

From Domain Descriptions to Requirement Prescriptions

A Different Approach to Requirements Engineering

Dines Bjørner

Uppsala, 18–20 May, 2016

Compiled: May 12, 2016, 10:32 am

- In [Manifest Domains: Analysis & Description] we introduced a method for analysing and describing manifest domains.
- In the next lectures of this seminar
 - ✧ we show how to systematically,
 - ✧ but, of course, not automatically,
 - ✧ “derive” *initial* 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.

- (i) **Domain requirements** are those requirements which can be expressed solely using technical terms of the domain.
- (ii) **Interface requirements** are those requirements which can be expressed using technical terms of both the domain and the machine.
- (iii) **Machine requirements** are those requirements which can be expressed solely using technical terms of the machine.

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

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

- We briefly review principles, techniques and tools for “deriving” *interface requirements* based on sharing domain
 - ❖ (ii.1) endurants, and
 - ❖ (ii.2) perdurants (i.e., actions, events and behaviours)with their machine correspondants.

- ❖ The unfolding of interface requirements lead to a number of machine concepts in terms of which the interface requirements are expressed.
 - ⊗ These machine concepts, both hardware and software,
 - ⊗ make possible the expression of a set of what we shall call ***derived requirements***.
 - ⊗ The paper explores this concept briefly.

- ❖ We do not cover machine requirements in this paper.
- ❖ The reason is that
 - ⊗ we find, cf. [Bjø06, Sect. 19.6], that
 - ⊗ when the individual machine requirements are expressed
 - ⊗ then references to domain phenomena
 - ⊗ are, in fact, abstract references, that is,
 - ⊗ they do not refer to the semantics of what they name.

- This paper claims only to structure the quest for requirements conception.
 - ❖ Instead of “discovering” requirements ‘ab initio’,
 - ⊗ for example, through interviews with stake-holders,
 - ⊗ we suggest to “derive” the requirements based on domain descriptions.
 - ❖ Instead of letting the individual requirements arise out of initial stake-holder interviews,
we suggest to structure these
 - ⊗ (i) around the structures of domain descriptions, and
 - ⊗ (ii) around the structures emerging from domain, interface and machine requirements.

- ❖ We shall refer to the requirements emerging from (i+ii)
- ❖ as the initial requirements.
- ❖ To these we add the *derived requirements* merging from interview with stakeholders:
 - ⊗ We are strongly of the opinion that the techniques and tools of, for example, [DvLF93, Jac01, ZH04, JHJ07, OD08, van09]
 - ⊗ can be smoothly integrated with those of this paper.
- We think that there is some clarification to be gained.

- We claim that our approach
 - ✧ contributes to a restructuring of
 - ✧ the field of requirements engineering
 - ✧ and its very many diverse concerns,
 - ✧ a structuring that is logically motivated
 - ✧ and is based on viewing
software specifications as mathematical objects.

1. Introduction

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

1.1. The Triptych Dogma of Software Development

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

1.2. Software As Mathematical Objects

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

- This base view entails, therefore,
 - ⋄ that specifications,
 - ⊗ whether *software design specifications*,
 - ⊗ or *requirements prescriptions*,
 - ⊗ or *domain descriptions*,
 - ⋄ must [also] be *formal specifications*.
- This is in contrast to considering *software design specifications*
 - ⋄ being artifacts of sociological,
 - ⋄ or even of psychological“nature”.

1.3. The Contribution of These Lectures

- We claim that the present lecture content contributes to our understanding and practice of *software engineering* as follows:
 - ⋄ (1) it shows how the new phase of engineering,
 - ⊗ domain engineering,
 - ⊗ as introduced in [Bjø16b],forms a prerequisite for requirements engineering;
 - ⋄ (2) it endows the “classical” form of requirements engineering with a structured set of development stages and steps:
 - ⊗ (a) first a domain requirements stage,
 - ⊗ (b) to be followed by an interface requirements stages, and
 - ⊗ (c) to be concluded by a machine requirements stage;

- ❖ (3) it further structures and gives a reasonably precise contents to the stage of domain requirements:
 - ⊗ (i) first a projection step,
 - ⊗ (ii) then an instantiation step,
 - ⊗ (iii) then a determination step,
 - ⊗ (iv) then an extension step, and
 - ⊗ (v) finally a fitting step —
- with these five steps possibly being iterated;

- ❖ (4) it also structures and gives a reasonably precise contents to the stage of interface requirements based on a notion of shared entities; and
- ❖ (5) it finally structures and gives a reasonably precise contents to the stage of machine requirements:
 - ⊗ (α) technology requirements and
 - ⊗ (β) development requirements.
- Each of the steps (i–v) open for the possibility of *simplifications*.
- Steps (a–c), (i–v), and (α – β), we claim, are new.
- They reflect a serious contribution, we claim, to a logical structuring of the field of requirements engineering and its very many otherwise seemingly diverse concerns.

1.4. Some Comments on the Lecture Content

- By *methodology* we understand the study and knowledge of one or more methods ■¹
- By a *method* understand the study and knowledge of the principles, techniques and tools for constructing some artifact, here (primarily) software ■

¹The ■ marks the end of definitions.

- These lectures are, perhaps, unusual in the following respects:
 - ❖ They are methodology lectures, hence there are no “neat” theories about development, no succinctly expressed propositions, lemmas nor theorems, and hence no proofs².
 - ❖ As a consequence the lectures are borne by many, and by extensive examples.
 - ❖ The examples of these lectures are all focused on a generic road transport net.

²— where these proofs would be about the development theories. The example development of requirements do imply properties, but formulation and proof of these do not constitute specifically new contributions — so are left out.

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

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

1.5. Structure of Lectures

- The structure of the paper is as follows:
 - ❖ Section 2. provides a fair-sized, hence realistic example.
 - ❖ Sections 3–5. covers our approach to requirements development.
 - ⊙ Section 3. overviews the issue of ‘requirements’; relates our approach (Sects. 4.–5.) to
 - * systems,
 - * user and external equipment and
 - * functional requirements;
 - and
 - ⊙ Sect. 3. also introduces the concepts of
 - * the machine to be requirements prescribed,
 - * the domain,
 - * the interface and
 - * the machine requirements.

- ⊗ Section 4. covers the domain requirements stages of
 - * projection (Sect. 4.1),
 - * instantiation (Sect. 4.2),
 - * determination (Sect. 4.3),
 - * extension (Sect. 4.3),
 - * fitting (Sect. 4.5).
- ⊗ Section 5. covers key features of *interface requirements*:
 - * *shared phenomena* (Sect. 5.1),
 - * *shared endurants* (Sect. 5.2),
 - * *shared actions,*
shared events
shared behaviours (Sect. 5.3).
 - * Section 5.3 further introduces the notion of *derived requirements*.
- ⊗ Section 7. concludes the paper.

2. An Example Domain: Transport

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

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

2.1. Endurants

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

2.1.1. Domain, Net, Fleet and Monitor

Applying `observe_part_sorts` [Bjø14d, Sect. 3.1.6] to to a transport domain $\delta:\Delta$ yields the following.

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

- 1 We analyse the composite traffic system into
 - a. a composite road net,
 - b. a composite fleet (of vehicles), and
 - c. an atomic monitor.

type

1 Δ

1a. N

1b. F

1c. M

value

1a. **obs_part_N**: $\Delta \rightarrow N$

1b. **obs_part_F**: $\Delta \rightarrow F$

1c. **obs_part_M**: $\Delta \rightarrow M$

Applying `observe_part_sorts` [Bjø14d, Sect. 3.1.6] to a net, $n:N$, yields the following.

2 The road net consists of two composite parts,

- a. an aggregation of hubs and
- b. an aggregation of links.

type

2a. HA

2b. LA

value

2a. **obs_part**_HA: $N \rightarrow HA$

2b. **obs_part**_LA: $N \rightarrow LA$

2.1.2. Hubs and Links

Applying `observe_part_types` [Bjø14d, Sect. 3.1.7] to hub and link aggregates yields the following.

3 Hub aggregates are sets of hubs.

4 Link aggregates are sets of links.

5 Fleets are set of vehicles.

type3 **H, HS = H-set**4 **L, LS = L-set**5 **V, VS = V-set****value**3 **obs_part_HS: HA \rightarrow HS**4 **obs_part_LS: LA \rightarrow LS**5 **obs_part_VS: F \rightarrow VS**

6 We introduce some auxiliary functions.

a. **links** extracts the links of a network.

b. **hubs** extracts the hubs of a network.

value6a. **links: $\Delta \rightarrow$ L-set**6a. **links(δ) \equiv obs_part_LS(obs_part_LA(obs**6b. **hubs: $\Delta \rightarrow$ H-set**6b. **hubs(δ) \equiv obs_part_HS(obs_part_HA(ob**

2.1.3. Unique Identifiers

Applying `observe_unique_identifier` [Bjø14d, Sect. 3.2] to the observed parts yields the following.

7 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.

type

7a. NI, HAI, LAI, HI, LI, FI, VI, MI

value

7c. **uid_NI**: $N \rightarrow NI$

7c. **uid_HAI**: $HA \rightarrow HAI$

7c. **uid_LAI**: $LA \rightarrow LAI$

7c. **uid_HI**: $H \rightarrow HI$

7c. **uid_LI**: $L \rightarrow LI$

7c. **uid_FI**: $F \rightarrow FI$

7c. **uid_VI**: $V \rightarrow VI$

7c. **uid_MI**: $M \rightarrow MI$

axiom

7b. $NI \cap HAI = \emptyset, NI \cap LAI = \emptyset, NI \cap HI = \emptyset$, etc.

where axiom 7b.. is expressed semi-formally, in mathematics.

We introduce some auxiliary functions:

- 8 `xtr_lis` extracts all link identifiers of a traffic system.
- 9 `xtr_his` extracts all hub identifiers of a traffic system.
- 10 Given an appropriate link identifier and a net `get_link` ‘retrieves’ the designated link.
- 11 Given an appropriate hub identifier and a net `get_hub` ‘retrieves’ the designated hub.

value

```

8  xtr_lis:  $\Delta \rightarrow \text{LI-set}$ 
8  xtr_lis( $\delta$ )  $\equiv$ 
8    let ls = links( $\delta$ ) in {uid_LI( $l$ ) |  $l:L \cdot l \in \text{ls}$ } end
9  xtr_his:  $\Delta \rightarrow \text{HI-set}$ 
9  xtr_his( $\delta$ )  $\equiv$ 
9    let hs = hubs( $\delta$ ) in {uid_HI( $h$ ) |  $h:H \cdot k \in \text{hs}$ } end
10 get_link:  $\text{LI} \rightarrow \Delta \xrightarrow{\sim} L$ 
10 get_link( $li$ )( $\delta$ )  $\equiv$ 
10   let ls = links( $\delta$ ) in
10   let  $l:L \cdot l \in \text{ls} \wedge li = \text{uid\_LI}(l)$  in  $l$  end end
10   pre:  $li \in \text{xtr\_lis}(\delta)$ 
11 get_hub:  $\text{HI} \rightarrow \Delta \xrightarrow{\sim} H$ 
11 get_hub( $hi$ )( $\delta$ )  $\equiv$ 
11   let hs = hubs( $\delta$ ) in
11   let  $h:H \cdot h \in \text{hs} \wedge hi = \text{uid\_HI}(h)$  in  $h$  end end
11   pre:  $hi \in \text{xtr\_his}(\delta)$ 

```

2.1.4. Mereology

Applying `observe_mereology` [Bjø14d, Sect. 3.3.2] to hubs, links, vehicles and the monitor yields the following.

- 12 Hub mereologies reflect that they are connected to zero, one or more links.
- 13 Link mereologies reflect that they are connected to exactly two distinct hubs.
- 14 Vehicle mereologies reflect that they are connected to the monitor.
- 15 The monitor mereology reflects that it is connected to all vehicles.
- 16 For all hubs of any net it must be the case that their mereology designates links of that net.
- 17 For all links of any net it must be the case that their mereologies designates hubs of that net.
- 18 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

12 **obs_mereo_H**: $H \rightarrow LI\text{-set}$

13 **obs_mereo_L**: $L \rightarrow HI\text{-set}$

axiom

13 $\forall l:L \cdot \text{card } \text{obs_mereo_L}(l) = 2$

value

14 **obs_mereo_V**: $V \rightarrow MI$

15 **obs_mereo_M**: $M \rightarrow VI\text{-set}$

axiom

16 $\forall \delta:\Delta, hs:HS \cdot hs = \text{hubs}(\delta), ls:LS \cdot ls = \text{links}(\delta) \cdot$

16 $\forall h:H \cdot h \in hs \cdot \text{obs_mereo_H}(h) \subseteq \text{xtr_lis}(\delta) \wedge$

17 $\forall l:L \cdot l \in ls \cdot \text{obs_mereo_L}(l) \subseteq \text{xtr_his}(\delta) \wedge$

18a. **let** $f:F \cdot f = \text{obs_part_F}(\delta) \Rightarrow$

18a. **let** $m:M \cdot m = \text{obs_part_M}(\delta),$

18a. $vs:VS \cdot vs = \text{obs_part_VS}(f)$ **in**

18a. $\forall v:V \cdot v \in vs \Rightarrow \text{uid_V}(v) \in \text{obs_mereo_M}(m)$

18b. $\wedge \text{obs_mereo_M}(m) = \{\text{uid_V}(v) \mid v:V \cdot v \in vs\}$

18b. **end end**

2.1.5. Attributes, I

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

- **Hubs:**

- ❖ locations are considered static,
- ❖ hub states and hub state spaces are considered programmable;

- **Links:**

- ❖ lengths and locations are considered static,
- ❖ link states and link state spaces 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

Applying `observe_attributes` [Bjø14d, Sect. 3.4.3] to hubs, links, vehicles and the monitor yields the following.

First hubs.

19 Hubs

- a. have geodetic locations, `GeoH`,
- b. have `hub states` which are sets of pairs of identifiers of links connected to the hub³,
- c. and have `hub state spaces` which are sets of hub states⁴.

20 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.

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

³A hub state “signals” which input-to-output link connections are open for traffic.

⁴A hub state space indicates which hub states a hub may attain over time.

type

19a. GeoH

19b. $H\Sigma = (LI \times LI)\text{-set}$ 19c. $H\Omega = H\Sigma\text{-set}$ **value**19a. **attr_GeoH**: $H \rightarrow \text{GeoH}$ 19b. **attr_HΣ**: $H \rightarrow H\Sigma$ 19c. **attr_HΩ**: $H \rightarrow H\Omega$ **axiom**20 $\forall \delta:\Delta,$ 20 **let** $hs = \text{hubs}(\delta)$ **in**20 $\forall h:H \cdot h \in hs \cdot$ 20a. $\text{xtr_lis}(h) \subseteq \text{xtr_lis}(\delta)$ 20b. $\wedge \text{attr_}\Sigma(h) \in \text{attr_}\Omega(h)$ 20 **end****value**21 $\text{xtr_lis}: H \rightarrow LI\text{-set}$ 21 $\text{xtr_lis}(h) \equiv$ 21 $\{li \mid li:LI, (li', li''): LI \times LI \cdot$ 21 $(li', li'') \in \text{attr_H}\Sigma(h) \wedge li \in \{li', li''\}\}$

Then links.

22 Links have lengths.

23 Links have geodetic location.

24 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

22 LEN

23 GeoL

24a. $L\Sigma = (HI \times HI)\text{-set}$ 24b. $L\Omega = L\Sigma\text{-set}$ **value**22 **attr_LEN**: $L \rightarrow \text{LEN}$ 23 **attr_GeoL**: $L \rightarrow \text{GeoL}$ 24a. **attr_LΣ**: $L \rightarrow L\Sigma$ 24b. **attr_LΩ**: $L \rightarrow L\Omega$ **axiom**24 $\forall n:N \cdot$ 24 **let** $ls = \text{xtr_links}(n)$, $hs = \text{xtr_hubs}(n)$ **in**24 $\forall l:L \cdot l \in ls \Rightarrow$ 24a. **let** $l\sigma = \text{attr_L}\Sigma(l)$ **in**24a. $0 \leq \text{card } l\sigma \leq 4$ 24a. $\wedge \forall (hi', hi''):(HI \times HI) \cdot (hi', hi'') \in l\sigma$ 24a. $\Rightarrow \{hi', hi''\} = \text{obs_mereo_L}(l)$ 24b. $\wedge \text{attr_L}\Sigma(l) \in \text{attr_L}\Omega(l)$ 24 **end end**

Then vehicles.

25 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

25 $VPos = onL \mid atH$

25a. $onL :: LI \ HI \ HI \ R$

25b. $R = \mathbf{Real} \quad \mathbf{axiom} \ \forall r:R \cdot 0 \leq r \leq 1$

25c. $atH :: HI \ LI \ LI$

value

25 $\mathbf{attr_VPos}: V \rightarrow VPos$

axiom

25a. $\forall n:N, onL(li, fhi, thi, r):VPos \cdot$

25a. $\exists l:L \cdot l \in \mathbf{obs_part_LS}(\mathbf{obs_part_N}(n))$

25a. $\Rightarrow li = \mathbf{uid_L}(l) \wedge \{fhi, thi\} = \mathbf{obs_mereo_L}(l),$

25c. $\forall n:N, atH(hi, fli, tli):VPos \cdot$

25c. $\exists h:H \cdot h \in \mathbf{obs_part_HS}(\mathbf{obs_part_N}(n))$

25c. $\Rightarrow hi = \mathbf{uid_H}(h) \wedge (fli, tli) \in \mathbf{attr_L\Sigma}(h)$

26 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 !

27 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

26b. $\text{MAP} = \text{VI} \xrightarrow{m'} \text{VPos}$

axiom

26b. $\forall \text{map}:\text{MAP} \cdot \text{card dom map} = \text{card rng map}$

value

26 distribute: $\text{VS} \rightarrow \text{N} \rightarrow \text{MAP}$

26 distribute(vs)(n) \equiv

26a. **let** (hs,ls) = (xtr_hubs(n),xtr_links(n)) **in**

26a. **let** vps = {onL(**uid**_(l),fhi,thi,r) |

26a. $l:L \cdot l \in \text{ls} \wedge \{fhi,thi\}$

26a. $\subseteq \text{obs_mereo_L}(l) \wedge 0 \leq r \leq 1 \}$

26a. $\cup \{ \text{atH}(\text{uid_H}(h),fli,tli) |$

26a. $h:H \cdot h \in \text{hs} \wedge \{fli,tli\}$

26a. $\subseteq \text{obs_mereo_H}(h) \}$ **in**

26b. $[\text{uid_V}(v) \mapsto vp | v:V, vp:\text{VPos} \cdot v \in \text{vs} \wedge vp \in \text{vps}]$

26 **end end**

And finally monitors. We consider only one monitor attribute.

28 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.

type

28 Traffic = $VI \xrightarrow{m} (T \times VPos)^*$

value

28 **attr**_Traffic: $M \rightarrow \text{Traffic}$

axiom

28b. $\forall \delta:\Delta \cdot$

28b. **let** m = **obs_part**_M(δ) **in**

28b. **let** tf = **attr**_Traffic(m) **in**

28b. **dom** tf $\subseteq \text{xtr_vis}(\delta) \wedge$

28b. $\forall vi:VI \cdot vi \in \text{dom tf} \cdot$

28b. **let** tr = tf(vi) **in**

28b. $\forall i,i+1:\text{Nat} \cdot \{i,i+1\} \subseteq \text{dom tr} \cdot$

28b. **let** (t,vp)=tr(i),(t',vp')=tr(i+1) **in**

28b. $t < t'$

28(b.)i \wedge **case** (vp,vp') **of**

28(b.)i (onL(li,fhi,thi,r),onL(li',fhi',thi',r'))

28(b.)i $\rightarrow li=li' \wedge fhi=fhi' \wedge thi=thi' \wedge r \leq r' \wedge li \in \text{xtr_lis}(\delta) \wedge \{fhi,thi\} = \text{obs_mereo_L}(\text{get_link}(li)(\delta)),$

28(b.)ii (atH(hi,fli,tli),atH(hi',fli',tli'))

28(b.)ii $\rightarrow hi=hi' \wedge fli=fli' \wedge tli=tli' \wedge hi \in \text{xtr_his}(\delta) \wedge (fli,tli) \in \text{obs_mereo_H}(\text{get_hub}(hi)(\delta)),$

28(b.)iii (onL(li,fhi,thi,1),atH(hi,fli,tli))

28(b.)iii $\rightarrow li=fli \wedge thi=hi \wedge \{li,tli\} \subseteq \text{xtr_lis}(\delta) \wedge \{fhi,thi\} = \text{obs_mereo_L}(\text{get_link}(li)(\delta))$

28(b.)iii $\wedge hi \in \text{xtr_his}(\delta) \wedge (fli,tli) \in \text{obs_mereo_H}(\text{get_hub}(hi)(\delta)),$

28(b.)iv (atH(hi,fli,tli),onL(li',fhi',thi',0))

28(b.)iv $\rightarrow \text{etcetera},$

28b. **—** $\rightarrow \text{false}$

28b. **end end end end end**

2.2. Perdurants

- Our presentation of example perdurants is not as systematic as that of example endurants.
- Give the simple basis of endurants covered above there is now a huge variety of perdurants, so we just select one example from each of the three classes of perdurants (as outline in [Bjø16b]):
 - ❖ a simple hub insertion action (Sect.),
 - ❖ a simple link disappearance event (Sect.) and
 - ❖ a not quite so simple behaviour, that of road traffic (Sect.).

2.2.1. Hub Insertion Action

29 Initially inserted hubs, h , are characterised

- a. by their unique identifier which not one of any hub in the net, n , into which the hub is being inserted,
- b. by a mereology, $\{\}$, of zero link identifiers, and
- c. by — whatever — attributes, $attrs$, are needed.

30 The result of such a hub insertion is a net, n' ,

- a. whose links are those of n , and
- b. whose hubs are those of n augmented with h .

value

29 $\text{insert_hub}: H \rightarrow N \rightarrow N$

30 $\text{insert_hub}(h)(n) \text{ as } n'$

29a. **pre:** $\text{uid_H}(h) \notin \text{xtr_his}(n)$

29b. $\wedge \text{obs_mereo_H} = \{\}$

29c. $\wedge \dots$

30a. **post:** $\text{obs_part_Ls}(n) = \text{obs_part_Ls}(n')$

30b. $\wedge \text{obs_part_Hs}(n) \cup \{h\} = \text{obs_part_Hs}(n')$

2.2.2. Link Disappearance Event

We formalise aspects of the link disappearance event:

- 31 The result net, $n':N'$, is not well-formed.
- 32 For a link to disappear there must be at least one link in the net;
- 33 and such a link may disappear such that
- 34 it together with the resulting net makes up for the “original” net.

value

31 **link_diss_event**: $N \times N' \times \mathbf{Bool}$

31 **link_diss_event**(n, n') **as** **tf**

32 **pre**: **obs_part_Ls**(**obs_part_LS**(n)) $\neq \{\}$

33 **post**: $\exists l:L.l \in \mathbf{obs_part_Ls}(\mathbf{obs_part_LS}(n)) \Rightarrow$

34 $l \notin \mathbf{obs_part_Ls}(\mathbf{obs_part_LS}(n'))$

34 $\wedge n' \cup \{l\} = \mathbf{obs_part_Ls}(\mathbf{obs_part_LS}(n))$

2.2.3. Road Traffic

- The analysis & description of the road traffic behaviour is composed
 - ⋄ (i) from the description of the global values of
 - ⊗ nets, links and hubs,
 - ⊗ vehicles,
 - ⊗ monitor,
 - ⊗ a clock, and
 - ⊗ an initial distribution, map, of vehicles, “across” the net;
 - ⋄ (ii) from the description of channels
 - ⊗ between vehicles and
 - ⊗ the monitor;

- ❖ (iii) from the description of behaviour signatures, that is, those of
 - ⊗ the overall road traffic system,
 - ⊗ the vehicles, and
 - ⊗ the monitor; and
- ❖ (iv) from the description of the individual behaviours, that is,
 - ⊗ the overall road traffic system, rts,
 - ⊗ the individual vehicles, veh, and
 - ⊗ the monitor, mon.

