xtr\_lis}\text{: } (\mathsf{L} \times \mathsf{L})^* - \mathsf{LI}\textbf{-set} \\
\mathsf{xtr\_lis(lll)} \equiv \\
\ \cup\{\mathsf{xtr\_lis(l',l'')} | \mathsf{(l',l'')}{:}(\mathsf{L} \times \mathsf{L}) \bullet \mathsf{(l',l'')} \in \textbf{elems}\ \mathsf{lll}\}
\end{array}
$$

7.2.2.3.1 Summary Well-formedness Predicate

175 The well-formedness of instantiated nets is now the conjunction of the individual well-formedness predicates above.

value

175. wf_instantiated_net: $\mathsf{N}_{\mathcal{I}} \rightarrow \mathbf{Bool}$
175. wf_instantiated_net(n'_Δ,hil,(exll,hl,lll))
166. wf_dist_toll_road_isect_hub_ids(hl)
167. $\wedge$ wf_dist_toll_road_u_d_link_ids(lll)
168. $\wedge$ wf_dist_e_e_link_ids(exll)
169. $\wedge$ wf_isolated_toll_road_isect_hubs(hil,hl)(n')
170. $\wedge$ wf_p_hubs_pt_of_ord_net(hil)(n')
171. $\wedge$ wf_p_hub_interf(n'_Δ,hil,(exll,_,_))
172. $\wedge$ wf_toll_road_isect_hub_iface(_,_,(exll,hl,lll))
173. $\wedge$ wf_exll(exll,hil,hl)
174. $\wedge$ wf_u_d_links(lll,hl)

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

7.2.2.4. Domain Instantiation — Abstraction

Example 80 . Domain Requirements. Instantiation Road Net, Abstraction:

- Domain instantiation has refined
  - an abstract definition of net sorts, n_Δ:N_Δ,
  - into a partially concrete definition of nets, $n_\mathcal{I}$:$N_\mathcal{I}$.
- We need to show the refinement relation:
  - abstraction($n_\mathcal{I}$) $= n_\Delta$.

value

176 abstraction: $\mathsf{N}_{\mathcal{I}} \to \mathsf{N}_{\Delta}$

177 abstraction(n'_{Δ},hil,(exll,hl,lll)) $\equiv$

178 **let** n_{Δ}:N_{Δ} $\cdot$

178 **let** hs = **obs_part**_HS_{Δ}(**obs_part**_HA_{Δ}(n'_{Δ})),

178 ls = **obs_part**_LS_{Δ}(**obs_part**_LA_{Δ}(n'_{Δ})),

178 ths = **elems** hl,

178 eells = xtr_links(eell), llls = xtr_links(lll) **in**

179 hs$\cup$ths=**obs_part**_HS_{Δ}(**obs_part**_HA_{Δ}(n_{Δ}))

180 $\wedge$ ls$\cup$eells$\cup$llls=**obs_part**_LS_{Δ}(**obs_part**_LA_{Δ}(n_{Δ}))

181 n_{Δ} **end end**

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

176 The abstraction function takes a concrete net, $\mathsf{n}_{\mathcal{I}}$:$\mathsf{N}_{\mathcal{I}}$, and yields an abstract net, n_{Δ}:N_{Δ}.

177 The abstraction function doubly decomposes its argument into constituent lists and sub-lists.

178 There is postulated an abstract net, n_{Δ}:N_{Δ}, such that

179 the hubs of the concrete net and toll-road equals those of the abstract net, and

180 the links of the concrete net and toll-road equals those of the abstract net.

181 And that abstract net, n_{Δ}:N_{Δ}, is postulated to be an abstraction of the concrete net.

7.2.2.5. An Instantiation Operator

- Domain instantiation take a requirements prescription, $\mathcal{R}_{\mathcal{P}}$, and yields a more **concrete requirements prescription** $\mathcal{R}_{\mathcal{I}}$.
  - **type instantiation**: $\mathcal{R}_{\mathcal{P}} \rightarrow \mathcal{R}_{\mathcal{I}}$
- Semantically
  - $\mathcal{R}_{\mathcal{P}}$ denotes a possibly infinite set of meanings, say $\mathbb{R}_{\mathbb{P}}$,
  - $\mathcal{R}_{\mathcal{I}}$ denotes a possibly infinite set of meanings, say $\mathbb{R}_{\mathbb{I}}$ and
  - such that some relation $\mathbb{R}_{\mathbb{I}} \sqsubseteq \mathbb{R}_{\mathbb{P}}$ is satisfied.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

7.2.3. Domain Determination

Definition 29 . 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".

7.2.3.1. 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 – May 23, 2015: 15:29

Example 81 . Domain Requirements. Determination Toll-roads: 7.2.3

- We focus only on the toll-road net.
- We single out only two 'determinations':

182 *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.

value

182. opposite_traffics: $(\mathsf{L}\times\mathsf{L})^* \times (\mathsf{L}\times\mathsf{L})^* \to$ **Bool**
182. opposite_traffics(exll,lll) $\equiv$
182. $\forall$ (lt,lf):(L×L) · (lt,lf) $\in$ **elems** exll$\widehat{\ }$lll $\Rightarrow$
182a.. **let** (ltσ,lfσ) = (**attr**_LΣ(lt),**attr**_LΣ(lf)) **in**
182a.$'$. **attr**_LΩ(lt)={ltσ}$\wedge$**attr**_LΩ(ft)={ftσ}
182a.$''$. $\wedge$ **card** ltσ = 1 = **card** lfσ
182. $\wedge$ **let** ({(hi,hi$'$)},{(hi$''$,hi$'''$)}) = (ltσ,lfσ) **in**
182c.. hi=hi$'''$ $\wedge$ hi$'$=hi$''$
182. **end end**

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

7.2.3.1.2 All Toll-road Hubs are Free-flow

183 *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*.

value
183. free_flow_toll_road_hubs: $(L \times L)^* \times (L \times L)^* \rightarrow$ **Bool**
183. free_flow_toll_road_hubs(exl,ll) $\equiv$
183. $\forall$ i:**Nat**•i $\in$ **inds** hl $\Rightarrow$
183. **attr**_HΣ(hl(i)) $=$
183a.. hσ_ex_ls(exl(i))
183b.. $\cup$ hσ_et_ls(exl(i),(i,ll))
183c.. $\cup$ hσ_tx_ls(exl(i),(i,ll))
183d.. $\cup$ hσ_tt_ls(i,ll)

183a.: from *e*ntry links back to the paired e*x*it *l*ink*s*:

value

183a.. $\mathsf{h}\sigma\_\mathsf{ex\_ls}$: $(\mathsf{L}\times\mathsf{L})\rightarrow\mathsf{L}\Sigma$

183a.. $\mathsf{h}\sigma\_\mathsf{ex\_ls}(\mathsf{e},\mathsf{x}) \equiv \{(\mathbf{uid}\_\mathsf{LI}(\mathsf{e}),\mathbf{uid}\_\mathsf{LI}(\mathsf{x}))\}$

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

183b.: from *e*ntry links to emanating *t*oll-road *l*ink*s*:

value

183b.. $\mathsf{h\sigma\_et\_ls}$: $(\mathsf{L}\times\mathsf{L})\times(\mathbf{Nat}\times(\mathsf{em{:}L}\times\mathsf{in{:}L})^*)\rightarrow\mathsf{L\Sigma}$
183b.. $\mathsf{h\sigma\_et\_ls((e,\_),(i,ll))} \equiv$
183b.. **case** i **of**
183b.. 2 $\rightarrow$ {(**uid**_Ll(e),**uid**_Ll(em(ll(1))))},
183b.. **len** ll+1 $\rightarrow$ {(**uid**_Ll(e),**uid**_Ll(em(ll(**len** ll))))},
183b.. _ $\rightarrow$ {(**uid**_Ll(e),**uid**_Ll(em(ll(i−1)))),
183b.. (**uid**_Ll(e),**uid**_Ll(em(ll(i))))}
183b.. **end**

- The *em* and *in* in the toll-road link list $(\mathsf{em{:}L}\times\mathsf{in{:}L})^*$ designate selectors for *em*anating, respectively *in*cident links.

183c.: from incident $t$oll-road links to e$x$it $l$ink$s$:

value

```
183c..  hσ_tx_ls: (L×L)×(Nat×(em:L×in:L)*)→LΣ
183c..  hσ_tx_ls((__,x),(i,ll)) ≡
183c..      case i of
183c..          2        → {(uid_Ll(in(ll(1))),uid_Ll(x))},
183c..          len ll+1 → {(uid_Ll(in(ll(len ll))),uid_Ll(x))},
183c..          _        → {(uid_Ll(in(ll(i−1))),uid_Ll(x)),
183c..                      (uid_Ll(in(ll(i))),uid_Ll(x))}
183c..      end
```

183d.: from incident $t$oll-road link to emanating $t$oll-road $l$ink$s$:

```
value
183d..  hσ_tt_ls: Nat×(em:L×in:L)*→LΣ
183d..  hσ_tt_ls(i,ll) ≡
183d..      case i of
183d..          2        → {(uid_Ll(in(ll(1))),uid_Ll(em(ll(1))))},
183d..          len ll+1 → {(uid_Ll(in(ll(len ll))),uid_Ll(em(ll(len ll))))},
183d..          _        → {(uid_Ll(in(ll(i−1))),uid_Ll(em(ll(i−1)))),
183d..                      (uid_Ll(in(ll(i))),uid_Ll(em(ll(i))))}
183d..      end
```

7.2.3.2. A Domain Determination Operator

- Domain determination take a requirements description, $\mathcal{R}_\mathcal{I}$, and yields a more **deterministic requirements prescription**, $\mathcal{R}_\mathcal{D}$.
  - **type** instantiation: $\mathcal{R}_\mathcal{I} \rightarrow \mathcal{R}_\mathcal{D}$
- Semantically
  - $\mathcal{R}_\mathcal{I}$ denotes a possibly infinite set of meanings, say $\mathbb{R}_\mathbb{I}$,
  - $\mathcal{R}_\mathcal{D}$ denotes a possibly infinite set of meanings, say $\mathbb{R}_\mathbb{D}$ and
  - such that some relation $\mathbb{R}_\mathbb{I} \sqsubseteq \mathbb{R}_\mathbb{D}$ is satisfied.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

Dines Bjørner's MAP-i Lecture # 8

End of MAP-i Lecture # 8:
Domain Requirements: Instantiation and Determination

Tuesday, 26 May 2015: 16:45–17:30

Dines Bjørner's MAP-i Lecture # 9

Domain Requirements: Extension and Fitting

Thursday, 28 May 2015: 10:00–11:15

7.2.4. Domain Extension

Definition 30 . 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 requirement,*
- *that what is introduced becomes*[27] *part of the unfolding requirements prescription* ■

[27]become or becomes ?

