From Domain Descriptions to Requirement Prescriptions

A Different Approach to Requirements Engineering

Dines Bjørner

Fredsvej 11, DK–2840 Holte, Denmark

August 31, 2015: 19:09

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

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

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

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

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

- We briefly review principles, techniques and tools for "deriving" interface requirements based on sharing domain
  - endurants,
  - actions.
  - events and
  - behaviours

  with their machine correspondants.
- Finally we
  - review machine requirements
  - while relating a number of machine requirements aspects
  - to domain phenomena.

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. The Contribution of These Lectures

- We claim that the present lecture content contributes to our understanding and practice of software engineering as follows:
  -
  -
  -

1.3. Structure of Lectures

- The structure of the paper is as follows:
  - Section 2. provides a fair-sized, hence realistic example
  - Sections 3–6. covers our approach to requirements development.
    - Section 3. overviews the issue of 'requirements', relates our approach (Sects. –) 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,
  - instantiation,
  - determination,
  - extension and
  - fitting.
- Section 5. covers key features of interface requirements; and
- Sect. 6. very briefly touches upon some machine requirements issues.

◈ 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 [Bjø16] and formalised in [Bjø14a].
  - 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` [Bjø14b, Sect. 3.1.6] to to a transport domain δ:Δ yields the following.

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

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` [Bjø14b, 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 → HA
2b. **obs_part**_LA: N → LA

2.1.2. Hubs and Links

Applying `observe_part_types` [Bjø14b, 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.

type
3 H, HS = H-**set**
4 L, LS = L-**set**
5 V, VS = V-**set**

6 We introduce some auxiliary functions.
   a. links extracts the links of a network.
   b. hubs extracts the hubs of a network.

value
3 **obs_part**_HS: HA → HS
4 **obs_part**_LS: LA → LS
5 **obs_part**_VS: F → VS

value
6a. links: Δ → L-**set**
6a. links(δ) ≡ **obs_part**_LS(**obs_part**_LA(δ))
6b. hubs: Δ → H-**set**
6b. hubs(δ) ≡ **obs_part**_HS(**obs_part**_HA(δ))

2.1.3. Unique Identifiers

Applying `observe_unique_identifier` [Bjø14b, 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 → NI
7c. **uid**_HAI: HA → HAI
7c. **uid**_LAI: LA → LAI
7c. **uid**_HI: H → HI
7c. **uid**_LI: L → LI
7c. **uid**_FI: F → FI
7c. **uid**_VI: V → VI
7c. **uid**_MI: M → MI
axiom
7b. NI∩HAI=Ø, NI∩LAI=Ø, NI∩HI=Ø, 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: Δ → LI-set
8  xtr_lis(δ) ≡
8    let ls = links(δ) in {uid_LI(l)|l:L•l ∈ ls} end
9  xtr_his: Δ → HI-set
9  xtr_his(δ) ≡
9    let hs = hubs(δ) in {uid_HI(h)|h:H•k ∈ hs} end
10 get_link: LI → Δ ~→ L
10 get_link(li)(δ) ≡
10   let ls = links(δ) in
10   let l:L • l ∈ ls ∧ li=uid_LI(l) in l end end
10   pre: li ∈ xtr_lis(δ)
11 get_hub: HI → Δ ~→ H
11 get_hub(hi)(δ) ≡
11   let hs = hubs(δ) in
11   let h:H • h ∈ hs ∧ hi=uid_HI(h) in h end end
11   pre: hi ∈ xtr_his(δ)
```

2.1.4. Mereology

We cover the mereologies of all part sorts introduced so far. We decide that nets, hub aggregates, link aggregates and fleets have no mereologies of interest. Applying `observe_mereology` [Bjø14b, 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 → LI-set
13  obs_mereo_L: L → HI-set
axiom
13  ∀ l:L·card obs_mereo_L(l)=2
value
14  obs_mereo_V: V → MI
15  obs_mereo_M: M → VI-set
axiom
16  ∀ δ:Δ, hs:HS·hs=hubs(δ), ls:LS·ls=links(δ) ·
16     ∀ h:H·h ∈ hs·obs_mereo_H(h)⊆xtr_his(δ) ∧
17     ∀ l:L·l ∈ ls·obs_mereo_L(l)⊆xtr_lis(δ) ∧
18a.    let f:F·f=obs_part_F(δ) ⇒
18a.       let m:M·m=obs_part_M(δ),
18a.           vs:VS·vs=obs_part_VS(f) in
18a.       ∀ v:V·v ∈ vs⇒uid_V(v) ∈ obs_mereo_M(m)
18b.     ∧ obs_mereo_M(m) = {uid_V(v)|v:V·v ∈ 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` [Bjø14b, 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 hub[1],
   c. and have hub state spaces which are sets of hub states[2].

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 Hubs have geodetic location.

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

[1] A hub state "signals" which input-to-output link connections are open for traffic.
[2] A hub state space indicates which hub states a hub may attain over time.

```
type
21    GeoH
19b.  HΣ = (LI×LI)-set
19c.  HΩ = HΣ-set
value
21    attr_GeoH: H → GeoH
19b.  attr_HΣ: H → HΣ
19c.  attr_HΩ: H → HΩ
axiom
20    ∀ δ:Δ,
20      let hs = hubs(δ) in
20      ∀ h:H · h ∈ hs ·
20a.          xtr_lis(h)⊆xtr_lis(δ)
20b.        ∧ attr_Σ(h) ∈ attr_Ω(h)
20      end
value
22  xtr_lis: H → LI-set
22  xtr_lis(h) ≡
22    {li | li:LI,(li′,li″):LI×LI ·
22        (li′,li″) ∈ attr_HΣ(h) ∧ li ∈ {li′,li″}}
```

Then links.

23 Links have lengths.

24 Links have geodetic location.

25 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
23 LEN
24 GeoL
25a. $L\Sigma = (HI \times HI)$**-set**
25b. $L\Omega = L\Sigma$**-set**
value
23 **attr**_LEN: $L \to LEN$
24 **attr**_GeoL: $L \to GeoL$
25a. **attr**_$L\Sigma$: $L \to L\Sigma$
25b. **attr**_$L\Omega$: $L \to L\Omega$
axiom
25 $\forall$ n:N ·
25 **let** ls = xtr−links(n), hs = xtr_hubs(n) **in**
25 $\forall$ l:L·l $\in$ ls $\Rightarrow$
25a. **let** lσ = **attr**_$L\Sigma$(l) **in**
25a. 0$\leq$**card** l$\sigma\leq$4
25a. $\wedge$ $\forall$ (hi′,hi″):(HI×HI)·(hi′,hi″) $\in$ lσ $\Rightarrow$
25a. {get_H(hi′)(n),get_H(hi″)(n)}=**obs_mereo**_L(l)
25b. $\wedge$ **attr**_$L\Sigma$(l) $\in$ **attr**_$L\Omega$(l)
25 **end end**

Then vehicles.

26 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
26 VPos = onL | atH
26a. onL :: LI HI HI R
26b. R = **Real** **axiom** ∀ r:R · 0≤r≤1
26c. atH :: HI LI LI
value
26 **attr**_VPos: V → VPos
axiom
26a. ∀ n:N, onL(li,fhi,thi,r):VPos ·
26a. ∃ l:L·l ∈**obs_part**_LS(**obs_part**_N(n))
26a. ⇒ li=**uid**_L(l)∧{fhi,thi}=**obs_mereo**_L(l),
26c. ∀ n:N, atH(hi,fli,tli):VPos ·
26c. ∃ h:H·h ∈**obs_part**_HS(**obs_part**_N(n))
26c. ⇒ hi=**uid**_H(h)∧(fli,tli) ∈ **attr**_LΣ(h)

27 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 !

28 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
27b.  MAP = VI ~m~> VPos
axiom
27b.  ∀ map:MAP · card dom map = card rng map
value
27  distribute: VS → N → MAP
27  distribute(vs)(n) ≡
27a.     let (hs,ls) = (xtr_hubs(n),xtr_links(n)) in
27a.     let vps = {onL(uid_(l),fhi,thi,r) |
27a.               l:L·l ∈ls∧{fhi,thi}
27a.               ⊆obs_mereo_L(l)∧0≤r≤1}
27a.             ∪ {atH(uid_H(h),fli,tli)|
27a.               h:H·h ∈hs∧{fli,tli}
27a.               ⊆obs_mereo_H(h)} in
27b.     [uid_V(v)↦vp|v:V,vp:VPos·v ∈vs∧vp∈vps]
27     end end
```

And finally monitors. We consider only one monitor attribute.

29 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
29  Traffic = VI ~m~> (T × VPos)*
value
29  attr_Traffic: M → Traffic
axiom
29b.  ∀ δ:Δ •
29b.    let m = obs_part_M(δ) in
29b.    let tf = attr_Traffic(m) in
29b.    dom tf ⊆ xtr_vis(δ) ∧
29b.    ∀ vi:VI • vi ∈ dom tf •
29b.      let tr = tf(vi) in
29b.      ∀ i,i+1:Nat • {i,i+1}⊆dom tr •
29b.        let (t,vp)=tr(i),(t′,vp′)=tr(i+1) in
29b.        t<t′
29(b.)i       ∧ case (vp,vp′) of
29(b.)i           (onL(li,fhi,thi,r),onL(li′,fhi′,thi′,r′))
29(b.)i             → li=li′∧fhi=fhi′∧thi=thi′∧r≤r′ ∧ li ∈ xtr_lis(δ) ∧ {fhi,thi} = obs_mereo_L(get_link(li)(δ)),
29(b.)ii          (atH(hi,fli,tli),atH(hi′,fli′,tli′))
29(b.)ii            → hi=hi′∧fli=fli′∧tli=tli′ ∧ hi ∈ xtr_his(δ) ∧ (fli,tli) ∈ obs_mereo_H(get_hub(hi)(δ)),
29(b.)iii         (onL(li,fhi,thi,1),atH(hi,fli,tli))
29(b.)iii           → li=fli∧thi=hi ∧ {li,tli} ⊆ xtr_lis(δ) ∧ {fhi,thi}=obs_mereo_L(get_link(li)(δ))
29(b.)iii            ∧ hi ∈ xtr_his(δ) ∧ (fli,tli) ∈ obs_mereo_H(get_hub(hi)(δ)),
29(b.)iv          (atH(hi,fli,tli),onL(li′,fhi′,thi′,0))
29(b.)iv            →  etcetera,
29b.          _  → false
29b.    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 [Bjø16]):
  - 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

30 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.