2.2.3.1 Global Values:

- There is given some globally observable parts.

35 besides the domain, $\delta:\Delta$,

36 a net, $n:N$,

37 a set of vehicles, $vs:V\text{-set}$,

38 a monitor, $m:M$, and

39 a clock, $clock$, behaviour.

40 From the net and vehicles we generate an initial distribution of positions of vehicles.

- The $n:N$, $vs:V\text{-set}$ and $m:M$ are observable from any road traffic system domain δ .

value

```

35   $\delta:\Delta$ 
36   $n:N = \mathbf{obs\_part\_N}(\delta),$ 
36   $ls:L\text{-set} = \mathbf{links}(\delta), hs:H\text{-set} = \mathbf{hubs}(\delta),$ 
36   $lis:LI\text{-set} = \mathbf{xtr\_lis}(\delta), his:HI\text{-set} = \mathbf{xtr\_his}(\delta)$ 
37   $va:VS = \mathbf{obs\_part\_VS}(\mathbf{obs\_part\_F}(\delta)),$ 
37   $vs:Vs\text{-set} = \mathbf{obs\_part\_Vs}(va),$ 
37   $vis:VI\text{-set} = \{\mathbf{uid\_VI}(v) \mid v:V \cdot v \in vs\},$ 
38   $m:\mathbf{obs\_part\_M}(\delta),$ 
38   $mi = \mathbf{uid\_MI}(m),$ 
38   $ma:\mathbf{attributes}(m)$ 
39   $\mathbf{clock}: \mathbb{T} \rightarrow \mathbf{out} \{\mathbf{clk\_ch}[vi \mid vi:VI \cdot vi \in vis]\} \quad \mathbf{Unit}$ 
40   $vm:MAP \cdot vpos\_map = \mathbf{distribute}(vs)(n);$ 

```

2.2.3.2 Channels:

41 We additionally declare a set of vehicle-to-monitor-channels indexed

- a. by the unique identifiers of vehicles
- b. and the (single) monitor identifier.⁵

and communicating vehicle positions.

channel

41 $\{v_m_ch[vi,mi] \mid vi:VI \cdot vi \in vis\}:VPos$

⁵Technically speaking: we could omit the monitor identifier.

2.2.3.3 Behaviour Signatures:

42 The road traffic system behaviour, `rts`, takes no arguments; and “behaves”, that is, continues forever.

43 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__attrs(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”.

44 The monitor 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

42 $\text{rts: Unit} \rightarrow \text{Unit}$

43 $\text{veh: vi:VI} \times \text{mi:MI} \rightarrow \text{vp:VPos} \rightarrow \text{out vm_ch[vi,mi]} \text{ Unit}$

44 $\text{mon: m:M} \times \text{vis:VI-set} \rightarrow \text{RTF} \rightarrow \text{in } \{\text{v_m_ch[vi,mi]} \mid \text{vi:VI} \cdot \text{vi} \in \text{vis}\}, \text{c}$

2.2.3.4 The Road Traffic System Behaviour:

45 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 monitor behaviour.

value

45 **rts()** =

45a. $\parallel \{ \text{veh}(\mathbf{uid_VI}(v), \text{mi})(\text{vm}(\mathbf{uid_VI}(v))) \mid v:V \cdot v \in \mathbf{vs} \}$

45b. $\parallel \text{mon}(\text{mi}, \mathbf{vis})([v_i \mapsto \langle \rangle \mid v_i:VI \cdot v_i \in \mathbf{vis}])$

- where, wrt, the monitor, we
 - ⋄ dispense with the mereology and the attribute state arguments
 - ⋄ and instead just have a monitor traffic argument which
 - ⊗ records the discrete road traffic, MAP,
 - ⊗ initially set to “empty” traces ($\langle \rangle$, of so far “no road traffic”!).
- In order for the monitor behaviour to assess the vehicle positions
 - ⋄ these vehicles communicate their positions
 - ⋄ to the monitor
 - ⋄ via a vehicle to monitor channel.
- In order for the monitor to time-stamp these positions
 - ⋄ it must be able to “read” a clock.

46 We describe here an abstraction of the vehicle behaviour **at** a Hub (hi).

- a. Either the vehicle remains at that hub informing the monitor of its position,
- b. or, internally non-deterministically,
 - i moves onto a link, tli, whose “next” hub, identified by thi, is obtained from the mereology of the link identified by tli;
 - ii informs the monitor, on channel $vm[vi,mi]$, that it is now at the very beginning (0) of the link identified by tli, whereupon the vehicle resumes the vehicle behaviour positioned at the very beginning of that link,
- c. or, again internally non-deterministically, the vehicle “disappears — off the radar” !

```
46  veh(vi,mi)(vp:atH(hi,fli,tli)) ≡  
46a.      v_m_ch[vi,mi]!vp ; veh(vi,mi)(vp)  
46b.      □  
46(b.)i    let {hi',thi}=obs_mereo_L(get_link(tli)(n)) in  
46(b.)i      assert: hi'=hi  
46(b.)ii    v_m_ch[vi,mi]!onL(tli,hi,thi,0) ;  
46(b.)ii    veh(vi,mi)(onL(tli,hi,thi,0)) end  
46c.      □ stop
```

47 We describe here an abstraction of the vehicle behaviour **on** a Link (ii).

Either

- a. the vehicle remains at that link position informing the monitor of its position,
- b. or, internally non-deterministically, if the vehicle's position on the link has not yet reached the hub,

- i then the vehicle moves an arbitrary increment ℓ_ε (less than or equal to the distance to the hub) along the link informing the monitor of this, or

- ii else,

- A while obtaining a “next link” from the mereology of the hub (where that next link could very well be the same as the link the vehicle is about to leave),

- B the vehicle informs the monitor that it is now at the hub identified by th_i , whereupon the vehicle resumes the vehicle behaviour positioned at that hub.

- c. or, internally non-deterministically, the vehicle “disappears — off the radar” !

```

47  veh(vi,mi)(vp:onL(li,fhi,thi,r)) ≡
47a.    v_m_ch[vi,mi]!vp ; veh(vi,mi,va)(vp)
47b.    □ if  $r + \ell_\varepsilon \leq 1$ 
47(b.)i      then
47(b.)i      v_m_ch[vi,mi]!onL(li,fhi,thi, $r + \ell_\varepsilon$ ) ;
47(b.)i      veh(vi,mi)(onL(li,fhi,thi, $r + \ell_\varepsilon$ ))
47(b.)ii     else
47(b.)iiA     let  $li':L \cdot li' \in \mathbf{obs\_mereo\_H}(\mathbf{get\_hub(thi)}(n))$  in
47(b.)iiB     v_m_ch[vi,mi]!atH(li,thi,li');
47(b.)iiB     veh(vi,mi)(atH(li,thi,li')) end end
47c.    □ stop

```

The Monitor Behaviour

48 The monitor behaviour evolves around

- a. the monitor identifier,
- b. the monitor mereology,
- c. and the attributes, `ma:ATTR`
- d. — where we have factored out as a separate arguments — a table of traces of time-stamped vehicle positions,
- e. while accepting messages
 - i about time
 - ii and about vehicle positions
- f. and otherwise progressing “in[de]finitely”.

49 Either the monitor “does own work”

50 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.

```

48  mon(mi,vis)(trf) ≡
49      mon(mi,vis)(trf)
50      □
50a.    □ {let tvp = (clk_ch?,v_m_ch[vi,mi]?) in
50b.      let trf' = trf † [vi ↦ trf(vi)^(tvp)] in
50c.      mon(mi,vis)(trf')
50d.      end end | vi:VI · vi ∈ vis}

```

- We are about to complete a long, i.e., a 50 slide example (!).
- We can now comment on the full example:
 - ⋄ The domain, $\delta : \Delta$ is a manifest part.
 - ⋄ The road net, $n : N$ is also a manifest part.
 - ⋄ The fleet, $f : F$, of vehicles, $vs : VS$, likewise, is a manifest part.
 - ⋄ But the monitor, $m : M$, is a concept.

- ⊗ One does not have to think of it as a manifest “observer”.
- ⊗ The vehicles are on — or off — the road (i.e., links and hubs).
- ⊗ We know that from a few observations and generalise to all vehicles.
- ⊗ They either move or stand still. We also, similarly, know that.
- ⊗ Vehicles move. Yes, we know that.
- ⊗ Based on all these repeated observations and generalisations we introduce the concept of vehicle traffic.
- ⊗ Unless positioned high above a road net — and with good binoculars — a single person cannot really observe the traffic.
- ⊗ There are simply too many links, hubs, vehicles, vehicle positions and times.
- ❖ Thus we conclude that, even in a richly manifest domain, we can also “speak of”, that is, describe concepts over manifest phenomena, including time !

2.3. Domain Facets

- The example of this section focuses on the *domain facet* [Bjø10a, 2008] of *intrinsic*s.
- It does not reflect the other *domain facets*:
 - ❖ domain support technologies,
 - ❖ domain rules, regulations & scripts,
 - ❖ organisation & management, and
 - ❖ human behaviour.

- The requirements examples, i.e., the rest of this paper, thus builds only on the *domain intrinsics*.
 - ❖ This means that we shall not be able to cover principles, technique and tools for the prescription of such important requirements that handle failures of support technology or humans.
 - ❖ We shall, however point out where we think such, for example, fault tolerance requirements prescriptions “fit in” and refer to relevant publications for their handling.

3. Requirements

- This and the next three sections, Sects. 3., 4. and 5., are the main sections of these lectures' coverage of requirements.
 - ❖ Section 4. is the most detailed and systematic section.
 - ❖ It covers the domain requirements operations of
 - ⊗ *projection*,
 - ⊗ *instantiation*,
 - ⊗ *determination*,
 - ⊗ *extension* and, less detailed,
 - ⊗ *fitting*.

- ❖ Section 5. surveys the interface requirements issues of shared phenomena:
 - ⊗ *shared endurants*,
 - ⊗ *shared actions*,
 - ⊗ *shared events* and
 - ⊗ *shared behaviour*,and “completes” the exemplification of the detailed domain extension of our requirements into a road pricing system.
- ❖ Section 5. also covers the notion of *derived requirements*.
- This, the initial, section captures main concepts and principles of requirements.


- Sections 3., 4., and 5. covers *initial requirements*.
 - ⋄ By ***initial requirements*** we shall, “operationally” speaking, understand the requirements that are derived from the general principles outlined in these sections [redacted]
 - ⋄ In contrast to these are the further requirements that are typically derived
 - ⊗ either from the *domain facet descriptions* of *intrinsic*, the *support technology*, the *rules & regulations*, the *organisation & management*, and the *human behaviour facets* [Bjø10a] — not covered in this paper,
 - ⊗ (and/)or by more conventional means [DvLF93, Jac01, ZH04, Lau02, JHJ07, OD08, van09].

Definition 1 *Requirements (I)*: By a *requirements* we understand (cf., IEEE Standard 610.12):

- “A condition or capability needed by a user to solve a problem or achieve an objective” ■■■
- The objective of requirements engineering is to create a *requirements prescription*:
 - ❖ A *requirements prescription* specifies observable properties of endurants and perdurants of *the machine* such as the requirements stake-holders wish them to be ■■■
 - ❖ The *machine* is what is required: that is, the hardware and software that is to be designed and which are to satisfy the requirements ■■■

- A requirements prescription thus (putatively) expresses what there should be.
- A requirements prescription expresses nothing about the design of the possibly desired (required) software.
- But as the requirements prescription is presented in the form of a model, one can base the design on that model.
- We shall show how a major part of a requirements prescription can be “derived” from “its” prerequisite domain description.

Rule 1. The “Golden Rule” of Requirements Engineering:

Prescribe only those requirements that can be objectively shown to hold for the designed software ⁶

- “Objectively shown” means that the designed software can
 - ❖ either be tested,
 - ❖ or be model checked,
 - ❖ or be proved (verified),to satisfy the requirements.
- Caveat⁷

⁶  marks the end of a rule.

⁷Will not be illustrated !

Rule 2. An “Ideal Rule” of Requirements Engineering: When prescribing (including formalising) requirements, also formulate tests and properties for model checking and theorems whose proof should show adherence to the requirements ■■■

- The rule is labelled “ideal” since such precautions will not be shown in this seminar.
- The rule is clear.
- It is a question for proper management to see that it is adhered to.
- See the “Caveat” above !

Rule 3. Requirements Adequacy: Make sure that requirements cover what users expect 

- That is,
 - ❖ do not express a requirement for which you have no users,
 - ❖ but make sure that all users' requirements are represented or somehow accommodated.
- In other words:
 - ❖ the requirements gathering process needs to be like an extremely “fine-meshed net”:
 - ❖ One must make sure that all possible stake-holders have been involved in the requirements acquisition process,
 - ❖ and that possible conflicts and other inconsistencies have been obviated.

Rule 4. Requirements Implementability: Make sure that requirements are implementable ■■■


- That is, do not express a requirement for which you have no assurance that it can be implemented.
- In other words,
 - ❖ although the requirements phase is not a design phase,
 - ❖ one must tacitly assume, perhaps even indicate, somehow, that an implementation is possible.
- But the requirements in and by themselves, may stay short of expressing such designs.
- Caveat !

Definition 2: **Requirements (II):**

- By **requirements** we shall understand
 - ⋄ a document
 - ⋄ which prescribes desired properties of
 - ⋄ a machine:
 - ⊗ what endurants the machine shall “maintain”, and
 - ⊗ what the machine shall (must; not should) offer of
 - * functions and of
 - * behaviours
 - ⊗ while also expressing which events the machine shall “handle”



- 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 3: 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 4: 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.

-
- Whereas domain descriptions may describe phenomena that cannot be computed,
 - requirements prescriptions must describe computable phenomena.

3.1. Some Requirements Aspects

- We shall unravel requirements in two stages —
 - ⋄ (i) the first stage is sketchy (and thus informal)
 - ⋄ (ii) while the last stage is systematic and both informal and formal.
 - ⋄ The sketchy stage consists of
 - ⊗ a narrative *problem/objective sketch*,
 - ⊗ a narrative *system requirements sketch*, and
 - ⊗ a narrative *user & external equipment requirements sketch*.


- The narrative and formal stage
 - ✧ consists of
 - ⊗ *design assumptions* and
 - ⊗ *design requirements*.
 - ✧ It is systematic, and mandates
 - ⊗ both strict narrative
 - ⊗ and formal prescriptions.
 - ✧ And it is “derivable” from the domain description.
- In a sense stage (i) is made superfluous once stage (ii) has been completed.
- The formal, engineering design work is to based on stage (ii).

- The purpose of the two stages (i–ii) is twofold:
 - ❖ to gently lead the requirements engineer and the reader into the requirements problems
 - ❖ while leading the requirements engineer and reader to focus on the very requirements essentials.

3.1.1. Requirements Sketches

3.1.1.1 Problem, Solution and Objective Sketch

Definition 5 *Problem, Solution and Objective Sketch*: By a problem, solution and objective sketch we understand


- a narrative which emphasises
- what the *problem* to be solved is,
- outlines a possible *solution*
- and sketches an *objective* of the solution 

Example 1 The Problem/Objective Requirements: A Sketch:

- The *problem* is that of traffic congestion.
- The chosen *solution* is to [build and] operate a toll-road system integrated into a road net and charge toll-road users a usage fee.
- The *objective* is therefore to create a **road-pricing product**.
 - ◇ By a road-pricing product
 - ⊗ we shall understand an IT-based system
 - ⊗ containing C&C equipment and software
 - ⊗ that enables the recording of *vehicle* movements
 - ⊗ within the *toll-road*
 - ⊗ and thus enables
 - * the *owner* of the road net to charge
 - * the *owner* of the vehicles
 - * *fees* for the usage of that toll-road

3.1.1.2 Systems Requirements

Definition 6 ***System Requirements:*** By a ***system requirements narrative*** we understand

- a narrative which emphasises
- the overall assumed and/or required
- hardware and software system equipment 

Example 2 The Road-pricing System Requirements: A Narrative:


- The requirements are based on the following constellation of system equipment:
 - ⋄ there is assumed a GNSS:
a GLOBAL NAVIGATION SATELLITE SYSTEM;
 - ⋄ there are *vehicles* equipped with GNSS receivers;
 - ⋄ there is a well-delineated road net called a *toll-road* net with specially equipped *toll-gates* with
 - ⊗ *vehicle identification sensors*,
 - ⊗ *exit barriers* which afford (only specially equipped) vehicles to exit⁸ from the toll-road net;
 - and
 - ⋄ there is a *road-pricing calculator*.

⁸We omit consideration of entry barriers.

- **The system to be designed (from the requirements) is the *road-pricing calculator*.**
- These four system elements are required to behave and interact as follows:
 - ❖ The GNSS is assumed to continuously offer vehicles information about their global position;
 - ❖ *vehicles* shall contain a `GNSS receiver` which based on the global position information shall regularly calculate their timed local position and offer this to the *calculator* — while otherwise cruising the general road net as well as the toll-road net, the latter while carefully moving through toll-gates;


- ❖ *toll-gates* shall register the identity of vehicles passing 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.

- The requirements are therefore to include **assumptions about**
 - ❖ the GNSS satellite and telecommunications equipment,
 - ❖ the vehicle GNSS receiver equipment,
 - ❖ the vehicle handling of GNSS input and forwarding, to the road pricing system, of its interpretation of GNSS input,
 - ❖ the toll-gate sensor equipment,
 - ❖ the toll-gate barrier equipment,
 - ❖ the toll-gate handling of entry, vehicle identification and exit sensors and the forwarding of vehicle identification to the road pricing calculator, and
 - ❖ the communications between toll-gates and vehicles, on “one side”, and the road pricing calculator, on the “other side”.

- It is in this sense that the requirements are for an information technology-based system
 - ◊ of both software and
 - ◊ hardware —
 - ⊗ not just hard computer and communications equipment,
 - ⊗ but also movement sensors
 - ⊗ and electro-mechanical “gear” 

3.1.1.3 User and External Equipment Requirements


Definition 7 *User and External Equipment Requirements:* By a ***user and external equipment requirements narrative*** we understand

- a narrative which emphasises assumptions about
 - ❖ the human user and
 - ❖ external equipment interfaces
- to the system components 
- The user and external equipment requirements
 - ❖ detail, and thus make explicit,
 - ❖ the assumptions listed in Example 2.

Example 3 The Road-pricing User and External Equipment

Requirements: Narrative:


- The human users of the road-pricing system are:
 - ⋄ *vehicle drivers,*
 - ⋄ *toll-gate sensor, actuator and barrier service staff, and*
 - ⋄ *the road-pricing calculator service staff.*
- The external equipment are:
 - ⋄ firstly, the GNSS satellites and the telecommunications equipment which enables *communication* between
 - ⊗ the GNSS satellites and vehicles,
 - ⊗ vehicles and the road-pricing calculator and
 - ⊗ toll-gates and the road-pricing calculator.

- ❖ Moreover, the external *equipment* are
 - ⊗ the toll-gates with their sensors:
 - * entry,
 - * vehicle identity, and
 - * exit,
 - and the barrier actuator.
- ❖ The external *equipment* are, finally, the vehicles ! 
- That is,
 - ❖ although we do indeed exemplify domain and requirements aspects of users and external equipment,
 - ❖ we do not expect to machine, i.e., to hardware or software design these elements;
 - ❖ *they are assumed already implemented !*

3.1.2. The Narrative and Formal Requirements Stage

3.1.2.1 Assumption and Design Requirements

Definition 8 *Assumption and Design Requirements:*

- By *assumption and design requirements* we understand precise prescriptions of
 - ❖ the endurants
 - ❖ and perdurantsof the (to be designed) system components
- and the assumptions which that design must rely upon 


- The specification principles, techniques and tools of expressing
- *design* and
- *assumptions*, upon which the design can be relied,
- will be covered and exemplified, extensively,
- in Sects. 4.–5.

3.2. The Three Phases of Requirements Engineering

- There are, as we see it, three kinds of design assumptions and requirements:
 - ❖ *domain requirements*,
 - ❖ *interface requirements* and
 - ❖ *machine requirements*.

- **Domain requirements** are those requirements which can be expressed solely using 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, in principle, can be expressed solely using terms of the machine ■■■

Definition 9 *Verification Paradigm:*

- Some preliminary designations:
 - ✧ let \mathcal{D} designate the the domain description;
 - ✧ let \mathcal{R} designate the requirements prescription, and
 - ✧ let \mathcal{S} designate the system design.
- Now $\mathcal{D}, \mathcal{S} \models \mathcal{R}$ shall be read:
 - ✧ it must be verified that the \mathcal{S} ystem design
 - ✧ satisfies the \mathcal{R} equirements prescription
 - ✧ in the context of the \mathcal{D} omain description 

- The “in the context of \mathcal{D} ...” term means that
 - ✧ proofs of \mathcal{S} oftware design correctness
 - ✧ with respect to \mathcal{R} equirements
 - ✧ will often have to refer to \mathcal{D} omain requirements assumptions.
- We refer to [GGJZ00, Gunter, Jackson and Zave, 2000] for an analysis of a varieties of forms in which \models relate to variants of \mathcal{D} , \mathcal{R} and \mathcal{S} .

3.3. Order of Presentation of Requirements Prescriptions

- The *domain requirements development* stage — as we shall see — can be sub-staged into:

- ◊ *projection*,
- ◊ *instantiation*,
- ◊ *determination*,
- ◊ *extension* and
- ◊ *fitting*.

- The *interface requirements development* stage — can be sub-staged into shared:


- ◊ *endurant*,
- ◊ *action*,
- ◊ *event* and
- ◊ *behaviour*


developments, where “sharedness” pertains to phenomena shared between, i.e., “present” in, both the domain (concretely, manifestly) and the machine (abstractly, conceptually).

- These development stages need not be pursued in the order of the three stages and their sub-stages.
- We emphasize that
 - ⋄ one thing is the stages and steps of development, as for example these:
 - ⊗ projection, instantiation, determination, extension, fitting,
 - ⊗ shared endurants, shared actions, shared events, shared behaviours,
 - ⊗ etcetera,
 - ⋄ another thing is the requirements prescription that results from these development stages and steps.
 - ⊗ The further software development,
 - ⊗ after and on the basis of the requirements prescription
 - ⊗ starts only when all stages and steps of the requirements prescription have been fully developed.

- The domain engineer is now free to rearrange the final prescription,
 - ⋄ irrespective of the order in which the various sections were developed,
 - ⋄ in such a way as to give a most
 - ⊗ pleasing,
 - ⊗ pedagogic and
 - ⊗ cohesivereading (i.e., presentation).
- From such a requirements prescription one can therefore not necessarily see in which order the various sections of the prescription were developed.