7.2.4.1. The Core Requirements Example: Domain Extension

Example 82 . Domain Requirements. Extension Vehicles: Parts, Properties and Channels:

184 There is a domain, $\delta_{\mathcal{E}}$:$\Delta_{\mathcal{E}}$, which contains

185 a fleet, $\mathsf{f}_{\mathcal{E}}$:$\mathsf{F}_{\mathcal{E}}$,

186 of a set, $\mathsf{vs}_{\mathcal{E}}$:$\mathsf{VS}_{\mathcal{E}}$, of

187 extended vehicles, $\mathsf{v}_{\mathcal{E}}$:$\mathsf{V}_{\mathcal{E}}$ — their extension amounting to

a. a dynamic, active and biddable attribute[28], whose value, ti-gpos:TiGpos, at any time, reflects that vehicle's *time-stamped global positions*

b. The vehicle's GNSS receiver calculates its local position, lpos:LPOS, based on these signals.

c. Vehicles access these external attributes via the external attribute channel, attr_TiGPos_ch, cf. Item 100 on Slide 273.

d. The vehicle can, on its own volition, offer the timed local position, ti-lpos:TiLPos to the price calculator, $\mathsf{c}_{\mathcal{E}}$:$\mathsf{C}_{\mathcal{E}}$ along a vehicles-to-calculator channel, v_c_ch.

[28] See Sect. Slide 187.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

type
184. $\Delta_{\mathcal{E}}$
185. $\mathsf{F}_{\mathcal{E}}$
186. $\mathsf{VS}_{\mathcal{E}} = \mathsf{V}_{\mathcal{E}}$**-set**
187. $\mathsf{V}_{\mathcal{E}}$
187a.. TiGPos = $\mathbb{T} \times$ GPOS
187a.. TiLPos = $\mathbb{T} \times$ LPOS
187b.. GPOS, LPOS
value
185. **obs_part**_$\mathsf{F}_{\mathcal{E}}$: $\Delta_{\mathcal{E}} \rightarrow \mathsf{F}_{\mathcal{E}}$
186. **obs_part**_$\mathsf{VS}_{\mathcal{E}}$: $\mathsf{F}_{\mathcal{E}} \rightarrow \mathsf{VS}_{\mathcal{E}}$
186. vs:**obs_part**_$\mathsf{VS}_{\mathcal{E}}(\mathsf{F}_{\mathcal{E}})$
channel
187c.. {attr_TiGPos_ch[vi]|viLVI•vi ∈ xtr_VIs(vs)}: TiGPos
187d.. {v_c_ch[vi,ci]
187d.. | vi:VI,ci:CI•vi∈vis∧ci=**uid**_C(c)}:(VI×TiLPos)
value
187a.. attr_TiGPos_ch[vi]?
187b.. loc_pos: GPOS → LPOS

- where vis:VI-**set** is the set unique vehicle identifiers of all vehicles of the requirements domain fleet, f:$\mathsf{F}_{\mathcal{R}_{\mathcal{E}}}$.

We define two auxiliary functions,

188 **xtr_vs**, which given a domain, or a fleet, extracts its set of vehicles, and

189 **xtr_vis** which given a set of vehicles generates their unique identifiers.

value

188. xtr_vs: $(\Delta_{\mathcal{E}}|\mathsf{F}_{\mathcal{E}}|\mathsf{VS}_{\mathcal{E}}) \rightarrow \mathsf{V}_{\mathcal{E}}$**-set**
188. xtr_vs(arg) $\equiv$
188.   **is**_$\Delta_{\mathcal{E}}$(arg) $\rightarrow$ **obs_part**_$\mathsf{VS}_{\mathcal{E}}$(**obs_part**_$\mathsf{F}_{\mathcal{E}}$(arg)),
188.   **is**_$\mathsf{F}_{\mathcal{E}}$(arg) $\rightarrow$ **obs_part**_$\mathsf{VS}_{\mathcal{E}}$(arg),
188.   **is**_$\mathsf{VS}_{\mathcal{E}}$(arg) $\rightarrow$ arg
189. xtr_vis: $(\Delta_{\mathcal{E}}|\mathsf{F}_{\mathcal{E}}|\mathsf{VS}_{\mathcal{E}}) \rightarrow$ VI**-set**
189. xtr_vis(arg) $\equiv$ {**uid**_VI(v)|v $\in$ xtr_vs(arg)}

Example 83 . Domain Requirements. Extension Toll-road Net: Parts, Properties and Channels:

- We extend the domain with toll-gates for vehicles entering and exiting the toll-road entry and exit links.
- Figure 8 illustrates the idea of gates.

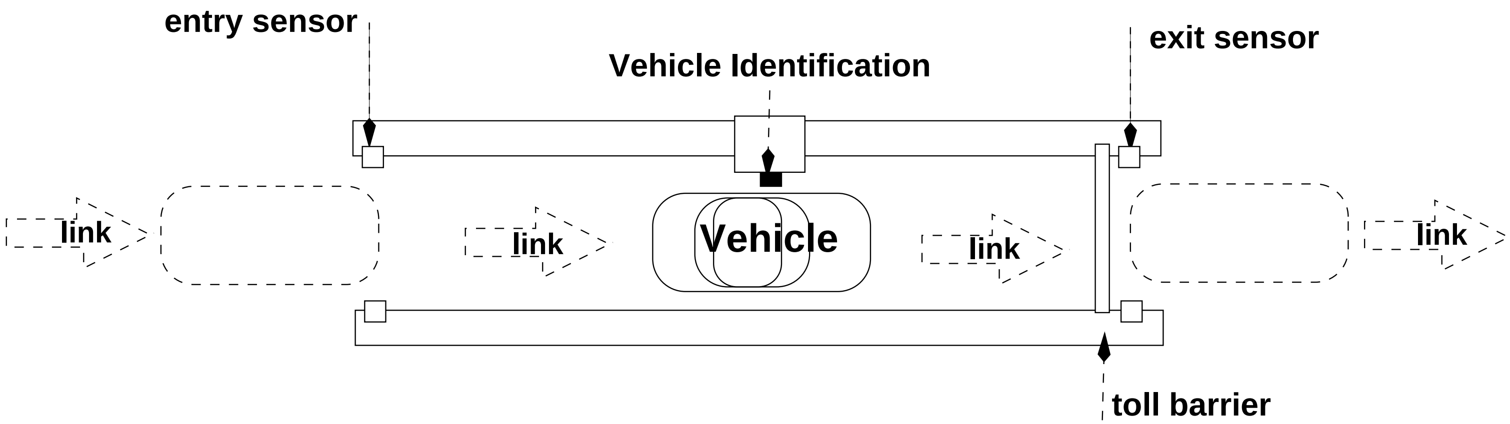

Figure 8: A toll plaza gate

- Figure 8 on the facing 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 identification 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.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

190 There is the domain, δ:$\triangle_{\mathcal{E}}$,

191 which contains the extended net, n:$\mathsf{N}_{\mathcal{E}}$, with the net extension amounting to the toll-road net, $\mathsf{TRN}_{\mathcal{E}}$,

192 that is, the instantiated toll-road net, trn:$\mathsf{TRN}_{\mathcal{I}}$, is extended, into trn:$\mathsf{TRN}_{\mathcal{E}}$, with entry, eg:EG, and exit, xg:XG, toll-gates.

From entry- and exit-gates we can observe

a. their unique identifier and their mereology: being paired with the entry-, respectively exit link and the calculator (by their unique identifiers); further

b. a pair of gate enter and leave sensors modeled as external attribute channels, (ges:ES,gls:XS), and

c. a time-stamped vehicle identity sensor modeled as external attribute channels.

type
190 $\Delta_{\mathcal{E}}$
191 $\mathsf{N}_{\mathcal{E}}$
192 $\mathsf{TRN}_{\mathcal{E}} = (\mathsf{EG}{\times}\mathsf{XG})^* \times \mathsf{TRN}_{\mathcal{I}}$
192a. GI
value
190 **obs_part**_$\mathsf{N}_{\mathcal{E}}$: $\Delta_{\mathcal{E}} \rightarrow \mathsf{N}_{\mathcal{E}}$
191 **obs_part**_$\mathsf{TRN}_{\mathcal{E}}$: $\mathsf{N}_{\mathcal{E}} \rightarrow \mathsf{TRN}_{\mathcal{E}}$
192a. **uid**_G: (EG|XG) → GI
192a. **obs_mereo**_G: (EG|XG) → (LI×CI)
channel
192b. {attr_enter_ch[gi]|gi:GI•...} "enter"
192b. {attr_leave_ch[gi]|gi:GI•...} "leave"
192c. {attr_passing_ch[gi]|gi:GI•...} TIVI
type
192c. TIVI = $\mathbb{T} \times$ VI

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

We define some auxiliary functions over toll-road nets, trn:$\mathsf{TRN}_{\mathcal{E}}$:

193 xtr_eGℓ extracts the ℓist of entry gates,

194 xtr_xGℓ extracts the ℓist of exit gates,

195 xtr_eGIds extracts the $s$et of entry gate identifiers,

196 xtr_xGIds extracts the $s$et of exit gate identifiers,

197 xtr_Gs extracts the $s$et of all gates, and

198 xtr_GIds extracts the $s$et of all gate identifiers.

value

193 $\mathsf{xtr\_eG}\ell$: $\mathsf{TRN}_{\mathcal{E}} \to \mathsf{EG}^*$

193 $\mathsf{xtr\_eG}\ell(\mathsf{pgl},\_) \equiv$

193 $\quad \{\mathsf{eg}|(\mathsf{eg},\mathsf{xg}):(\mathsf{EG},\mathsf{XG})\cdot(\mathsf{eg},\mathsf{xg}) \in \textbf{elems}\ \mathsf{pgl}\}$

194 $\mathsf{xtr\_xG}\ell$: $\mathsf{TRN}_{\mathcal{E}} \to \mathsf{XG}^*$

194 $\mathsf{xtr\_xG}\ell(\mathsf{pgl},\_) \equiv$

194 $\quad \{\mathsf{xg}|(\mathsf{eg},\mathsf{xg}):(\mathsf{EG},\mathsf{XG})\cdot(\mathsf{eg},\mathsf{xg}) \in \textbf{elems}\ \mathsf{pgl}\}$

195 $\mathsf{xtr\_eGIds}$: $\mathsf{TRN}_{\mathcal{E}} \to \mathsf{GI}\textbf{-set}$

195 $\mathsf{xtr\_eGIds}(\mathsf{pgl},\_) \equiv$

195 $\quad \{\textbf{uid}\_\mathsf{GI}(\mathsf{g})|\mathsf{g}:\mathsf{EG}\cdot\mathsf{g} \in \mathsf{xtr\_eGs}(\mathsf{pgl},\_)\}$

196 $\mathsf{xtr\_xGIds}$: $\mathsf{TRN}_{\mathcal{E}} \to \mathsf{GI}\textbf{-set}$