31 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
30 insert_hub: H → N → N
31 insert_hub(h)(n) **as** n′
30a. **pre**: **uid**_H(h) ∉ xtr_his(n)
30b. ∧ **obs_mereo**_H= {}
30c. ∧ ...
31a. **post**: **obs_part**_Ls(n) = **obs_part**_Ls(n′)
31b. ∧ **obs_part**_Hs(n) ∪ {h} = **obs_part**_Hs(n′)

2.2.2. Link Disappearance Event

We formalise aspects of the link disappearance event:

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

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

34 and such a link may disappear such that

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

```
value
32  link_diss_event: N × N′ × Bool
32  link_diss_event(n,n′) as tf
33      pre: obs_part_Ls(obs_part_LS(n))≠{}
34      post: ∃ l:L·l ∈ obs_part_Ls(obs_part_LS(n)) ⇒
35              l ∉ obs_part_Ls(obs_part_LS(n′))
35              ∧ n′ ∪ {l} = obs_part_Ls(obs_part_LS(n))
```

2.2.3. Road Traffic

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

- (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.

36 besides the domain, δ:Δ,

37 a net, n:N,

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

39 a monitor, m:M, and

40 a clock, clock, behaviour.

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

- The n:N, vs:V-**set** and m:M are observable from any road traffic system domain δ.

```
value
36  δ:Δ
37  n:N = obs_part_N(δ),
37  ls:L-set=links(δ),hs:H-set=hubs(δ),
37  lis:LI-set=xtr_lis(δ),his:HI-set=xtr_his(δ)
38  va:VS=obs_part_VS(obs_part_F(δ)),
38  vs:Vs-set=obs_part_Vs(va),
38  vis:VI-set = {uid_VI(v)|v:V·v ∈ vs},
39  m:obs_part_M(δ),
39  mi=uid_MI(m),
39  ma:attributes(m)
40  clock: 𝕋 → out {clk_ch[vi|vi:VI·vi ∈ vis]}  Unit
41  vm:MAP·vpos_map = distribute(vs)(n);
```

2.2.3.2 Channels:

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

a. by the unique identifiers of vehicles

b. and the (single) monitor identifier.[3]

and communicating vehicle positions.

channel

42 $\{v\_m\_ch[vi,mi]|vi:VI \cdot vi \in vis\}:VPos$

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

2.2.3.3 Behaviour Signatures:

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

44 The vehicle behaviour

a. is indexed by the unique identifier, uid_V(v):VI,
b. the vehicle mereology, in this case the single monitor identifier mi:MI,
c. the vehicle attributes, obs_attribs(v)
d. and — factoring out one of the vehicle attributes — the current vehicle position.
e. The vehicle behaviour offers communication to the monitor behaviour; and behaves "forever".

45 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
43  rts: Unit → Unit
44  veh: vi:VI × mi:MI → vp:VPos → out vm_ch[vi,mi] Unit
45  mon: m:M × vis:VI-set → RTF → in {v_m_ch[vi,mi]|vi:VI·vi ∈ vis},c
```

2.2.3.4 The Road Traffic System Behaviour:

46 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

46 rts() ≡

46a. ‖ {veh(**uid**_VI(v),mi)(vm(**uid**_VI(v)))|v:V·v ∈ vs}

46b. ‖ mon(mi,vis)([vi↦⟨⟩|vi:VI·vi ∈ 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.

47 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" !

```
47   veh(vi,mi)(vp:atH(hi,fli,tli)) ≡
47a.        v_m_ch[vi,mi]!vp ; veh(vi,mi)(vp)
47b.      ⌈⌉
47(b.)i     let {hi′,thi}=obs_mereo_L(get_link(tli)(n)) in
47(b.)i                          assert: hi′=hi
47(b.)ii    v_m_ch[vi,mi]!onL(tli,hi,thi,0) ;
47(b.)ii    veh(vi,mi)(onL(tli,hi,thi,0)) end
47c.      ⌈⌉ stop
```

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

a. the vehicle remains at that link position informing the monitor of its position,
b. or, internally non-deterministically, if the vehicle's position on the link has not yet reached the hub,
    i then the vehicle moves an arbitrary increment ℓ_ε (less than or equal to the distance to the hub) along the link informing the monitor of this, or
    ii else,
        A while obtaining a "next link" from the mereology of the hub (where that next link could very well be the same as the link the vehicle is about to leave),
        B the vehicle informs the monitor that it is now at the hub identified by thi, whereupon the vehicle resumes the vehicle behaviour positioned at that hub.
c. or, internally non-deterministically, the vehicle "disappears — off the radar" !

48 veh(vi,mi)(vp:onL(li,fhi,thi,r)) ≡
48a. v_m_ch[vi,mi]!vp ; veh(vi,mi,va)(vp)
48b. ⌈ **if** r + $\ell_\varepsilon \leq 1$
48(b.)i **then**
48(b.)i v_m_ch[vi,mi]!onL(li,fhi,thi,r+ℓ_ε) ;
48(b.)i veh(vi,mi)(onL(li,fhi,thi,r+ℓ_ε))
48(b.)ii **else**
48(b.)iiA **let** li′:LI·li′ ∈ **obs_mereo**_H(get_hub(thi)(n)) **in**
48(b.)iiB v_m_ch[vi,mi]!atH(li,thi,li′);
48(b.)iiB veh(vi,mi)(atH(li,thi,li′)) **end end**
48c. ⌈ **stop**

The Monitor Behaviour

49 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".

50 Either the monitor "does own work"

51 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.