3.4. Design Requirements and Design Assumptions


- A crucial distinction is between *design requirements* and *design assumptions*.
 - ❖ The ***design requirements***
 - ⊗ are those requirements for which
 - ⊗ the system designer **has to** implement
 - ⊗ hardware or software
 - ⊗ in order satisfy system user expectations 

- ❖ The ***design assumptions***
 - ⊗ are those requirements for which
 - ⊗ the system designer
 - does not** have to implement hardware or software,
 - ⊗ but whose properties
 - ⊗ the designed hardware, respectively software relies on for proper functioning 


Example 4 Road Pricing System — Design Requirements:

- The design requirements for the road pricing calculator of these lectures are for the design
 - ❖ of that part of the vehicle software which interfaces the GNSS receiver and the road pricing calculator (cf. Items 129–132),
 - ❖ of that part of the toll-gate software which interfaces the toll-gate and the road pricing calculator (cf. Items 137–139) and
 - ❖ of the road pricing calculator (cf. Items 168–181) ■■■


Example 5 Road Pricing System — Design Assumptions:

- The design assumptions for the road pricing calculator include:
 - ❖ that *vehicles* behave as prescribed in Items 128–132,
 - ❖ that the GNSS regularly offers vehicles correct information as to their global position (cf. Item 129),
 - ❖ that *toll-gates* behave as prescribed in Items 134–139, and
 - ❖ that the *road net* is formed and well-formed as defined in Examples 10–12 

Example 6 Toll-Gate System — Design Requirements:

- The design requirements for the toll-gate system of these lectures are for the design of
 - ❖ software for the toll-gate
 - ❖ and its interfaces to the road pricing system,
 - ❖ i.e., Items 133–134 

Example 7 Toll-Gate System — Design Assumptions:

- The design assumptions for the toll-gate system include
 - ❖ that the vehicles behave as per Items 128–132, and
 - ❖ that the road pricing calculator behave as per Items 168–181 

3.5. Derived Requirements

- In building up the domain, interface and machine requirements a number of machine concepts are introduced.
 - ❖ These machine concepts enable the expression of additional requirements.
 - ❖ It is these we refer to as derived requirements.
 - ❖ Techniques and tools espoused in such classical publications as [DvLF93, Jac01, ZH04, Lau02, van09] can in those cases be used to advantage.

4. Domain Requirements

- Domain requirements primarily express the assumptions
 - ❖ that a design must rely upon
 - ❖ in order that that design can be verified.
- Although domain requirements firstly express assumptions
 - ❖ it appears that the software designer is well-advised
 - ❖ in also implementing, as data structures and procedures,
 - ❖ the endurants, respectively perdurants
 - ❖ expressed in the domain requirements prescriptions.
- Whereas
 - ❖ domain endurants are “real-life” phenomena
 - ❖ they are now, in domain requirements prescriptions,
 - ❖ abstract concepts (to be represented by a machine).

Definition 10 *Domain Requirements Prescription:* A *domain requirements prescription*

- is that subset of the requirements prescription
- whose technical terms are defined in a domain description 

- To determine a relevant subset all we need is collaboration with requirements, cum domain stake-holders.
- Experimental evidence,
 - ⋄ in the form of example developments
 - ⊗ of requirements prescriptions
 - ⊗ from domain descriptions,appears to show
 - ⋄ that one can formulate techniques for such developments
 - ⋄ around a few domain-description-to-requirements-prescription operations.
 - ⋄ We suggest these:
 - ⊗ *projection*,
 - ⊗ *instantiation*,
 - ⊗ *determination*,
 - ⊗ *extension* and
 - ⊗ *fitting*.

- In Sect. 3.3

- ❖ we mentioned that the order in which one performs
- ❖ these description-to-prescription operations
- ❖ is not necessarily the order in which we have listed them here,
- ❖ but, with notable exceptions, one is well-served in starting out requirements development
- ❖ by following this order.

4.1. Domain Projection

Definition 11 *Domain Projection*: By a *domain projection* we mean

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

⋄ *endurants*:

- ⊗ *parts,*
- ⊗ *materials and*
- ⊗ *components,*
- as well as*

⋄ *perdurants*:

- ⊗ *actions,*
- ⊗ *events and*
- ⊗ *behaviours*

that the stake-holders do not wish represented or relied upon by the machine 

- The resulting document is a *partial domain requirements prescription*.
- In determining an appropriate subset
 - ⋄ the requirements engineer must secure
 - ⋄ that the final “projection prescription”
 - ⋄ is complete and consistent — that is,
 - ⊗ that there are no “dangling references”,
 - ⊗ i.e., that all entities
 - ⊗ and their internal properties
 - ⊗ that are referred to
 - ⊗ are all properly defined.

4.1.1. Domain Projection — Narrative


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

Example 8 Domain Requirements. Projection: A Narrative Sketch:

- We require that the road pricing system shall [at most] relate to the following domain entities – and only to these⁹:
 - ⋄ the net,
 - ⊗ its links and hubs,
 - ⊗ and their properties
(unique identifiers, mereologies and some attributes),
 - ⋄ the vehicles, as endurants, and
 - ⋄ the general vehicle behaviours, as perdurants.

⁹By ‘relate to ... these’ we mean that the required system does not rely on domain phenomena that have been “projected away”.

- We treat projection together with a concept of *simplification*.
- The example simplifications are
 - ❖ vehicle positions and,
 - ❖ related to the simpler vehicle position,
 - ❖ vehicle behaviours.

- To prescribe and formalise this we copy the domain description.
- From that domain description we remove all mention of
 - ◊ the hub insertion action,
 - ◊ the link disappearance event, and
 - ◊ the monitor 
- As a result we obtain $\Delta_{\mathcal{P}}$, the projected version of the domain requirements prescription¹⁰.

¹⁰Restrictions of the net to the toll road nets, hinted at earlier, will follow in the next domain requirements steps.

4.1.2. Domain Projection — Formalisation

- The requirements prescription hinges, crucially,
 - ✧ not only on a systematic narrative of all the
 - ⊗ projected, ⊗ determinated, ⊗ fitted
 - ⊗ instantiated, ⊗ extended and
 - specifications,
 - ✧ but also on their formalisation.
- In the formal domain projection example we, regretfully, omit the narrative texts.
 - ✧ In bringing the formal texts we keep the item numbering from Sect. 2.,
 - ✧ where you can find the associated narrative texts.

Example 9 Domain Requirements — Projection:

Main Sorts

type

1 $\Delta_{\mathcal{P}}$

1a. $N_{\mathcal{P}}$

1b. $F_{\mathcal{P}}$

value

1a. **obs_part** $N_{\mathcal{P}}: \Delta_{\mathcal{P}} \rightarrow N_{\mathcal{P}}$

1b. **obs_part** $F_{\mathcal{P}}: \Delta_{\mathcal{P}} \rightarrow F_{\mathcal{P}}$

type

2a. $HA_{\mathcal{P}}$

2b. $LA_{\mathcal{P}}$

value

2a. **obs_part** $HA: N_{\mathcal{P}} \rightarrow HA$

2b. **obs_part** $LA: N_{\mathcal{P}} \rightarrow LA$

Concrete Types

type

3 $H_{\mathcal{P}}, HS_{\mathcal{P}} = H_{\mathcal{P}}\text{-set}$

4 $L_{\mathcal{P}}, LS_{\mathcal{P}} = L_{\mathcal{P}}\text{-set}$

5 $V_{\mathcal{P}}, VS_{\mathcal{P}} = V_{\mathcal{P}}\text{-set}$

value

3 **obs_part_HS** $_{\mathcal{P}}: HA_{\mathcal{P}} \rightarrow HS_{\mathcal{P}}$

4 **obs_part_LS** $_{\mathcal{P}}: LA_{\mathcal{P}} \rightarrow LS_{\mathcal{P}}$

5 **obs_part_VS** $_{\mathcal{P}}: F_{\mathcal{P}} \rightarrow VS_{\mathcal{P}}$

6a. **links**: $\Delta_{\mathcal{P}} \rightarrow L\text{-set}$

6a. **links**($\delta_{\mathcal{P}}$) \equiv **obs_part_LS** $_{\mathcal{R}}(\text{obs_part_LA}_{\mathcal{R}}(\delta_{\mathcal{R}}))$

6b. **hubs**: $\Delta_{\mathcal{P}} \rightarrow H\text{-set}$

6b. **hubs**($\delta_{\mathcal{P}}$) \equiv **obs_part_HS** $_{\mathcal{P}}(\text{obs_part_HA}_{\mathcal{P}}(\delta_{\mathcal{P}}))$

Unique Identifiers

type

7a. HI, LI, VI, MI

value

7c. **uid_HI**: $H_{\mathcal{D}} \rightarrow HI$

7c. **uid_LI**: $L_{\mathcal{D}} \rightarrow LI$

7c. **uid_VI**: $V_{\mathcal{D}} \rightarrow VI$

7c. **uid_MI**: $M_{\mathcal{D}} \rightarrow MI$

axiom

7b. $HI \cap LI = \emptyset, HI \cap VI = \emptyset, HI \cap MI = \emptyset,$

7b. $LI \cap VI = \emptyset, LI \cap MI = \emptyset, VI \cap MI = \emptyset$

Mereology

value

12 **obs_mereo_H** \mathcal{D} : $H_{\mathcal{D}} \rightarrow \text{LI-set}$

13 **obs_mereo_L** \mathcal{D} : $L_{\mathcal{D}} \rightarrow \text{HI-set}$

13 **axiom** $\forall l:L_{\mathcal{D}} \cdot \text{card } \text{obs_mereo_L}_{\mathcal{D}}(l)=2$

14 **obs_mereo_V** \mathcal{D} : $V_{\mathcal{D}} \rightarrow \text{MI}$

15 **obs_mereo_M** \mathcal{D} : $M_{\mathcal{D}} \rightarrow \text{VI-set}$

axiom

16 $\forall \delta_{\mathcal{D}}:\Delta_{\mathcal{D}}, \text{hs}:\text{HS} \cdot \text{hs}=\text{hubs}(\delta), \text{ls}:\text{LS} \cdot \text{ls}=\text{links}(\delta_{\mathcal{D}}) \Rightarrow$

16 $\forall h:H_{\mathcal{D}} \cdot h \in \text{hs} \Rightarrow$

16 **obs_mereo_H** $\mathcal{D}(h) \subseteq \text{xtr_his}(\delta_{\mathcal{D}}) \wedge$

17 $\forall l:L_{\mathcal{D}} \cdot l \in \text{ls} \cdot$

16 **obs_mereo_L** $\mathcal{D}(l) \subseteq \text{xtr_lis}(\delta_{\mathcal{D}}) \wedge$

18a. **let** $f:F_{\mathcal{D}} \cdot f=\text{obs_part_F}_{\mathcal{D}}(\delta_{\mathcal{D}}) \Rightarrow$

18a. $\text{vs}:V_{\mathcal{S}} \cdot \text{vs}=\text{obs_part_VS}_{\mathcal{D}}(f) \text{ in}$

18a. $\forall v:V_{\mathcal{D}} \cdot v \in \text{vs} \Rightarrow$

18a. **uid_V** $\mathcal{D}(v) \in \text{obs_mereo_M}_{\mathcal{D}}(m) \wedge$

18b. **obs_mereo_M** $\mathcal{D}(m)$

18b. $= \{\text{uid_V}_{\mathcal{D}}(v) \mid v:V \cdot v \in \text{vs}\}$

18b. **end**

Attributes: We project attributes of hubs, links and vehicles.
 First **hubs**:

type

19a. GeoH

19b. $H\Sigma_{\mathcal{P}} = (LI \times LI)\text{-sett}$

19c. $H\Omega_{\mathcal{P}} = H\Sigma_{\mathcal{P}}\text{-set}$

value

19b. **attr** $_{H\Sigma_{\mathcal{P}}}: H_{\mathcal{P}} \rightarrow H\Sigma_{\mathcal{P}}$

19c. **attr** $_{H\Omega_{\mathcal{P}}}: H_{\mathcal{P}} \rightarrow H\Omega_{\mathcal{P}}$

axiom

20 $\forall \delta_{\mathcal{P}}: \Delta_{\mathcal{P}},$

20 **let** $hs = \text{hubs}(\delta_{\mathcal{P}})$ **in**

20 $\forall h: H_{\mathcal{P}} \cdot h \in hs \cdot$

20a. $\text{xtr_lis}(h) \subseteq \text{xtr_lis}(\delta_{\mathcal{P}})$

20b. $\wedge \text{attr}_{\Sigma_{\mathcal{P}}}(h) \in \text{attr}_{\Omega_{\mathcal{P}}}(h)$

20 **end**

Then **links**:

type

23 **GeoL**

24a. $L\Sigma_{\mathcal{P}} = (HI \times HI)\text{-set}$

24b. $L\Omega_{\mathcal{P}} = L\Sigma_{\mathcal{P}}\text{-set}$

value

23 **attr_GeoL**: $L \rightarrow \text{GeoL}$

24a. **attr_L** $\Sigma_{\mathcal{P}}$: $L_{\mathcal{P}} \rightarrow L\Sigma_{\mathcal{P}}$

24b. **attr_L** $\Omega_{\mathcal{P}}$: $L_{\mathcal{P}} \rightarrow L\Omega_{\mathcal{P}}$

axiom

24a.— 24b. on Slide 47.

Finally **vehicles**:

- For ‘road pricing’ we need vehicle positions.
 - ◇ But, for “technical reasons”,
 - ◇ we must abstain from the detailed description
 - ◇ given in Items 25–25c.¹¹
- We therefore *simplify* vehicle positions.

¹¹The ‘technical reasons’ are that we assume that the *GNSS* cannot provide us with direction of vehicle movement and therefore we cannot, using only the *GNSS* provide the details of ‘offset’ along a link (*onL*) nor the “from/to link” at a hub (*atH*).

51 A simplified vehicle position designates

- a. either a link
- b. or a hub,

type

51 $\text{SVPos} = \text{SonL} \mid \text{SatH}$

51a. $\text{SonL} :: \text{LI}$

51b. $\text{SatH} :: \text{HI}$

axiom

25a.' $\forall n:N, \text{SonL}(li):\text{SVPos} \cdot$

25a.' $\exists l:L.l \in \text{obs_part_LS}(\text{obs_part_N}(n)) \Rightarrow li = \text{uid_L}(l)$

25c.' $\forall n:N, \text{SatH}(hi):\text{SVPos} \cdot$

25c.' $\exists h:H.h \in \text{obs_part_HS}(\text{obs_part_N}(n)) \Rightarrow hi = \text{uid_H}(h)$

Global Values

value

```
35   $\delta_{\mathcal{P}}:\Delta_{\mathcal{P}},$   
36   $n:N_{\mathcal{P}} = \mathbf{obs\_part\_N}_{\mathcal{P}}(\delta_{\mathcal{P}}),$   
36   $ls:L_{\mathcal{P}\text{-set}} = \mathbf{links}(\delta_{\mathcal{P}}),$   
36   $hs:H_{\mathcal{P}\text{-set}} = \mathbf{hubs}(\delta_{\mathcal{P}}),$   
36   $lis:LI\text{-set} = \mathbf{xtr\_lis}(\delta_{\mathcal{P}}),$   
36   $his:HI\text{-set} = \mathbf{xtr\_his}(\delta_{\mathcal{P}})$ 
```

Behaviour Signatures: We omit the monitor behaviour.

52 We leave the vehicle behaviours' attribute argument undefined.

type

52 ATTR

value

42 $\text{trs}_{\mathcal{P}}: \mathbf{Unit} \rightarrow \mathbf{Unit}$

43 $\text{veh}_{\mathcal{P}}: \mathbf{VI} \times \mathbf{MI} \times \mathbf{ATTR} \rightarrow \dots \mathbf{Unit}$

The System Behaviour: We omit the monitor behaviour.

value

45a. $\text{trs}_{\mathcal{P}}() = \parallel \{ \text{veh}_{\mathcal{P}}(\mathbf{uid_VI}(v), \mathbf{obs_mereo_V}(v), _) \mid v:V_{\mathcal{P}} \cdot v \in \mathbf{vs} \}$

The Vehicle Behaviour:

- Given the simplification of vehicle positions
- we *simplify* the vehicle behaviour given in Items 46–47

46' $\text{veh}(\text{vi}, \text{mi})(\text{vp}:\text{SatH}(\text{hi})) \equiv$

46a.' $\text{v_m_ch}[\text{vi}, \text{mi}]!\text{SatH}(\text{hi}) ; \text{veh}(\text{vi}, \text{mi})(\text{SatH}(\text{hi}))$

46(b.)i' $\sqcap \text{let li:L} \cdot \text{li} \in \mathbf{obs_mereo_H}(\text{get_hub}(\text{hi})(n)) \text{ in}$

46(b.)ii' $\text{v_m_ch}[\text{vi}, \text{mi}]!\text{SonL}(\text{li}) ; \text{veh}(\text{vi}, \text{mi})(\text{SonL}(\text{li})) \text{ end}$

46c.' $\sqcap \text{stop}$

47' $\text{veh}(\text{vi}, \text{mi})(\text{vp}:\text{SonL}(\text{li})) \equiv$

47a.' $\text{v_m_ch}[\text{vi}, \text{mi}]!\text{SonL}(\text{li}) ; \text{veh}(\text{vi}, \text{mi}, \text{va})(\text{SonL}(\text{li}))$

47(b.)iiA' $\sqcap \text{let hi:H} \cdot \text{hi} \in \mathbf{obs_mereo_L}(\text{get_link}(\text{li})(n)) \text{ in}$

47(b.)iiB' $\text{v_m_ch}[\text{vi}, \text{mi}]!\text{SatH}(\text{hi}) ; \text{veh}(\text{vi}, \text{mi})(\text{atH}(\text{hi})) \text{ end}$

47c.' $\sqcap \text{stop}$

- We can simplify Items 46'–47c.' further.

```

53  veh(vi,mi)(vp) ≡
54    v_m_ch[vi,mi]!vp ; veh(vi,mi,va)(vp)
55    [] case vp of
55      SatH(hi) →
56        let li:L·li ∈ obs_mereo_H(get_hub(hi)(n)) in
57        v_m_ch[vi,mi]!SonL(li) ; veh(vi,mi)(SonL(li)) end,
55      SonL(li) →
58        let hi:H·hi ∈ obs_mereo_L(get_link(li)(n)) in
59        v_m_ch[vi,mi]!SatH(hi) ; veh(vi,mi)(atH(hi)) end end
60    [] stop

```

53 This line coalesces Items 46' and 47'.

54 Coalescing Items 46a.' and 47'.

55 Captures the distinct parameters of Items 46' and 47'.


56 Item 46(b.)i'.

57 Item 46(b.)ii'.

58 Item 47(b.)iiA'.

59 Item 47(b.)iiB'.

60 Coalescing Items 46c.' and 47c.'.

- The above vehicle behaviour definition
 - ❖ will be transformed (i.e., further “refined”)
 - ❖ in Sect. 5.4’s Example 18;
 - ❖ cf. Items 128– 132 on Slide 208 

4.1.3. Discussion

- Domain projection can also be achieved by
 - ❖ developing a “completely new” domain description —
 - ❖ typically on the basis of one or more existing domain description(s) —
 - ❖ where that “new” description now takes the rôle
 - ❖ of being the project domain requirements.

4.2. Domain Instantiation

Definition 12 *Instantiation*: By ***domain instantiation*** we mean

- a **refinement** of the partial domain requirements prescription
- (resulting from the projection step)
- in which the refinements aim at rendering the


⋄ *endurants*:

- ⊗ *parts*,
- ⊗ *materials and*
- ⊗ *components*,
- as well as the*

⋄ *perdurants*:

- ⊗ *actions*,
- ⊗ *events and*
- ⊗ *behaviours*

of the domain requirements prescription

- *more concrete, more specific* 

- Properties that hold of the projected domain shall also hold of the (therefrom) instantiated domain.
- Refinement of endurants can be expressed
 - ✧ either in the form of concrete types,
 - ✧ or of further “delineating” axioms over sorts,
 - ✧ or of a combination of concretisation and axioms.
- We shall exemplify the third possibility.
- Example 10 express requirements that the road net
 - ✧ (on which the road-pricing system is to be based)must satisfy.
- Refinement of perdurants will not be illustrated (other than the simplification of the vehicle projected behaviour).

4.2.1. Domain Instantiation

Example 10 Domain Requirements. Instantiation Road Net:

- We now require that there is, as before, a road net, $n_{\mathcal{J}}:N_{\mathcal{J}}$, which can be understood as consisting of two, “connected sub-nets”.
 - ⋄ A toll-road net, $trn_{\mathcal{J}}:TRN_{\mathcal{J}}$, cf. Fig. 1 on the following slide,
 - ⋄ and an ordinary road net, $n_{\mathcal{P}'}$.
 - ⋄ The two are connected as follows:
 - ⊗ The toll-road net, $trn_{\mathcal{J}}$, borders some toll-road plazas, in Fig. 1 on the next slide shown by white filled circles.
 - ⊗ These toll-road plaza hubs are proper hubs of the ‘ordinary’ road net, $n'_{\mathcal{P}}$.

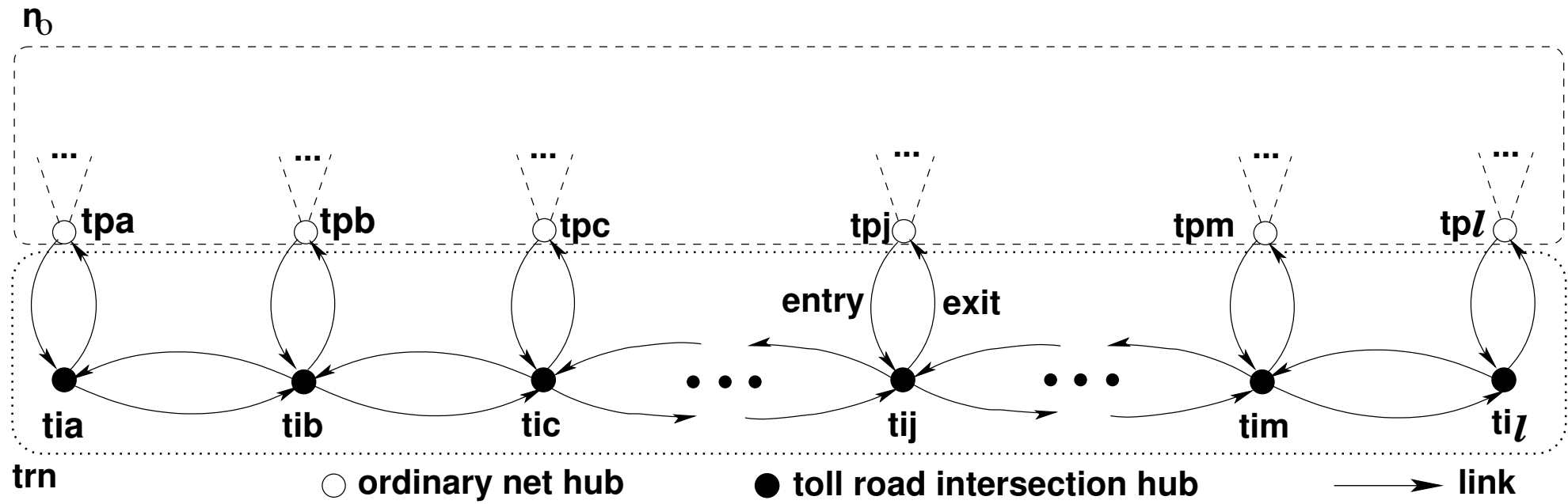


Figure 1: A simple, linear toll-road net

61 The instantiated domain, $\delta_{\mathcal{J}}:\Delta_{\mathcal{J}}$ has just the net, $n_{\mathcal{J}}:N_{\mathcal{J}}$ being instantiated.

62 The road net consists of two “sub-nets”

- a. an “ordinary” road net, $n_o:N_{\mathcal{P}'}$ and
- b. a toll-road net proper, $\text{trn}:\text{TRN}_{\mathcal{J}}$ —

- 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, $trn:TRN_{\mathcal{J}}^{12}$, has each plaza, $hil[i]$, connected to a pair of toll-road links: an entry and an exit link: $(l_e:L, l_x:L)$.
 - iii The toll-road plaza interface to the ‘ordinary’ net, $n_o:N_{\mathcal{P}'}$, has each plaza, i.e., the hub designated by the hub identifier $hil[i]$, connected to one or more ordinary net links, $\{l_{i_1}, l_{i_2}, \dots, l_{i_k}\}$.

¹²We (sometimes) omit the subscript \mathcal{J} when it should be clear from the context what we mean.