196 $\mathsf{xtr\_xGIds}(\mathsf{pgl},\_) \equiv$

196 $\quad \{\textbf{uid}\_\mathsf{GI}(\mathsf{g})|\mathsf{g}:\mathsf{EG}\cdot\mathsf{g} \in \mathsf{xtr\_xGs}(\mathsf{pgl},\_)\}$

197 $\mathsf{xtr\_Gs}$: $\mathsf{TRN}_{\mathcal{E}} \to \mathsf{G}\textbf{-set}$

197 $\mathsf{xtr\_Gs}(\mathsf{pgl},\_) \equiv$

197 $\quad \mathsf{xtr\_eGs}(\mathsf{pgl},\_) \cup \mathsf{xtr\_xGs}(\mathsf{pgl},\_)$

198 $\mathsf{xtr\_GIds}$: $\mathsf{TRN}_{\mathcal{E}} \to \mathsf{GI}\textbf{-set}$

198 $\mathsf{xtr\_GIds}(\mathsf{pgl},\_) \equiv$

198 $\quad \mathsf{xtr\_eGIds}(\mathsf{pgl},\_) \cup \mathsf{xtr\_xGIds}(\mathsf{pgl},\_)$

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

199 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

200 the unique identifier, ci:CI, of the road pricing calculator, c:C, and

201 the unique identifiers, vis:**VI-set**, of the fleet vehicles.

value
200 ci:CI
201 vis:VI-**set**
axiom
199 ∀ n:$N_{\mathcal{R}_3}$, trn:$TRN_{\mathcal{R}_3}$ ·
199 **let** (exgl,(exl,hl,lll)) = **obs_part**_$TRN_{\mathcal{R}_3}$(n) **in**
199a. **len** exgl = **len** exl = **len** hl = **len** lll + 1
199b. ∧ **card** xtr_GIds(exgl) = 2 ∗ **len** exgl
199c. ∧ ∀ i:**Nat**·i ∈ **inds** exgl·
199c. **let** ((eg,xg),(el,xl)) = (exgl(i),exl(i)) **in**
199c. **obs_mereo**_G(eg) = (**uid**_U(el),ci,vis)
199c. ∧ **obs_mereo**_G(xg) = (**uid**_U(xl),ci,vis) **end end**

Example 84 . Domain Requirements. Extension Parts, Properties and Channels:

202 The road pricing calculator repeatedly receives

a. information, (vi,(τ,pos)):VITIPOS,

b. sent by vehicles as to their identify and time-stamped position

c. over a channel, v_c_ch indexed by the c:$\mathsf{C}_{\mathcal{E}}$ and the vehicle identities.

203 The road pricing calculator has a number of attributes:

a. a 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,vp), and

b. a (total) road location function, vplf:VPLF.

i The $v$ehicle $p$osition $l$ocation $f$unction, vplf:VPLF, is subject to another function, locate_VPos, which, given a local position, lpos:LPos, yields the vehicle position designated by the `GNSS`-provided position, or yields the response that the provided position is off the toll-road net.

ii This result is used by the road-pricing calculator to conditionally

A either update the traffic map, trm:TRM, recording also the relevant time,

B or reset that vehicle's traffic recording while send a bill for the just completed journey.

type

202a. $\mathsf{VITIPos} = \mathsf{VI} \times (\mathbb{T} \times \mathsf{LPos})$

value

202a. ... v_c_ch[ci,vi] ? ...

202b. ... v_c_ch[ci,vi] ! (vi,(τ,p)) ...

channel

202c. $\{\mathsf{v\_c\_ch}[\,\mathsf{ci,vi}\,] | \mathsf{vi{:}VI}\cdot\mathsf{vi} \in \mathsf{vis}\}\mathsf{:VITIPos}$

type

203a. $\mathsf{TRM} = \mathsf{VI} \xrightarrow[m]{} (\mathbb{T} \times \mathsf{VPos})^*$

203b. $\mathsf{VPLF} = \mathsf{LPos} \to \mathsf{VPos}$ | ″off_TRN″

value

203(b.)i locate_LH: $\mathsf{LPos}\times\mathsf{RLF} \to$ (VPos|″off_TRN″)

203(b.)iiA update_TRM: $\mathsf{VI}\times(\mathbb{T}\times\mathsf{VPos})\to\mathsf{TRM}\to\mathsf{TRM}$

203(b.)iiB reset_TRM: $\mathsf{VI}\to\mathsf{TRM}\to\mathsf{TRM}$

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

Example 85 . Domain Requirements. Extension Main Sorts:

204 The main sorts of the road-pricing domain, $\Delta_{\mathcal{E}}$, are

a. the net, projected, instantiated (to include the specific toll-road net), made more determinate and now extended, $\mathsf{N}_{\mathcal{E}}$, with toll-gates;

b. the fleet, $\mathsf{F}_{\mathcal{E}}$,

c. of sets, VS, of extended vehicles, $\mathsf{V}_{\mathcal{E}}$;

d. the extended toll-road net, $\mathsf{TRN}_{\mathcal{E}}$, extending the instantiated toll-road net, $\mathsf{TRN}_{\mathcal{I}}$, with toll-gates; and

e. the road pricing calculator, $\mathsf{C}_{\mathcal{E}}$.

type

204. $\Delta_{\mathcal{E}}$

204a.. $\mathsf{N}_{\mathcal{E}}$

204b.. $\mathsf{F}_{\mathcal{E}}$

204c.. $\mathsf{VS}_{\mathcal{E}} = \mathsf{V}_{\mathcal{E}}$**-set**

204d.. $\mathsf{TRN}_{\mathcal{E}} = (\mathsf{EG}\times\mathsf{XG})^* \times \mathsf{TRN}_{\mathcal{I}}$

204e.. $\mathsf{C}_{\mathcal{E}}$

value

204a.. **obs_part**_$\mathsf{N}_{\mathcal{E}}$: $\Delta \rightarrow \mathsf{N}_{\mathcal{E}}$

204b.. **obs_part**_$\mathsf{F}_{\mathcal{E}}$: $\Delta \rightarrow \mathsf{F}_{\mathcal{E}}$

204c.. **obs_part**_$\mathsf{VS}_{\mathcal{E}}$: $\Delta \rightarrow \mathsf{VS}_{\mathcal{E}}$

204d.. **obs_part**_$\mathsf{TRN}_{\mathcal{E}}$: $\mathsf{N}_{\mathcal{E}} \rightarrow \mathsf{TRN}_{\mathcal{E}}$

204e.. **obs_part**_$\mathsf{C}_{\mathcal{E}}$: $\Delta \rightarrow \mathsf{C}_{\mathcal{E}}$

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

Example 86 . Domain Requirements. Extension Global Values:

- We exemplify a road-pricing system behaviour, in Example 87 on Slide 442,
- based on the following global values.

205 There is a given domain, $\delta_{\mathcal{E}}$:$\Delta_{\mathcal{E}}$;

206 there is the net, $\mathsf{n}_{\mathcal{E}}$:$\mathsf{N}_{\mathcal{E}}$, of that domain;

207 there is toll-road net, $\mathsf{trn}_{\mathcal{E}}$:$\mathsf{TRN}_{\mathcal{E}}$, of that net;

208 there is a set, $\mathsf{egs}_{\mathcal{E}}$:$\mathsf{EG}_{\mathcal{E}}$-**set**, of entry gates;

209 there is a set, $\mathsf{xgs}_{\mathcal{E}}$:$\mathsf{XG}_{\mathcal{E}}$-**set**, of exit gates;

210 there is a set, $\mathsf{gis}_{\mathcal{E}}$:$\mathsf{GI}_{\mathcal{E}}$-**set**, ofgate identifiers;

211 there is a set, $\mathsf{vs}_{\mathcal{E}}$:$\mathsf{V}_{\mathcal{E}}$-**set**, of vehicles;

212 there is a set, $\mathsf{vis}_{\mathcal{E}}$:$\mathsf{VI}_{\mathcal{E}}$-**set**, of vehicle identifiers;

213 there is the road-pricing calculator, $\mathsf{c}_{\mathcal{E}}$:$\mathsf{C}_{\mathcal{E}}$ and

214 there is its unique identifier, $\mathsf{ci}_{\mathcal{E}}$:CI.

value

205. $\delta_{\mathcal{E}}$:$\Delta_{\mathcal{E}}$
206. $\mathsf{n}_{\mathcal{E}}$:$\mathsf{N}_{\mathcal{E}} = \textbf{obs\_part\_}\mathsf{N}_{\mathcal{E}}(\delta_{\mathcal{E}})$
207. $\mathsf{trn}_{\mathcal{E}}$:$\mathsf{TRN}_{\mathcal{E}} = \textbf{obs\_part\_}\mathsf{TRN}_{\mathcal{E}}(\mathsf{n}_{\mathcal{E}})$
208. $\mathsf{egs}_{\mathcal{E}}$:$\mathsf{EG}\textbf{-set} = \mathsf{xtr\_egs}(\mathsf{trn}_{\mathcal{E}})$
209. $\mathsf{xgs}_{\mathcal{E}}$:$\mathsf{XG}\textbf{-set} = \mathsf{xtr\_xgs}(\mathsf{trn}_{\mathcal{E}})$
210. $\mathsf{gis}_{\mathcal{E}}$:$\mathsf{XG}\textbf{-set} = \mathsf{xtr\_gis}(\mathsf{trn}_{\mathcal{E}})$
211. $\mathsf{vs}_{\mathcal{E}}$:$\mathsf{V}_{\mathcal{E}}\textbf{-set} = \textbf{obs\_part\_}\mathsf{VS}(\textbf{obs\_part\_}\mathsf{F}_{\mathcal{E}}(\delta_{\mathcal{E}}))$
212. $\mathsf{vis}_{\mathcal{E}}$:$\mathsf{VI}\textbf{-set} = \{\textbf{uid\_}\mathsf{VI}(\mathsf{v}_{\mathcal{E}})|\mathsf{v}_{\mathcal{E}}:\mathsf{V}_{\mathcal{E}}\cdot\mathsf{v}_{\mathcal{E}} \in \mathsf{vs}_{\mathcal{E}}\}$
213. $\mathsf{c}_{\mathcal{E}}$:$\mathsf{C}_{\mathcal{E}} = \textbf{obs\_part\_}\mathsf{C}_{\mathcal{E}}(\delta_{\mathcal{E}})$
214. $\mathsf{ci}_{\mathcal{E}}$:$\mathsf{CI}_{\mathcal{E}} = \textbf{uid\_}\mathsf{CI}(\mathsf{c}_{\mathcal{E}})$

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

Example 87 . Domain Requirements. Extension System Behaviour:

- We shall model the behaviour of the road-pricing system as follows:
  - we shall only model behaviours related to atomic parts;
  - we shall not model behaviours of hubs and links;
  - thus we shall model only
    - the set of behaviours of vehicles, veh,
    - the set of behaviours of toll-gates, gate, and
    - the behaviour of the road-pricing calculator, calc.