```
49  mon(mi,vis)(trf) ≡
50      mon(mi,vis)(trf)
51    ⌈⌉
51a.    □{let tvp = (clk_ch?,v_m_ch[vi,mi]?) in
51b.      let trf′ = trf † [vi ↦ trf(vi)^<tvp>] in
51c.      mon(mi,vis)(trf′)
51d.      end end | vi:VI · vi ∈ vis}
```

● ◈

TO BE RESOLVED: INITIAL DISTRIBUTION, MAP, OF VEHICLES

2.3. Domain Facets

- The example of this section does not reflect the concepts of domain facets such as
  - domain intrinsics,
  - domain support technologies,
  - domain rules, regulations & scripts,
  - organisation & management, and
  - human behaviour.
  - These facets are covered in [Bjø10, 2008].

3. Requirements

- This and the next sections are the main sections of these lectures.
  - Section 3. is the most detailed and systematic section. It covers the domain requirements operations of projection, instantiation, determination, extension, and, less detailed, fitting.
  - Section 4. surveys the interface requirements issues of shared phenomena and "completes" the exemplification of the detailed domain extension of our requirements into a road pricing system.
  - Section 5. – very, very briefly – relates some machine requirements issues, i.e., safety criticality and fault tolerance, to the overall design of the road pricing system.
- This initial section captures main concepts and principles of requirements.

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.
- Note that requirements is about systems.

Rule 1 *The "Golden Rule" of Requirements Engineering:*

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

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

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.

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

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

Rule 4 *Requirements Implementability:* *Make sure that requirements are implementable*

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

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

- By **validation** we mean that requirements stake-holders, i.e., "ordinary" domain staff, can understand the requirements and accept them
- By **verification** we mean that requirements developers, i.e., professional engineers can formally text, model check and prove properties of the requirements
- Thus, do not express a requirement for which you have no assurance that it can be validated and verified.

- For example,
  - once a first-level software design has been proposed,
  - one must verify that it satisfies the requirements.
- Thus specific parts of even abstract software designs are usually provided with references to specific parts of the requirements that they are (thus) claimed to implement.
- This latter verification is made simpler when a design is based on a model specification — such as we propose.

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.

3.1. Four Requirements Facets

- We shall unravel requirements in two stages —
  the first stage is sketchy (and thus informal) while
  the last stage is systematic and both informal and formal.
- The sketchy stage consists of
  - a narrative problem/objective sketch sketch,
  - a narrative system requirements sketch, and
  - a narrative user & external equipment requirements sketch,
- The narrative and formal design requirements prescription stage
  - is systematic, and mandates both strict narrative,
  - and formal descriptions;
  - and is “derivable” from the domain description.

3.1.1. Problem and/or Objective Sketch

Definition 5 ***Problem/Objective Sketch:*** By a **problem/objective sketch** we understand

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

Example 1 **The Problem/Objective Requirements: A Sketch**:

- The objective is 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 a well-delineated *road net*
    - and thus enables
      - the *owner* of the road net to charge
      - the *owner* of the vehicles
      - *fees* for the usage of that road net ■

3.1.2. Systems Requirements

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

- a narrative which emphasises
- the overall 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-gate*s with
    - *vehicle identification sensor*s,
    - *barrier*s which afford (only specially equipped) vehicles to enter into and exit from the toll-road net;

    and
  - there is a *road-pricing calculator*.

- **The system to be designed (from the requirements) is the *road-pricing calculator*.**
- These four system elements are required to behave and interact as follows:
  - The `GNSS` is assumed to continuously offer vehicles information about their global position;
  - *vehicles* shall contain a `GNSS receiver` which based on the global position information shall regularly calculate their timed local position and offer this to the *calculator* — while otherwise cruising the general road net as well as the toll-road net, the latter while carefully moving through toll-gates;
  - *toll-gates* shall register the identity of vehicles entering and exiting the toll-road and offer this information to the calculator; and
  - the *calculator* shall accept all messages from vehicles and gates and use this information to record the movements of vehicles and bill these whenever they exit the toll-road.

- 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”

3.1.3. 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 this 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 describe domain and requirements aspects of users and external equipment,
  - we do not expect to machine (i.e., hardware or software) design these elements;
  - they are assumed already implemented !

3.1.4. Design Requirements

Definition 8 ***Design Requirements:***

- By **design requirements** we understand precise prescriptions of
  - the endurants
  - and perdurants

  of the (to be designed) system components
- and the assumptions which that design must rely upon ■

- We shall not here exemplify design requirements.
  - This will be done, extensively, in the examples of Sects. 5.–6.
  - The assumptions upon which the design can be relied upon, that is, shall be verified ("against") are illustrated in Sect. 4.

3.2. The Three Phases of Software Engineering

- There are, as we see it, three kinds of design assumptions and requirements:
  - domain requirements (the primary design assumptions) ,
  - interface requirements and
  - machine requirements.
  - The last two being the primary design requirements.

- The **domain requirements** are those requirements which can be expressed sôlely using technical terms of the domain ■
- The **interface requirements** are those requirements which can be expressed only using technical terms of both the domain and the machine ■
- The **machine requirements** are those requirements which, in principle, can be expressed sôlely using technical terms of the machine ■

Definition 9 *A Verification Paradigm:*

- Some preliminary designations:
  - let $\mathscr{D}$ designate the the domain requirements;
  - let $\mathscr{R}$ designate the interface and machine requirements and
  - let $\mathscr{S}$ designate the system design.
- Now $\mathscr{D}, \mathscr{S} \models \mathscr{R}$ shall be read:
  - it must be verified that the $\mathscr{S}$ystem design
  - satisfies the interface and machine $\mathscr{R}$equirements
  - in the context of the $\mathscr{D}$omain requirements •
- The "in the context of $\mathscr{D}$..." term means that
  - proofs of $\mathscr{S}$oftware design correctness
  - with respect to $\mathscr{R}$equirements
  - will often have to refer to $\mathscr{D}$omain requirements assumptions.

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, i.e., "present", between both the domain (concretely, manifestly) and the machine (abstractly, conceptually).

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

- The domain engineer is now free to rearrange the final prescription,
  - irrespective of the order in which the various sections were developed,
  - in such a way as to give a most
    - pleasing,
    - pedagogic and
    - cohesive

    reading (i.e., presentation).
- From such a requirements prescription one can therefore not necessarily see in which order the various sections of the prescription were developed.

3.4. Design Requirements and Design Assumptions

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

- 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:
  - for that part of the vehicle software which interfaces the GNSS receiver and the road pricing calculator (cf. Items 129– 132 on Slide 179),
  - for that part of the toll-gate software which interfaces the toll-gate and the road pricing calculator (cf. Items 137– 139 on Slide 184) and
  - the road pricing calculator (cf. Items 168– 181 on Slide 235)

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 on Slide 179,
  - that the GNSS regularly offers vehicles correct information as to their global position (cf. Item 129 on Slide 179),
  - that *toll-gates* behave as prescribed in Items 134– 139 on Slide 184, 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 on Slide 184

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

4. Domain Requirements

- Domain requirements express the assumptions
  - that a design must rely upon
  - in order that that design can be verified correct.

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

- is that subset of the requirements prescription
- which can be expressed sôlely using terms from the domain description ■
- To determine a relevant subset all we need is collaboration with requirements stake-holders.

- 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 & Simplification

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

- *a subset of the domain description,*
- *one which projects out all those*
  - *endurants:*
    - *parts,*
    - *materials and*
    - *components,*

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

  that the stake-holders do not wish represented or relied upon by the machine.
- *Simplification means that we simplify (refine) some internal qualities, like mereologies and/or attributes* ■

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

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 these[4]:
  - 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.

[4] By 'relate to … these' we mean that the required system does not rely on domain phenomena that have been "projected away".

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

- 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 $\Delta_{\mathscr{P}}$, the projected version of the domain requirements prescription[5].

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

4.1.2. Domain Projection — Formalisation

- The requirements prescription hinges, crucially,
  - not only on a systematic narrative of all the
    - projected,
    - instantiated,
    - determinated,
    - extended and
    - fitted

    specifications,
  - but also on their formalisation.
- In the formal domain projection example we, regretfully, omit the narrative texts.
  - In bringing the formal texts we keep the item numbering from Sect. 2.,
  - where you can find the associated narrative texts.

Example 9 Domain Requirements — Projection:

Main Sorts

type
1 $\Delta_{\mathscr{P}}$
1a. $\mathsf{N}_{\mathscr{P}}$
1b. $\mathsf{F}_{\mathscr{P}}$
value
1a. **obs_part**_$\mathsf{N}_{\mathscr{P}}$: $\Delta_{\mathscr{P}} \to \mathsf{N}_{\mathscr{P}}$
1b. **obs_part**_$\mathsf{F}_{\mathscr{P}}$: $\Delta_{\mathscr{P}} \to \mathsf{F}_{\mathscr{P}}$
type
2a. $\mathsf{HA}_{\mathscr{P}}$
2b. $\mathsf{LA}_{\mathscr{P}}$
value
2a. **obs_part**_HA: $\mathsf{N}_{\mathscr{P}} \to \mathsf{HA}$
2b. **obs_part**_LA: $\mathsf{N}_{\mathscr{P}} \to \mathsf{LA}$

Concrete Types

type
3 $\mathsf{H}_{\mathscr{P}}$, $\mathsf{HS}_{\mathscr{P}} = \mathsf{H}_{\mathscr{P}}$**-set**
4 $\mathsf{L}_{\mathscr{P}}$, $\mathsf{LS}_{\mathscr{P}} = \mathsf{L}_{\mathscr{P}}$**-set**
5 $\mathsf{V}_{\mathscr{P}}$, $\mathsf{VS}_{\mathscr{P}} = \mathsf{V}_{\mathscr{P}}$**-set**
value
3 **obs_part**_$\mathsf{HS}_{\mathscr{P}}$: $\mathsf{HA}_{\mathscr{P}} \to \mathsf{HS}_{\mathscr{P}}$
4 **obs_part**_$\mathsf{LS}_{\mathscr{P}}$: $\mathsf{LA}_{\mathscr{P}} \to \mathsf{LS}_{\mathscr{P}}$
5 **obs_part**_$\mathsf{VS}_{\mathscr{P}}$: $\mathsf{F}_{\mathscr{P}} \to \mathsf{VS}_{\mathscr{P}}$
6a. links: $\Delta_{\mathscr{P}} \to$ L**-set**
6a. links($\delta_{\mathscr{P}}$) $\equiv$ **obs_part**_$\mathsf{LS}_{\mathscr{R}}$(**obs_part**_$\mathsf{LA}_{\mathscr{R}}$($\delta_{\mathscr{R}}$))
6b. hubs: $\Delta_{\mathscr{P}} \to$ H**-set**
6b. hubs($\delta_{\mathscr{P}}$) $\equiv$ **obs_part**_$\mathsf{HS}_{\mathscr{P}}$(**obs_part**_$\mathsf{HA}_{\mathscr{P}}$($\delta_{\mathscr{P}}$))

Unique Identifiers

type
7a. HI, LI, VI, MI
value
7c. **uid**_HI: $H_{\mathscr{P}} \to$ HI
7c. **uid**_LI: $L_{\mathscr{P}} \to$ LI
7c. **uid**_VI: $V_{\mathscr{P}} \to$ VI
7c. **uid**_MI: $M_{\mathscr{P}} \to$ MI
axiom
7b. HI$\cap$LI=Ø, HI$\cap$VI=Ø, HI$\cap$MI=Ø,
7b. LI$\cap$VI=Ø, LI$\cap$MI=Ø, VI$\cap$MI=Ø

Mereology