- 62b. The toll-road net, $\text{trn:TRN}_{\mathcal{J}}$, consists of three collections (modeled as lists) of links and hubs:
- i a list of pairs of toll-road entry/exit links:
 $\langle (l_{e_1}, l_{x_1}), \dots, (l_{e_\ell}, l_{x_\ell}) \rangle,$
 - ii a list of toll-road intersection hubs: $\langle h_{i_1}, h_{i_2}, \dots, h_{i_\ell} \rangle,$ and
 - iii a list of pairs of main toll-road (“up” and “down”) links:
 $\langle (ml_{i_{1u}}, ml_{i_{1d}}), (m_{i_{2u}}, m_{i_{2d}}), \dots, (m_{i_{\ell u}}, m_{i_{\ell d}}) \rangle.$
- d. The three lists have commensurate lengths (ℓ).

type

61 $\Delta_{\mathcal{J}}$

62 $N_{\mathcal{J}} = N_{\mathcal{P}'} \times \text{HIL} \times \text{TRN}$

62a. $N_{\mathcal{P}'}$

62b. $\text{TRN}_{\mathcal{J}} = (L \times L)^* \times H^* \times (L \times L)^*$

62c. $\text{HIL} = H^*$

axiom

62d. $\forall n_{\mathcal{J}} : N_{\mathcal{J}} \cdot$

62d. **let** $(n_{\Delta}, \text{hil}, (\text{exll}, \text{hl}, \text{lll})) = n_{\mathcal{J}}$ **in**

62d. **len** $\text{hil} = \mathbf{len} \text{ exll} = \mathbf{len} \text{ hl} = \mathbf{len} \text{ lll} + 1$

62d. **end**

[Lecturer explains $N_{\mathcal{P}'}$]

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

63 The toll-road intersection hubs all¹³ have distinct identifiers.

63 $\text{wf_dist_toll_road_isect_hub_ids} : H^* \rightarrow \mathbf{Bool}$

63 $\text{wf_dist_toll_road_isect_hub_ids}(\text{hl}) \equiv$

63 **len** $\text{hl} = \mathbf{card} \text{ xtr_his}(\text{hl})$

¹³A ‘must’ can be inserted in front of all ‘all’s,

64 The toll-road links all have distinct identifiers.

64 $\text{wf_dist_toll_road_u_d_link_ids}: (L \times L)^* \rightarrow \mathbf{Bool}$

64 $\text{wf_dist_toll_road_u_d_link_ids}(\text{lll}) \equiv$

64 $2 \times \mathbf{len} \text{ lll} = \mathbf{card} \text{ xtr_lis}(\text{lll})$

65 The toll-road entry/exit links all have distinct identifiers.

65 $\text{wf_dist_e_x_link_ids}: (L \times L)^* \rightarrow \mathbf{Bool}$

65 $\text{wf_dist_e_x_link_ids}(\text{exll}) \equiv$

65 $2 \times \mathbf{len} \text{ exll} = \mathbf{card} \text{ xtr_lis}(\text{exll})$

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

66 $\text{wf_isoltd_toll_road_isect_hubs}: H^* \times H^* \rightarrow N_{\mathcal{J}} \rightarrow \mathbf{Bool}$

66 $\text{wf_isoltd_toll_road_isect_hubs}(\text{hil}, \text{hl})(n_{\mathcal{J}}) \equiv$

66 $\mathbf{let} \text{ ls} = \text{xtr_links}(n_{\mathcal{J}}) \mathbf{in}$

66 $\mathbf{let} \text{ his} = \bigcup \{ \mathbf{obs_mereo_L}(l) \mid l:L.l \in \text{ls} \} \mathbf{in}$

66 $\text{his} \cap \text{xtr_his}(\text{hl}) = \{ \} \mathbf{end} \mathbf{end}$

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

67 $\text{wf_p_hubs_pt_of_ord_net}: \text{HI}^* \rightarrow \text{N}'_{\Delta} \rightarrow \mathbf{Bool}$

67 $\text{wf_p_hubs_pt_of_ord_net}(\text{hil})(\text{n}'_{\Delta}) \equiv$

67 $\mathbf{elems\ hil} \subseteq \text{xtr_his}(\text{n}'_{\Delta})$

68 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.

68 $\text{wf_p_hub_interf}: \text{N}'_{\Delta} \rightarrow \mathbf{Bool}$

68 $\text{wf_p_hub_interf}(\text{n}_o, \text{hil}, (\text{exll}, _, _)) \equiv$

68 $\forall i: \mathbf{Nat} \cdot i \in \mathbf{inds\ exll} \Rightarrow$

68 $\mathbf{let\ h = get_H(hil(i))(n'_{\Delta})\ in}$

68 $\mathbf{let\ lis = obs_mereo_H(h)\ in}$

68 $\mathbf{let\ lis' = lis \setminus xtr_lis(n')\ in}$

68 $\mathbf{lis' = xtr_lis(exll(i))\ end\ end\ end}$

69 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.

69 **wf_toll_road_isect_hub_iface**: $\mathbf{N}_{\mathcal{J}} \rightarrow \mathbf{Bool}$

69 **wf_toll_road_isect_hub_iface**(__,__,(exll,hl,lll)) \equiv

69 $\forall i:\mathbf{Nat} \cdot i \in \mathbf{inds} \text{ hl} \Rightarrow$

69 **obs_mereo_H**(hl(i)) =

69a. $\text{xtr_lis}(\text{exll}(i)) \cup$

69 **case** i **of**

69b. $1 \rightarrow \text{xtr_lis}(\text{lll}(1)),$

69b. $\mathbf{len} \text{ hl} \rightarrow \text{xtr_lis}(\text{lll}(\mathbf{len} \text{ hl} - 1))$

69b. $_ \rightarrow \text{xtr_lis}(\text{lll}(i)) \cup \text{xtr_lis}(\text{lll}(i - 1))$

69 **end**

70 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.

70 **wf_exll**: $(L \times L)^* \times Hl^* \times H^* \rightarrow \mathbf{Bool}$

70 **wf_exll**(exll,hil,hl) \equiv

70 $\forall i:\mathbf{Nat} \cdot i \in \mathbf{len} \text{ exll}$

70 **let** (hi,(el,xl),h) = (hil(i),exll(i),hl(i)) **in**

70 **obs_mereo_L**(el) = **obs_mereo_L**(xl)

70 = {hi} \cup {**uid_H**(h)} **end**

70 **pre**: **len** eell = **len** hil = **len** hl

71 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.

71 **wf_u_d_links**: $(L \times L)^* \times H^* \rightarrow \mathbf{Bool}$

71 **wf_u_d_links**(lll,hl) \equiv

71 $\forall i:\mathbf{Nat} \cdot i \in \mathbf{inds} \text{ lll} \Rightarrow$

71 **let** (ul,dl) = lll(i) **in**

71 **obs_mereo_L**(ul) = **obs_mereo_L**(dl) =

71a. **uid_H**(hl(i)) \cup **uid_H**(hl(i+1)) **end**

71 **pre**: **len** lll = **len** hl+1

- We have used some additional auxiliary functions:

$\text{xtr_his}: H^* \rightarrow H\mathbf{l}\text{-set}$

$\text{xtr_his}(hl) \equiv \{\mathbf{uid_Hl}(h) \mid h:H \cdot h \in \mathbf{elems\ hl}\}$

$\text{xtr_lis}: (L \times L) \rightarrow L\mathbf{l}\text{-set}$

$\text{xtr_lis}(l', l'') \equiv \{\mathbf{uid_Ll}(l')\} \cup \{\mathbf{uid_Ll}(l'')\}$

$\text{xtr_lis}: (L \times L)^* \rightarrow L\mathbf{l}\text{-set}$

$\text{xtr_lis}(lll) \equiv$
 $\cup \{\text{xtr_lis}(l', l'') \mid (l', l'') : (L \times L) \cdot (l', l'') \in \mathbf{elems\ lll}\}$

72 The well-formedness of instantiated nets is now the conjunction of the individual well-formedness predicates above.

72 $\text{wf_instantiated_net}: N_{\mathcal{J}} \rightarrow \mathbf{Bool}$

72 $\text{wf_instantiated_net}(n'_{\Delta}, \text{hil}, (\text{exll}, \text{hl}, \text{lll}))$

63 $\text{wf_dist_toll_road_isect_hub_ids}(\text{hl})$

64 $\wedge \text{wf_dist_toll_road_u_d_link_ids}(\text{lll})$

65 $\wedge \text{wf_dist_e_e_link_ids}(\text{exll})$

66 $\wedge \text{wf_isolated_toll_road_isect_hubs}(\text{hil}, \text{hl})(n')$

67 $\wedge \text{wf_p_hubs_pt_of_ord_net}(\text{hil})(n')$

68 $\wedge \text{wf_p_hub_interf}(n'_{\Delta}, \text{hil}, (\text{exll}, _, _))$

69 $\wedge \text{wf_toll_road_isect_hub_iface}(_, _, (\text{exll}, \text{hl}, \text{lll}))$

70 $\wedge \text{wf_exll}(\text{exll}, \text{hil}, \text{hl})$

71 $\wedge \text{wf_u_d_links}(\text{lll}, \text{hl})$

4.2.2. Domain Instantiation — Abstraction

Example 11 Domain Requirements. Instantiation Road Net, Abstraction:

- Domain instantiation has refined
 - ✧ an abstract definition of net sorts, $n_{\mathcal{P}}:N_{\mathcal{P}}$,
 - ✧ into a partially concrete definition of nets, $n_{\mathcal{I}}:N_{\mathcal{I}}$.
- We need to show the refinement relation:
 - ✧ $\text{abstraction}(n_{\mathcal{I}}) = n_{\mathcal{P}}$.

value

```

73  abstraction:  $N_{\mathcal{I}} \rightarrow N_{\mathcal{P}}$ 
74  abstraction( $n'_{\Delta}, hl, (exll, hl, lll)$ )  $\equiv$ 
75    let  $n_{\mathcal{P}}: N_{\mathcal{P}}$  .
75      let  $hs = \text{obs\_part\_HS}_{\mathcal{P}}(\text{obs\_part\_HA}_{\mathcal{P}}(n'_{\mathcal{P}})),$ 
75       $ls = \text{obs\_part\_LS}_{\mathcal{P}}(\text{obs\_part\_LA}_{\mathcal{P}}(n'_{\mathcal{P}})),$ 
75       $ths = \text{elems } hl,$ 
75       $eells = \text{xtr\_links}(eell), llls = \text{xtr\_links}(lll)$  in
76       $hs \cup ths = \text{obs\_part\_HS}_{\mathcal{P}}(\text{obs\_part\_HA}_{\mathcal{P}}(n_{\mathcal{P}}))$ 
77       $\wedge \quad ls \cup eells \cup llls = \text{obs\_part\_LS}_{\mathcal{P}}(\text{obs\_part\_LA}_{\mathcal{P}}(n_{\mathcal{P}}))$ 
78     $n_{\mathcal{P}}$  end end

```

- 73 The abstraction function takes a concrete net, $n_{\mathcal{J}}:N_{\mathcal{J}}$, and yields an abstract net, $n_{\mathcal{P}}:N_{\mathcal{P}}$.
- 74 The abstraction function doubly decomposes its argument into constituent lists and sub-lists.
- 75 There is postulated an abstract net, $n_{\mathcal{P}}:N_{\mathcal{P}}$, such that
- 76 the hubs of the concrete net and toll-road equals those of the abstract net, and
- 77 the links of the concrete net and toll-road equals those of the abstract net.
- 78 And that abstract net, $n_{\mathcal{P}}:N_{\mathcal{P}}$, is postulated to be an abstraction of the concrete net.


4.2.3. Discussion

- Domain descriptions, such as illustrated in [Bjø16b, Manifest Domains: Analysis & Description] and in this paper,
 - ❖ model families of concrete, i.e., specifically occurring domains.
 - ❖ Domain instantiation, as exemplified in this section (i.e., Sect.), “narrow down” these families.
 - ❖ Domain instantiation, such as it is defined, cf. Definition 12 on Slide 147,
allows the requirements engineer to instantiate
to a concrete instance of a very specific domain,
 - ❖ that, for example, of the toll-road between *Bolzano Nord* and *Trento Sud* in Italy (i.e., $n=7$)¹⁴.

¹⁴Here we disregard the fact that this toll-road does not start/end in neither *Bolzano Nord* nor *Trento Sud*.

4.3. Domain Determination

Definition 13 ***Determination:*** By ***domain determination*** we mean

- *a refinement of the partial domain requirements prescription,*
 - *resulting from the instantiation step,*
 - *in which the refinements aim at rendering the*
 - ◊ *endurants:*
 - ⊗ *parts,*
 - ⊗ *materials and*
 - ⊗ *components, as well as the*
 - ◊ *perdurants:*
 - ⊗ *functions,*
 - ⊗ *events and*
 - ⊗ *behaviours*
- of the partial domain requirements prescription*
- *less non-determinate, more determinate* 

- Determinations usually render these concepts less general.
 - ❖ That is, the value space
 - ⊗ of endurants that are made more determinate
 - ⊗ is “smaller”, contains fewer values,
 - ⊗ as compared to the endurants before determination has been “applied”.

4.3.1. Domain Determination: Example

- We show an example of ‘domain determination’.
 - ❖ It is expressed solely in terms of
 - ❖ axioms over the concrete toll-road net type.

Example 12 Domain Requirements. Determination Toll-roads:

- We focus only on the toll-road net.
- We single out only two 'determinations':

All Toll-road Links are One-way Links

79 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.


```

79 opposite_traffics:  $(L \times L)^* \times (L \times L)^* \rightarrow \mathbf{Bool}$ 
79 opposite_traffics(exl, lll)  $\equiv$ 
79    $\forall (lt, lf): (L \times L) \cdot (lt, lf) \in \mathbf{elems} \text{ exl} \wedge \text{lll} \Rightarrow$ 
79a.   let  $(lt\sigma, lf\sigma) = (\mathbf{attr\_L\Sigma}(lt), \mathbf{attr\_L\Sigma}(lf))$  in
79a.'.    $\mathbf{attr\_L\Omega}(lt) = \{lt\sigma\} \wedge \mathbf{attr\_L\Omega}(lf) = \{lf\sigma\}$ 
79a.".    $\wedge \mathbf{card} \text{ } lt\sigma = 1 = \mathbf{card} \text{ } lf\sigma$ 
79    $\wedge$  let  $(\{(hi, hi')\}, \{(hi'', hi''')\}) = (lt\sigma, lf\sigma)$  in
79c.    $hi = hi''' \wedge hi' = hi''$ 
79   end end

```

All Toll-road Hubs are Free-flow

80 *The hub state spaces* are singleton sets of the toll-road hub states which always allow exactly these (and only these) crossings:

- a. from *entry* links back to the paired *exit* *links*,
- b. from *entry* links to emanating *toll-road* *links*,
- c. from incident *toll-road* links to *exit* *links*, and
- d. from incident *toll-road* link to emanating *toll-road* *links*.

80 `free_flow_toll_road_hubs: (L × L)* × (L × L)* → Bool`

80 `free_flow_toll_road_hubs(exl, ll) ≡`

80 `∀ i: Nat · i ∈ inds hl ⇒`

80 `attr_HΣ(hl(i)) =`

80a. `hσ_ex_ls(exl(i))`

80b. `∪ hσ_et_ls(exl(i), (i, ll))`

80c. `∪ hσ_tx_ls(exl(i), (i, ll))`

80d. `∪ hσ_tt_ls(i, ll)`

80a.: from *entry* links back to the paired exit *links*:

80a. $h\sigma_ex_ls: (L \times L) \rightarrow L\Sigma$

80a. $h\sigma_ex_ls(e,x) \equiv \{(\mathbf{uid_LI}(e), \mathbf{uid_LI}(x))\}$

80b.: from *entry* links to emanating *toll-road* *links*:

80b. $h\sigma_et_ls: (L \times L) \times (\mathbf{Nat} \times (em:L \times in:L)^*) \rightarrow L\Sigma$

80b. $h\sigma_et_ls((e, _), (i, ll)) \equiv$

80b. **case** i **of**

80b. $2 \rightarrow \{(\mathbf{uid_Ll}(e), \mathbf{uid_Ll}(em(ll(1))))\},$

80b. $\mathbf{len\ ll} + 1 \rightarrow \{(\mathbf{uid_Ll}(e), \mathbf{uid_Ll}(em(ll(\mathbf{len\ ll}))))\},$

80b. $_ \rightarrow \{(\mathbf{uid_Ll}(e), \mathbf{uid_Ll}(em(ll(i-1))))\},$

80b. $(\mathbf{uid_Ll}(e), \mathbf{uid_Ll}(em(ll(i))))\}$

80b. **end**

- The *em* and *in* in the toll-road link list $(em:L \times in:L)^*$ designate selectors for *emanating*, respectively *incident* links.

80c.: from incident *toll-road* links to exit *links*:

80c. $h\sigma_tx_ls: (L \times L) \times (\mathbf{Nat} \times (em:L \times in:L)^*) \rightarrow L\Sigma$

80c. $h\sigma_tx_ls((_,x),(i,ll)) \equiv$

80c. **case** *i* **of**

80c. 2 $\rightarrow \{(\mathbf{uid_LI}(in(ll(1))), \mathbf{uid_LI}(x))\},$

80c. **len** *ll* + 1 $\rightarrow \{(\mathbf{uid_LI}(in(ll(\mathbf{len} \ ll))), \mathbf{uid_LI}(x))\},$

80c. — $\rightarrow \{(\mathbf{uid_LI}(in(ll(i-1))), \mathbf{uid_LI}(x)),$

80c. $(\mathbf{uid_LI}(in(ll(i))), \mathbf{uid_LI}(x))\}$

80c. **end**

80d.: from incident *toll-road* link to emanating *toll-road* links:

80d. $h\sigma_tt_ls: \mathbf{Nat} \times (\mathbf{em}:L \times \mathbf{in}:L)^* \rightarrow L\Sigma$

80d. $h\sigma_tt_ls(i, ll) \equiv$

80d. **case** *i* **of**

80d. 2 $\rightarrow \{(\mathbf{uid_Ll}(\mathbf{in}(ll(1))), \mathbf{uid_Ll}(\mathbf{em}(ll(1))))\},$

80d. **len** *ll* + 1 $\rightarrow \{(\mathbf{uid_Ll}(\mathbf{in}(ll(\mathbf{len}\ ll))), \mathbf{uid_Ll}(\mathbf{em}(ll(\mathbf{len}\ ll))))\},$

80d. — $\rightarrow \{(\mathbf{uid_Ll}(\mathbf{in}(ll(i-1))), \mathbf{uid_Ll}(\mathbf{em}(ll(i-1))))),$

80d. $(\mathbf{uid_Ll}(\mathbf{in}(ll(i))), \mathbf{uid_Ll}(\mathbf{em}(ll(i))))\}$

80d. **end**

- The example above illustrated ‘domain determination’ with respect to endurants.
 - ❖ Typically “endurant determination” is expressed in terms of axioms that limit state spaces —
 - ❖ where “endurant instantiation” typically “limited” the mereology of endurants: how parts are related to one another.


- We shall not exemplify domain determination with respect to perdurants.

4.3.2. Discussion

- ⋄ The borderline between instantiation and determination is fuzzy.
 - ⊗ Whether, as an example, fixing the number of toll-road intersection hubs to a constant value, e.g., $n=7$,
 - ⊗ is instantiation
 - ⊗ or determination,
- is really a matter of choice !

4.4. Domain Extension

Definition 14 *Extension*: By *domain extension* we understand the

- *introduction of*
 - ◇ *endurants (see Sect.) and*
 - ◇ *perdurants (see Sect.)**that were not feasible in the original domain,*
- *but for which, with computing and communication,*
- *and with new, emerging technologies,*
- *for example, sensors, actuators and satellites,*
- *there is the possibility of feasible implementations,*
- *hence the requirements,*
- *that what is introduced becomes part of the unfolding requirements prescription* 

4.4.1. Endurant Extensions

Definition 15 *Endurant Extension*:

- By an *endurant extension* we understand
 - ✧ the introduction of one or more endurants
 - ✧ into the projected, instantiated and determined domain $\mathcal{D}_{\mathcal{R}}$
 - ✧ resulting in domain $\mathcal{D}_{\mathcal{R}}'$,
 - ✧ such that these form a *conservative extension*
 - ✧ of the theory, $\mathcal{T}_{\mathcal{D}_{\mathcal{R}}}$
 - ✧ denoted by the domain requirements $\mathcal{D}_{\mathcal{R}}$ (i.e., “before” the extension),
 - ✧ that is: every theorem of $\mathcal{T}_{\mathcal{D}_{\mathcal{R}}}$
 - ✧ is still a theorem of $\mathcal{T}_{\mathcal{D}_{\mathcal{R}}'}$.

- Usually domain extensions involve one or more of the already introduced sorts.
- In Example 13 we introduce (i.e., “extend”)
 - ❖ vehicles with GPSS-like sensors,
 - ❖ and introduce toll-gates with
 - ⊗ entry sensors,
 - ⊗ vehicle identification sensors,
 - ⊗ gate actuators and
 - ⊗ exit sensors.
 - ❖ Finally road pricing calculators are introduced.

Example 13 Domain Requirements — Endurant Extension:

- We present the extensions in several steps.
 - ❖ Some of them will be developed in this section.
 - ❖ Development of the remaining will be deferred to Sect. .
 - ❖ The reason for this deferment is that those last steps are examples of *interface requirements*.

- The initial extension-development steps are:
 - ❖ [a] vehicle extension,
 - ❖ [b] sort and unique identifiers of road price calculators,
 - ❖ [c] vehicle to road pricing calculator channel,
 - ❖ [d] sorts and dynamic attributes of toll-gates,
 - ❖ [e] road pricing calculator attributes,
 - ❖ [f] “total” system state, and
 - ❖ [g] the overall system behaviour.
- This decomposition establishes system interfaces in “small, easy steps”.