215 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,

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

value

215. sys: **Unit** $\rightarrow$ **Unit**
215. sys() $\equiv$
215a.. $\parallel$ {veh(**uid**_V(v),(ci,gis),$\mathbb{U}$TiGPos)|v:V•v $\in$ vs$_{\mathcal{E}}$}
215b.. $\parallel$ $\parallel$ {gate(″`Entry`″)(**uid**_EG(eg),**obs_mereo**_G(eg),($\mathbb{U}$enter,$\mathbb{U}$passing,$\mathbb{U}$leave))|eg:EG
215c.. $\parallel$ $\parallel$ {gate(″`Exit`″)(**uid**_EG(xg),**obs_mereo**_G(xg),($\mathbb{U}$enter,$\mathbb{U}$passing,$\mathbb{U}$leave))|xg:XG•
215d.. $\parallel$ calc(ci$_{\mathcal{E}}$,(vis$_{\mathcal{E}}$,gis$_{\mathcal{E}}$))(rlf)(trm)

Example 88 . Domain Requirements. Extension Vehicle Behaviour:

216 Instead of moving around by explicitly expressed internal non-determinism[29] vehicles move around by unstated internal non-determinism and instead receive their current position from the global positioning subsystem.

a. At each moment the vehicle receives its time-stamped local position, tilpos:TiLPos,

b. which it then proceeds to communicate, with its vehicle identification, (vi,tilpos), to the road pricing subsystem —

c. whereupon it resumes its vehicle behaviour.

[29] We refer to Items 157b., 157c. on Slide 343 and 158b., 158(c.)ii, 159 on Slide 345

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

value

216. veh: vi:VI×(ci:CI×gis:GI-**set**)×$\mathbb{U}$TiGPos →
216. **out** v_c_ch[ci,vi] **Unit**
216. veh(vi,(ci,gis),attr_TiGPos_ch[vi]) ≡
216a.. **let** (τ,gpos) = attr_TiGPos_ch[vi]? **in**
216a.. **let** lpos = loc_pos(gpos) **in**
216b.. v_c_ch[ci,vi] ! (vi,(τ,lpos)) ;
216c.. veh(vi,(ci,gis),attr_TiGPos_ch[vi]) **end end**
216. **pre** vi ∈ vis$_{\mathcal{E}}$ ∧ ci = ci$_{\mathcal{E}}$ ∧ gis = gis$_{\mathcal{E}}$

Example 89 . Domain Requirements. Extension Gate Behaviour:

- The entry and the exit gates have "vehicle enter", "vehicle leave" and "vehicle time and identification" sensors.
  - The following assumption can now be made:
    - during the time interval between
    - a gate's vehicle "enter" sensor having first sensed a vehicle entering that gate
    - and that gate's "leave" sensor having last sensed that vehicle leaving that gate
    - that gate's "vehicle time and identification" sensor registers the time when the vehicle is entering the gate and that vehicle's unique identification.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

- We sketch the toll-gate behaviour:

217 We parameterise the toll-gate behaviour as either an entry or an exit gate.

218 Toll-gates

  a. inform the calculator of place (i.e., link) and time of entering and exiting of identified vehicles

  b. over an appropriate array of channels.

219 Toll-gates operate autonomously and cyclically.

  a. The **attr**_Enter event "triggers" the behaviour specified in formula line Item 219b.–219d..

  b. The time-of-entry and the identity of the entering (or exiting) vehicle is sensed via external attribute channel inputs.

  c. Then the road pricing calculator is informed of time-of-entry and of vehicle vi entering (or exiting) link li.

  d. And finally, after that vehicle has left the entry or exit gate that toll-gate's behaviour is resumed.

- The toll-gate behaviour, gate:

type
217 EE = "Enter" | "Exit"
218a. GCM = EE × (𝕋 × VI × LI)
channel
218b. {g_c_ch[**uid**_GI(g),ci]|g:G,ci:CI•g ∈ gates(trn)} GCM
value
219 gate: ee:EE×gi:GI×(ci:CI×VI-**set**×LI)×(𝕌enter×𝕌passing×𝕌leave) → **out** g_c_ch[gi
219 gate(ee,gi,(ci,vis,li),ea:(attr_enter_ch[gi],attr_passing_ch[gi],attr_leave_ch[gi])) ≡
219a. attr_enter_ch[gi] ? ;
219b. **let** (τ,vi) = attr_passing_ch[gi] ? **in assert** vi ∈ vis
219c. g_c_ch[gi,ci] ! (ee,(τ,(vi,li)));
219d. attr_leave_ch[gi] ?
219d. gate(ee)(gi,(ci,vis,li),ea)
219 **end**
219 **pre** ci = $ci_{\mathcal{E}}$ ∧ vis = $vis_{\mathcal{E}}$ ∧ li ∈ $lis_{\mathcal{E}}$

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

Example 90 . Domain Requirements. Extension Calculator Behaviour:

220 The road-pricing calculator alternates between (offering to accept communication with)

a. either any vehicle

b. or any toll-gate.

220. calc: ci:CI×(vis:VI-**set**×gis:GI-**set**)→RLF→TRM→
220a.. **in** {v_c_ch[ci,vi]|vi:VI•vi ∈ vis},
220b.. {g_c_ch[ci,gi]|gi:GI•gi ∈ gis} **Unit**
220. calc(ci,(vis,gis))(rlf)(trm) ≡
220a.. react_to_vehicles(ci,(vis,gis))(rlf)(trm)
220. ⌊⌋⌈⌉
220b.. react_to_gates(ci,(vis,gis))(rlf)(trm)
220. **pre** ci = $ci_{\mathcal{E}}$ ∧ vis = $vis_{\mathcal{E}}$ ∧ gis = $gis_{\mathcal{E}}$

221 If the communication is from a vehicle inside the toll-road net

a. then its toll-road net position, vp, is found from the road location function, rlf,

b. and the calculator resumes its work with the traffic map, trm, suitable updated,

c. otherwise the calculator resumes its work with no changes.

```
220a..  react_to_vehicles(ci,(vis,gis))(rlf)(trm) ≡
220a..    let (vi,(τ,lpos)) =
220a..       ⌈⌉⌈{v_c_ch[ ci,vi ]|vi:VI•vi∈ vis} in
221.      if vi ∈ dom trm
221a..       then let vp = rlf(lpos) in
221b..            calc(ci,(vis,gis))(rlf)(trm†[ vi↦trm⌢⟨(τ,vp)⟩ ]) end
221c..       else calc(ci,(vis,gis))(rlf)(trm) end end
```

222 If the communication is from a gate,

a. then that gate is either an entry gate or an exit gate;
b. if it is an entry gate
c. then the calculator resumes its work with the vehicle (that passed the entry gate) now recorded, afresh, in the traffic map, **trm**.
d. Else it is an exit gate and
e. the calculator concludes that the vehicle has ended its to-be-paid for journey inside the toll-road net, and hence to be billed;
f. then the calculator resumes its work with the vehicle (that passed the exit gate) now removed from the traffic map, **trm**.

220b.. react_to_gates(ci,(vis,gis))(rlf)(trm) $\equiv$
220b.. **let** (ee,(τ,(vi,li))) =
220b.. $\lceil\rceil\lceil\rceil$\{g_c_ch[ci,gi]|gi:GI•gi$\in$ gis\} **in**
222a.. **case** ee **of**
222b.. "Enter" $\rightarrow$
222c.. calc(ci,(vis,gis))(rlf)(trm$\cup$[vi$\mapsto\langle$(τ,(li,0))$\rangle$]),
222d.. "Exit" $\rightarrow$
222e.. billing(vi,trm(vi)$^\frown\langle$(τ,(li,1))$\rangle$);
222f.. calc(ci,(vis,gis))(rlf)(trm$\backslash$\{vi\}) **end end**

• • •

- We have made relevant external attributes explicit parameters of their (corresponding part) processes.
- We refer to Sect. 1.3.7.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

7.2.4.2. A Domain Extension Operator

- Domain extension takes a (more-or-less) deterministic requirements description, $\mathcal{R}_\mathcal{D}$, and yields an **extended requirements prescription**, $\mathcal{R}_\mathcal{E}$, which extends the domain description, $\mathcal{D}$, and, "at the same time", "extends" the requirements prescription, $\mathcal{R}_\mathcal{D}$,
  - **type extension**: $\mathcal{R}_\mathcal{D} \to \mathcal{R}_\mathcal{E}$
- Semantically
  - $\mathcal{R}_\mathcal{D}$ denotes a possibly infinite set of meanings, say $\mathbb{R}_\mathbb{D}$, and
  - $\mathcal{R}_\mathcal{E}$ denotes a possibly infinite set of meanings, say $\mathbb{R}_\mathbb{E}$,
  - but now the relation $\mathbb{R}_\mathcal{E} \sqsubseteq \mathbb{R}_\mathcal{D}$ is not necessarily satisfied —
  - but instead some **conservative extension** relation $\mathbb{R}_\mathbb{E} \sqsupseteq \mathbb{D}_\mathbb{D}$ is satisfied.

7.2.5. 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 – May 23, 2015: 15:29

- 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.

7.2.5.1. Some Definitions

- We thus assume
  - that there are n domain requirements developments, d_{r_1}, d_{r_2}, ..., 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 – May 23, 2015: 15:29

Definition 31 . 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***', $p_{d_{r_1}}$, $p_{d_{r_2}}$, ..., $p_{d_{r_n}}$, and*
    - *m* **shared domain requirements***, $s_{d_{r_1}}$, $s_{d_{r_2}}$, ..., $s_{d_{r_m}}$,*
    - *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 32 . 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.*

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

- *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 is 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 – May 23, 2015: 15:29

- *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 $s$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 ■

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

7.2.5.2. Requirements Fitting Procedure — A Sketch

- Requirements fitting consists primarily of a pragmatically determined sequence of analytic and synthetic ('fitting') steps.
  - It is first decided which n domain requirements documents to fit.
  - Then a 'manual' analysis is made of the selected, n domain requirements.
  - During this analysis tentative shared domain requirements are identified.
  - It is then decided which m shared domain requirements to single out.
  - This decision results in a tentative construction of n partial domain requirements.
  - An analysis is made of the tentative partial and shared domain requirements.
  - A decision is then made
    - whether to accept the resulting documents
    - or to iterate the steps above.

7.2.5.3. Requirements Fitting – An Example

Example 91 . Domain Requirements. Fitting A Sketch:

- We postulate two domain requirements:
  - We have outlined a domain requirements development for software support for a road-pricing system.
  - We have earlier hinted at domain operations related to insertion of new and removal of existing links and hubs.
- We can therefore postulate that there are two domain requirements developments, both based on the transport domain:
- one, $d_{r_{\text{toll}}}$, for a road-pricing system, and,
- another, $d_{r_{\text{maint.}}}$, for a toll-road link and hub building and maintenance system monitoring and controlling link and hub quality and for development.