```
value
12  obs_mereo_H𝒫: H𝒫 → LI-set
13  obs_mereo_L𝒫: L𝒫 → HI-set
13      axiom ∀ l:L𝒫 · card obs_mereo_L𝒫(l)=2
14  obs_mereo_V𝒫: V𝒫 → MI
15  obs_mereo_M𝒫: M𝒫 → VI-set
axiom
16  ∀ δ𝒫:Δ𝒫, hs:HS·hs=hubs(δ), ls:LS·ls=links(δ𝒫) ⇒
16    ∀ h:H𝒫·h ∈ hs ⇒
16      obs_mereo_H𝒫(h)⊆xtr_his(δ𝒫) ∧
17    ∀ l:L𝒫·l ∈ ls ·
16      obs_mereo_L𝒫(l)⊆xtr_lis(δ𝒫) ∧
18a.  let f:F𝒫·f=obs_part_F𝒫(δ𝒫) ⇒
18a.        vs:VS𝒫·vs=obs_part_VS𝒫(f) in
18a.      ∀ v:V𝒫·v ∈ vs ⇒
18a.        uid_V𝒫(v) ∈ obs_mereo_M𝒫(m) ∧
18b.      obs_mereo_M𝒫(m)
18b.        = {uid_V𝒫(v)|v:V·v ∈ vs}
18b.  end
```

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

type
21 GeoH
19b. $\mathsf{H}\Sigma_{\mathscr{P}} = (\mathsf{LI}\times\mathsf{LI})$**-sett**
19c. $\mathsf{H}\Omega_{\mathscr{P}} = \mathsf{H}\Sigma_{\mathscr{P}}$**-set**
value
19b. **attr**_$\mathsf{H}\Sigma_{\mathscr{P}}$: $\mathsf{H}_{\mathscr{P}} \to \mathsf{H}\Sigma_{\mathscr{P}}$
19c. **attr**_$\mathsf{H}\Omega_{\mathscr{P}}$: $\mathsf{H}_{\mathscr{P}} \to \mathsf{H}\Omega_{\mathscr{P}}$
axiom
20 $\forall\ \delta_{\mathscr{P}}{:}\Delta_{\mathscr{P}},$
20 **let** hs = hubs($\delta_{\mathscr{P}}$) **in**
20 $\forall$ h:$\mathsf{H}_{\mathscr{P}}$ · h $\in$ hs ·
20a. xtr_lis(h)$\subseteq$xtr_lis($\delta_{\mathscr{P}}$)
20b. $\wedge$ **attr**_$\Sigma_{\mathscr{P}}$(h) $\in$ **attr**_$\Omega_{\mathscr{P}}$(h)
20 **end**

Then links:

type
24 GeoL
25a. $\mathsf{L}\Sigma_{\mathscr{P}} = (\mathsf{HI}\times\mathsf{HI})$**-set**
25b. $\mathsf{L}\Omega_{\mathscr{P}} = \mathsf{L}\Sigma_{\mathscr{P}}$**-set**
value
24 **attr**_GeoL: $\mathsf{L} \to$ GeoL
25a. **attr**_$\mathsf{L}\Sigma_{\mathscr{P}}$: $\mathsf{L}_{\mathscr{P}} \to \mathsf{L}\Sigma_{\mathscr{P}}$
25b. **attr**_$\mathsf{L}\Omega_{\mathscr{P}}$: $\mathsf{L}_{\mathscr{P}} \to \mathsf{L}\Omega_{\mathscr{P}}$
axiom
25a.– 25b. on Slide 34.

Finally **vehicles:**

- For ‘road pricing’ we need vehicle positions.
  - But, for “technical reasons”,
  - we must abstain from the detailed description
  - given in Items 26–26c.
  - The ‘technical reasons’ are that we assume that the *GNSS* cannot provide us with direction of vehicle movement
  - and therefore we cannot, using only the *GNSS* provide the details of ‘offset’ along a link (*onL*) nor the “from/to link” at a hub (*atH*).
- We therefore simplify vehicle positions.

52 A simplified vehicle position designates

a. either a link

b. or a hub,

type

52 SVPos = SonL | SatH

52a. SonL :: LI

52b. SatH :: HI

axiom

26a.' ∀ n:N, SonL(li):SVPos ·

26a.' ∃ l:L·l ∈**obs_part**_LS(**obs_part**_N(n)) ⇒ li=**uid**_L(l)

26c.' ∀ n:N, atH(hi):SVPos ·

26c.' ∃ h:H·h ∈**obs_part**_HS(**obs_part**_N(n)) ⇒ hi=**uid**_H(h)

Global Values

value

36 $\delta_{\mathscr{P}}$:$\Delta_{\mathscr{P}}$,
37 n:N$_{\mathscr{P}}$ = **obs_part_**N$_{\mathscr{P}}$($\delta_{\mathscr{P}}$),
37 ls:L$_{\mathscr{P}}$**-set** = links($\delta_{\mathscr{P}}$),
37 hs:H$_{\mathscr{P}}$**-set** = hubs($\delta_{\mathscr{P}}$),
37 lis:LI**-set** = xtr_lis($\delta_{\mathscr{P}}$),
37 his:HI**-set** = xtr_his($\delta_{\mathscr{P}}$)

Behaviour Signatures: We omit the monitor behaviour.

53 We leave the vehicle behaviours' attribute argument undefined.

type
53 ATTR
value
43 $\mathsf{trs}_{\mathscr{P}}$: **Unit** → **Unit**
44 $\mathsf{veh}_{\mathscr{P}}$: VI×MI×ATTR → ... **Unit**

The System Behaviour: We omit the monitor behaviour.

value
46a. $\mathsf{trs}_{\mathscr{P}}() = \| \{ \mathsf{veh}_{\mathscr{P}}(\mathbf{uid\_VI}(v), \mathbf{obs\_mereo\_V}(v), \_) \mid v{:}V_{\mathscr{P}} \cdot v \in vs \}$

The Vehicle Behaviour:

- Given the simplification of vehicle positions
- we *simplify* the vehicle behaviour given in Items 47–48

47' veh(vi,mi)(vp:SatH(hi)) ≡
47a.' v_m_ch[vi,mi]!SatH(hi) ; veh(vi,mi)(SatH(hi))
47(b.)i' ⌈ **let** li:LI·li ∈ **obs_mereo**_H(get_hub(hi)(n)) **in**
47(b.)ii' v_m_ch[vi,mi]!SonL(li) ; veh(vi,mi)(SonL(li)) **end**
47c.' ⌉ **stop**

48' veh(vi,mi)(vp:SonL(li)) ≡
48a.' v_m_ch[vi,mi]!SonL(li) ; veh(vi,mi,va)(SonL(li))
48(b.)iiA' ⌈ **let** hi:HI·hi ∈ **obs_mereo**_L(get_link(li)(n)) **in**
48(b.)iiB' v_m_ch[vi,mi]!SatH(hi) ; veh(vi,mi)(atH(hi)) **end**
48c.' ⌉ **stop**

- We can simplify Items 47′–48c.′ further.

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

54 This line coalesces Items 47′ and 48′.

55 Coalescing Items 47a.′ and 48′.

56 Captures the distinct parameters of Items 47′ and 48′.

57 Item 47(b.)i′.

58 Item 47(b.)ii′.

59 Item 48(b.)iiA′.

60 Item 48(b.)iiB′.

61 Coalescing Items 47c.′ and 48c.′.

- 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 179

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, $n_{\mathscr{I}}$:$N_{\mathscr{I}}$, which can be understood as consisting of two, "connected sub-nets".
  - A toll-road net, $trn_{\mathscr{I}}$:$TRN_{\mathscr{I}}$, cf. Fig. 1 on the next slide,
  - and an ordinary road net, $n_{\mathscr{P}'}$.
  - The two are connected as follows:
    - The toll-road net, $trn_{\mathscr{I}}$, borders some toll-road plazas, in Fig. 1 on the following slide shown by white filled circles.
    - These toll-road plaza hubs are proper hubs of the 'ordinary' road net, $n'_{\mathscr{P}}$.

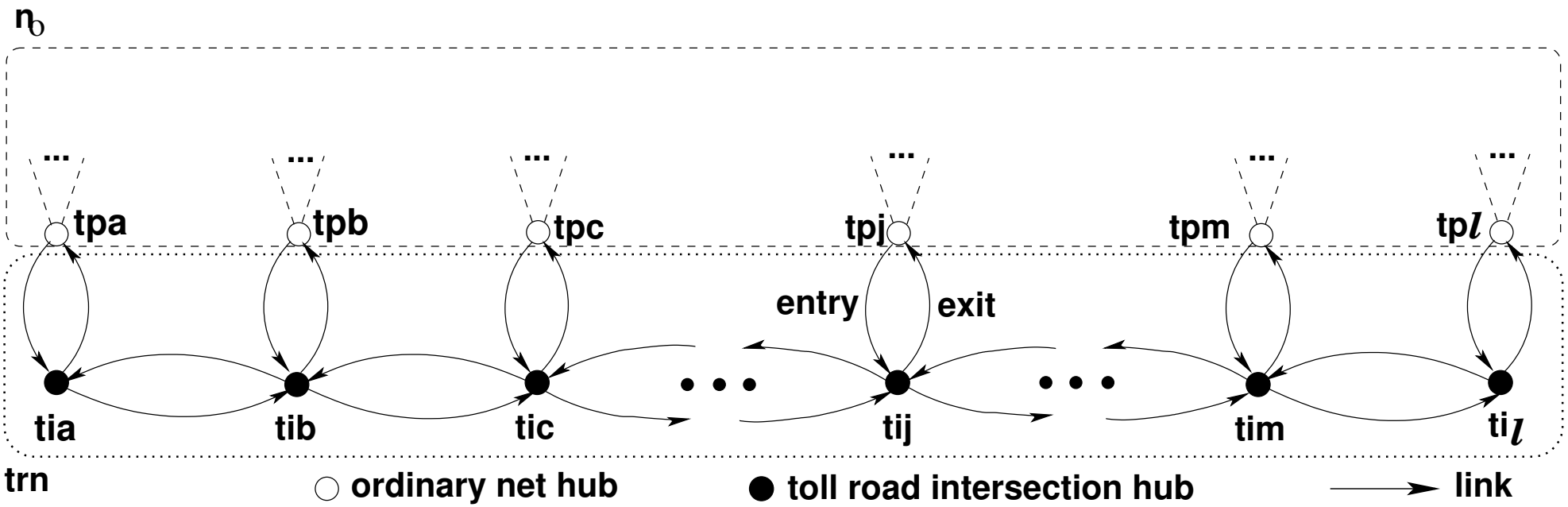

Figure 1: A simple, linear toll-road net

62 The instantiated domain, $\delta_{\mathscr{I}}$:$\Delta_{\mathscr{I}}$ has just the net, $n_{\mathscr{I}}$:$N_{\mathscr{I}}$ being instantiated.

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

a. an “ordinary” road net, n_o:$N_{\mathscr{P}'}$ and

b. a toll-road net proper, trn:$TRN_{\mathscr{I}}$ —

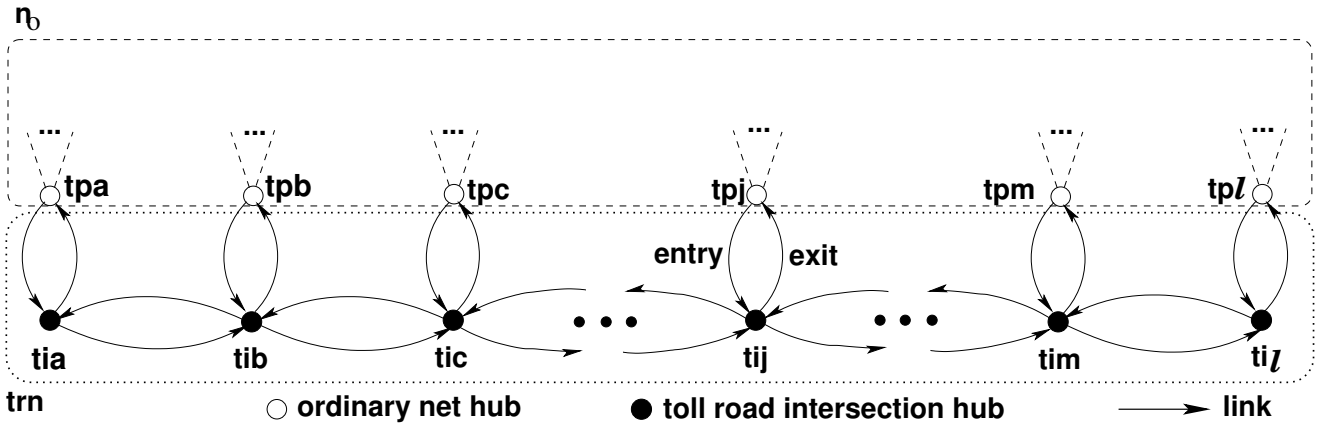

Figure 2: The Instantiated Road Net

c. "connected" by an interface hil:HIL:

i That interface consists of a number of toll-road plazas (i.e., hubs), modeled as a list of hub identifiers, hil:HI*.

ii The toll-road plaza interface to the toll-road net, trn:TRN$_{\mathscr{I}}$[6], 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, $n_o{:}N_{\mathscr{P}'}$, has each plaza, i.e., the hub designated by the hub identifier hil[i], connected to one or more ordinary net links, $\{l_{i_1}, l_{i_2}, \cdots, l_{i_k}\}$.

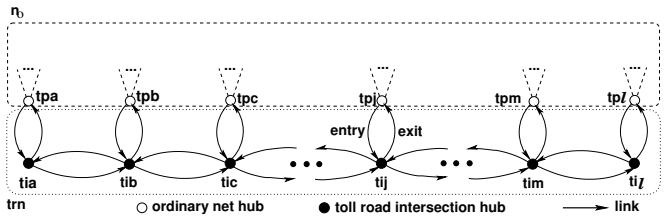

Figure 3: The Instantiated Road Net

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

63b. The toll-road net, trn:TRN$_{\mathscr{I}}$, consists of three collections (modeled as lists) of links and hubs:

i a list of pairs of toll-road entry/exit links: $\langle(l_{e_1}, l_{x_1}), \cdots, (l_{e_\ell}, l_{x_\ell})\rangle$,

ii a list of toll-road intersection hubs: $\langle h_{i_1}, h_{i_2}, \cdots, h_{i_\ell}\rangle$, and

iii a list of pairs of main toll-road ("up" and "$d$own") links: $\langle(ml_{i_{1u}}, ml_{i_{1d}}), (m_{i_{2u}}, m_{i_{2d}}), \cdots, (m_{i_{\ell u}}, m_{i_{\ell d}})\rangle$.

d. The three lists have commensurate lengths (ℓ).

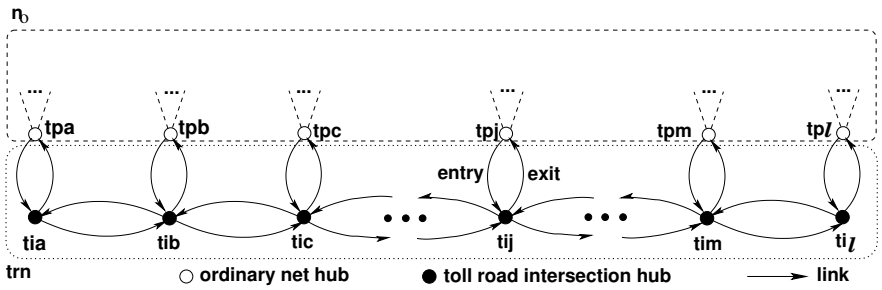

Figure 4: The Instantiated Road Net

type

62 $\Delta_{\mathscr{I}}$

63 $N_{\mathscr{I}} = N_{\mathscr{P}'} \times HIL \times TRN$

63a. $N_{\mathscr{P}'}$

63b. $TRN_{\mathscr{I}} = (L \times L)^* \times H^* \times (L \times L)^*$

63c. $HIL = HI^*$

axiom

63d. $\forall\ n_{\mathscr{I}}{:}N_{\mathscr{I}} \cdot$

63d. **let** $(n_{\Delta}, hil, (exll, hl, lll)) = n_{\mathscr{I}}$ **in**

63d. **len** hil = **len** exll = **len** hl = **len** lll + 1

63d. **end**

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

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

64 The toll-road intersection hubs all[7] have distinct identifiers.

64 wf_dist_toll_road_isect_hub_ids: $H^* \to$ **Bool**

64 wf_dist_toll_road_isect_hub_ids(hl) ≡

64 **len** hl = **card** xtr_his(hl)

[7] A 'must' can be inserted in front of all 'all's,

65 The toll-road links all have distinct identifiers.