4.4.1.1 [a] Vehicle Extension:

- 81 There is a domain, $\delta_{\mathcal{E}}:\Delta_{\mathcal{E}}$, which contains
- 82 a fleet, $f_{\mathcal{E}}:\mathbf{F}_{\mathcal{E}}$, that is,
- 83 a set, $vs_{\mathcal{E}}:\mathbf{VS}_{\mathcal{E}}$, of
- 84 extended vehicles, $v_{\mathcal{E}}:\mathbf{V}_{\mathcal{E}}$ — their extension amounting to
- 85 a dynamic reactive attribute, whose value, $ti_gpos:\mathbf{TiGpos}$, at any time, reflects that vehicle's time-stamped global position.
- 86 The vehicle's GNSS receiver calculates, loc_pos , its local position, $lpos:\mathbf{LPos}$, based on these signals.
- 87 Vehicles access these *external attributes* via the *external attribute channel*, $attr_TiGPos_ch$.

type

```

81   $\Delta_{\mathcal{E}}$ 
82   $F_{\mathcal{E}}$ 
83   $VS_{\mathcal{E}} = V_{\mathcal{E}}\text{-set}$ 
84   $V_{\mathcal{E}}$ 
85   $TiGPos = \mathbb{T} \times GPos$ 
86   $GPos, LPos$ 

```

value

```

81   $\delta_{\mathcal{E}}: \Delta_{\mathcal{E}}$ 
82  obs_part $F_{\mathcal{E}}: \Delta_{\mathcal{E}} \rightarrow F_{\mathcal{E}}$ 
82   $f = \mathbf{obs\_part}F_{\mathcal{E}}(\delta_{\mathcal{E}})$ 
83  obs_part $VS_{\mathcal{E}}: F_{\mathcal{E}} \rightarrow VS_{\mathcal{E}}$ 
83   $vs = \mathbf{obs\_part}VS_{\mathcal{E}}(f)$ 
83   $vis = \mathbf{xtr\_vis}(vs)$ 
85   $\mathbf{attr\_TiGPos\_ch}[vi]?$ 
86   $\mathbf{loc\_pos}: GPos \rightarrow LPos$ 

```

channel

```

86   $\{\mathbf{attr\_TiGPos\_ch}[vi] \mid vi:VI \cdot vi \in vis\}: TiGPos$ 

```


We define two auxiliary functions,

88 **xtr_vs**, which given a domain, or a fleet, extracts its set of vehicles,
and

89 **xtr_vis** which given a set of vehicles generates their unique
identifiers.

value

88 **xtr_vs**: $(\Delta_{\mathcal{E}} | F_{\mathcal{E}} | VS_{\mathcal{E}}) \rightarrow V_{\mathcal{E}}\text{-set}$

88 **xtr_vs**(arg) \equiv

88 **is** $_{\Delta_{\mathcal{E}}}$ (arg) \rightarrow **obs_part** $_{VS_{\mathcal{E}}}(\text{obs_part}_{F_{\mathcal{E}}}(\text{arg}))$,

88 **is** $_{F_{\mathcal{E}}}$ (arg) \rightarrow **obs_part** $_{VS_{\mathcal{E}}}(\text{arg})$,

88 **is** $_{VS_{\mathcal{E}}}$ (arg) \rightarrow arg

89 **xtr_vis**: $(\Delta_{\mathcal{E}} | F_{\mathcal{E}} | VS_{\mathcal{E}}) \rightarrow VI\text{-set}$

89 **xtr_vis**(arg) $\equiv \{\mathbf{uid}_{VI}(v) | v \in \mathbf{xtr_vs}(\text{arg})\}$

4.4.1.2 [b] Road Pricing Calculator: Basic Sort and Unique Identifier:

90 The domain $\delta_{\mathcal{E}}:\Delta_{\mathcal{E}}$, also contains a pricing calculator, $c:C_{\delta_{\mathcal{E}}}$, with unique identifier $ci:CI$.

type

90 C, CI

value

90 **obs_part_C**: $\Delta_{\mathcal{E}} \rightarrow C$

90 **uid_CI**: $C \rightarrow CI$

90 $c = \mathbf{obs_part_C}(\delta_{\mathcal{E}})$

90 $ci = \mathbf{uid_CI}(c)$

4.4.1.3 [c] Vehicle to Road Pricing Calculator Channel:

91 Vehicles can, on their own volition, offer the timed local position,
viti-lpos:VITiLPos

92 to the pricing calculator, $c:C_{\mathcal{E}}$ along a vehicles-to-calculator
channel, v_c_ch .

type

91 $VITiLPos = VI \times (\mathbb{T} \times LPos)$

channel

92 $\{v_c_ch[vi,ci] \mid vi:VI, ci:C \mid vi \in vis \wedge ci = \mathbf{uid_C}(c)\}:VITiLPos$

4.4.1.4 [d] Toll-gate Sorts and Dynamic Types:

- We extend the domain with toll-gates for vehicles entering and exiting the toll-road entry and exit links.
- Figure 2 illustrates the idea of gates.

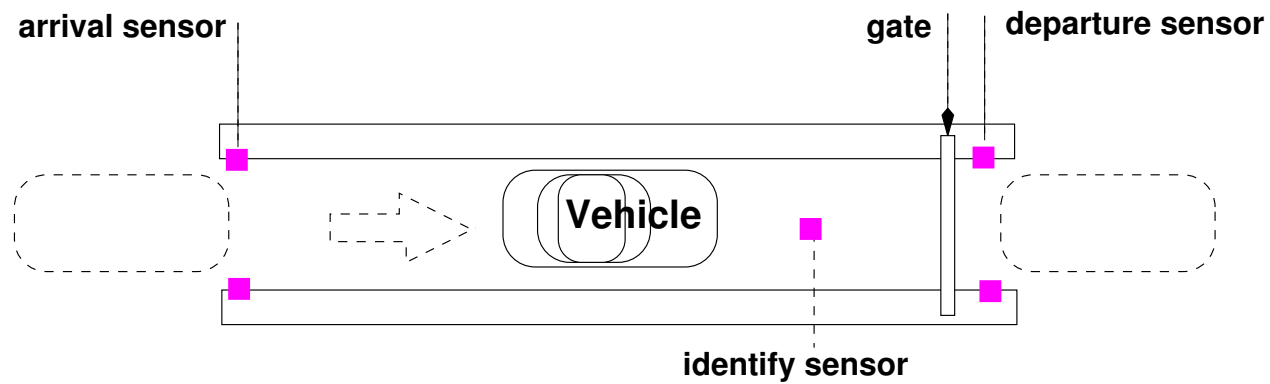


Figure 2: A toll plaza gate

- Figure 2 is intended to illustrate a vehicle entering (or exiting) a toll-road arrival link.

- ❖ The toll-gate is equipped with three sensors:
an arrival sensor, a vehicle identification sensor and an departure sensor.
- ❖ The arrival sensor serves to prepare the vehicle identification sensor.
- ❖ The departure sensor serves to prepare the gate for closing when a vehicle has passed.
- ❖ The vehicle identify sensor identifies the vehicle and “delivers” a pair: the current time and the vehicle identifier.
- ❖ Once the vehicle identification sensor has identified a vehicle
 - ⊗ the gate opens and
 - ⊗ a message is sent to the road pricing calculator as to the passing vehicle’s identity and the identity of the link associated with the toll-gate (see Items 109- 110 on Slide 197).

- 93 The domain contains the extended net, $n:N_{\mathcal{E}}$,
94 with the net extension amounting to the toll-road net, $TRN_{\mathcal{E}}$, that is, the
instantiated toll-road net, $trn:TRN_{\mathcal{J}}$, is extended, into $trn:TRN_{\mathcal{E}}$, with *entry*,
 $eg:EG$, and *exit*, $xg:XG$, toll-gates.

From entry- and exit-gates we can observe

- 95 their unique identifier and
96 their mereology: pairs of entry-, respectively exit link and calculator unique
identifiers; further
97 a pair of gate entry and exit sensors modeled as *external attribute* channels,
($ges:ES, gls:XS$), and
98 a time-stamped vehicle identity sensor modeled as *external attribute* channels.

type

93 $N_{\mathcal{E}}$
 94 $TRN_{\mathcal{E}} = (EG \times XG)^* \times TRN_{\mathcal{J}}$
 95 GI

value

93 **obs_part** $N_{\mathcal{E}}: \Delta_{\mathcal{E}} \rightarrow N_{\mathcal{E}}$
 94 **obs_part** $TRN_{\mathcal{E}}: N_{\mathcal{E}} \rightarrow TRN_{\mathcal{E}}$
 95 **uid** $G: (EG|XG) \rightarrow GI$
 96 **obs_mereo** $G: (EG|XG) \rightarrow (LI \times CI)$
 94 $trn: TRN_{\mathcal{E}} = \mathbf{obs_part_}TRN_{\mathcal{E}}(\delta_{\mathcal{E}})$

channel

97 $\{\text{attr_entry_ch}[gi] | gi: GI \cdot \text{xtr_eGlds}(trn)\}$ "enter"
 97 $\{\text{attr_exit_ch}[gi] | gi: GI \cdot \text{xtr_xGlds}(trn)\}$ "exit"
 98 $\{\text{attr_identity_ch}[gi] | gi: GI \cdot \text{xtr_Glds}(trn)\}$ TIVI

type

98 $TIVI = \mathbb{T} \times VI$

We define some **auxiliary functions** over toll-road nets, $\text{trn:TRN}_{\mathcal{E}}$:

99 xtr_eG extracts the *list* of entry gates,

100 xtr_xG extracts the *list* of exit gates,

101 xtr_eGlds extracts the set of entry gate identifiers,

102 xtr_xGlds extracts the set of exit gate identifiers,

103 xtr_Gs extracts the set of all gates, and

104 xtr_Glds extracts the set of all gate identifiers.

value

99 $\text{xtr_eGl}: \text{TRN}_{\mathcal{E}} \rightarrow \text{EG}^*$
 99 $\text{xtr_eGl}(\text{pgl}, _) \equiv \{\text{eg} \mid (\text{eg}, \text{xg}): (\text{EG}, \text{XG}) \cdot (\text{eg}, \text{xg}) \in \text{elems pgl}\}$
 100 $\text{xtr_xGl}: \text{TRN}_{\mathcal{E}} \rightarrow \text{XG}^*$
 100 $\text{xtr_xGl}(\text{pgl}, _) \equiv \{\text{xg} \mid (\text{eg}, \text{xg}): (\text{EG}, \text{XG}) \cdot (\text{eg}, \text{xg}) \in \text{elems pgl}\}$
 101 $\text{xtr_eGlds}: \text{TRN}_{\mathcal{E}} \rightarrow \text{Gl-set}$
 101 $\text{xtr_eGlds}(\text{pgl}, _) \equiv \{\text{uid_Gl}(g) \mid g: \text{EG} \cdot g \in \text{xtr_eGs}(\text{pgl}, _)\}$
 102 $\text{xtr_xGlds}: \text{TRN}_{\mathcal{E}} \rightarrow \text{Gl-set}$
 102 $\text{xtr_xGlds}(\text{pgl}, _) \equiv \{\text{uid_Gl}(g) \mid g: \text{EG} \cdot g \in \text{xtr_xGs}(\text{pgl}, _)\}$
 103 $\text{xtr_Gs}: \text{TRN}_{\mathcal{E}} \rightarrow \text{G-set}$
 103 $\text{xtr_Gs}(\text{pgl}, _) \equiv \text{xtr_eGs}(\text{pgl}, _) \cup \text{xtr_xGs}(\text{pgl}, _)$
 104 $\text{xtr_Glds}: \text{TRN}_{\mathcal{E}} \rightarrow \text{Gl-set}$
 104 $\text{xtr_Glds}(\text{pgl}, _) \equiv \text{xtr_eGlds}(\text{pgl}, _) \cup \text{xtr_xGlds}(\text{pgl}, _)$

105 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

106 the unique identifier, $ci:CI$, of the road pricing calculator, $c:C$, and

107 the unique identifiers, $vis:VI$ -set, of the fleet vehicles.

axiom

```

105   $\forall n:N_{\mathcal{R}_3}, \text{trn}: \text{TRN}_{\mathcal{R}_3} \cdot$ 
105    let (exgl,(exl,hl,lll)) = obs_part_TRN $_{\mathcal{R}_3}$ (n) in
105a.    len exgl = len exl = len hl = len lll + 1
105b.     $\wedge$  card xtr_Glds(exgl) = 2 * len exgl
105c.     $\wedge \forall i:\text{Nat} \cdot i \in \text{inds exgl} \cdot$ 
105c.        let ((eg,xg),(el,xl)) = (exgl(i),exl(i)) in
105c.        obs_mereo_G(eg) = (uid_U(el),ci,vis)
105c.         $\wedge$  obs_mereo_G(xg) = (uid_U(xl),ci,vis)
105    end end

```

4.4.1.5 [e] Toll-gate to Calculator Channels:

108 We distinguish between *entry* and *exit* gates.

109 Toll road entry and exit gates offers the road pricing calculator a pair: whether it is an entry or an exit gates, and pair of the passing vehicle's identity and the time-stamped identity of the link associated with the toll-gate

110 to the road pricing calculator via a (gate to calculator) channel.

type

108 $EE = \text{"entry"} | \text{"exit"}$

109 $EEVITiLI = EE \times (VI \times (\mathbb{T} \times \text{SonL}))$

channel

110 $\{g_c_ch[gi,ci] | gi:GI \cdot gi \in gis\} : EETiVILI$

4.4.1.6 [f] Road Pricing Calculator Attributes:

- 111 The road pricing attributes include a programmable traffic map, trm:TRM , which, for each vehicle inside the toll-road net, records a chronologically ordered list of each vehicle's timed position, (τ, lpos) , and
- 112 a static (total) road location function, vplf:VPLF . The vehicle position location function, vplf:VPLF , which, given a local position, lpos:LPos , yields *either* the simple vehicle position, svpos:SVPos , designated by the GNSS-provided position, *or* yields the response that the provided position is off the toll-road net. The vplf:VPLF function is constructed, construct_vplf ,
- 113 from awareness, of a geodetic road map, GRM , of the topology of the extended net, $\text{n}_{\mathcal{E}}:\text{N}_{\mathcal{E}}$, including the mereology and the geodetic attributes of links and hubs.

type

111 $\text{TRM} = \text{VI} \xrightarrow{\overline{m}} (\mathbb{T} \times \text{SVPos})^*$

112 $\text{VPLF} = \text{GRM} \rightarrow \text{LPos} \rightarrow (\text{SVPos} \mid \text{"off_N"})$

113 GRM

value

111 $\text{attr_TRM}: \text{C}_{\mathcal{E}} \rightarrow \text{TRM}$

112 $\text{attr_VPLF}: \text{C}_{\mathcal{E}} \rightarrow \text{VPLF}$

- The geodetic road map maps geodetic locations into hub and link identifiers.

23 Geodetic link locations represent the set of point locations of a link.

19a. Geodetic hub locations represent the set of point locations of a hub.

114 A geodetic road map maps geodetic link locations into link identifiers and geodetic hub locations into hub identifiers.

115 We sketch the construction, *geo_GRM*, of geodetic road maps.

type

114 $\text{GRM} = (\text{GeoL} \xrightarrow{m} \text{LI}) \cup (\text{GeoH} \xrightarrow{m} \text{HI})$

value

115 $\text{geo_GRM}: \text{N} \rightarrow \text{GRM}$

115 $\text{geo_GRM}(n) \equiv$

115 **let** $\text{ls} = \text{xtr_links}(n)$, $\text{hs} = \text{xtr_hubs}(n)$ **in**

115 $[\text{attr_GeoL}(l) \mapsto \text{uid_LI}(l) \mid l:\text{L}.l \in \text{ls}]$

115 \cup

115 $[\text{attr_GeoH}(h) \mapsto \text{uid_HI}(h) \mid h:\text{H}.h \in \text{hs}]$ **end**

116 The `vplf:VPLF` function obtains a simple vehicle position, `svpos`, from a geodetic road map, `grm:GRM`, and a local position , `lpos`:

value

116 `obtain_SVPos: GRM → LPos → SVPos`

116 `obtain_SVPos(grm)(lpos) as svpos`

116 **post:** `case svpos of`

116 `SatH(hi) → within(lpos,grm(hi)),`

116 `SonL(li) → within(lpos,grm(li)),`

116 `"off_N" → true end`

4.4.1.7 [g] “Total” System State:

Global values:

- 117 There is a given domain, $\delta_{\mathcal{E}}:\Delta_{\mathcal{E}}$;
- 118 there is the net, $n_{\mathcal{E}}:N_{\mathcal{E}}$, of that domain;
- 119 there is toll-road net, $trn_{\mathcal{E}}:TRN_{\mathcal{E}}$, of that net;
- 120 there is a set, $egs_{\mathcal{E}}:EG_{\mathcal{E}}\text{-set}$, of entry gates;
- 121 there is a set, $xgs_{\mathcal{E}}:XG_{\mathcal{E}}\text{-set}$, of exit gates;
- 122 there is a set, $gis_{\mathcal{E}}:GI_{\mathcal{E}}\text{-set}$, of gate identifiers;
- 123 there is a set, $vs_{\mathcal{E}}:V_{\mathcal{E}}\text{-set}$, of vehicles;
- 124 there is a set, $vis_{\mathcal{E}}:VI_{\mathcal{E}}\text{-set}$, of vehicle identifiers;
- 125 there is the road-pricing calculator, $c_{\mathcal{E}}:C_{\mathcal{E}}$ and
- 126 there is its unique identifier, $ci_{\mathcal{E}}:CI$.

value

```

117   $\delta_{\mathcal{E}}:\Delta_{\mathcal{E}}$ 
118   $n_{\mathcal{E}}:N_{\mathcal{E}} = \mathbf{obs\_part\_N}_{\mathcal{E}}(\delta_{\mathcal{E}})$ 
119   $trn_{\mathcal{E}}:TRN_{\mathcal{E}} = \mathbf{obs\_part\_TRN}_{\mathcal{E}}(n_{\mathcal{E}})$ 
120   $egs_{\mathcal{E}}:EG\text{-set} = \mathbf{xtr\_egs}(trn_{\mathcal{E}})$ 
121   $xgs_{\mathcal{E}}:XG\text{-set} = \mathbf{xtr\_xgs}(trn_{\mathcal{E}})$ 
122   $gis_{\mathcal{E}}:XG\text{-set} = \mathbf{xtr\_gis}(trn_{\mathcal{E}})$ 
123   $vs_{\mathcal{E}}:V_{\mathcal{E}}\text{-set} = \mathbf{obs\_part\_VS}(\mathbf{obs\_part\_F}_{\mathcal{E}}(\delta_{\mathcal{E}}))$ 
124   $vis_{\mathcal{E}}:VI\text{-set} = \{\mathbf{uid\_VI}(v_{\mathcal{E}}) \mid v_{\mathcal{E}}:V_{\mathcal{E}} \cdot v_{\mathcal{E}} \in vs_{\mathcal{E}}\}$ 
125   $c_{\mathcal{E}}:C_{\mathcal{E}} = \mathbf{obs\_part\_C}_{\mathcal{E}}(\delta_{\mathcal{E}})$ 
126   $ci_{\mathcal{E}}:CI_{\mathcal{E}} = \mathbf{uid\_CI}(c_{\mathcal{E}})$ 

```

4.4.1.8 [h] “Total” System Behaviour:

The signature and definition of the system behaviour is sketched as are the signatures of the vehicle, toll-gate and road pricing calculator.

- We shall model the behaviour of the road pricing system as follows:
 - ⋄ we shall not model behaviours nets, hubs and links;
 - ⋄ thus we shall model only
 - ⊗ the behaviour of vehicles, **veh**,
 - ⊗ the behaviour of toll-gates, **gate**, and
 - ⊗ the behaviour of the road-pricing calculator, **calc**,
 - ⋄ The behaviours of vehicles and toll-gates are presented here.
 - ⋄ But the behaviour of the road-pricing calculator is “deferred” till Sect. 5.4 since it reflects an interface requirements.

127 The road pricing system behaviour, **sys**, is expressed as

- a. the parallel, \parallel , (distributed) composition of the behaviours of all vehicles,
- b. with the parallel composition of the parallel (likewise distributed) composition of the behaviours of all entry gates,
- c. with the parallel composition of the parallel (likewise distributed) composition of the behaviours of all exit gates,
- d. with the parallel composition of the behaviour of the road-pricing calculator,

value

127 sys: **Unit** \rightarrow **Unit**

127 sys() \equiv

127a. $\parallel \{ \text{veh}(\text{uid_V}(v), (ci, gis)) \mid v:V \cdot v \in vs_{\mathcal{E}} \}$

127b. $\parallel \parallel \{ \text{gate}(gi, \text{”entry”}, \text{obs_mereo_G}(eg)) \mid eg:EG \cdot eg \in \text{egs}_{\mathcal{E}} \wedge gi = \text{uid_EG}(eg) \}$

127c. $\parallel \parallel \{ \text{gate}(gi, \text{”exit”}, \text{obs_mereo_G}(xg)) \mid xg:XG \cdot xg \in \text{xgs}_{\mathcal{E}} \wedge gi = \text{uid_XG}(xg) \}$

127d. $\parallel \text{calc}(ci_{\mathcal{E}}, (vis_{\mathcal{E}}, gis_{\mathcal{E}}))(rlf)(trm)$

128 veh: $vi:VI \times (ci:CI \times gis:GI\text{-set}) \rightarrow \text{out } v_c_ch[vi, ci] \text{ **Unit**}$

134 gate: $gi:GI \times ee:EE \times (ci:CI \times VI\text{-set} \times LI) \rightarrow \text{in } \text{attr_entry_ch}[gi, ci], \text{attr_id_ch}[gi, ci], \text{attr_exit_ch}[gi, ci] \bullet$

168 calc: $ci:CI \times (vis:VI\text{-set} \times gis:GI\text{-set}) \times VPLF \rightarrow TRM \rightarrow \text{in } \{v_c_ch[vi, ci] \mid vi:VI \cdot vi \in vis\}, \{g_c_ch[gi, ci] \mid g:GI \cdot g \in gis\}$

- We consider ”entry” or ”exit” to be a static attribute of toll-gates.
- The behaviour signatures were determined as per the techniques presented in [Bjø16b, Sect. 4.1.1 and 4.5.2].

Vehicle Behaviour:

- 128 Instead of moving around by explicitly expressed internal non-determinism¹⁵ vehicles move around by unstated internal non-determinism and instead receive their current position from the global positioning subsystem.
- 129 At each moment the vehicle receives its time-stamped global position, (τ, gpos) :TiGPos,
- 130 from which it calculates the local position, lpos :VPos
- 131 which it then communicates, with its vehicle identification, $(\text{vi}, (\tau, \text{lpos}))$, to the road pricing subsystem —
- 132 whereupon it resumes its vehicle behaviour.

¹⁵We refer to Items 46b., 46c. on Slide 68 and 47b., 47(b.)ii, 47c. on Slide 70

value

```

128  veh:  $vi:VI \times (ci:CI \times gis:GI\text{-set}) \rightarrow$ 
128      in attr_TiGPos_ch[vi] out v_c_ch[vi,ci] Unit
128  veh(vi,(ci,gis))  $\equiv$ 
129      let ( $\tau$ ,gpos) = attr_TiGPos_ch[vi]? in
130      let lpos = loc_pos(gpos) in
131      v_c_ch[vi,ci] ! (vi,( $\tau$ ,lpos)) ;
132      veh(vi,(ci,gis)) end end
128      pre  $vi \in vis_{\mathcal{E}} \wedge ci = ci_{\mathcal{E}} \wedge gis = gis_{\mathcal{E}}$ 

```

- The above behaviour represents an assumption about the behaviour of vehicles.
 - ⋄ If we were to design software for the monitoring and control of vehicles
 - ⋄ then the above vehicle behaviour would have to be refined in order to serve as a proper interface requirements.
 - ⋄ The refinement would include handling concerns
 - ⊗ about the drivers’ behaviour when entering, passing and exiting toll-gates,
 - ⊗ about the proper function of the GNSS equipment, and
 - ⊗ about the safe communication with the road price calculator.

- ❖ The above concerns would already have been addressed
 - ⊗ in a model of domain facets such as
 - * *human behaviour*,
 - * *technology support*,
 - * proper tele-communications *scripts*,
 - * etcetera.
 - ⊗ We refer to [Bjø10a].

Gate Behaviour:

- The entry and the exit gates have “vehicle enter”, “vehicle exit” and “timed vehicle identification” sensors.
 - ⋄ The following assumption can now be made:
 - ⊗ during the time interval between
 - ⊗ a gate’s vehicle “entry” sensor having first sensed a vehicle entering that gate
 - ⊗ and that gate’s “exit” sensor having last sensed that vehicle leaving that gate
 - ⊗ that gate’s vehicle time and “identify” sensor registers the time when the vehicle is entering the gate and that vehicle’s unique identification.

- We sketch the toll-gate behaviour:

133 We parameterise the toll-gate behaviour as either an entry or an exit gate.

134 Toll-gates operate autonomously and cyclically.

135 The `attr_enter_ch` event “triggers” the behaviour specified in formula line Item 136–138 starting with a “Raise” barrier action.

136 The time-of-passing and the identity of the passing vehicle is sensed by `attr_passing_ch` channel events.

137 Then the road pricing calculator is informed of time-of-passing and of the vehicle identity `vi` and the link `li` associated with the gate – and with a “Lower” barrier action.