- The fitting procedure now identifies the shared awareness by both dr_{toll} and $dr_{\text{maint.}}$ of nets (N), hubs (H) and links (L).
  - We conclude from this that we can single out a common requirements for software that manages net, hubs and links.
  - Such software requirements basically amounts to requirements for a database system.
  - A suitable such system, say a relational database management system, DB_{rel}, may already be available with the customer.

- In any case, where there before were two requirements ($d_{r_{\text{toll}}}, d_{r_{\text{maint.}}}$) there are now four:
  - $d'_{r_{\text{toll}}}$, a modification of $d_{r_{\text{toll}}}$ which omits the description sections pertaining to the net;
  - $d'_{r_{\text{maint.}}}$, a modification of $d_{r_{\text{maint.}}}$ which likewise omits the description sections pertaining to the net;
  - $d_{r_{\text{net}}}$, which contains what was basically omitted in $d'_{r_{\text{toll}}}$ and $d'_{r_{\text{maint.}}}$; and
  - $d_{r_{\text{db:i/f}}}$ (db:i/f for database interface) which prescribes a mapping between type names of $d_{r_{\text{net}}}$ and relation and attribute names of DB_{rel} ■

- Much more can and should be said, but this suffices as an example in a software engineering methodology paper.

7.2.6. Domain Requirements Consolidation

- After projection, instantiation, determination, extension and fitting,
  - it is time to review, consolidate and possibly restructure (including re-specify)
  - the domain requirements prescription
  - before the next stage of requirements development.

Dines Bjørner's MAP-i Lecture # 9

End of MAP-i Lecture # 9: Domain Requirements: Extension and Fitting

Thursday, 28 May 2015: 10:00–11:15

Dines Bjørner's MAP-i Lecture # 10

Interface Requirements

Thursday, 28 May 2015: 12:15–13:00

7.3. Interface Requirements

- By an **interface requirements** we 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) ■

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

7.3.1. Shared Phenomena

- By **sharing** we mean
  - that an **endurant** is represented both
    - in the domain and
    - "inside" the machine, and
    - that its machine representation
    - must at suitable times
    - reflect its 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

- 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 ■

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

- 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.

- Each such shared phenomenon shall then be individually dealt with:
  - **endurant sharing** shall lead to interface requirements for data initialisation and refreshment;
  - **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.

• • •

- We shall now illustrate these domain interface requirements
- development steps with respect to our ongoing example.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

7.3.2. Shared Endurants

- We "split" our interface requirements development into two separate steps:
  - the development of $d_{r_{\text{net}}}$
    - (the common domain requirements for the shared hubs and links),
  - and the co-development of $d_{r_{\text{db:i/f}}}$
    - (the common domain requirements for the interface between $d_{r_{\text{net}}}$ and DB_{rel} —
- under the assumption of an available relational database system DB_{rel})

Example 92 . Interface Requirements. Shared Endurants:

- The main shared endurants are
  - the net (hubs, links) and
  - the vehicles.
- As domain endurants hubs and links undergo changes,
  - all the time,
  - with respect to the values of several attributes:
    - *length, cadestral information, names,*
    - *wear and tear* (where-ever applicable),
    - *last/next scheduled maintenance* (where-ever applicable),
    - *state* and *state space*,
    - and many others.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

- Similarly for vehicles:
  - their position,
  - velocity and acceleration, and
  - many other attributes.
- When planning the common domain requirements for the net, i.e., the hubs and links,
  - we enlarge our scope of requirements concerns beyond the two so far treated ($d_{r_{\text{toll}}}$, $d_{r_{\text{maint.}}}$)
  - in order to make sure that the shared relational database of nets, their hubs and links, may be useful beyond those requirements.

- 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 DB_{rel}

 © Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

7.3.2.1. Data Initialisation

- As part of $d_{r_{\text{net}}}$ 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.
- Essentially these prescriptions concretise the insert and remove link and hub actions.

Example 93 . Interface Requirements. Shared Endurant Initialisation:

- The domain is that of the road net, n:N, say of Chapter 6 — see also Example 92 on Slide 475
- 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) , and
  - the initial values of all their attributes, **attributes**(l) and **attributes**(h).

223 There are r_l and r_h "recorders" recording link, respectively hub properties with each recorder having a unique identity,

224 Each recorder is charged with a set of links or a set of hubs according to some partitioning of all such.

225 The recorders inform a central data base, net_db, of their recordings:

a. (ri,nol,(u$_j$,m$_j$,attrs$_j$)) where
b. ri is the identity of the recorder,
c. nol is either `link` or `hub`,
d. u$_j$ = **uid**_L(l) or **uid**_H(h) for some link or hub,
e. m$_j$ = **obs_mereo**_L(l) or **obs_mereo**_H(h) for that link or hub and
f. attrs$_j$ = **attributes**(l) or **attributes**(h) for that link or hub.

type

223. RI

value

223. rl,rh:NAT **axiom** rl>0 $\wedge$ rh>0

type

225a.. M = RI×″`link`″×LNK | RI×″`hub`″×HUB

225a.. LNK = LI × HI-**set** × LATTRS

225a.. HUB = HI × LI-**set** × HATTRS

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

value

224. partitioning: L-**set** $\to$ **Nat** $\to$ (L-**set**)*
224. $\quad\quad\quad\quad$ | H-**set** $\to$ **Nat** $\to$ (H-**set**)*
224. partitioning(s)(r) **as** sl
224. $\quad$ **post**: **len** sl = r
224. $\quad\quad$ $\wedge \cup$ **elems** sl = s
224. $\quad\quad$ $\wedge \forall$ si,sj:(L-**set**|H-**set**) $\cdot$
224. $\quad\quad\quad$ si$\neq$\{\}
224. $\quad\quad\quad$ $\wedge$ sj$\neq$\{\}
224. $\quad\quad\quad$ $\wedge$ \{si,sj\}$\subseteq$**elems** ss $\Rightarrow$ si $\cap$ sj = \{\}

226 The $r_l + r_h$ recorder behaviours interact with the one net_db behaviour

channel
226. r_db: RI×(LNK|HUB)
value
226. LNK_recorder: RI → L-**set** → **out** r_db **Unit**
226. HUB−recorder: RI → H-**set** → **out** r_db **Unit**
226. net_db: **Unit** → **in** r_db **Unit**

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

227 The data base behaviour, **net_db**, offers to receive messages from the link an hub recorders.

228 And the data base behaviour, **net_db**, deposits these messages in respective variables.

229 Initially there is a net, $n : N$,

230 from which is observed its links and hubs.

231 These sets are partitioned into r_l, respectively r_h length lists of non-empty links and hubs.

232 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.

233 We construct, for technical reasons, as the listener will soon see, disjoint lists of link, respectively hub recorder identities.

value
227. net_db:
variable
228. lnk_db: $(\mathsf{RI}\times\mathsf{LNK})$**-set**
228. hub_db: $(\mathsf{RI}\times\mathsf{HUB})$**-set**
value
229. n:N
230. ls:L**-set** = obs_Ls(obs_LS(n))
230. hs:H**-set** = obs_Hs(obs_HS(n))
231. lsl:(L**-set**)* = partition(ls)(rl)
231. lhl:(H**-set**)* = partition(hs)(rh)
233. rill:RI* **axiom len** rill = rl = **card elems** rill
233. rihl:RI* **axiom len** rihl = rh = **card elems** rihl

232. ab_initio_data: $\mathbf{Unit} \to \mathbf{Unit}$
232. ab_initio_data() $\equiv$
232. $\quad\|$ {lnk_rec(rill[i])(lsl[i])|i:$\mathbf{Nat}$·$1 \leq$i$\leq$rl}
232. $\quad\|$ {hub_rec(rihl[i])(lhl[i])|i:$\mathbf{Nat}$·$1 \leq$i$\leq$rh}
232. $\quad\|$ net_db()

234 The link and the hub recorders are near-identical behaviours.

a. 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

b. the unique identifier,

c. the mereology, and

d. the attributes.

e. These “data” are sent, as a message, prefixed the senders identity, to the data base behaviour.

f. We presently leave the ... unexplained.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

value

226. link_rec: RI → L-**set** → **Unit**

234. link_rec(ri,ls) ≡

234a.. **for** ∀ l:L·l ∈ ls **do** **uid**_L(l)

234b.. **let** lnk = (**uid**_L(l),

234c.. **obs_mereo**_L(l),

234d.. **attributes**(l)) **in**

234e.. rdb ! (ri,"`link`",lnk);

234f.. ... **end**

234a.. **end**

```
226.   hub_rec: RI × H-set → Unit
234.   hub_rec(ri,hs) ≡
234a..     for ∀ h:H•h ∈ hs do uid_H(h)
234b..        let hub = (uid_L(h),
234c..                   obs_mereo_H(h),
234d..                   attributes(h)) in
234e..        rdb ! (ri,"hub",hub);
234f..        ... end
234a..    end
```

235 The **net_db** data base behaviour revolves around a seemingly "never-ending" cyclic process.

236 Each cycle "starts" with acceptance of some,

237 either link or hub data.

238 If link data then it is deposited in the link data base,

239 if hub data then it is deposited in the hub data base.

value

235. net_db() ≡
236. **let** (ri,loh,data) = r_db ? **in**
237. **case** loh **of**
238. "`link`" → ... ; lnk_db := lnk_db ∪ (ri,data),
239. "`hub`" → ... ; hub_db := hub_db ∪ (ri,data)
237. **end end** ;
235′. ... ;
235. net_db()

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

- 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 Item 234f., Slide 488);
    - (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 Item 235′, Slide 491) ■

- 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., $(ui, me, attrs)$.

 © Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

7.3.2.2. Data Refreshment

- As part of $d_{r_{\text{net}}}$ 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.
- These prescriptions also concretise insert and remove link and hub actions.

7.3.3. Shared Actions, Events and Behaviours

- We illustrate the ideas of
  - shared actions, events and behaviours
  - through the domain requirements extension
  - of Sect. 7.2.4,
  - more specifically Examples 87–89
    Slides 442–449.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

Example 94 . Interface Requirements. Shared Actions, Events and Behaviours:

This Example has yet to be written

Examples 88–90, Slides 445–453,
illustrate shared interactive actions, events and behaviours.

7.4. Machine Requirements

7.4.1. Delineation of Machine Requirements

7.4.1.1. On Machine Requirements

Definition 33 . Machine Requirements: *By* **machine requirements** *we shall understand*

- *such requirements*
- *which can be expressed "sôlely" using terms*
- *from, or of the machine* ■

Definition 34 . The Machine: *By the* **machine** *we shall understand*

- *the hardware*
- *and software*
- *to be built from the requirements* ■

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

- 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.