```
65 wf_dist_toll_road_u_d_link_ids: (L×L)*→Bool
65 wf_dist_toll_road_u_d_link_ids(lll) ≡
65     2 × len lll = card xtr_lis(lll)
```

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

```
66 wf_dist_e_x_link_ids: (L×L)*→Bool
66 wf_dist_e_x_link_ids(exll) ≡
66     2 × len exll = card xtr_lis(exll)
```

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

```
67 wf_isoltd_toll_road_isect_hubs: Hl*×H*→N𝒥→Bool
67 wf_isoltd_toll_road_isect_hubs(hil,hl)(n𝒥) ≡
67     let ls=xtr_links(n𝒥) in
67     let his = ∪ {obs_mereo_L(l)|l:L·l ∈ ls} in
67     his ∩ xtr_his(hl) = {} end end
```

68 The plaza hub identifiers must designate hubs of the 'ordinary' net.

68 wf_p_hubs_pt_of_ord_net: HI*→N$'_\Delta$→**Bool**
68 wf_p_hubs_pt_of_ord_net(hil)(n'$_\Delta$) ≡
68 **elems** hil ⊆ xtr_his(n$'_\Delta$)

69 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.

69 wf_p_hub_interf: N$'_\Delta$→**Bool**
69 wf_p_hub_interf(n$_o$,hil,(exll,_,_)) ≡
69 ∀ i:**Nat** · i ∈ **inds** exll ⇒
69 **let** h = get_H(hil(i))(n$'_\Delta$) **in**
69 **let** lis = **obs_mereo**_H(h) **in**
69 **let** lis$'$ = lis \ xtr_lis(n$'$) **in**
69 lis$'$ = xtr_lis(exll(i)) **end end end**

70 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.

```
70   wf_toll_road_isect_hub_iface: N_𝒥→Bool
70   wf_toll_road_isect_hub_iface(_,_,(exll,hl,lll)) ≡
70      ∀ i:Nat · i ∈ inds hl ⇒
70         obs_mereo_H(hl(i)) =
70a.          xtr_lis(exll(i)) ∪
70         case i of
70b.           1 → xtr_lis(lll(1)),
70b.           len hl → xtr_lis(lll(len hl−1))
70b.           _ → xtr_lis(lll(i)) ∪ xtr_lis(lll(i−1))
70         end
```

71 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.

```
71  wf_exll: (L×L)*×Hl*×H*→Bool
71  wf_exll(exll,hil,hl) ≡
71      ∀ i:Nat · i ∈ len exll
71          let (hi,(el,xl),h) = (hil(i),exll(i),hl(i)) in
71          obs_mereo_L(el) = obs_mereo_L(xl)
71          = {hi} ∪ {uid_H(h)} end
71      pre: len eell = len hil = len hl
```

72 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.

```
72  wf_u_d_links: (L×L)*×H*→Bool
72  wf_u_d_links(lll,hl) ≡
72      ∀ i:Nat · i ∈ inds lll ⇒
72          let (ul,dl)  = lll(i) in
72          obs_mereo_L(ul) = obs_mereo_L(dl) =
72a.          uid_H(hl(i)) ∪ uid_H(hl(i+1)) end
72      pre: len lll = len hl+1
```

- We have used some additional auxiliary functions:

xtr_his: H* → HI-set
xtr_his(hl) ≡ {uid_HI(h)|h:H·h ∈ elems hl}
xtr_lis: (L×L) → LI-set
xtr_lis(l′,l″) ≡ {uid_LI(l′)}∪{uid_LI(l″)}
xtr_lis: (L×L)* − LI-set
xtr_lis(lll) ≡
∪{xtr_lis(l′,l″)|(l′,l″):(L×L)·(l′,l″)∈ elems lll}

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

73 wf_instantiated_net: $N_{\mathscr{J}} \to \mathbf{Bool}$
73 wf_instantiated_net(n′$_\Delta$,hil,(exll,hl,lll))
64 wf_dist_toll_road_isect_hub_ids(hl)
65 $\wedge$ wf_dist_toll_road_u_d_link_ids(lll)
66 $\wedge$ wf_dist_e_e_link_ids(exll)
67 $\wedge$ wf_isolated_toll_road_isect_hubs(hil,hl)(n′)
68 $\wedge$ wf_p_hubs_pt_of_ord_net(hil)(n′)
69 $\wedge$ wf_p_hub_interf(n′$_\Delta$,hil,(exll,_,_))
70 $\wedge$ wf_toll_road_isect_hub_iface(_,_,(exll,hl,lll))
71 $\wedge$ wf_exll(exll,hil,hl)
72 $\wedge$ wf_u_d_links(lll,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, $n_{\mathscr{P}}$:$N_{\mathscr{P}}$,
  - into a partially concrete definition of nets, $n_{\mathscr{I}}$:$N_{\mathscr{I}}$.
- We need to show the refinement relation:
  - abstraction($n_{\mathscr{I}}$) = $n_{\mathscr{P}}$.

```
value
74  abstraction: N_𝒥 → N_𝒫
75  abstraction(n′_Δ,hil,(exll,hl,lll)) ≡
76    let n_𝒫:N_𝒫 ·
76      let hs = obs_part_HS_𝒫(obs_part_HA_𝒫(n′_𝒫)),
76          ls = obs_part_LS_𝒫(obs_part_LA_𝒫(n′_𝒫)),
76          ths = elems hl,
76          eells = xtr_links(eell), llls = xtr_links(lll) in
77          hs∪ths=obs_part_HS_𝒫(obs_part_HA_𝒫(n_𝒫))
78       ∧  ls∪eells∪llls=obs_part_LS_𝒫(obs_part_LA_𝒫(n_𝒫))
79    n_𝒫 end end
```

74 The abstraction function takes a concrete net, $n_{\mathscr{I}}:N_{\mathscr{I}}$, and yields an abstract net, $n_{\mathscr{P}}:N_{\mathscr{P}}$.

75 The abstraction function doubly decomposes its argument into constituent lists and sub-lists.

76 There is postulated an abstract net, $n_{\mathscr{P}}:N_{\mathscr{P}}$, such that

77 the hubs of the concrete net and toll-road equals those of the abstract net, and

78 the links of the concrete net and toll-road equals those of the abstract net.

79 And that abstract net, $n_{\mathscr{P}}:N_{\mathscr{P}}$, is postulated to be an abstraction of the concrete net.

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 sôlely 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

80 *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.

80 opposite_traffics: $(L \times L)^* \times (L \times L)^* \to \mathbf{Bool}$
80 opposite_traffics(exll,lll) $\equiv$
80 $\quad \forall$ (lt,lf):$(L \times L) \cdot$ (lt,lf) $\in$ **elems** exll$^\frown$lll $\Rightarrow$
80a. $\quad$ **let** (ltσ,lfσ) = (**attr**_LΣ(lt),**attr**_LΣ(lf)) **in**
80a.′. $\quad$ **attr**_LΩ(lt)=$\{$lt$\sigma\}\wedge$**attr**_LΩ(ft)=$\{$ft$\sigma\}$
80a.″. $\quad \wedge$ **card** ltσ = 1 = **card** lfσ
80 $\quad \wedge$ **let** $(\{$(hi,hi′)$\},\{$(hi″,hi‴)$\})$ = (ltσ,lfσ) **in**
80c. $\quad$ hi=hi‴ $\wedge$ hi′=hi″
80 $\quad$ **end end**

All Toll-road Hubs are Free-flow

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

a. from *e*ntry links back to the paired e*x*it *l*ink*s*,

b. from *e*ntry links to emanating *t*oll-road *l*ink*s*,

c. from incident *t*oll-road links to e*x*it *l*ink*s*, and

d. from incident *t*oll-road link to emanating *t*oll-road *l*ink*s*.