138 And finally, after that vehicle has left the entry or exit gate the barrier is again “Lower”ered and

139 that toll-gate’s behaviour is resumed.

type

133 $EE = \text{"enter"} \mid \text{"exit"}$

value

134 $\text{gate: } gi:GI \times ee:EE \times (ci:CI \times VI\text{-set} \times LI) \rightarrow$

134 **in** $\text{attr_enter_ch}[gi], \text{attr_passing_ch}[gi], \text{attr_leave_ch}[gi]$

134 **out** $\text{attr_barrier_ch}[gi], g_c_ch[gi, ci]$ **Unit**

134 $\text{gate}(gi, ee, (ci, vis, li)) \equiv$

135 $\text{attr_enter_ch}[gi] ? ; \text{attr_barrier_ch}[gi] ! \text{"Lower"}$

136 **let** $(\tau, vi) = \text{attr_passing_ch}[gi] ?$ **in assert** $vi \in vis$

137 $(\text{attr_barrier_ch}[gi] ! \text{"Raise"} \parallel g_c_ch[gi, ci] ! (ee, (vi, (\tau, \text{SonL}(li)))));$

138 $\text{attr_leave_ch}[gi] ? ; \text{attr_barrier_ch}[gi] ! \text{"Lower"}$

139 $\text{gate}(gi, ee, (ci, vis, li))$


134 **end**

134 **pre** $ci = ci_{\mathcal{E}} \wedge vis = vis_{\mathcal{E}} \wedge li \in lis_{\mathcal{E}}$

- The *gate* signature's $attr_enter_ch[gi]$, $attr_passing_ch[gi]$, $attr_barrier_ch[gi]$ and $attr_leave_ch[gi]$ model respective *external attributes* [Bjø16b, Sect. 4.1.1 and 4.5.2] (the $attr_barrier_ch[gi]$ models reactive (i.e., output) attribute), while
- $g_c_ch[gi,ci]$ models the *embedded attribute sharing* between gates (their identification of vehicle positions) and the calculator road map.

- The above behaviour represents an assumption about the behaviour of toll-gates.
 - ⋄ If we were to design software for the monitoring and control of toll-gates
 - ⋄ then the above gate behaviour would have to be refined in order to serve as a proper interface requirements.
 - ⋄ The refinement would include handling concerns
 - ⊗ about the drivers’ behaviour when entering, passing and exiting toll-gates,
 - ⊗ about the proper function of the entry, passing and exit sensors,
 - ⊗ about the proper function of the gate barrier (opening and closing), and
 - ⊗ about the safe communication with the road price calculator.

The above concerns would already have been addressed

- in a model of domain facets such as
 - ✧ *human behaviour*,
 - ✧ *technology support*,
 - ✧ proper tele-communications *scripts*,
 - ✧ etcetera.
- We refer to [Bjø10a] 

4.5. Requirements Fitting

- Often a domain being described
- “fits” onto, is “adjacent” to, “interacts” in some areas with,
- another domain:
 - ❖ transportation with logistics,
 - ❖ health-care with insurance,
 - ❖ banking with securities trading and/or insurance,
 - ❖ and so on.

- The issue of requirements fitting arises
 - ❖ when two or more software development projects
 - ❖ are based on what appears to be the same domain.
- The problem then is
 - ❖ to harmonise the two or more software development projects
 - ❖ by harmonising, if not too late, their requirements developments.

- We thus assume
 - ❖ that there are n domain requirements developments, $d_{r_1}, d_{r_2}, \dots, d_{r_n}$, being considered, and
 - ❖ that these pertain to the same domain — and can hence be assumed covered by a same domain description.

Definition 16 *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*', $p_{dr_1}, p_{dr_2}, \dots, p_{dr_n}$, and
 - ⊗ m *shared domain requirements*, $s_{dr_1}, s_{dr_2}, \dots, s_{dr_m}$,
 - ⊗ 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 17 *Requirements Harmonisation:*

- By *requirements harmonisation* we mean
 - ⋄ a number of alternative and/or co-ordinated prescription actions,
 - ⋄ one set for each of the domain requirements actions:
 - ⊗ Projection,
 - ⊗ Instantiation,
 - ⊗ Determination and
 - ⊗ Extension.




- They are – we assume n separate software product requirements:
 - ⋄ Projection:
 - ⊗ If the n product requirements do not have the same projections,
 - ⊗ then identify a common projection which they all share,
 - ⊗ and refer to it as the *common projection*.
 - ⊗ Then develop, for each of the n product requirements,
 - ⊗ if required,
 - ⊗ a *specific projection* of the common one.
 - ⊗ Let there be m such specific projections, $m \leq n$.

❖ Instantiation:

- ⊗ First instantiate the common projection, if any instantiation is needed.
- ⊗ Then for each of the m specific projections
- ⊗ instantiate these, if required.

❖ Determination:

- ⊗ Likewise, if required, “perform” “determination” of the possibly instantiated common projection,
- ⊗ and, similarly, if required,
- ⊗ “perform” “determination” of the up to m possibly instantiated projections.

- ❖ 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 requirement***s we mean a domain requirements which is short of (that is, is missing) some prescription parts: text and formula 
- By a ***shared domain requirement***s we mean a domain requirements 

- By **requirements fitting** m shared domain requirements texts, $sdrs$, into n partial domain requirements we mean that
 - ✧ there is for each partial domain requirements, pdr_i ,
 - ✧ an identified, non-empty subset of $sdrs$ (could be all of $sdrs$), $ssdrs_i$,
 - ✧ such that textually conjoining $ssdrs_i$ to pdr_i ,
 - ✧ i.e., $ssdrs_i \oplus pdr_i$
 - ✧ can be claimed to yield the “original” d_{r_i} ,
 - ✧ that is, $\mathcal{M}(ssdrs_i \oplus pdr_i) \subseteq \mathcal{M}(d_{r_i})$,
 - ✧ where \mathcal{M} is a suitable meaning function over prescriptions ■

4.6. Discussion


- **Facet-oriented Fittings:**

- ❖ An altogether different way of looking at domain requirements
 - ⊗ may be achieved when also considering domain facets
 - ⊗ not covered in neither the example of Sect. nor in this section (i.e., Sect.)
 - ⊗ nor in the following two sections.
- ❖ We refer to [Bjø10a].

Example 14 Domain Requirements — Fitting:

- Example 13 hints at three possible sets of interface requirements:
 - ❖ (i) for a road pricing [sub-]system, as will be illustrated in Sect. ;
 - ❖ (ii) for a vehicle monitoring and control [sub-]system, and
 - ❖ (iii) for a toll-gate monitoring and control [sub-]system.
- The vehicle monitoring and control [sub-]system would focus on implementing the **vehicle** behaviour, see Items 128- 132 on Slide 208.
- The toll-gate monitoring and control [sub-]system would focus on implementing the **calculator** behaviour, see Items 134- 139 on Slide 213.


The fitting amounts to


- (a) making precise the (narrative and formal) texts that are specific to each of the three (i–iii) separate sub-system requirements are kept separate;
- (b) ensuring that (meaning-wise) shared texts that have different names for (meaning-wise) identical entities have these names renamed appropriately;
- (c) that these texts are subject to commensurate and ameliorated further requirements development;
- etcetera 

5. Interface and Derived Requirements

- We remind the listener that
 - ❖ ***interface requirements***
 - ⊗ can be expressed only using terms from
 - ⊗ both the domain
 - ⊗ and the machine ■
- Users are not part of the machine.
 - ❖ So no reference can be made to users, such as
 - ❖ “*the system must be user friendly*”,
 - ❖ and the like !¹⁶

¹⁶So how do we cope with the statement: “*the system must be user friendly*”? We refer to Sect. on Slide 284 for a discussion of this issue.


- By **interface requirements** we [also] mean
 - ❖ *requirements prescriptions*
which refines and extends the domain requirements
 - ❖ *by considering those requirements*
of the domain requirements whose
 - ⊗ *endurants (parts, materials) and*
 - ⊗ *perdurants (actions, events and behaviours)*
 - ❖ *are “shared”*
 - ❖ *between the domain and the machine*
(being requirements prescribed) 
 - ❖ The two *interface requirements* definitions above go hand-in-hand, i.e., complement one-another.

- By ***derived requirements*** we mean
 - ❖ *requirements prescriptions*
 - ❖ *which are expressed in terms of the machine concepts and facilities*
 - ❖ *introduced by the emerging requirements* 

5.1. Interface Requirements

5.1.1. Shared Phenomena

- By **sharing** we mean
 - ⋄ that some or all properties of an **endurant** is represented both
 - ⊗ in the domain and
 - ⊗ “inside” the machine, and
 - ⊗ that their machine representation
 - ⊗ must at suitable times
 - ⊗ reflect their state in the domain;and/or
 - ⋄ that an **action**
 - ⊗ requires a sequence of several “on-line” interactions
 - ⊗ between the machine (being requirements prescribed) and
 - ⊗ the domain, usually a person or another machine;and/or

- ❖ that an **event**
 - ⊗ arises either in the domain,
that is, in the environment of the machine,
 - ⊗ or in the machine,
 - ⊗ and need be communicated to the machine, respectively to the environment;and/or
- ❖ that a **behaviour** is manifested both
 - ⊗ by actions and events of the domain and
 - ⊗ by actions and events of the machine 

- So a systematic reading of the domain requirements shall
 - ⋄ result in an identification of all shared
 - ⊗ endurants,
 - * parts,
 - * materials and
 - * components;
 - and
 - ⊗ perdurants
 - * actions,
 - * events and
 - * behaviours.

- Each such shared phenomenon shall then be individually dealt with:
 - ❖ **endurant sharing** shall lead to interface requirements for data initialisation and refreshment as well as for access to endurant attributes;
 - ❖ **action sharing** shall lead to interface requirements for interactive dialogues between the machine and its environment;
 - ❖ **event sharing** shall lead to interface requirements for how such event are communicated between the environment of the machine and the machine; and
 - ❖ **behaviour sharing** shall lead to interface requirements for action and event dialogues between the machine and its environment.

5.1.1.1 Environment–Machine Interface:

- Domain requirements extension, Sect. 4.4,
 - ⋄ usually introduce new endurants into (i.e., ‘extend’ the) domain.
 - ⋄ Some of these endurants may become elements of the domain requirements.
 - ⋄ Others are to be projected “away”.
 - ⋄ Those that are let into the domain requirements
 - ⊗ either have their endurants represented, somehow, also in the machine,
 - ⊗ or have (some of) their properties, usually some attributes, accessed by the machine.

- ❖ Similarly for perdurants.
 - ⊗ Usually the machine representation of shared perdurants access (some of) their properties, usually some attributes.
- ❖ The interface requirements must spell out which domain extensions are shared.
- ❖ Thus domain extensions may necessitate a review of domain
 - ⊗ projection,
 - ⊗ instantiations and
 - ⊗ determination.

- In general, there may be several of the projection–eliminated parts (etc.) whose dynamic attributes need be accessed in the usual way, i.e., by means of `attr_XYZ_ch` channel communications (where `XYZ` is a projection–eliminated part attribute).

Example 15 Interface Requirements — Projected Extensions: We refer to Fig. 2 on Slide 189.

- We do not represent the GNSS system in the machine:
 - ❖ only its “effect”: the ability to record global positions
 - ❖ by accessing the GNSS attribute (channel):

channel

87 $\{\text{attr_TiGPos_ch}[vi] \mid vi:VI \cdot vi \in \text{xtr_VIs}(vs)\}: \text{TiGPos}$

- And we do not really represent the gate nor its sensors and actuator in the machine.
- But we do give an idealised description of the gate behaviour, see Items 134–139
- Instead we represent their dynamic gate attributes:
 - (97) the vehicle entry sensors (leftmost ■s),
 - (97) the vehicle identity sensor (center ■), and
 - (98) the vehicle exit sensors (rightmost ■s)
- by channels — we refer to Example 13 (Sect. , Slide 192):

channel

```

97      {attr_entry_ch[gi]|gi:Gl·xtr_eGlds(trn)} "enter"
97      {attr_exit_ch[gi]|gi:Gl·xtr_xGlds(trn)} "exit"
98      {attr_identity_ch[gi]|gi:Gl·xtr_Glds(trn)} TIVI ■■■

```

5.1.2. Shared Endurants

Example 16 Interface Requirements. Shared Endurants:

- The main shared endurants are
 - ⋄ the vehicles,
 - ⋄ the net (hubs, links, toll-gates) and
 - ⋄ the price calculator.
- As domain endurants hubs and links undergo changes,
 - ⋄ all the time,
 - ⋄ with respect to the values of several attributes:
 - ⊗ length, geodetic information, names,
 - ⊗ wear and tear (where-ever applicable),
 - ⊗ last/next scheduled maintenance (where-ever applicable),
 - ⊗ state and state space,
 - ⊗ and many others.

- Similarly for vehicles:
 - ✧ their position,
 - ✧ velocity and acceleration, and
 - ✧ many other attributes.
- We then come up with something like
 - ✧ hubs and links are to be represented as tuples of relations;
 - ✧ each net will be represented by a pair of relations
 - ⊗ a hubs relation and a links relation;
 - ⊗ each hub and each link may or will be represented by several tuples;
 - ✧ etcetera.
- In this database modeling effort it must be secured that “standard” operations on nets, hubs and links can be supported by the chosen relational database system

5.1.2.1 Data Initialisation:

- In general, one must prescribe data initialisation, that is provision for
 - ❖ an interactive user interface dialogue with a set of proper display screens,
 - ⊗ one for establishing net, hub or link attributes names and their types, and, for example,
 - ⊗ two for the input of hub and link attribute values.
 - ❖ Interaction prompts may be prescribed:
 - ⊗ next input,
 - ⊗ on-line vetting and
 - ⊗ display of evolving net, etc.
 - ❖ These and many other aspects may therefore need prescriptions.

Example 17 Interface Requirements. Shared Endurant Initialisation:

- The domain is that of the road net, $n:N$.
- By ‘shared road net initialisation’ we mean the “ab initio” establishment, “from scratch”, of a data base recording the properties of all links, $l:L$, and hubs, $h:H$,
 - ❖ their unique identifications, **uid**_L(l) and **uid**_H(h),
 - ❖ their mereologies, **obs_mereo**_L(l) and **obs_mereo**_H(h),
 - ❖ the initial values of all their static and programmable attributes and
 - ❖ the access values, that is, channel designations for all other attribute categories.

- 140 There are r_l and r_h “recorders” recording link, respectively hub properties – with each recorder having a unique identity.
- 141 Each recorder is charged with the recording of a set of links or a set of hubs according to some partitioning of all such.
- 142 The recorders inform a central data base, `net_db`, of their recordings $(ri, hol, (u_j, m_j, attrs_j))$ where
- 143 ri is the identity of the recorder,
- 144 hol is either a hub or a `link` literal,
- 145 $u_j = \mathbf{uid_L}(l)$ or $\mathbf{uid_H}(h)$ for some link or hub,
- 146 $m_j = \mathbf{obs_mereo_L}(l)$ or $\mathbf{obs_mereo_H}(h)$ for that link or hub and
- 147 $attrs_j$ are *attributes* for that link or hub — where *attributes* is a function which “records” all respective static and dynamic attributes (left undefined).

type

140 RI

value

140 rl,rh:NAT **axiom** $rl > 0 \wedge rh > 0$

type

142 $M = RI \times \text{"link"} \times LNK \mid RI \times \text{"hub"} \times HUB$

142 $LNK = LI \times HI\text{-set} \times LATTRS$

142 $HUB = HI \times LI\text{-set} \times HATTRS$

value

```

141 partitioning: L-set  $\rightarrow$  Nat  $\rightarrow$  (L-set)* | H-set  $\rightarrow$  Nat  $\rightarrow$  (H-set)*
141 partitioning(s)(r) as sl
141 post: len sl = r  $\wedge$   $\bigcup$  elems sl = s
141       $\wedge \forall si,sj:(L\text{-set}|H\text{-set}) \cdot$ 
141       $si \neq \{\} \wedge sj \neq \{\} \wedge \{si,sj\} \subseteq \text{elems } ss \Rightarrow si \cap sj = \{\}$ 

```


148 The $r_l + r_h$ recorder behaviours interact with the one net_db behaviour

channel

148 r_db: $RI \times (LNK|HUB)$

value

148 link_rec: $RI \rightarrow L\text{-set} \rightarrow \text{out r_db Unit}$

148 hub_rec: $RI \rightarrow H\text{-set} \rightarrow \text{out r_db Unit}$

148 net_db: $\text{Unit} \rightarrow \text{in r_db Unit}$

- 149 The data base behaviour, `net_db`, offers to receive messages from the link and hub recorders.
- 150 The data base behaviour, `net_db`, deposits these messages in respective variables.
- 151 Initially there is a net, $n : N$,
- 152 from which is observed its links and hubs.
- 153 These sets are partitioned into r_l , respectively r_h length lists of non-empty links and hubs.
- 154 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.
- 155 We construct, for technical reasons, as the listener will soon see, disjoint lists of link, respectively hub recorder identities.

value

149 net_db:

variable

150 lnk_db: (RI \times LNK)-set

150 hub_db: (RI \times HUB)-set

value

151 n:N

152 ls:L-set = obs_Ls(obs_LS(n))

152 hs:H-set = obs_Hs(obs_HS(n))

153 ls:(L-set)* = partitioning(ls)(rl)

153 hl:(H-set)* = partitioning(hs)(rh)

155 rll:RI* **axiom** len rll = rl = **card** elems rll

155 rhl:RI* **axiom** len rhl = rh = **card** elems rhl

154 ab_initio_data: **Unit** \rightarrow **Unit**

154 ab_initio_data() \equiv

154 || {lnk_rec(rll[i])(ls[i])|i:Nat.1 \leq i \leq rl} ||

154 || {hub_rec(rhl[i])(hl[i])|i:Nat.1 \leq i \leq rh}

154 || net_db()

156 The link and the hub recorders are near-identical behaviours.

157 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

158 the unique identifier,

159 the mereology, and

160 the attributes.

161 These “data” are sent, as a message, prefixed the senders identity, to the data base behaviour.

162 We presently leave the ... unexplained.

value

```
148 link_rec:  $RI \rightarrow L\text{-set} \rightarrow \mathbf{Unit}$ 
156 link_rec(ri,ls)  $\equiv$ 
157   for  $\forall l:L.l \in ls$  do uid_L(l)
158     let lnk = (uid_L(l),
159               obs_mereo_L(l),
160               attributes(l)) in
161     rdb ! (ri,"link",lnk);
162   ... end
157 end
```

```
148  hub_rec: RI  $\times$  H-set  $\rightarrow$  Unit
156  hub_rec(ri,hs)  $\equiv$ 
157      for  $\forall h:H.h \in hs$  do uid_H(h)
158          let hub = (uid_L(h),
159                      obs_mereo_H(h),
160                      attributes(h)) in
161          rdb ! (ri,"hub",hub);
162      ... end
157  end
```


163 The net_db data base behaviour revolves around a seemingly
“never-ending” cyclic process.

164 Each cycle “starts” with acceptance of some,
165 either link or hub data.

166 If link data then it is deposited in the link data base,
167 if hub data then it is deposited in the hub data base.

value

```
163 net_db() ≡  
164   let (ri,hol,data) = r_db ? in  
165   case hol of  
166     "link" → ... ; lnk_db := lnk_db ∪ (ri,data),  
167     "hub"  → ... ; hub_db := hub_db ∪ (ri,data)  
165   end end ;  
163'   ... ;  
163   net_db()
```


- The above model is an idealisation.
 - ⋄ It assumes that the link and hub data represent a well-formed net.
 - ⋄ Included in this well-formedness are the following issues:
 - ⊗ (a) that all link or hub identifiers are communicated exactly once,
 - ⊗ (b) that all mereologies refer to defined parts, and
 - ⊗ (c) that all attribute values lie within an appropriate value range.
 - ⋄ If we were to cope with possible recording errors then we could, for example, extend the model as follows:
 - ⊗ (i) when a link or a hub recorder has completed its recording then it increments an initially zero counter (say at formula Item 162);
 - ⊗ (ii) before the net data base recycles it tests whether all recording sessions has ended and then proceeds to check the data base for well-formedness issues (a–b–c) (say at formula Item 163') 

- The above example illustrates the ‘interface’ phenomenon:
 - ⋄ In the formulas, for example, we show both
 - ⊗ manifest domain entities, viz., n, l, h etc., and
 - ⊗ abstract (required) software objects, viz., $(ui, me, attrs)$.

5.1.2.2 Data Refreshment:

- One must also prescribe data refreshment:
 - ❖ an interactive user interface dialogue with a set of proper display screens
 - ⊗ one for selecting the updating of net, of hub or of link attribute names and their types and, for example,
 - ⊗ two for the respective update of hub and link attribute values.
 - ❖ Interaction-prompts may be prescribed:
 - ⊗ next update,
 - ⊗ on-line vetting and
 - ⊗ display of revised net, etc.
 - ❖ These and many other aspects may therefore need prescriptions.

5.1.3. Shared Perdurants

- We can expect that
 - ❖ for every part in the domain
 - ❖ that is shared with the machine and
 - ❖ for which there is a corresponding behaviour of the domain
 - ❖ there might be a corresponding process of the machine.
- If a projected, instantiated, ‘determined’ and possibly extended domain part is dynamic,
then it is definitely a candidate for being shared
and having an associated machine process.

- We now illustrate the concept of shared perdurants
 - ❖ via the domain requirements extension example
 - ❖ of Sect. 4.4, i.e. Example 13 Slides 181–217.

Example 18 Interface Requirements — Shared Behaviours: Road Pricing Calculator Behaviour:

168 The road-pricing calculator alternates between offering to accept
communication from

169 either any vehicle

170 or any toll-gate.

168 $\text{calc: ci:CI} \times (\text{vis:VI-set} \times \text{gis:GI-set}) \rightarrow \text{RLF} \rightarrow \text{TRM} \rightarrow$

169 $\text{in } \{v_c_ch[ci,vi] \mid vi:VI.vi \in \text{vis}\},$

170 $\{g_c_ch[ci,gi] \mid gi:GI.gi \in \text{gis}\} \quad \mathbf{Unit}$

168 $\text{calc}(ci,(\text{vis},\text{gis}))(\text{rlf})(\text{trm}) \equiv$

169 $\text{react_to_vehicles}(ci,(\text{vis},\text{gis}))(\text{rlf})(\text{trm})$

168 \square

170 $\text{react_to_gates}(ci,(\text{vis},\text{gis}))(\text{rlf})(\text{trm})$

168 $\mathbf{pre} \text{ ci} = \text{ci}_{\mathcal{E}} \wedge \text{vis} = \text{vis}_{\mathcal{E}} \wedge \text{gis} = \text{gis}_{\mathcal{E}}$

171 If the communication is from a vehicle inside the toll-road net
 172 then its toll-road net position, vp , is found from the road location
 function, rlf ,

173 and the calculator resumes its work with the traffic map, trm ,
 suitably updated,

174 otherwise the calculator resumes its work with no changes.

```

169  react_to_vehicles( $ci, (vis, gis), vplf$ )( $trm$ )  $\equiv$ 
169      let ( $vi, (\tau, lpos)$ ) =  $\sqcap \{v\_c\_ch[ci, vi] ? | vi:Vl.vi \in vis\}$  in
171          if  $vi \in \text{dom } trm$ 
172              then let  $vp = vplf(lpos)$  in
173                   $\text{calc}(ci, (vis, gis), vplf)(trm \dagger [vi \mapsto trm \hat{\langle (\tau, vp) \rangle}])$  end
174              else  $\text{calc}(ci, (vis, gis), vplf)(trm)$  end end
```

175 If the communication is from a gate,
176 then that gate is either an entry gate or an exit gate;
177 if it is an entry gate
178 then the calculator resumes its work with the vehicle (that passed
the entry gate) now recorded, afresh, in the traffic map, trm.
179 Else it is an exit gate and
180 the calculator concludes that the vehicle has ended its
to-be-paid-for journey inside the toll-road net, and hence to be
billed;
181 then the calculator resumes its work with the vehicle now removed
from the traffic map, trm.


```

170  react_to_gates(ci,(vis,gis),vplf)(trm)  $\equiv$ 
170    let (ee,( $\tau$ ,(vi,li))) =  $\sqcap$  {g_c_ch[ci,gi]?|gi:Gl.gi $\in$  gis} in
176    case ee of
177      "Enter"  $\rightarrow$ 
178        calc(ci,(vis,gis),vplf)(trm  $\cup$  [vi $\mapsto$   $\langle(\tau, \text{SonL}(li))\rangle$  ]),
179      "Exit"  $\rightarrow$ 
180        billing(vi,trm(vi) $\wedge$   $\langle(\tau, \text{SonL}(li))\rangle$ );
181        calc(ci,(vis,gis),vplf)(trm  $\setminus$  {vi}) end end

```


The \sqcap operator is the slide version of the non-deterministic choice operator \sqcap .