7.4.1.2. Machine Requirements Facets

- We shall, in particular, consider the following five kinds of machine requirements:
  - *performance requirements*,
  - *dependability requirements*,
  - *maintenance requirements*,
  - *platform requirements* and
  - *documentation requirements.*

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

7.4.2. Performance Requirements

Definition 35 . 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 95 . Machine Requirements. Road-pricing System Performance:

- Possible road pricing system performance requirements could evolve around:
  - maximum number of cars entering and leaving the sum total of all gates within a minimum period —
    for example 10.000 maximum within any interval of 10 seconds minimum;
  - maximum time between a car entering a gate and the raising of the gate barrier —
    for example 3 seconds;
  - etcetera,
- We cannot be more specific:
  - that would require more details about
  - gate sensors and
  - gate barriers.

 © Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

7.4.3. Dependability Requirements

MORE TO COME

7.4.3.1. 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 36 . 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 – May 23, 2015: 15:29

Definition 37 . 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 38 . 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 – May 23, 2015: 15:29

Definition 39 . 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 40 . 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 – May 23, 2015: 15:29

- *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 – May 23, 2015: 15:29

- 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 41 . Dependability Attribute: *By a dependability attribute we shall mean either one of the following:*

- *accessibility,*
- *availability,*
- *integrity,*
- *reliability,*
- *robustness,*
- *safety and*
- *security.*

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

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.).

7.4.3.2. 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 never get access to the machine.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

Definition 42 . 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 96 . Machine Requirements. Road-pricing System Accessibility:

- Fairness of the calculator behaviour, cf. formula Item 220 on Slide 450 (⟦⟧)
  - shall mean that "earlier" (wrt. time-stamped) messages
  - from either vehicles
  - or from gates
  - shall be accepted by the calculator
  - before "later" such messages.
- This is guaranteed by the semantics of RSL.
  - And, hence, shall be guaranteed
  - by any implementation of the deterministic choice⌊⌈

 © Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

7.4.3.3. 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"!

Definition 43 . 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 — hence some form of "time slicing" is to be effected*

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

Example 97 . Machine Requirements. Road-pricing System Availability:

- Formula Item 216b. (Slide 445) specify that
  - vehicles "continuously" inform
  - the calculator (cf. formula Items 220 on Slide 450)
  - of their time-stamped local position.
- This may lead you to think that these messages
  - may effectively "block out"
  - "concurrent" messages from toll-road gates.
- In an implementation we may choose
  - to discretize vehicle-to-calculator messages.
  - That is, to "space them apart",
  - some time interval —
  - so long as an "intentional semantics is maintained"

7.4.3.4. Integrity

Definition 44 . 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 – May 23, 2015: 15:29

Example 98 . Machine Requirements. Road-pricing System Integrity:

- We divide the integrity concerns for
  the road-pricing computing and communications system
  into two "spheres":
  - the integrity of the sensor and actuator equipment
    attached to
    - vehicles (i.e., their GNSS attributes), and to
    - toll-road gates:
      - in/out sensors,
      - vehicle identifiers and
      - gates;

    and
  - the software of the road-pricing computing and communications system,
    - that is, the software which interfaces with
      - vehicles,
      - toll-gates and
      - the calculator.

- As for the integrity of the the sensor and actuator equipment we do not require
  - that the road-pricing computing and communications system
  - is 100% dependable,
  - It is satisfactory if it retains its
    - accessibility,
    - availability,
    - reliability,
    - safety and
    - security

  in the presence of maintenance.

 © Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

- As for the integrity of the software we require that it
  - is **proven correct**
    with respect to domain and requirements specifications
    under the assumption that
    sensor and actuator equipment functions
    with 100%'s integrity;
  - and where correctness proofs
    may not be feasible or possible,
    that the software is appropriately **model-checked**;
  - and where "complete" model-checks
    may not be feasible or possible,
    that the software is **formally tested**

Definition 45 . Reliability: *A system being reliable — in the context of a machine being dependable — means*

- *some measure of continuous correct service,*
- *that is, measure of time to failure*

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

Example 99 . Machine Requirements. Road-pricing System Reliability:

- *Mean-time between failures*, `MTBF`,
  - (i) of any vehicle's `GNSS` correct recording of local position must be at least 30.000 hours;
  - (ii) of any toll-gate complex, that is,
    - it's ability to correctly identify a passing vehicle, or
    - it's ability to correctly close and open gates

    must be at least 20.000 hours

7.4.3.5. Safety

Definition 46 . 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* ■

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

Example 100 . Machine Requirements. Road-pricing System Safety:

- *Mean time to catastrophic failure*, MTCF,
  - (i) for a vehicle's GNSS to function properly shall be 60.000 hours; and
  - (ii) of any toll-gate complex, that is,
    - it's ability to correctly identify a passing vehicle, or
    - it's ability to correctly close and open gates

  must be at least 40.000 hours

7.4.3.6. 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.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

Definition 47 . 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 101 . Machine Requirements. Road-pricing System Security:

- Vehicles are authorised
  - to receive GNSS timed global positions,
    but not to tamper with, e.g. misrepresent them,

  are authorised
  - to, and shall correctly compute
    their local positions
    based on the received global positions,

  and are finally authorised
  - to, and shall correctly
    inform the calculator of their timed local positions

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

7.4.3.7. Robustness

Definition 48 . 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 102 . Machine Requirements. Road-pricing System Robustness:

- The road-pricing computing and communications system shall retain its
  - performance and
  - dependability, that is,
    - accessibility,
    - availability,
    - reliability, and
    - safety

  requirements
- in the presence of maintenance.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

7.4.4. Maintenance Requirements

TO BE TYPED

7.4.4.1. Delineation and Facets of Maintenance Requirements

Definition 49 . 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 61 on Slide 553.

 © Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

7.4.4.2. Adaptive Maintenance

Definition 50 . 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 103 . Machine Requirements. Road-pricing System Adaptive Maintenance:

- Two forms of adaptive maintenance occur in connection with the road-pricing computing and communication system:
  - adaptive maintenance of vehicle and toll-gate sensors and actuators, and
  - adaptive maintenance of the "interfacing" software, that is,
    - the vehicle software as prescribed by Item 216 on Slide 445,
    - the toll-gate software as prescribed by Item 219 on Slide 448, and
    - the calculator software as prescribed by Item 220 on Slide 450.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

- Adaptive maintenance of vehicle and toll-gate sensors and actuators occurs when
  - existing sensors or actuators
  - are replaced due to failure.
- Adaptive maintenance of interfacing software is required when
  - existing sensors or actuators have been replaced and their characteristics are different from those of the replaced equipment,
  - hence requires modifications of interfacing software

7.4.4.3. Corrective Maintenance

Definition 51 . Corrective Maintenance: *By corrective maintenance we understand such maintenance which*

- *corrects a software error* ■

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

Example 104 . Machine Requirements. Road-pricing System Corrective Maintenance:

- Corrective maintenance of the road-pricing computing and communications system is required in two "spheres":
  - when system, that is, toll-gate and vehicles sensors or actuators fail, and
  - when, despite all verification efforts, the interfacing, that is,
    - the vehicle,
    - the gate, or
    - the calculator

    software fails.

- In the former case (equipment failure)
  - the failing sensor or actuator is replaced
  - possibly implying adaptive maintenance.
- In the latter case (software failure)
  - the failing software is analysed
  - in order to locate the erroneous code,
  - whereupon that code is replaced by such code
  - that can lead to a verification of the full system

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

7.4.4.4. Perfective Maintenance

Definition 52 . 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 105 . Machine Requirements. Road-pricing System Perfective Maintenance:

- We focus on perfective maintenance of
  - vehicle,
  - toll-gate and
  - calculator

  software.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

- We focus, in particular, on
  - the reaction time in connection with response to external stimuli for the gate software
    - the timed local position, Item 216a. on Slide 445, of vehicles;
    - the attr_enter_ch[gi] event from a toll-gate's in coming sensor, Item 219a. on Slide 448;
    - the timed vehicle identity for a attr_TIVI_ch[gi] event form a toll-gate sensor, Item 219b. on Slide 448; and
    - the attr_leave_ch[gi] event from a toll-gate's out going sensor, Item 219d. on Slide 448;

- the reaction time, of the calculator, Item 220 on Slide 450, to incoming, alternating, communications from
  - either vehicles, Item 220a. on Slide 450,
  - or gates, Item 220b. on Slide 450.
- and the calculation time of the calculator
  - for billing, cf. Item 222e. on Slide 452.

 © Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

7.4.4.5. Preventive Maintenance

Definition 53 . Preventive Maintenance: *By preventive maintenance we understand such maintenance which*

- *helps detect, i.e., forestall, future occurrence*
- *of software or hardware failures* ■

Example 106 . Machine Requirements. Road-pricing System Preventive Maintenance:

TO BE WRITTEN

7.4.4.6. Extensional Maintenance

Definition 54 . Extensional Maintenance: *By extensional maintenance we understand such maintenance which adds new functionalities to the software, i.e., which implements additional requirements* ■

Example 107 . Machine Requirements. Road-pricing System Extensional Maintenance:

TO BE WRITTEN

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

7.4.5. Platform Requirements

TO BE WRITTEN

7.4.5.1. Delineation and Facets of Platform Requirements

Definition 55 . 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 56 . Platform Requirements: *By platform requirements we mean a combination of the following:*

- *development platform requirements,*
- *execution platform requirements,*
- *maintenance platform requirements and*
- *demonstration platform requirements* ■

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

7.4.5.2. Development Platform

Definition 57 . 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* ■

7.4.5.3. Execution Platform

Definition 58 . 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* ■

7.4.5.4. Maintenance Platform

Definition 59 . 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* ■

7.4.5.5. Demonstration Platform

Definition 60 . 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 – May 23, 2015: 15:29

Example 108 . Machine Requirements. Road-pricing System Platform Requirements:

- The platform requirements are the following:
  - the **development platform** $\boxed{\text{to be typed}}$
  - the **execution platform** $\boxed{\text{to be typed}}$
  - the **maintenance platform** $\boxed{\text{to be typed}}$

    and
  - the **demonstration platform** $\boxed{\text{to be typed}}$.

7.4.6. Documentation Requirements

Definition 61 . 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., test) documents.

- *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 62 . Documentation Requirements: *By documentation requirements*

- *we mean requirements*
- *of any of the software documents*
- *that together make up*
  - *software and*
  - *hardware*[30]

Example 109 . Machine Requirements — Documentation:

TO BE WRITTEN

[30]— we omit a definition of what we mean by hardware such as the one we gave for software, cf. Definition 61 on Slide 553.