```
81   free_flow_toll_road_hubs: (L×L)*×(L×L)*→Bool
81   free_flow_toll_road_hubs(exl,ll) ≡
81      ∀ i:Nat·i ∈ inds hl ⇒
81          attr_HΣ(hl(i)) =
81a.              hσ_ex_ls(exl(i))
81b.            ∪ hσ_et_ls(exl(i),(i,ll))
81c.            ∪ hσ_tx_ls(exl(i),(i,ll))
81d.            ∪ hσ_tt_ls(i,ll)
```

81a.: from *e*ntry links back to the paired e*x*it *l*inks*s*:

81a. hσ_ex_ls: (L×L)→LΣ
81a. hσ_ex_ls(e,x) ≡ {(**uid**_LI(e),**uid**_LI(x))}

81b.: from *e*ntry links to emanating *t*oll-road *l*ink*s*:

```
81b.  hσ_et_ls: (L×L)×(Nat×(em:L×in:L)*)→LΣ
81b.  hσ_et_ls((e,_),(i,ll)) ≡
81b.      case i of
81b.          2        → {(uid_LI(e),uid_LI(em(ll(1))))},
81b.          len ll+1 → {(uid_LI(e),uid_LI(em(ll(len ll))))},
81b.          _        → {(uid_LI(e),uid_LI(em(ll(i−1)))),
81b.                      (uid_LI(e),uid_LI(em(ll(i))))}
81b.      end
```

- The *em* and *in* in the toll-road link list (em:L×in:L)* designate selectors for *em*anating, respectively *in*cident links.

81c.: from incident *t*oll-road links to e*x*it *l*ink*s*:

```
81c.  hσ_tx_ls: (L×L)×(Nat×(em:L×in:L)*)→LΣ
81c.  hσ_tx_ls((_,x),(i,ll)) ≡
81c.      case i of
81c.          2          → {(uid_LI(in(ll(1))),uid_LI(x))},
81c.          len ll+1   → {(uid_LI(in(ll(len ll))),uid_LI(x))},
81c.          _          → {(uid_LI(in(ll(i−1))),uid_LI(x)),
81c.                         (uid_LI(in(ll(i))),uid_LI(x))}
81c.      end
```

81d.: from incident *t*oll-road link to emanating *t*oll-road *l*ink*s*:

```
81d.  hσ_tt_ls: Nat×(em:L×in:L)*→LΣ
81d.  hσ_tt_ls(i,ll) ≡
81d.      case i of
81d.          2          → {(uid_LI(in(ll(1))),uid_LI(em(ll(1))))},
81d.          len ll+1 → {(uid_LI(in(ll(len ll))),uid_LI(em(ll(len ll))))},
81d.          _          → {(uid_LI(in(ll(i−1))),uid_LI(em(ll(i−1)))),
81d.                        (uid_LI(in(ll(i))),uid_LI(em(ll(i))))}
81d.      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.
  - Typically perdurants are expressed in terms of expressions and statements.
  - And, typically, perdurant non-determinism is expressed in terms of the choice or parallelism operators: $\mathsf{C}_{i_a} \sqcap \mathsf{C}_{i_b}$ or $\mathsf{C}_{x_a} \square \mathsf{C}_{x_b}$ or $\mathsf{C}_{p_a} \| \mathsf{C}_{p_b}$.
  - 'Perdurant determination' is then, typically, a matter of making the choice conditional (i,x) or of "sequentializing" parallelism (p).

    (i) $\mathsf{C}_{i_a} \sqcap \mathsf{C}_{i_b} \Rightarrow$ **if** B_i **then** C_{i_a} **else** C_{i_b} **end**
    (x) $\mathsf{C}_{x_a} \square \mathsf{C}_{x_b} \Rightarrow$ **if** B_x **then** C_{x_a} **else** C_{x_b} **end**
    (p) $\mathsf{C}_{p_a} \| \mathsf{C}_{p_b} \Rightarrow \mathsf{C}_{p_a} ; \mathsf{C}_{p_b}$

4.4. Domain Extension

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

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

- Usually extensions involving one of the main sorts entails extensions involving several of the main sorts.
- In our example we introduce (i.e., “extend”)
  - vehicles with GPSS-like sensors,
  - and introduce toll-gates with
    - entry sensors,
    - vehicle identification sensors,
    - gate actuators and
    - exit sensors.
  - Finally road pricing calculators are introduced.

4.4.1. The Requirements Example: Domain Extension

Example 13 Domain Requirements — Extension:

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

82 There is a domain, $\delta_{\mathscr{E}}$:$\Delta_{\mathscr{E}}$, which contains

83 a fleet, $\mathsf{f}_{\mathscr{E}}$:$\mathsf{F}_{\mathscr{E}}$, that is,

84 a set, $\mathsf{vs}_{\mathscr{E}}$:$\mathsf{VS}_{\mathscr{E}}$, of

85 extended vehicles, $\mathsf{v}_{\mathscr{E}}$:$\mathsf{V}_{\mathscr{E}}$ — their extension amounting to

86 a dynamic reactive attribute, whose value, ti-gpos:TiGpos, at any time, reflects that vehicle's time-stamped global position.

87 The vehicle's GNSS receiver calculates, *loc_pos*, its local position, lpos:LPos, based on these signals.

88 Vehicles access these external attributes via the external attribute channel, attr_TiGPos_ch.

type
82 $\Delta_{\mathscr{E}}$
83 $\mathsf{F}_{\mathscr{E}}$
84 $\mathsf{VS}_{\mathscr{E}} = \mathsf{V}_{\mathscr{E}}$-**set**
85 $\mathsf{V}_{\mathscr{E}}$
86 TiGPos $= \mathbb{T} \times$ GPos
87 GPos, LPos
value
82 $\delta_{\mathscr{E}}$:$\Delta_{\mathscr{E}}$
83 **obs_part**_$\mathsf{F}_{\mathscr{E}}$: $\Delta_{\mathscr{E}} \to \mathsf{F}_{\mathscr{E}}$
83 f $=$ **obs_part**_$\mathsf{F}_{\mathscr{E}}(\delta_{\mathscr{E}})$
84 **obs_part**_$\mathsf{VS}_{\mathscr{E}}$: $\mathsf{F}_{\mathscr{E}} \to \mathsf{VS}_{\mathscr{E}}$
84 vs $=$ **obs_part**_$\mathsf{VS}_{\mathscr{E}}$(f)
84 vis $=$ xtr_vis(vs)
86 attr_TiGPos_ch[vi]?
87 loc_pos: GPos $\to$ LPos
channel
87 {attr_TiGPos_ch[vi]|vi:VI·vi $\in$ vis}:TiGPos

We define two auxiliary functions,

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

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

```
value
89  xtr_vs: (Δ_ℰ|F_ℰ|VS_ℰ) → V_ℰ-set
89  xtr_vs(arg) ≡
89      is_Δ_ℰ(arg) → obs_part_VS_ℰ(obs_part_F_ℰ(arg)),
89      is_F_ℰ(arg) → obs_part_VS_ℰ(arg),
89      is_VS_ℰ(arg) → arg
90  xtr_vis: (Δ_ℰ|F_ℰ|VS_ℰ) → VI-set
90  xtr_vis(arg) ≡ {uid_VI(v)|v ∈ xtr_vs(arg)}
```

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

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

type
91 C, CI
value
91 **obs_part**_C: $\Delta_{\mathscr{E}} \rightarrow$ C
91 **uid**_CI: C $\rightarrow$ CI
91 c = **obs_part**_C($\delta_{\mathscr{E}}$)
91 ci = **uid**_CI(c)

4.4.1.3 [c] Vehicle to Road Pricing Calculator Channel:

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

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

type

92 VITiLPos = VI × (𝕋 × LPos)

channel

93 {v_c_ch[vi,ci]|vi:VI,ci:CI·vi∈vis∧ci=**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 5 illustrates the idea of gates.

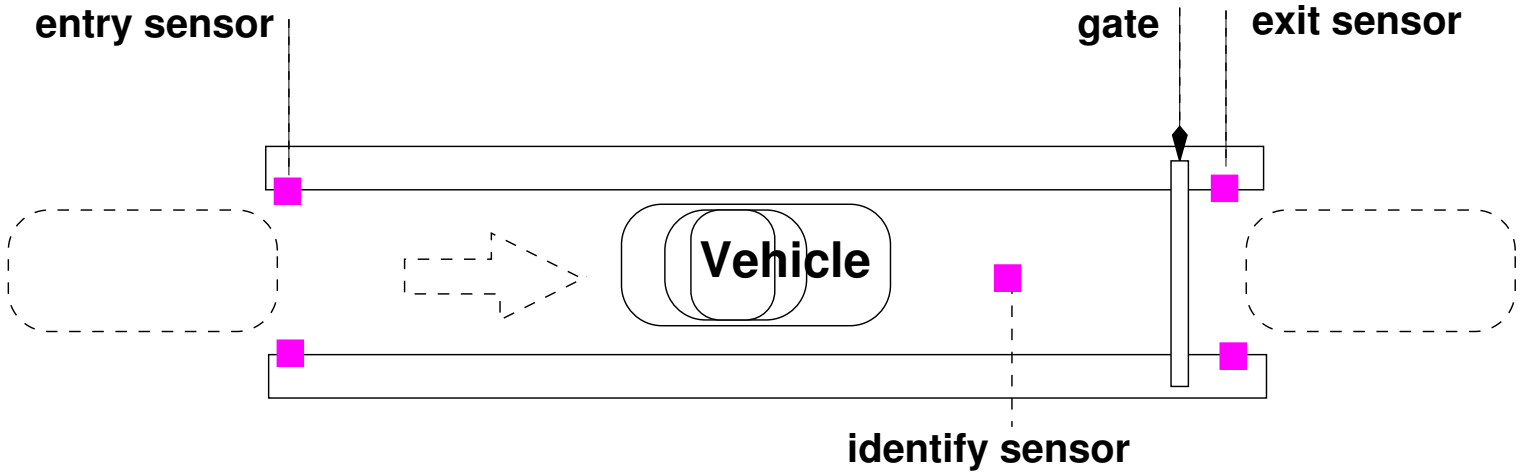

Figure 5: A toll plaza gate

- Figure 5 on the previous slide is intended to illustrate a vehicle entering (or exiting) a toll-road entry link.
  - The toll-gate is equipped with three sensors:
    an entry sensor, a vehicle identification sensor and an exit sensor.
  - The entry sensor serves to prepare
    the vehicle identification sensor.
  - The exit sensor serves to prepare
    the gate for closing when a vehicle has passed.
  - The vehicle identify sensor identifies the vehicle and "delivers" a pair: the current time and the vehicle identifier.
  - Once the vehicle identification sensor has identified a vehicle
    - the gate opens and
    - a message is sent to the road pricing calculator as to the passing vehicle's identity and the identity of the link associated with the toll-gate (see Items 109- 110 on Slide 168).