- The above behaviour is the one for which we are to design software



5.2. Derived Requirements


Definition 18 *Derived Perdurant*: By a *derived perdurant* we shall understand

- a perdurant which is not shared with the domain,
- but which focus on exploiting facilities
- of the software or hardware of the machine 

- “Exploiting facilities of the software”, to us,
 - ❖ means that requirements,
 - ❖ imply the presence, in the machine, of concepts (i.e., hardware and/or software),
 - ❖ and that it is these concepts
 - ❖ that the **derived requirements** “rely” on.
- We illustrate all three forms of perdurant extensions:
 - ❖ derived actions,
 - ❖ derived events and
 - ❖ derived behaviours.

5.2.1. Derived Actions

Definition 19 *Derived Action*:

- By a *derived action* we shall understand
 - ⋄ (a) a conceptual action
 - ⋄ (b) that calculates
 - ⊗ a usually non-Boolean valued property from,
 - ⊗ and possibly changes to
 - ⋄ (c) a machine behaviour state
 - ⋄ (d) as instigated by some actor 

Example 19 Domain Requirements. Derived Action: Tracing Vehicles:

- The example is based on the *Road Pricing Calculator Behaviour* of Example 18 on Slide 262.
 - ⋄ The “external” actor, i.e., a user of the *Road Pricing Calculator* system
 - ⋄ wishes to trace specific vehicles “cruising” the toll-road.
 - ⋄ That user (a *Road Pricing Calculator* staff),
 - ⊗ issues a command to the *Road Pricing Calculator* system,
 - ⊗ with the identity of a vehicle
 - ⊗ not already being traced.
 - ⋄ As a result the *Road Pricing Calculator* system
 - ⊗ augments a possibly void trace of
 - ⊗ the timed toll-road positions of vehicles.

- We augment the definition of the *calculator* definition Items 168–181, Slides 262–265.

182 Traces are modeled by a pair of dynamic attributes:

- a. as a programmable attribute, *tra:TRA*, of the set of identifiers of vehicles being traced, and
- b. as a reactive attribute, *vdu:VDU*¹⁷, that maps vehicle identifiers into time-stamped sequences of simple vehicle positions, i.e., as a subset of the *trm:TRM* programmable attribute.

183 The actor-to-calculator *begin* or *end* trace command, *cmd:Cmd*, is modeled as an autonomous dynamic attribute of the *calculator*.

184 The *calculator* signature is furthermore augmented with the three attributes mentioned above.

¹⁷ VDU: visual display unit

185 The occurrence and handling of an actor trace command is modeled as a non-deterministic external choice and a *react_to_trace_cmd* behaviour.

186 The reactive attribute value (*attr_vdu_ch?*) is that subset of the traffic map (*trm*) which records just the time-stamped sequences of simple vehicle positions being traced (*tra*).

type

182a. $\text{TRA} = \text{VI-set}$

182b. $\text{VDU} = \text{TRM}$

183 $\text{Cmd} = \text{BTr} \mid \text{ETr}$

183 $\text{BTr} :: \text{VI}$

183 $\text{ETr} :: \text{VI}$

value


```

184  calc: ci:CI × (vis:VI-set × gis:GI-set) → RLF → TRM → TRA
169,170  in {v_c_ch[ci,vi]|vi:VI·vi ∈ vis},
169,170    {g_c_ch[ci,gi]|gi:GI·gi ∈ gis},
185,186    attr_cmd_ch,attr_vdu_ch  Unit
168  calc(ci,(vis,gis))(rlf)(trm)(tra) ≡
169    react_to_vehicles(ci,(vis,gis),)(rlf)(trm)(tra)
170    □ react_to_gates(ci,(vis,gis))(rlf)(trm)(tra)
185    □ react_to_trace_cmd(ci,(vis,gis))(rlf)(trm)(tra)
168    pre ci = ciℰ ∧ vis = visℰ ∧ gis = gisℰ
186    axiom □ attr_vdu_ch[ci]? = trm|tra

```


- 187 The *react_to_trace_cmd* alternative behaviour is either a "Begin" or an "End" request which identifies the affected vehicle.
- 188 If it is a "Begin" request
- 189 and the identified vehicle is already being traced then we do not prescribe what to do !
- 190 Else we resume the calculator behaviour, now recording that vehicle as being traced.
- 191 If it is an "End" request
- 192 and the identified vehicle is already being traced then we do not prescribe what to do !
- 193 Else we resume the calculator behaviour, now recording that vehicle as no longer being traced.


```
187 react_to_trace_cmd(ci,(vis,gis))(vplf)(trm)(tra) ≡  
187   case attr_cmd_ch[ci]? of  
188,189,190   mkBTr(vi) → if vi ∈ tra then chaos else calc(ci,(vis,gis))(vplf)(trm)(tra ∪ {vi}) end  
191,192,193   mkETr(vi) → if vi ∉ tra then chaos else calc(ci,(vis,gis))(vplf)(trm)(tra \ {vi}) end  
187   end
```

- The above behaviour, Items 168–193, is the one for which we are to design software 

- Example 19 exemplifies an action requirement as per definition 19:
 - ❖ (a) the action is conceptual,
it has no physical counterpart in the domain;
 - ❖ (b) it calculates (186) a visual display (vdu);
 - ❖ (c) the vdu value is based on a conceptual notion of traffic road maps (trm), an element of the calculator state;
 - ❖ (d) the calculation is triggered by an actor (attr_cmd_ch).

5.2.2. Derived Events

Definition 20 *Derived Event*: By a *derived event* we shall understand

- (a) a conceptual event,
- (b) that calculates a property or some non-Boolean value
- (c) from a machine behaviour state change 

Example 20 Domain Requirements. Derived Event: Current Maximum Flow:

- The example is based on the *Road Pricing Calculator Behaviour* of Examples 19 and 18 on Slide 262.
 - ⋄ By “the current maximum flow” we understand
 - ⊗ a time-stamped natural number, the number
 - ⊗ representing the highest number of vehicles which
 - * at the time-stamped moment cruised
 - * or now cruisesaround the toll-road net.
- We augment the definition of the *calculator* definition Items 168–193, Slides 262–274.

- 194 We augment *calculator* signature with
- 195 a time-stamped natural number valued dynamic programmable attribute, $(t:\mathbb{T}, max:Max)$.
- 196 Whenever a vehicle enters the toll-road net, through one of its [entry] gates,
- a. it is checked whether the resulting number of vehicles recorded in the *road traffic map* is higher than the hitherto *maximum* recorded number.
 - b. If so, that programmable attribute has its number element “upped” by one.
 - c. Otherwise not.
- 197 No changes are to be made to the `react_to_gates` behaviour (Items 170–181 Slide 265) when a vehicle exits the toll-road net.

type

195 $\text{MAX} = \mathbb{T} \times \text{NAT}$

value

184,194 $\text{calc: ci:CI} \times (\text{vis:VI-set} \times \text{gis:GI-set}) \rightarrow \text{RLF} \rightarrow \text{TRM} \rightarrow \text{TRA} \rightarrow \text{MAX}$

169,170 **in** $\{v_c_ch[ci,vi] \mid vi:VI \cdot vi \in \text{vis}\},$

169,170 $\{g_c_ch[ci,gi] \mid gi:GI \cdot gi \in \text{gis}\},$

169,170 $\text{attr_cmd_ch, attr_vdu_ch} \quad \textbf{Unit}$

...

170 $\text{react_to_gates}(ci,(\text{vis},\text{gis}))(\text{vplf})(\text{trm})(\text{tra})(t,m) \equiv$

170 $\text{let } (ee,(\tau,(vi,li))) = \square \{g_c_ch[ci,gi] \mid gi:GI \cdot gi \in \text{gis}\} \text{ in}$

176 **case ee of**

177 "Enter" \rightarrow


196 $\text{calc}(ci,(\text{vis},\text{gis}))(\text{vplf})(\text{trm} \cup [vi \mapsto \langle (\tau, \text{SonL}(li)) \rangle])(\text{tra})(\tau, \text{if card dom trm} = m \text{ then } m+1 \text{ else } m \text{ end},$

197 "Exit" \rightarrow

180 $\text{billing}(vi, \text{trm}(vi) \wedge \langle (\tau, \text{SonL}(li)) \rangle);$

181 $\text{calc}(ci,(\text{vis},\text{gis}))(\text{vplf})(\text{trm} \setminus \{vi\})(\text{tra})(t,m) \text{ end}$

176 **end**

- The above behaviour, Items 168 on Slide 262 through 196c. on the preceding slide, is the one for which we are to design software 

- Example 20 exemplifies a derived event requirement as per Definition 20:
 - ❖ (a) the event is conceptual, it has no physical counterpart in the domain;
 - ❖ (b) it calculates (196b.) the max value based on a conceptual notion of traffic road maps (trm),
 - ❖ (c) which is an element of the calculator state.

5.2.3. No Derived Behaviours

- There are no derived behaviours. The reason is as follows.
 - ❖ Behaviours are associated with parts.
 - ❖ A possibly ‘derived behaviour’ would entail the introduction of an ‘associated’ part.
 - ❖ And if such a part made sense it should – in all likelihood – already have been either a proper domain part or become a domain extension.
- If the domain–to–requirements engineer insist on modeling some interface requirements as a process then we consider that a technical matter, a choice of abstraction.

5.3. Discussion

5.3.1. Derived Requirements

- Formulation of derived actions or derived events usually involves technical terms not only from the domain but typically from such conceptual ‘domains’ as mathematics, economics, engineering or their visualisation.

- Derived requirements may, for some requirements developments, constitute “sizable” requirements compared to “all the other” requirements.
 - ❖ For their analysis and prescription it makes good sense to first having developed “the other” requirements:
 - ⊗ domain,
 - ⊗ interface and
 - ⊗ machine requirements.
 - ❖ The treatment of the present paper does not offer special techniques and tools for the conception, &c., of derived requirements.
 - ❖ Instead we refer to the seminal works of [DvLF93, Lau02, van09].


5.3.2. Introspective Requirements

- Humans, including human users are, in this paper, considered to never be part of the domain for which a requirements prescription is being developed.
 - ❖ If it is necessary to involve humans
 - ❖ in the domain description or the requirements prescription
 - ❖ then their prescription is to reflect assumptions
 - ❖ upon whose behaviour the machine rely.


- It is therefore that we, above, have stated, in passing, that we cannot accept requirements of the kind:
“*the machine must be user friendly*”,
because, in reality, it means “*the user must rely upon the machine being ‘friendly’*”
whatever that may mean.
- We are not requirements prescribing humans, nor their sentiments !

6. Machine Requirements

Definition 21 *Machine Requirements:* By *machine requirements* we shall understand

- such requirements
- which can be expressed “sôlely” using terms
- from, or of **the machine** 

Definition 22 *The Machine:* By the *machine* we shall understand

- the hardware
- and software
- to be built from the requirements 

- The expression
 - ⋄ which can be expressed
 - ⋄ “sôlely” using terms
 - ⋄ from, or of the machine

shall be understood with “a grain of salt”.


- ⋄ Let us explain.
 - ⊗ The *machine requirements* statements
 - ⊗ may contain references to domain entities
 - ⊗ but these are meant to be generic references,
 - ⊗ that is, references to certain classes of entities in general.

We shall illustrate this “genericity” in some of the examples below.

- Analysis of different kinds of requirements,
 - ⋄ such as exemplified
 - ⋄ but not so classified
 - ⋄ in seminal textbooks [Lau02, van09]
 - ⋄ suggests the following categories of machine requirements:
 - ⊗ (i) *technology requirements* and
 - ⊗ (ii) *development requirements*.

6.1. Technology Requirements


Definition 23 *Technology Requirements:* By *technology requirements* we shall understand

- such machine requirements
- which primarily focus on alleviating
 - ❖ physical deficiencies of the hardware or
 - ❖ inefficiencies of the softwareof the machine —
- cf. Items (i–ii), i.e., Sects. – below 


- We shall, in particular, consider the following kinds of technology requirements:
 - ❖ (i) *performance requirements* and
 - ❖ (ii) *dependability requirements*
- with dependability requirements being concerned with either
 - ❖ (a) *accessibility*,
 - ❖ (b) *availability*,
 - ❖ (c) *integrity*,
 - ❖ (d) *reliability*,
 - ❖ (e) *safety*,
 - ❖ (f) *security* and/or
 - ❖ (g) *robustness*.

6.1.1. Performance Requirements

Definition 24 *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 21 Machine Requirements. Technology: Performance:

- The road pricing system shall be able
 - ❖ to keep records of up to 50.000 vehicles at any time,
 - ❖ to record up to 10.000 vehicle positions per second, and
 - ❖ to bill up to 1000 (distinct) vehicles per second.
- A vehicle is assumed to access the road pricing calculator with a mean time between accesses of 5 seconds.
- A toll-gate is assumed to access the road pricing calculator with a mean time between accesses of 5 seconds 


6.1.2. Dependability Requirements

- Dependability is a complex notion.


6.1.2.1 Failures, Errors and Faults

- To properly define the concept of *dependability* we need first introduce and define the concepts of
 - ❖ *failure*,
 - ❖ *error*, and
 - ❖ *fault*.


Definition 25 *Failure*:

- A machine *failure* occurs
- when the delivered service
- deviates from fulfilling the machine function,
- the latter being what the machine is aimed at 


Definition 26 **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 27 *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*.

Definition 28 *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 29 *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.

- *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.
- We consider the following attributes:

- *Accessibility*
- *Availability*
- *Integrity*
- *Reliability*
- *Safety*
- *Security*

- 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 30 ***Dependability Attribute:*** By a *dependability attribute* we shall mean either one of the following:

- *accessibility,*
- *availability,*
- *integrity,*
- *reliability,*
- *robustness,*
- *safety and*
- *security.*

That is, a machine is dependable if it satisfies some degree of “mixture” of being accessible, available, having integrity, and being reliable, safe and secure

- The crucial term above is “satisfies”.
- The issue is: To what “degree”?
- As we shall see — in a later later lecture — to cope properly
 - ❖ with dependability requirements and
 - ❖ their resolutionrequires that we deploy
 - ❖ mathematical formulation techniques,
 - ❖ including analysis and simulation,from statistics (stochastics, etc.).

6.1.2.2 Accessibility

- Usually a desired, i.e., the required, computing system, i.e., the machine, will be used by many users — over “near-identical” time intervals.
- Their being granted access to computing time is usually specified, at an abstract level, as being determined by some internal nondeterministic choice, that is: essentially by “*tossing a coin*”!
- If such internal nondeterminism was carried over, into an implementation, some “*coin tossers*” might not get access to the machine “for a long- long time”.

Definition 31 *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 22 Machine Requirements. Technology: Accessibility:

- No vehicle access to the road pricing calculator shall wait more than 2 seconds.
- No toll-gate access to the road pricing calculator shall wait more than 2 seconds.


6.1.2.3 Availability

- Usually a desired, i.e., the required, computing system, i.e., the machine, will be used by many users — over “near-identical” time intervals.
- Once a user has been granted access to machine resources, usually computing time, that user’s computation may effectively make the machine unavailable to other users —
- by “going on and on and on”!

Definition 32 *Availability*: By *availability* — in the context of a machine being dependable — we mean

- its readiness for usage.
- That is, that some form of “*guaranteed percentage of computing time*” per time interval (or percentage of some other computing resource consumption)
- is achieved, for example, in the form of “*time slicing*” 

Example 23 Machine Requirements. Technology: Availability:


- We simplify the availability requirements due to the apparent simplicity of the vehicle movement records and billings.
 - ❖ The complete handling of the recording or billing of a vehicle movement shall be done without interference from other recordings or billings 

6.1.2.4 Integrity

Definition 33 *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***.

Example 24 Machine Requirements. Technology: Integrity:

- We do not require an explicit formulation of integrity.
- We instead refer to the
 - ❖ reliability,
 - ❖ safety,
 - ❖ security and
 - ❖ robustnessmeasures (below) 

6.1.2.5 Reliability

Definition 34 *Reliability*: A system being *reliable* — in the context of a machine being dependable — means


- some measure of continuous correct service,
- that is, measure of (mean) time to failure (MTTF)

Example 25 Machine Requirements. Technology: Reliability:


- A road pricing calculator shall have a MTTF of least 10^8 seconds or approx. 40 months.

6.1.2.6 Safety

Definition 35 *Safety*: By *safety* — in the context of a machine being dependable — we mean

- some measure of continuous delivery of service of
 - ❖ either correct service, or incorrect service after benign failure,
- that is: Measure of time to catastrophic failure 

Example 26 Machine Requirements. Technology: Safety:


- The road pricing system, now including the
 - ❖ vehicle global position system and the
 - ❖ toll-gate sensors and barrier actuator
- shall have
 - ❖ a mean time to catastrophic failure
 - ❖ equal to the MTTF, 10^8 seconds 

6.1.2.7 Security

We shall take a rather limited view of security. We are not including any consideration of security against brute-force terrorist attacks. We consider that an issue properly outside the realm of software engineering.

- Security, then, in our limited view, requires a notion of *authorised user*,
- with authorised users being fine-grained authorised to access only a well-defined subset of system resources (data, functions, etc.).
- An *unauthorised user* (for a resource) is anyone who is not authorised access to that resource.

Definition 36 *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 27 Machine Requirements. Technology: Security:


- We omit exemplifying road pricing system security 

6.1.2.8 Robustness

Definition 37 *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 28 Machine Requirements. Technology: Robustness:

- We restrict ourselves to consider only the software of the road pricing system.
 - ⋄ For every instance of
 - ⊗ restart after failure
 - ⊗ it shall be verified
 - ⊗ that all attributes have retained their appropriate values;
 - ⋄ and for every instance of
 - ⊗ software maintenance, see Sect. ,
 - ⊗ the whole system shall be verified, i.e.,
 - * tested,
 - * model checked and
 - * proven correct,
 - ⊗ to the same and full extent that the original system delivery was verified 


6.2. Development Requirements

Definition 38 *Development Requirement:*

- By *development requirements* we shall understand
 - ❖ (i) *process requirements* (Sect. 6.4.1),
 - ❖ (ii) *maintenance requirements* (Sect. 6.4.2),
 - ❖ (iii) *platform requirements* (Sect. 6.4.3),
 - ❖ (iv) *management requirements* (Sect. 6.4.4) and
 - ❖ (v) *documentation requirements* (Sect. 6.4.5) 

6.2.1. Process Requirements

Definition 39 *Process Requirement:*


- By a *development process requirements* we shall understand requirements which are concerned with the development process to be followed by the development engineers:
 - ❖ whether pursuing formal methods, and to which degree, and (compatibly)/or
 - ❖ whether pursuing best practices, and possible details thereof, and (compatibly)/or
 - ❖ whether adhering otherwise to established, e.g., IEEE standards,
 - ❖ etcetera 

Example 29 Machine Requirements. Development: Road Pricing System:

- The road pricing system is to be developed
 - ❖ according to the triptych approach;
 - ❖ based on, or developing itself, a generic transport domain description,
as per the approach outlined in [Bjø16b],
expressing suitable predicates about that description,
and testing, model checking and proving satisfaction of these verifications;
 - ❖ accurately detailing the requirements prescriptions as per the approach outlined in [Bjø16a, this paper (!)];
 - ❖ etcetera;
 - ❖ finally testing, model checking and proving satisfaction of $\mathcal{D}, \mathcal{S} \models \mathcal{R}$.

6.2.2. Maintenance Requirements

Definition 40 ***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 53 on Slide 337.

6.2.2.1 Adaptive Maintenance

Definition 41 *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 30 Machine Requirements. Development: Adaptive Maintenance:

- Road pricing system adaptive maintenance shall conclude with a full set of successful
 - ❖ formal software tests,
 - ❖ model checks, and
 - ❖ correctness proofs 

6.2.2.2 Corrective Maintenance

Definition 42 *Corrective Maintenance:* By *corrective maintenance* we understand such maintenance which


- corrects a software error ■

Example 31 Machine Requirements. Development: Corrective Maintenance:


- Road pricing system corrective maintenance shall conclude with a full set of successful
 - ❖ formal software tests,
 - ❖ model checks, and
 - ❖ correctness proofs ■

6.2.2.3 Perfective Maintenance

Definition 43 *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 32 Machine Requirements. Development: Perfective Maintenance:

- Road pricing system perfective maintenance shall conclude with a full set of successful
 - ❖ formal software tests,
 - ❖ model checks, and
 - ❖ correctness proofs 

6.2.2.4 Preventive Maintenance

Definition 44 *Preventive Maintenance:* By *preventive maintenance* we understand such maintenance which

- helps detect, i.e., forestall, future occurrence
- of software or hardware failures ■

Example 33 Machine Requirements. Development: Preventive Maintenance:

- Road pricing system preventive maintenance shall conclude with a full set of successful
 - ❖ formal software tests,
 - ❖ model checks, and
 - ❖ correctness proofs ■

6.2.2.5 Extensional Maintenance

Definition 45 *Extensional Maintenance*: By *extensional maintenance* we understand such maintenance which adds new functionalities to the software, i.e., which implements additional requirements ■


Example 34 Machine Requirements. Development: Extensional Maintenance:

- Road pricing system extensional maintenance shall conclude with a full set of successful
 - ❖ formal software tests,
 - ❖ model checks, and
 - ❖ correctness proofs ■

6.2.3. Platform Requirements

6.2.3.1 Delineation and Facets of Platform Requirements

Definition 46 *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 47 *Platform Requirements*: By *platform requirements* we mean a combination of the following:

- *execution platform requirements*,
- *demonstration platform requirements*,
- *development platform requirements* and
- *maintenance platform requirements*


6.2.3.2 Execution Platform

Definition 48 *Execution Platform Requirements:* By *execution platform requirements* we shall understand such machine requirements which

- detail the specific (other) software and hardware
- for the platform on which the software
- is to be executed 


6.2.3.3 Demonstration Platform

Definition 49 *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 


6.2.3.4 Development Platform

Definition 50 *Development Platform Requirements:* By *development platform requirements* we shall understand such machine requirements which

- detail the specific software and hardware
- for the platform on which the software
- is to be developed 

6.2.3.5 Maintenance Platform

Definition 51 *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 

Example 35 Machine Requirements. Development: Platform:


- The DDC [BGOR10] Ada Compiler [BO80, CO84, Oes86] development platform requirements were: the system
 - ❖ shall execute on the US Military Space computer MI1750 running its proprietary MI1750 operating system;
 - ❖ shall be developed on SUN Sparc Workstations using UNIX;
 - ❖ shall be demonstrated and maintained on the NASA Houston TX installation of MI1750 emulating SUN Sparc workstation software ■■■
 - ❖

6.2.4. Management Requirements

Definition 52 *Management Requirements:*

- By *management requirements* we shall understand requirements that express principles of
 - ❖ project preparation: staffing, budgetting and financing;
 - ❖ development: formal/informal, verification (testing, etc.);
 - ❖ product transfer: maketing, sales, maintenance;
 - ❖ etcetera 

Example 36 Machine Requirements. Development: Management:

- We shall refer to [Bjø11a, Believable Software Management]
- for the principles to be followed for the
- development phase(s) 


6.2.5. Documentation Requirements

Definition 53 *Software*: By *software* we shall understand


- not only *code* that may be the basis for executions by a computer,
- but also its full *development documentation*:

- ❖ the stages and steps of *application domain description*,
- ❖ the stages and steps of *requirements prescription*, and
- ❖ the stages and steps of *software design* prior to code,


with all of the above including all *validation* and *verification* (incl., *formal test* [test model, test suite, test result, etc.], *model-checking* and *proof*) *documents*.