7.4.7. Discussion

TO BE TYPED

Dines Bjørner's MAP-i Lecture # 10

End of MAP-i Lecture # 10:
Interface Requirements

Thursday, 28 May 2015: 12:15–13:00

Dines Bjørner's MAP-i Lecture # 11

Conclusion

Thursday, 28 May 2015: 15:30–16:30

8. Conclusion

8.1. Various Observations

8.1.1. 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[31]:

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

"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.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

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 dealt with in [10, Part IV].

8.1.2. Beauty Is Our Business

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

It's life that matters, nothing but life –
the process of discovering, the everlasting and perpetual process,
not the discovery itself, at all.[32]

- 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.

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

8.2. Acknowledgements

- I thank Dr. Luís Soares Barbosa
  for having organiused my lectures.
  I truly much appreciate the huge amount of work he has done.
- Preparing for these lectures has taken quite some time.
  But it has been fun to do that work —
  including quite some additional research —
  not (yet) enough I am {afraid|glad} to say !
- I thank Prof. José Nuno Oliveira for having supported my being here.
  He has indeed established a truly remarkable department.
  We can all be very proud of him.
- I thank my wife, Kari, of 50 years, for holding my arm
  when crossing the streets of Braga this week !

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

Dines Bjørner's MAP-i Lecture # 11

End of MAP-i Lecture # 11: Conclusion

Thursday, 28 May 2015: 15:30–16:30

Dines Bjørner's MAP-i Lecture # 12

Discussion of Research Topics

Thursday, 28 May 2015: 16:45–17:30

9. Discussion of Research Topics

- There are a number of research topics:
  - some relate to domain analysis & description, cf. Chapter 1, and some of these are listed in Sect. 8.1,
  - other relate to requirements engineering, cf. Chapter 7, and some of these are listed in Sect. 8.2.

9.1. Domain Science & Engineering Topics

- The `TripTych` approach to software development,
  - based on an initial, serious phase of domain engineering,
  - a new phase of software engineering,
  - for which we claim to now have laid
    a solid foundation for domain engineering —
- opens up for a variety of issues that need further study.
- The entries in this section are not ordered
  according to any specific principle.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

9.1.1. Analysis & Description Calculi for Other Domains

- The analysis and description calculus of this paper appears suitable for manifest domains.
- For other domains other calculi appears necessary.
  - There is the introvert, composite domain 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 – May 23, 2015: 15:29

- It seems straightforward:
  - to base a method for analysing & describing a category of domains
  - on the idea of prompts like those developed in this lecture.

9.1.2. On Domain Description Languages

- We have in this seminar expressed the domain descriptions in the RAISE [40] specification language RSL [39].
- With what is thought of as basically inessential, editorial changes, one can reformulate these domain description texts in either of
  - Alloy [45] or
  - The B-Method [1] or
  - VDM [30, 31, 37] or
  - Z [55].

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

- 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 `CafeOBJ` texts.

- We did not go into much detail with respect to perdurants, let alone behaviours.
  - For all the very many domain descriptions, covered elsewhere, `RSL` (with its `CSP` sub-language) suffices.
  - But there are cases where we have conjoined our `RSL` domain descriptions with descriptions in
    - `Petri Nets` [52] or
    - `MSC` [44] or
    - `StateCharts` [42].

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

- Since this seminar only focused on endurants there was no need, it appears, to get involved in temporal issues.
- When that becomes necessary, in a study or description of perdurants, then we either deploy
  - `DC: The Duration Calculus` [56] or
  - `TLA+: Temporal Logic of Actions` [48].

9.1.3. Ontology Relations

- A more exact understanding of the relations between
  - the “classical” AI/information science/ontology view of domains [4, 5, 46], and
  - the algorithmic view of domains, as presented in the current paper,
  - seems required.
- The almost disparate jargon of the two “camps” seems, however, to be a hindrance.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

9.1.4. Analysis of Perdurants

- A study of perdurants, as detailed as that of our study of endurants, ought be carried out.
- One difficulty, as we see it, is the choice of formalisms:
  - whereas the basic formalisms for the expression of endurants and their qualities was type theory and simple functions and predicates,
  - there is no such simple set of formal constructs that can "carry" the expression of behaviours.
    - Besides the textual `CSP`, [43], there is graphic notations of
    - Petri Nets, [52],
    - Message Sequence Charts, [44],
    - State-charts, [42], and others.

9.1.5. Commensurate Discrete and Continuous Models

- Section 5.3.7 Slides 268–270 hinted at
  - co-extensive descriptions of discrete and continuous behaviours,
  - the former in, for example, `RSL`,
  - the latter in, typically, the calculus mathematics of partial different equations (`PDE`s).
  - The problem that arises in this situation is the following:
    - there will be, say variable identifiers, e.g., x, y, ..., z
    - which in the `RSL` formalisation has one set of meanings, but
    - which in the `PDE` "formalisation" has another set of meanings.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

  - Current formal specification languages[33] do not cope with continuity.

- Some research is going on.
- But to substantially cover, for example, the proper description of laminar and turbulent flows in networks (e.g., pipelines, Example 61 on Slide 269) requires more substantial results.

[33] `Alloy` [45],`Event B` [1],`RSL` [39], `VDM-SL` [30, 31, 37], `Z` [55], etc.

9.1.6. Interplay between Parts, Materials and Components

- Examples 49 on Slide 215, 50 on Slide 219, 51 on Slide 222 and 61 on Slide 269 revealed but a small fraction of the problems that may arise in connection with modeling the interplay between parts and materials.
- Subject to proper formal specification language and, for example `PDE` specification, we may expect more interesting
  - laws, as for example those of Examples 50 on Slide 219, 51 on Slide 222,
  - and even proof of these as if they were theorems.
- Formal specifications have focused on verifying properties of requirements and software designs.
- With co-extensive (i.e., commensurate) formal specifications of both discrete and continuous behaviours we may expect formal specifications to also serve as bases for predictions.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

9.1.7. Dynamics

- There is a serious limitation in what can be modeled with the present approach.
  - Although we can model the dynamic introduction of new atomic or removal of existing parts, when members of a composite set of such parts,
  - we cannot model the dynamic introduction or removal of the processes corresponding to such parts.
  - Also we have not shown how to model global time.
  - And, although we can model spatial positions,
  - we have not shown how to model spatial locations.

- These deliberate omissions are due to the facts
  - that the description language, `RSL`, cannot model continuity and
  - that it cannot provide for arbitrary models of time.
- Here is an area worth studying.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

9.1.8. Precise Descriptions of Manifest Domains

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

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

  — may serve to

  - improve the "business processes" of these domains,
  - make them more palatable for the human actors,
  - make them more efficient wrt. resource-usage.
- Descriptions of domains are descriptions of the syntax and semantics of the technical languages used in speaking about and in the domain.

 © Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

- The domain analysis required for the design of programming languages is based on computability: mathematical logic and recursive function theory.
- The domain analysis required for "real-world" domains is not based on computability: that "world" is not computable.
- Requirements engineering based on domain descriptions is based on deriving computable subsets of refined domain descriptions.
- The classical theory and practice of programming language semantics and compiler development [6] and [9, Part VII (Chapters 16–19)] can now be further developed into a theory and practice for deriving general software from formal domain descriptions [12].
- Descriptions of domains are descriptions of the syntax and semantics of the technical languages used in speaking about and in the domain.

- The domain analysis required for the design of programming languages is based on computability: mathematical logic and recursive function theory.
- The domain analysis required for "real-world" domains is not based on computability: that "world" is not computable.
- Requirements engineering based on domain descriptions is based on deriving computable subsets of refined domain descriptions.
- The classical theory and practice of programming language semantics and compiler development [6] and [9, Part VII (Chapters 16–19)] can now be further developed into a theory and practice for deriving general software from formal domain descriptions [12].

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

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

9.1.9. Towards Mathematical Models of Domain Analysis & Description

- There are two aspects to a precise description of the **domain analysis prompt**s and **domain description prompt**s.
  - There is that of describing
    - the individual prompts
    - as if they were "machine instructions"
    - for an albeit strange machine;
  - and there is that of describing
    - the interplay between prompts:
      - the sequencing of **domain description prompt**s
      - as determined by the outcome of the **domain analysis prompt**s.

- We have
  - described and formalised the latter in [25, Processes];
  - and we are in the midst of describing and formalising the former in [19, Prompts].

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

9.1.10. Laws of Descriptions: A Calculus of Prompts

- Laws of descriptions deal with the order and results of applying the domain analysis and description prompts.
- Some laws are covered in [17].
- It is expected that establishing formal models of the prompts, for example as outlined in [19, 25], will help identify such laws.

- The various description prompts apply to parts (etc.) of specified sorts (etc.) and to a "hidden state".
  - The "hidden state" has two major elements:
    - the domain and
    - the evolving description texts.
  - An "execution" of a prompt potentially changes that "hidden state".

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

- Let P, PA and PB be composite part sorts where PA and PB are derived from P.
- Let $\Re_i$, $\Re_j$, etc., be suitable functions which rename sort, type and attribute names.
- In a proper prompt calculus
  - we would expect
  - `observe_part_sorts_PA`;`observe_part_sorts_PB`,
  - when "executed" by one and the same domain engineer,
  - to yield the same "hidden state" as
  - `observe_part_sorts_PB`;$\Re_i$;`observe_part_sorts_PA`;$\Re_j$.

- Also one would expect
  - `observe_part_sorts_PA`;$\Re_i$;`observe_part_sorts_PA`;$\Re_j$.
  - to yield the same state as just
  - `observe_part_sorts_PA`
  - given suitable renaming functions.
- Well ? or does one really ?

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

- There are some assumptions that are made here.
- One pair of assumptions is
  - that the domain is fixed
  - and to one observer.
  - yields the same analysis and description results
  - no matter in which order prompts are "executed".
- Another assumption is that the domain engineer
  - does not get wiser as analysis and description progresses.
- If, as one can very well expect, the domain engineer does get wiser,
  - then former results may be discarded and
  - either replaced by newer analysis and descriptions
  - or prompts repeated.
- In such cases these laws do not hold.

9.1.11. Domains and Galois Connections

- Section 1.1.8 very briefly mentioned that formal concepts form Galois Connections.
- In the seminal [38] a careful study is made of this fact and beautiful examples show the implications for domains.
- It seems that our examples have all been too simple.
- They do not easily lead on to the "discovery" of "new" domain concepts from appropriate concept lattices.
- We refer to [29, Section 9].
- Further study need be done.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

9.1.12. Laws of Domain Description Prompts

- Typically `observe_part_sorts` applies to a composite part, p:P, and yield descriptions of one or more part sorts: p_1:P_1,p_2:P_2,...,p_m:P_m.
- Let p_i:P_i,p_j:P_j,...,p_k:P_k (of these) be composite.
- Now `observe_part_sorts`(p_i) and `observe_part_sorts`(p_j), etc., can be applied and yield texts ***text***$_i$, respectively ***text***$_j$.
- A law of domain description prompts now expresses that the order in which the two or more observers is applied is immaterial, that is, they commute.
- In [17] we made an early exploration of such laws of domain description prompts.
- More work, hear also next, need be done.