94 The domain contains the extended net, n:$N_{\mathscr{E}}$,

95 with the net extension amounting to the toll-road net, $TRN_{\mathscr{E}}$, that is, the instantiated toll-road net, trn:$TRN_{\mathscr{J}}$, is extended, into trn:$TRN_{\mathscr{E}}$, with entry, eg:EG, and exit, xg:XG, toll-gates.

From entry- and exit-gates we can observe

96 their unique identifier and

97 their mereology: pairs of entry-, respectively exit link and calculator unique identifiers; further

98 a pair of gate entry and exit sensors modeled as external attribute channels, (ges:ES,gls:XS), and

99 a time-stamped vehicle identity sensor modeled as external attribute channels.

type
94 $N_{\mathscr{E}}$
95 $TRN_{\mathscr{E}} = (EG \times XG)^* \times TRN_{\mathscr{I}}$
96 GI
value
94 **obs_part**_$N_{\mathscr{E}}$: $\Delta_{\mathscr{E}} \to N_{\mathscr{E}}$
95 **obs_part**_$TRN_{\mathscr{E}}$: $N_{\mathscr{E}} \to TRN_{\mathscr{E}}$
96 **uid**_G: (EG|XG) → GI
97 **obs_mereo**_G: (EG|XG) → (LI×CI)
95 trn:$TRN_{\mathscr{E}}$ = **obs_part**_$TRN_{\mathscr{E}}(\delta_{\mathscr{E}})$
channel
98 {attr_entry_ch[gi]|gi:GI·xtr_eGIds(trn)} "enter"
98 {attr_exit_ch[gi]|gi:GI·xtr_xGIds(trn)} "exit"
99 {attr_identity_ch[gi]|gi:GI·xtr_GIds(trn)} TIVI
type
99 TIVI = $\mathbb{T} \times$ VI

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

100 xtr_eGℓ extracts the ℓist of entry gates,

101 xtr_xGℓ extracts the ℓist of exit gates,

102 xtr_eGIds extracts the $s$et of entry gate identifiers,

103 xtr_xGIds extracts the $s$et of exit gate identifiers,

104 xtr_Gs extracts the $s$et of all gates, and

105 xtr_GIds extracts the $s$et of all gate identifiers.

```
value
100  xtr_eGℓ: TRN_ℰ → EG*
100  xtr_eGℓ(pgl,_) ≡ {eg|(eg,xg):(EG,XG)·(eg,xg)∈ elems pgl}
101  xtr_xGℓ: TRN_ℰ → XG*
101  xtr_xGℓ(pgl,_) ≡ {xg|(eg,xg):(EG,XG)·(eg,xg)∈ elems pgl}
102  xtr_eGIds: TRN_ℰ → GI-set
102  xtr_eGIds(pgl,_) ≡ {uid_GI(g)|g:EG·g ∈ xtr_eGs(pgl,_)}
103  xtr_xGIds: TRN_ℰ → GI-set
103  xtr_xGIds(pgl,_) ≡ {uid_GI(g)|g:EG·g ∈ xtr_xGs(pgl,_)}
104  xtr_Gs: TRN_ℰ → G-set
104  xtr_Gs(pgl,_) ≡ xtr_eGs(pgl,_) ∪ xtr_xGs(pgl,_)
105  xtr_GIds: TRN_ℰ → GI-set
105  xtr_GIds(pgl,_) ≡ xtr_eGIds(pgl,_) ∪ xtr_xGIds(pgl,_)
```

106 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

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

108 the unique identifiers, vis:VI-**set**, of the fleet vehicles.

axiom

106 $\forall$ n:$\mathsf{N}_{\mathscr{R}_3}$, trn:$\mathsf{TRN}_{\mathscr{R}_3}$ ·
106 **let** (exgl,(exl,hl,lll)) = **obs_part_TRN**$_{\mathscr{R}_3}$(n) **in**
106a. **len** exgl = **len** exl = **len** hl = **len** lll + 1
106b. ∧ **card** xtr_Glds(exgl) = 2 ∗ **len** exgl
106c. ∧ ∀ i:**Nat**·i ∈ **inds** exgl·
106c. **let** ((eg,xg),(el,xl)) = (exgl(i),exl(i)) **in**
106c. **obs_mereo**_G(eg) = (**uid**_U(el),ci,vis)
106c. ∧ **obs_mereo**_G(xg) = (**uid**_U(xl),ci,vis)
106 **end end**

4.4.1.5 [e] Toll-gate to Calculator Channels:

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

110 to the road pricing calculator via channel.

type

109 $\mathsf{EEVITiLI} = (\text{"Entry"}|\text{"Exit"})\times(\mathsf{VI}\times(\mathbb{T}\times\mathsf{SonL}))$

channel

110 $\{\mathsf{g\_c\_ch}[\mathsf{gi},\mathsf{ci}]|\mathsf{gi}:\mathsf{GI}\cdot\mathsf{gi}\in\mathsf{gis}\}:\mathsf{EEVITiLI}$

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 *v*ehicle *p*osition *l*ocation *f*unction, vplf:VPLF, which, given a local position, lpos:LPos, yields *either* the simple vehicle position, svpos:SVPos, designated by the GNSS-provided position, *or* yields the response that the provided position is off the toll-road net The vplf:VPLF function is constructed, construct_vplf,

113 from awareness, of a geodetic road map, GRM, of the topology of the extended net, $\mathsf{n}_{\mathscr{E}}$:$\mathsf{N}_{\mathscr{E}}$, including the mereology and the geodetic attributes of links and hubs.

type

111 TRM = VI $\xrightarrow[m]{}$ ($\mathbb{T}$ × SVPos)*

112 VPLF = GRM → LPos → (SVPos | "off_N")

113 GRM

value

111 **attr**_TRM: $C_{\mathscr{E}}$ → TRM

112 **attr**_VPLF: $C_{\mathscr{E}}$ → VPLF

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

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

21 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  GRM = (GeoL →m LI) ∪ (GeoH →m HI)
value
115  geo_GRM: N → GRM
115  geo_GRM(n) ≡
115    let ls = xtr_links(n), hs = xtr_hubs(n) in
115    [attr_GeoL(l)↦uid_LI(l)|l:L·l ∈ ls]
115    ∪
115    [attr_GeoH(h)↦uid_HI(h)|h:H·h ∈ 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, $\delta_{\mathscr{E}}$:$\Delta_{\mathscr{E}}$;

118 there is the net, $\mathsf{n}_{\mathscr{E}}$:$\mathsf{N}_{\mathscr{E}}$, of that domain;

119 there is toll-road net, $\mathsf{trn}_{\mathscr{E}}$:$\mathsf{TRN}_{\mathscr{E}}$, of that net;

120 there is a set, $\mathsf{egs}_{\mathscr{E}}$:$\mathsf{EG}_{\mathscr{E}}$**-set**, of entry gates;

121 there is a set, $\mathsf{xgs}_{\mathscr{E}}$:$\mathsf{XG}_{\mathscr{E}}$**-set**, of exit gates;

122 there is a set, $\mathsf{gis}_{\mathscr{E}}$:$\mathsf{GI}_{\mathscr{E}}$**-set**, ofgate identifiers;

123 there is a set, $\mathsf{vs}_{\mathscr{E}}$:$\mathsf{V}_{\mathscr{E}}$**-set**, of vehicles;

124 there is a set, $\mathsf{vis}_{\mathscr{E}}$:$\mathsf{VI}_{\mathscr{E}}$**-set**, of vehicle identifiers;

125 there is the road-pricing calculator, $\mathsf{c}_{\mathscr{E}}$:$\mathsf{C}_{\mathscr{E}}$ and

126 there is its unique identifier, $\mathsf{ci}_{\mathscr{E}}$:$\mathsf{CI}$.

value

117 $\delta_{\mathscr{E}}$:$\Delta_{\mathscr{E}}$
118 $n_{\mathscr{E}}$:$N_{\mathscr{E}}$ = **obs_part**_$N_{\mathscr{E}}$($\delta_{\mathscr{E}}$)
119 $trn_{\mathscr{E}}$:$TRN_{\mathscr{E}}$ = **obs_part**_$TRN_{\mathscr{E}}$($n_{\mathscr{E}}$)
120 $egs_{\mathscr{E}}$:EG-**set** = xtr_egs($trn_{\mathscr{E}}$)
121 $xgs_{\mathscr{E}}$:XG-**set** = xtr_xgs($trn_{\mathscr{E}}$)
122 $gis_{\mathscr{E}}$:XG-**set** = xtr_gis($trn_{\mathscr{E}}$)
123 $vs_{\mathscr{E}}$:$V_{\mathscr{E}}$-**set** = **obs_part**_VS(**obs_part**_$F_{\mathscr{E}}$($\delta_{\mathscr{E}}$))
124 $vis_{\mathscr{E}}$:VI-**set** = {**uid**_VI($v_{\mathscr{E}}$)|$v_{\mathscr{E}}$:$V_{\mathscr{E}}$·$v_{\mathscr{E}}$ ∈ $vs_{\mathscr{E}}$}
125 $c_{\mathscr{E}}$:$C_{\mathscr{E}}$ = **obs_part**_$C_{\mathscr{E}}$($\delta_{\mathscr{E}}$)
126 $ci_{\mathscr{E}}$:$CI_{\mathscr{E}}$ = **uid**_CI($c_{\mathscr{E}}$)

4.4.1.8 [h] "Total" System Behaviour:

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

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

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

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

```
value
127  sys: Unit → Unit
127  sys() ≡
127a.      ‖ {veh(uid_V(v),(ci,gis),attr_TiGPos_ch)|v:V·v ∈ vs_ℰ}
127b.    ‖ ‖ {gate("Entry")(gi,obs_mereo_G(eg),
127b.            (attr_entry_ch[gi],attr_identify_ch[gi],attr_exit_ch[gi]))
127b.            |eg:EG·eg ∈ egs_ℰ∧gi=uid_EG(eg)}
127c.    ‖ ‖ {gate("Exit")(gi,obs_mereo_G(xg),
127c.            (attr_entry_ch[gi],attr_identify_ch[gi],attr_exit_ch[gi]))
127c.            |xg:XG·xg ∈ xgs_ℰ∧gi=uid_XG(xg)}
127d.      ‖ calc(ci_ℰ,(vis_ℰ,gis_ℰ))(rlf)(trm)

128  veh: vi:VI×(ci:CI×gis:GI-set)×𝕌TiGPos → out v_c_ch[vi,ci] Unit
134  gate: ee:EE×gi:GI×(ci:CI×VI-set×LI)×
134          (𝕌entry×𝕌identify×𝕌exit) → out g_c_ch[gi,ci]  Unit
168  calc: ci:CI×(vis:VI-set×gis:GI-set)×VPLF→TRM →
168          in {v_c_ch[vi,ci]|vi:VI·vi ∈ vis},{g_c_ch[gi,ci]|gi:GI·gi ∈ gis}  Unit
```

Vehicle Behaviour:

128 Instead of moving around by explicitly expressed internal non-determinism[8] 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, (vi,(τ,lpos)), to the road pricing subsystem —

132 whereupon it resumes its vehicle behaviour.