- 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 54 *Documentation Requirements:* By documentation requirements

- we mean requirements
- of any of the software documents
- that together make up
 - ❖ software and
 - ❖ hardware¹⁸ 

Example 37 Machine Requirements. Development: Documentation:

- The road pricing system documentation requirements shall include
 - ❖ all of the software documents
 - ❖ implied by Definition 53 on Slide 337 above 

¹⁸— we omit a definition of what we mean by hardware such as the one we gave for software, cf. Definition 53 on Slide 337.

6.3. Discussion

TO BE WRITTEN

7. Conclusion

- Conventional requirements engineering considers the domain only rather implicitly.
 - ❖ Requirements gathering (‘acquisition’) is not structured by any pre-existing knowledge of the domain,
 - ❖ instead it is “structured” by a number of relevant techniques and tools [Jac01, van09, Jac10]
 - ❖ which, when applied, “fragment-by-fragment” “discovers” such elements of the domain that are immediately relevant to the requirements.

- The present work turns this requirements prescription process “up-side-down”.
 - ❖ Now the process is guided (“steered”, “controlled”) almost exclusively by the domain description
 - ❖ which is assumed to be existing before the requirements development starts.

- In conventional requirements engineering
 - ❖ many of the relevant techniques and tools can be said to take into account
 - ❖ *sociological* and *psychological* facets of gathering the requirements and
 - ❖ *linguistic* facets of expressing these requirements.
 - ❖ That is, the focus is rather much on the *process*.

- In the present paper's requirements “derivation” from domain descriptions
 - ❖ the focus is all the time on the descriptions and prescriptions,
 - ❖ in particular on their formal expressions and
 - ❖ the “transformation” of these.
 - ❖ That is (descriptions and) prescriptions are considered formal, *mathematical* objects.
 - ❖ That is, the focus is rather much on the *objects*.



- We conclude by briefly reviewing
 - ❖ what has been achieved,
 - ❖ present shortcomings,
 - ❖ possible research challenges, and
 - ❖ a few words on relations to “classical requirements engineering”.

7.0.0.1 What has been Achieved ?

- We have shown how to systematically “derive” initial aspects of requirements prescriptions from domain descriptions.
 - ⋄ The stages¹⁹ and steps²⁰ of this “derivation”²¹ are new.
 - ⋄ We claim that current requirements engineering approaches, although they may refer to a or the ‘domain’, are not really ‘serious’ about this:
 - ⊗ they do not describe the domain, and
 - ⊗ they do not base their techniques and tools
 - ⊗ on a reasoned understanding of the domain.

¹⁹(a) domain, (b) interface and (c) machine requirements

²⁰For domain requirements: (i) projection, (ii) instantiation, (iii) determination, (iv) extension and (v) fitting; etc.

²¹We use double quotation marks: “...” to indicate that the derivation is not automatable.

- ❖ In contrast we have identified, we claim, a logically motivated decomposition of
 - ⊗ requirements into three phases, cf. Footnote 19 on the preceding slide.,
 - ⊗ of domain requirements into five steps, cf. Footnote 20 on the previous slide., and
 - ⊗ of interface requirements, based on a concept of shared entities,
 - tentatively into (α) shared endurants, (β) shared actions, (γ) shared events, and (δ) shared behaviours
 - (with more research into the $(\alpha-\delta)$ techniques needed).

7.0.0.2 Present Shortcomings and Research Challenges:

- We see three shortcomings:
 - ⋄ (1) The “derivation” techniques have yet to consider “extracting” requirements from *domain facet descriptions*.
 - ⊗ We plan to rewrite [Bjø10a] and extend it to include requirements considerations.
 - ⊗ Only by including *domain facet descriptions*
 - * can we, in “deriving” *requirements prescriptions*,
 - * include failures of, for example, support technologies and humans,
 - * in the design of fault-tolerant software.

- ❖ (2) The “derivation” principles, techniques and tools should be given a formal treatment.
- ❖ (3) There is a serious need for relating the approach of the present paper to that of the seminal text book of [van09, Axel van Lamsweerde].
 - ⊗ [van09] is not being “replaced” by the present work. It tackles a different set of problems.
 - ⊗ We refer to the penultimate paragraph before the **Acknowledgement** closing.

7.0.0.3 Comparison to “Classical” Requirements Engineering:

- Except for a few, represented by two, we are not going to compare the contributions of the present paper with published journal or conference papers on the subject of requirements engineering.

- The reason for this is the following.
 - ❖ The present paper, rather completely, we claim, reformulates requirements engineering,
 - ❖ giving it a ‘foundation’, in *domain engineering*,
 - ❖ and then developing *requirements engineering* from there,
 - ❖ viewing requirements prescriptions as “derived” from domain descriptions.
 - ❖ We do not see any of the papers, except those reviewed below [JHJ07] and [DvLF93], referring in any technical sense to ‘domains’ such as we understand them.

4 [JHJ07, Deriving Specifications for Systems That Are Connected to the Physical World]

- The paper that comes closest to the present paper in its serious treatment of the [problem] domain as a precursor for requirements development is that of [JHJ07, Jones, Hayes & Jackson].
 - ❖ A purpose of [JHJ07] (Sect. 1.1, Page 367, last §) is to see “how little can one say” (about the problem domain) when expressing assumptions about requirements.
 - ❖ This is seen by [JHJ07] (earlier in the same paragraph) as in contrast to our form of domain modeling.
 - ❖ [JHJ07] reveals assumptions about the domain when expressing *rely guarantees* in tight conjunction with expressing the *guarantee* (requirements).
 - ❖ That is, analysing and expressing requirements, in [JHJ07], goes hand-in-hand with analysing and expressing fragments of the domain.

- The current paper takes the view that
 - ❖ since, as demonstrated in [Bjø16b], it is possible to model sizable aspects of domains,
 - ❖ then it would be interesting to study how one might “derive” — and which — requirements prescriptions from domain descriptions;
 - ❖ and having demonstrated that (i.e., the “how much can be derived”)
 - ❖ it seems of scientific interest to see how that new start (i.e., starting with a priori given domain descriptions or starting with first developing domain descriptions) can be combined with existing approaches, such as [JHJ07].

- We do appreciate the “tight coupling” of rely–guarantees of [JHJ07].
 - ❖ But perhaps one loses understanding the domain
 - ❖ due to its fragmented presentation.
 - ❖ If the ‘relies’ are not outright, i.e., textually directly expressed
 - ❖ in our domain descriptions, then they obviously
 - ❖ must be provable properties of what our domain descriptions express.

- Our, i.e., the present, paper — with its background in [Bjø16b, Sect. 4.7] —
 - ❖ develops — with a background in [Jac95, M.A. Jackson] —
 - ❖ a set of principles and techniques for the access of attributes.
 - ❖ The “discovery” of the CM and SG channels of [JHJ07] and of the type of their messages,
 - ❖ seems, compared to our approach, less systematic.

- Also, it is not clear how the [JHJ07] case study “scales” up to a larger domain.
 - ❖ The *sluice gate* of [JHJ07] is but part of a large (‘irrigation’) system of reservoirs (water sources), canals, sluice gates and the fields (water sinks) to be irrigated.
 - ❖ We obviously would delineate such a larger system
 - ❖ and research & develop an appropriate, both informal, a narrative, and formal domain description for such a class of irrigation systems based on assumptions of precipitation and evaporation.
 - ❖ Then the users’ requirements, in [JHJ07], that the sluice gate, over suitable time intervals, is open 20% of the time and otherwise closed, could now be expressed more pertinently, in terms of the fields being appropriately irrigated.

7.0.0.5 [DvLF93, Goal-directed Requirements Acquisition]

- outlines an approach to requirements acquisition that starts with fragments of domain description.
 - ❖ The domain description is captured in terms of predicates over *actors, actions, events, entities* and (their) *relations*.

- ❖ Our approach to domain modeling differs from that of [DvLF93] as follows:
 - ⊗ Agents, actions, entities and relations are, in [DvLF93], seen as specialisations of a concept of *objects*.
 - ⊗ The nearest analogy to relations, in [Bjø16b], as well as in this paper, is the signatures of perdurants.
 - ⊗ Our ‘agents’ relate to discrete endurants, i.e., parts, and are the behaviours that evolve around these parts: one agent per part !
 - ⊗ [DvLF93] otherwise include describing parts, relations between parts, actions and events much like [Bjø16b] and this paper does.

- [DvLF93] then introduces a notion of *goal*.
 - ❖ A **goal**, in [DvLF93], is defined as
 - ❖ "*a nonoperational objective*
 - ❖ *to be achieved by the desired system.*
 - ❖ *Nonoperational means that the objective is not formulated in terms*
 - ❖ *of objects and actions “available” to some agent of the system*

22"

²²We have reservations about this definition: Firstly, it is expressed in terms of some of the “things” it is not! (To us, not a very useful approach.) Secondly, we can imagine goals that are indeed formulated in terms of objects and actions ‘available’ to some agent of the system. For example, wrt. the ongoing library examples of [DvLF93], *the system shall automate the borrowing of books*, etcetera. Thirdly, we assume that by “‘available’ to some agent of the system” is meant that these agents, actions, entities, etc., are also required.

- [DvLF93] then goes on to exemplify goals.
 - ⋄ In this, the current paper, we are not considering *goals*, also a major theme of [van09].²³
 - ⋄ Typically the expression of goals of [DvLF93, van09], are “within” computer & computing science and involve the use of temporal logic.²⁴

²³An example of a goal — for the road pricing system — could be that of

- ⊗ *shortening travel times of motorists,*
- ⊗ *reducing gasoline consumption and air pollution,*
- ⊗ *while recouping investments on toll-road construction.*
- ⋄ We consider techniques for ensuring the above kind of goals
 - ⊗ “outside” the realm of computer & computing science
 - ⊗ but “inside” the realm of operations research (OR) —
 - ⊗ while securing that the OR models are commensurate with our domain models.

²⁴In this paper we do not exemplify goals, let alone the use of temporal logic. We cannot exemplify all aspects of domain description and requirements prescription, but, if we were, would then use the temporal logic of [ZH04, The Duration Calculus].

- ❖ *"Constraints are operational objectives*
 - ⊗ *to be achieved by the desired (i.e., required) system, . . . ,*
 - ⊗ *formulated in terms of objects and actions “available”*
 - ⊗ *to some agents of the system. . . .*
- ❖ *Goals are made operational through constraints. . . .*
- ❖ *A constraint operationalising a goal amounts to some abstract “implementation” of this goal” [DvLF93].*
- ❖ [DvLF93] then goes on to express goals and constraints operationalising these.
- ❖ [DvLF93] is a fascinating paper²⁵ as it shows how to build goals and constraints on domain description fragments.

²⁵— that might, however, warrant a complete rewrite.



- These papers, [JHJ07] and [DvLF93], as well as the current paper,
 - ✧ together with such seminal monographs as [ZH04, OD08, van09],
 - ✧ clearly shows that there are many diverse ways
 - ✧ in which to achieve precise requirements prescriptions.
 - ✧ The [ZH04, OD08] monographs primarily study the $\mathcal{D}, \mathcal{S} \models \mathcal{R}$ specification and proof techniques from the point of view of the specific tools of their specification languages²⁶.

²⁶The Duration Calculus [DC], respectively DC, Timed Automata and Z

- ❖ Physics, as a natural science, and its many engineering ‘renditions’,
are manifested in many separate sub-fields: Electricity, mechanics, statics, fluid dynamics — each with further sub-fields.
- ❖ It seems, to this author, that there is a need to study
- ❖ the [ZH04, OD08, van09] approaches
and the approach taken in this paper
in the light of identifying sub-fields of requirements engineering.
- ❖ The title of the present paper suggests one such sub-field.

7.0.0.6 Acknowledgments:

- This paper has been many years underway.
- Earlier versions have been the basis for (“innumerable”) PhD lectures and seminars around the world — after I retired, in 2007, from The Technical University of Denmark.
- I thank the many organisers of those events for their willingness to “hear me out”:
 - ❖ Jin Song Dong, NUS, Singapore;
 - ❖ Kokichi Futatsugi and Kazuhiro Ogata, JAIST, Japan;
 - ❖ Dominique Méry, Univ. of Nancy, France;
 - ❖ Franz Wotawa and Bernhard K. Aichernig, Techn. Univ. of Graz, Austria;
 - ❖ Wolfgang J. Paul, Univ. of Saarland, Germany;

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

- ❖ Sun Meng, Peking Univ., China;
- ❖ He JiFeng and Zhu HuiBiao, East China Normal Univ., Shanghai, China;
- ❖ Zhou Chaochen, Lin Huimin and Zhan Naijun, Inst. of Softw., CAS, Beijing, China;
- ❖ Victor P. Ivannikov, Inst. of Sys. Prgr., RAS, Moscow, Russia;
- ❖ Luís Soares Barbosa and Jose Nuno Oliveira, Univ. of Minho, Portugal; and
- ❖ Steeve McKeever and Andreas Hamfeldt, Uppsala University.

/sf

7.1. Bibliographical Notes

- I have thought about domain engineering for more than 20 years.
- But serious, focused writing only started to appear since [Bjø06, Part IV] — with [Bjø03, Bjø97] being exceptions:
 - ✧ [Bjø07] suggests a number of domain science and engineering research topics;
 - ✧ [Bjø10a] covers the concept of domain facets;
 - ✧ [BE10] explores compositionality and Galois connections.
 - ✧ [Bjø08, Bjø10c] show how to systematically, but, of course, not automatically, “derive” requirements prescriptions from domain descriptions;

- ❖ [Bjø11a] takes the triptych software development as a basis for outlining principles for believable software management;
- ❖ [Bjø09, Bjø14a] presents a model for Stanisław Leśniewski's [CV99] concept of mereology;
- ❖ [Bjø10b, Bjø11b] present an extensive example and is otherwise a precursor for the present paper;
- ❖ [Bjø11c] presents, based on the `Triptych` view of software development as ideally proceeding from domain description via requirements prescription to software design, concepts such as software demos and simulators;
- ❖ [Bjø13] analyses the `Triptych`, especially its domain engineering approach, with respect to [Mas43, Mas54, Maslow]'s and [PS04, Peterson's and Seligman's]'s notions of humanity: how can computing relate to notions of humanity;

- ❖ the first part of [Bjø14b] is a precursor for [Bjø16b] with the second part of [Bjø14b] 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 with
- ❖ [Bjø14c] focus on domain safety criticality.
- ❖ The present paper, [Bjø16a], marks, for me, a high point, with
- ❖ [Bjø16b] now constituting the base introduction to domain science & engineering.

7.2. References

- [BE10] Dines Bjørner and Asger Eir. Compositionality: Ontology and Mereology of Domains. Some Clarifying Observations in the Context of Software Engineering in July 2008, eds. Martin Steffen, Dennis Dams and Ulrich Hannemann. In *Festschrift for Prof. Willem Paul de Roever Concurrency, Compositionality, and Correctness*, volume 5930 of *Lecture Notes in Computer Science*, pages 22–59, Heidelberg, July 2010. Springer.
- [BGOR10] Dines Bjørner, Chr. Gram, Ole N. Oest, and Leif Rystromb. Dansk Datamatik Center. In Benkt Wangler and Per Lundin, editors, *History of Nordic Computing*, Stockholm, Sweden, 18-20 October 2010. Springer.
- [BjØ97] Dines Bjørner. Michael Jackson’s Problem Frames: Domains, Requirements and Design. In Li ShaoYang and Michael Hinchley, editors, *ICFEM’97: International*

Conference on Formal Engineering Methods, Los Alamitos, November 12–14 1997. IEEE Computer Society. Final Version.

- [Bjø03] Dines Bjørner. Domain Engineering: A "Radical Innovation" for Systems and Software Engineering ? In *Verification: Theory and Practice*, volume 2772 of *Lecture Notes in Computer Science*, Heidelberg, October 7–11 2003. Springer–Verlag. The Zohar Manna International Conference, Taormina, Sicily 29 June – 4 July 2003. .
- [Bjø06] Dines Bjørner. *Software Engineering, Vol. 3: Domains, Requirements and Software Design*. Texts in Theoretical Computer Science, the EATCS Series. Springer, 2006.
- [Bjø07] Dines Bjørner. Domain Theory: Practice and Theories, Discussion of Possible Research Topics. In *ICTAC'2007*, volume 4701 of *Lecture Notes in Computer Science* (eds.

J.C.P. Woodcock et al.), pages 1–17, Heidelberg, September 2007. Springer.

- [Bjø08] Dines Bjørner. From Domains to Requirements. In *Montanari Festschrift*, volume 5065 of *Lecture Notes in Computer Science* (eds. Pierpaolo Degano, Rocco De Nicola and José Meseguer), pages 1–30, Heidelberg, May 2008. Springer.
- [Bjø09] Dines Bjørner. On Mereologies in Computing Science. In *Festschrift: Reflections on the Work of C.A.R. Hoare, History of Computing* (eds. Cliff B. Jones, A.W. Roscoe and Kenneth R. Wood), pages 47–70, London, UK, 2009. Springer.
- [Bjø10a] Dines Bjørner. Domain Engineering. In Paul Boca and Jonathan Bowen, editors, *Formal Methods: State of the Art*

and New Directions, Eds. Paul Boca and Jonathan Bowen, pages 1–42, London, UK, 2010. Springer.

[Bjø10b] Dines Bjørner. Domain Science & Engineering – *From Computer Science to The Sciences of Informatics, Part I of II: The Engineering Part*. *Kibernetika i sistemny analiz*, (4):100–116, May 2010.

[Bjø10c] Dines Bjørner. The Rôle of Domain Engineering in Software Development. Why Current Requirements Engineering Seems Flawed! In *Perspectives of Systems Informatics*, volume 5947 of *Lecture Notes in Computer Science*, pages 2–34, Heidelberg, Wednesday, January 27, 2010. Springer.

[Bjø11a] Dines Bjørner. Believable Software Management. *Encyclopedia of Software Engineering*, 1(1):1–32, 2011.

[Bjø11b] Dines Bjørner. Domain Science & Engineering – *From Computer Science to The Sciences of Informatics Part II of II: The Science Part*. *Kibernetika i sistemny*

analiz, (2):100–120, May 2011.

- [Bjø11c] Dines Bjørner. Domains: Their Simulation, Monitoring and Control – A Divertimento of Ideas and Suggestions. In *Rainbow of Computer Science, Festschrift for Hermann Maurer on the Occasion of His 70th Anniversary.*, Festschrift (eds. C. Calude, G. Rozenberg and A. Saloma), pages 167–183. Springer, Heidelberg, Germany, January 2011.
- [Bjø13] Dines Bjørner. *Domain Science and Engineering as a Foundation for Computation for Humanity*, chapter 7, pages 159–177. Computational Analysis, Synthesis, and Design of Dynamic Systems. CRC [Francis & Taylor], 2013. (eds.: Justyna Zander and Pieter J. Mosterman).
- [Bjø14a] Dines Bjørner. *A Rôle for Mereology in Domain Science and Engineering*. Synthese Library (eds. Claudio Calosi

and Pierluigi Graziani). Springer, Amsterdam, The Netherlands, October 2014.

- [Bjø14b] Dines Bjørner. Domain Analysis: Endurants – An Analysis & Description Process Model. In Shusaku Iida, José Meseguer, and Kazuhiro Ogata, editors, *Specification, Algebra, and Software: A Festschrift Symposium in Honor of Kokichi Futatsugi*. Springer, May 2014.
- [Bjø14c] Dines Bjørner. Domain Engineering – A Basis for Safety Critical Software. Invited Keynote, ASSC2014: Australian System Safety Conference, Melbourne, 26–28 May, December 2014.
- [Bjø14d] Dines Bjørner. Manifest Domains: Analysis & Description. Research Report, 2014. Superseded by [Bjø16b].

- [Bjø16a] Dines Bjørner. From Domain Descriptions to Requirements Prescriptions – A Different Approach to Requirements Engineering. *Submitted for consideration by Formal Aspects of Computing*, 2016.
- [Bjø16b] Dines Bjørner. Manifest Domains: Analysis & Description. *Expected published by Formal Aspects of Computing*, 2016.
- [BO80] Dines Bjørner and Ole N. Oest, editors. *Towards a Formal Description of Ada*, volume 98 of *LNCS*. Springer, 1980.
- [CO84] G.B. Clemmensen and O. Oest. Formal specification and development of an Ada compiler – a VDM case study. In *Proc. 7th International Conf. on Software Engineering*, 26.-29. March 1984, Orlando, Florida, pages 430–440. IEEE, 1984.

- [CV99] R. Casati and A. Varzi. *Parts and Places: the structures of spatial representation*. MIT Press, 1999.
- [DvLF93] Anne Dardenne, Axel van Lamsweerde, and Stephen Fickas. Goal-directed requirements acquisition. *Sci. Comput. Program.*, 20(1-2):3–50, April 1993.
- [GGJZ00] Carl A. Gunter, Elsa L. Gunter, Michael A. Jackson, and Pamela Zave. A Reference Model for Requirements and Specifications. *IEEE Software*, 17(3):37–43, May–June 2000.
- [Jac95] Michael A. Jackson. *Software Requirements & Specifications: a lexicon of practice, principles and prejudices*. ACM Press. Addison-Wesley, Reading, England, 1995.
- [Jac01] Michael A. Jackson. *Problem Frames — Analyzing and Structuring Software Development Problems*. ACM Press,

Pearson Education. Addison-Wesley, England, 2001.

- [Jac10] Michael A. Jackson. Program Verification and System Dependability. In Paul Boca and Jonathan Bowen, editors, *Formal Methods: State of the Art and New Directions*, pages 43–78, London, UK, 2010. Springer.
- [JHJ07] Cliff B. Jones, Ian Hayes, and Michael A. Jackson. Deriving Specifications for Systems That Are Connected to the Physical World. In Cliff Jones, Zhiming Liu, and James Woodcock, editors, *Formal Methods and Hybrid Real-Time Systems: Essays in Honour of Dines Bjørner and Zhou Chaochen on the Occasion of Their 70th Birthdays*, volume 4700 of *Lecture Notes in Computer Science*, pages 364–390. Springer, 2007.
- [Lau02] Søren Lauesen. *Software Requirements - Styles and Techniques*. Addison-Wesley, UK, 2002.

- [Mas43] Abraham Maslow. A Theory of Human Motivation. *Psychological Review*, 50(4):370–96, 1943.
<http://psychclassics.yorku.ca/Maslow/motivation.htm>.
- [Mas54] Abraham Maslow. *Motivation and Personality*. Harper and Row Publishers, 3rd ed., 1954.
- [OD08] Ernst-Rüdiger Olderog and Henning Dierks. *Real-Time Systems: Formal Specification and Automatic Verification*. Cambridge University Press, UK, 2008.
- [Oes86] O. Oest. VDM From Research to Practice. In H.-J. Kugler, editor, *Information Processing '86*, pages 527–533. IFIP World Congress Proceedings, North-Holland Publ.Co., Amsterdam, 1986.
- [PS04] Christopher Peterson and Martin E.P. Seligman. *Character strengths and virtues: A handbook and classification*. Oxford University Press, 2004.

- [van09] Axel van Lamsweerde. *Requirements Engineering: From System Goals to UML Models to Software Specifications*. Wiley, 2009.
- [ZH04] Chao Chen Zhou and Michael R. Hansen. *Duration Calculus: A Formal Approach to Real-time Systems*. Monographs in Theoretical Computer Science. An EATCS Series. Springer–Verlag, 2004.