9.1.13. Domain Theories:

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

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

Example 110 . A Law of Train Traffic at Stations:

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

- Now let us (idealistically) assume
  - that actual trains arrive at and depart from train stations according the train timetable and
  - that the train traffic includes all and only such trains as are listed in the train timetable.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

- Now a law of train traffic expresses
  - *"Over the modulo time interval of a train timetable it is the case that*
    - *the number of trains arriving at a station*
    - *minus the number of trains ending their journey at that station*
    - *plus the number of trains starting their journey at that station*
    - *equals number of trains departing from that station."*

9.1.14. External Attributes

- More study is needed in order to clarify
  - the relations between the various external attributes
  - and control theory.

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

9.2. Requirements Topics

9.2.1. Domain Requirements Methodology

- Further principles, techniques and tools
- for the projection, instantiation, determination, extension and fitting operations.

9.2.2. Domain Requirements Operator Theory

- A model of the domain to domain-to-requirements operators:
- projection, instantiation, determination, extension and fitting. (Sect. 4).

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

9.2.3. Methodology for Interface Requirements

- Sect. 7.3 did not go into sufficient detail as to method principles, techniques and tools.

9.3. Final Words

$\boxed{\text{Have a Happy \& Fruitful R\&D Career!}}$

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

Dines Bjørner's MAP-i Lecture # 12

End of MAP-i Lecture # 12: Discussion of Research Topics

Thursday, 28 May 2015: 16:45–17:30

10. Bibliography

10.1. Bibliographical Notes

- Web page **www.imm.dtu.dk/˜dibj/domains/** lists the published papers and reports mentioned in the next two subsections.

10.1.1. Published Papers

- I have thought about domain engineering for more than 20 years.
- But serious, focused writing only started to appear since [10, Part IV] — with [8, 7] being exceptions:
  - [11] suggests a number of domain science and engineering research topics;
  - [14] covers the concept of domain facets;
  - [29] explores compositionality and Galois connections.
  - [12, 28] show how to systematically, but, of course, not automatically, "derive" requirements prescriptions from domain descriptions;
  - [16] takes the triptych software development as a basis for outlining principles for believable software management;
  - [13, 21] presents a model for Stanisław Leśniewski's [32] concept of mereology;

- [15, 17] present an extensive example and is otherwise a precursor for the present paper;
- [18] 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;
- [20] analyses the **TripTych**, especially its domain engineering approach, with respect to Maslow's [34] and Peterson's and Seligman's [35] notions of humanity: how can computing relate to notions of humanity;
- the first part of [22] 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
- [23] focus on domain safety criticality.

[34] *Theory of Human Motivation.* Psychological Review 50(4) (1943):370-96; and *Motivation and Personality,* Third Edition, Harper and Row Publishers, 1954.
[35] *Character strengths and virtues: A handbook and classification.* Oxford University Press, 2004

The present paper basically replaces the domain analysis and description section of all of the above reference — including [10, Part IV].

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

10.1.2. 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.p` (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)

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

10.2. References

[1] J.-R. 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.

[2] A. Badiou. *Being and Event*. Continuum, 2005. (Lêtre et l'événements, Edition du Seuil, 1988).

[3] G. Birkhoff. *Lattice Theory*. American Mathematical Society, Providence, R.I., 3 edition, 1967.

[4] T. Bittner, M. Donnelly, and B. Smith. Endurants and Perdurants in Directly Depicting Ontologies. *AI Communications*, 17(4):247–258, December 2004. IOS Press, in [53].

[5] T. Bittner, M. Donnelly, and B. Smith. Individuals, Universals, Collections: On the Foundational Relations of Ontology. In A. Varzi and L. Vieu, editors, *Formal Ontology in Information Systems*,

Proceedings of the Third International Conference, pages 37–48. IOS Press, 2004.

[6] D. Bjørner. Programming Languages: Formal Development of Interpreters and Compilers. In *International Computing Symposium 77 (eds. E. Morlet and D. Ribbens)*, pages 1–21. European ACM, North-Holland Publ.Co., Amsterdam, 1977.

[7] D. Bjørner. Michael Jackson's Problem Frames: Domains, Requirements and Design. In L. ShaoYang and M. Hinchley, editors, *ICFEM'97: International Conference on Formal Engineering Methods*, Los Alamitos, November 12–14 1997. IEEE Computer Society. Final Version.

[8] D. 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. Final draft version.

[9] D. Bjørner. *Software Engineering, Vol. 2: Specification of Systems and Languages*. Texts in Theoretical Computer Science, the EATCS Series. Springer, 2006. Chapters 12–14 are primarily authored by Christian Krog Madsen.

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

[11] D. 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.

[12] D. 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.

[13] D. 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.

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

[15] D. 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.

[16] D. Bjørner. Believable Software Management. *Encyclopedia of Software Engineering*, 1(1):1–32, 2011.

[17] D. 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.

[18] D. 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.

[19] D. Bjørner. Domain Analysis: A Model of Prompts (paper[36], slides[37]). Research Report 2013-6, DTU Compute and Fredsvej 11, DK-2840 Holte, Denmark, Fall 2013.

[20] D. Bjørner. *Domain Science and Engineering as a Foundation for Computation for Humanity*, chapter 7, pages 159–177. Com-

[36] http://www.imm.dtu.dk/˜dibj/da-mod-p.pdf
[37] http://www.imm.dtu.dk/˜dibj/da-mod-s.pdf

putational Analysis, Synthesis, and Design of Dynamic Systems. CRC [Francis & Taylor], 2013. (eds.: Justyna Zander and Pieter J. Mosterman).

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

[22] D. Bjørner. Domain Analysis: Endurants – An Analysis & Description Process Model. In S. Iida, J. Meseguer, and K. Ogata, editors, *Specification, Algebra, and Software: A Festschrift Symposium in Honor of Kokichi Futatsugi*. Springer, May 2014.

[23] D. Bjørner. Domain Engineering – A Basis for Safety Critical Software. Invited Keynote, ASSC2014: Australian System Safety Conference, Melbourne, 26–28 May, December 2014.

[24] D. Bjørner. Manifest Domains: Analysis & Description. Research Report, 2014. Part of a series of research reports: [26, 27], Being

submitted.

[25] D. Bjørner. Domain Analysis: Endurants – a Consolidated Model of Prompts. Research Report, DTU Compute and Fredsvej 11, DK-2840 Holte, Denmark, May 2015.

[26] D. Bjørner. Domain Analysis & Description: Models of Processes and Prompts. Research Report, To be completed early 2014. Part of a series of research reports: [24, 27].

[27] D. Bjørner. From Domains to Requirements – A Different View of Requirements Engineering. Research Report, To be completed mid 2015. Part of a series of research reports: [24, 26].

[28] D. Bjørner. The Role 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.

[29] D. Bjørner and A. 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.

[30] D. Bjørner and C. B. Jones, editors. *The Vienna Development Method: The Meta-Language*, volume 61 of *LNCS*. Springer, 1978.

[31] D. Bjørner and C. B. Jones, editors. *Formal Specification and Software Development*. Prentice-Hall, 1982.

[32] R. Casati and A. Varzi. *Parts and Places: the structures of spatial representation*. MIT Press, 1999.

[33] R. Casati and A. Varzi. Events. In E. N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Spring 2010 edition, 2010.

[34] R. Casati and A. 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.

[35] D. Davidson. *Essays on Actions and Events*. Oxford University Press, 1980.

[36] F. Dretske. Can Events Move? *Mind*, 76(479-492), 1967. reprinted in [34], pp. 415-428.

[37] J. Fitzgerald and P. G. 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.

[38] B. Ganter and R. Wille. *Formal Concept Analysis — Mathematical Foundations*. Springer-Verlag, January 1999. ISBN: 3540627715, 300 pages, Amazon price: US $ 44.95.

[39] C. W. George, P. Haff, K. Havelund, A. E. Haxthausen, R. Milne, C. B. Nielsen, S. Prehn, and K. R. Wagner. *The RAISE Specification Language*. The BCS Practitioner Series. Prentice-Hall, Hemel Hampstead, England, 1992.

[40] C. W. George, A. E. Haxthausen, S. Hughes, R. Milne, S. Prehn, and J. S. Pedersen. *The RAISE Development Method*. The BCS Practitioner Series. Prentice-Hall, Hemel Hampstead, England, 1995.

[41] P. Hacker. Events and Objects in Space and Time. *Mind*, 91:1–19, 1982. reprinted in [34], pp. 429-447.

[42] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, 1987.

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

[44] ITU-T. CCITT Recommendation Z.120: Message Sequence Chart (MSC), 1992, 1996, 1999.

[45] D. Jackson. *Software Abstractions: Logic, Language, and Analysis*. The MIT Press, Cambridge, Mass., USA, April 2006. ISBN 0-262-10114-9.

[46] I. Johansson. Qualities, Quantities, and the Endurant-Perdurant Distinction in Top-Level Ontologies. In D. A. B. R. N. M. R.-B. T. 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.

[47] J. Kim. *Supervenience and Mind*. Cambridge University Press, 1993.

[48] L. Lamport. *Specifying Systems*. Addison–Wesley, Boston, Mass., USA, 2002.

[49] D. Mellor. Things and Causes in Spacetime. *British Journal for the Philosophy of Science*, 31:282–288, 1980.

[50] C.-Y. T. Pi. *Mereology in Event Semantics*. Phd, McGill University, Montreal, Canada, August 1999.

[51] A. Quinton. Objects and Events. *Mind*, 88:197–214, 1979.

[52] W. Reisig. *Petrinetze: Modellierungstechnik, Analysemethoden, Fallstudien*. Leitfäden der Informatik. Vieweg+Teubner, 1st edition, 15 June 2010. 248 pages; ISBN 978-3-8348-1290-2.

[53] J. Renz and H. W. Guesgen, editors. *Spatial and Temporal Reasoning*, volume 14, vol. 4, Journal: AI Communications, Amsterdam, The Netherlands, Special Issue. IOS Press, December 2004.

[54] G. Wilson and S. Shpall. Action. In E. N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Summer 2012 edition, 2012.

[55] J. C. P. Woodcock and J. Davies. *Using Z: Specification, Proof and Refinement*. Prentice Hall International Series in Computer Science, 1996.

[56] C. C. Zhou and M. R. Hansen. *Duration Calculus: A Formal Approach to Real–time Systems*. Monographs in Theoretical Computer Science. An EATCS Series. Springer–Verlag, 2004.

11. Table of Contents

Contents

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

© Dines Bjørner 2015, Fredsvej 11, DK–2840 Holte, Denmark – May 23, 2015: 15:29

Last page