[8] We refer to Items 47b., 47c. on Slide 55 and 48b., 48(b.)ii, 48c. on Slide 57

```
value
128  veh: vi:VI×(ci:CI×gis:GI-set)×𝕌TiGPos →
128      out v_c_ch[vi,ci] Unit
128  veh(vi,(ci,gis),attr_TiGPos_ch[vi]) ≡
129      let (τ,gpos) = attr_TiGPos_ch[vi]? in
130      let lpos = loc_pos(gpos) in
131      v_c_ch[vi,ci] ! (vi,(τ,lpos)) ;
132      veh(vi,(ci,gis),attr_TiGPos_ch[vi]) end end
128      pre vi ∈ vis_ℰ ∧ ci = ci_ℰ ∧ gis = gis_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 [Bjø10].

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.

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.

138 And finally, after that vehicle has left the entry or exit gate

139 that toll-gate's behaviour is resumed.

```
type
133  EE = "Enter" | "Exit"
value
134  gate: ee:EE×gi:GI×(ci:CI×VI-set×LI)×
134          (𝕌enter×𝕌passing×𝕌leave) →
134            out g_c_ch[gi,ci] Unit
134  gate(ee,gi,(ci,vis,li),
134          ea:(attr_enter_ch[gi],attr_passing_ch[gi],attr_leave_ch[gi])) ≡
135    attr_enter_ch[gi] ? ;
136    let (τ,vi) = attr_passing_ch[gi] ? in assert vi ∈ vis
137    g_c_ch[gi,ci] ! (ee,(vi,(τ,SonL(li)))) ;
138    attr_leave_ch[gi] ? ;
139    gate(ee,gi,(ci,vis,li),ea)
134      end
134      pre ci = ci_ℰ ∧ vis = vis_ℰ ∧ li ∈ lis_ℰ
```

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

The above concerns would already have been addressed

- in a model of domain facets such as
  - *human behaviour*,
  - *technology support*,
  - proper tele-communications *scripts*,
  - etcetera.
- We refer to [Bjø10]

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}, ..., 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 15 *Requirements Fitting:*

- By **requirements fitting** we mean
  - a harmonisation of $n > 1$ domain requirements
  - that have overlapping (shared) not always consistent parts and
  - which results in
    - n partial domain requirements', $p_{d_{r_1}}, p_{d_{r_2}}, \ldots, p_{d_{r_n}}$, and
    - m shared domain requirements, $s_{d_{r_1}}, s_{d_{r_2}}, \ldots, s_{d_{r_m}}$,
    - that "fit into" two or more of the partial domain requirements

- The above definition pertains to the result of 'fitting'.
- The next definition pertains to the act, or process, of 'fitting'.

Definition 16 *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 *s*hared *d*omain *r*equirements texts, $sdrs$, into n partial domain requirements we mean that
  - there is for each *p*artial *d*omain *r*equirements, pdr_i,
  - an identified, non-empty *s*ubset of $sdrs$ (could be all of $sdrs$), $ssdrs_i$,
  - such that textually conjoining $ssdrs_i$ to pdr_i,
  - i.e., $ssdrs_i \oplus pdr_i$
  - can be claimed to yield the "original" d_{r_i},
  - that is, $\mathscr{M}(ssdrs_i \oplus pdr_i) \subseteq \mathscr{M}(d_{r_i})$,
  - where $\mathscr{M}$ 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 [Bjø10].

Example 14 Domain Requirements — Fitting:

- Example 13 hints at three possible sets of interface requirements:
  - (i) for a road pricing system, as will be illustrated in Sect. ;
  - (iii) for a vehicle monitoring and control system, and
  - (ii) for a toll-gate monitoring and control system.
- The vehicle monitoring and control system would focus on implementing the vehicle behaviour, see Items 128- 132 on Slide 179.
- The toll-gate monitoring and control system would focus on implementing the calculator behaviour, see Items 134- 139 on Slide 184.

The fitting amounts to ...

-
  - TO BE WRITTEN
- 

4.6.1. Discussion

- ◈
- ◈
- ◈

5. Interface Requirements

- We remind the listener that
  - **interface requirements**
    - can be expressed only using terms from
    - both the domain
    - and the machine ■

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

5.1. Shared Phenomena

- By **sharing** we mean
  - that some or all properties of an **endurant** is represented both
    - in the domain and
    - "inside" the machine, and
    - that their machine representation
    - must at suitable times
    - reflect their state in the domain;

    and/or
  - that an **action**
    - requires a sequence of several "on-line" interactions
    - between the machine (being requirements prescribed) and
    - the domain, usually a person or another machine;

    and/or

- 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.

5.2. Environment–Machine Interface

- Domain requirements extension, Sect. ,
  - 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. 5 on Slide 160.

- 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

88 {attr_TiGPos_ch[vi]|vi:VI·vi ∈ xtr_VIs(vs)}: TiGPos

- And we do not really represent the gate nor its sensors and actuator in the machine.
- But we do give an idealised description of the gate behaviour, see Items 134–139
- Instead we represent their dynamic gate attributes:
  - (98) the vehicle entry sensors (leftmost ■s),
  - (98) the vehicle identity sensor (center ■), and
  - (99) the vehicle exit sensors (rightmost ■s)
- by channels — we refer to Example 13 (Sect. , Slide 163):

channel
98 {attr_entry_ch[gi]|gi:GI·xtr_eGIds(trn)} "enter"
98 {attr_exit_ch[gi]|gi:GI·xtr_xGIds(trn)} "exit"
99 {attr_identity_ch[gi]|gi:GI·xtr_GIds(trn)} TIVI ■

5.3. 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

5.3.1. 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 = **uid**_L(l) or **uid**_H(h) for some link or hub,

146 m_j = **obs_mereo**_L(l) or **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 ∧ rh>0
type
142  M = RI×"link"×LNK | RI×"hub"×HUB
142  LNK = LI × HI-set × LATTRS
142  HUB = HI × LI-set × HATTRS
```

```
value
141  partitioning: L-set → Nat → (L-set)* | H-set → Nat → (H-set)*
141  partitioning(s)(r) as sl
141  post: len sl = r ∧ ∪ elems sl = s
141        ∧ ∀ si,sj:(L-set|H-set) ·
141              si≠{}∧sj≠{}∧{si,sj}⊆elems ss⇒si ∩ sj={}
```

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

channel
148 r_db: RI×(LNK|HUB)
value
148 link_rec: RI → L-**set** → **out** r_db **Unit**
148 hub_rec: RI → H-**set** → **out** r_db **Unit**
148 net_db: **Unit** → **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×LNK)-set
150  hub_db: (RI×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  lsl:(L-set)* = partitioning(ls)(rl)
153  lhl:(H-set)* = partitioning(hs)(rh)
155  rill:RI* axiom len rill = rl = card elems rill
155  rihl:RI* axiom len rihl = rh = card elems rihl
154  ab_initio_data: Unit → Unit
154  ab_initio_data() ≡
154     ‖ {lnk_rec(rill[i])(lsl[i])|i:Nat·1≤i≤rl} ‖
154     ‖ {hub_rec(rihl[i])(lhl[i])|i:Nat·1≤i≤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 → L-set → Unit
156  link_rec(ri,ls) ≡
157      for ∀ l:L·l ∈ 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 × H-set → Unit
156  hub_rec(ri,hs) ≡
157      for ∀ h:H·h ∈ 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., $(ui, me, attrs)$.

5.3.2. 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.

5.4. Shared Actions, Events and Behaviours

- TO BE WRITTEN

- We illustrate ideas of
  - shared actions, events and behaviours
  - through the domain requirements extension
  - of Sect. 7.2.4,
  - Example 13
    Slides 154–187.

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  calc: ci:CI×(vis:VI-set×gis:GI-set)→RLF→TRM→
169        in {v_c_ch[ci,vi]|vi:VI·vi ∈ vis},
170           {g_c_ch[ci,gi]|gi:GI·gi ∈ gis}  Unit
168  calc(ci,(vis,gis))(rlf)(trm) ≡
169     react_to_vehicles(ci,(vis,gis))(rlf)(trm)
168   □
170     react_to_gates(ci,(vis,gis))(rlf)(trm)
168     pre ci = ci_ℰ ∧ vis = vis_ℰ ∧ gis = gis_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) ≡
169    let (vi,(τ,lpos)) = ⨆{v_c_ch[ci,vi]|vi:VI·vi∈ vis} in
171      if vi ∈ dom trm
172        then let vp = vplf(lpos) in
173               calc(ci,(vis,gis),vplf)(trm†[vi↦trm⌢⟨(τ,vp)⟩]) end
174        else  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.

```
react_to_gates(ci,(vis,gis),vplf)(trm) ≡
  let (ee,(τ,(vi,li))) =
      ⨆{g_c_ch[ci,gi]|gi:GI·gi∈ gis} in
    case ee of
      "Enter" →
        calc(ci,(vis,gis),vplf)(trm∪[vi↦⟨(τ,SonL(li))⟩]),
      "Exit" →
        billing(vi,trm(vi)⌢⟨(τ,SonL(li))⟩);
        calc(ci,(vis,gis),vplf)(trm\{vi}) end end
```

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

6. Machine Requirements

Definition 17 ***Machine Requirements:*** By **machine requirements** we shall understand

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

Definition 18 ***The Machine:*** By the **machine** we shall understand

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

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

  shall be understood with "a grain of salt".
  - Let us explain.
    - The machine requirements statements
    - may contain references to domain entities
    - but these are meant to be generic references,
    - that is, references to certain classes of entities in general.

  We shall illustrate this "genericitiy" in some of the examples below.

- We shall, in particular, consider the following five kinds of machine requirements:
  - *performance requirements*,
  - *dependability requirements*,
  - *maintenance requirements*,
  - *platform requirements* and
  - *documentation requirements*.

6.1. Performance Requirements

Definition 19 ***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 19 Machine Requirements. Road Pricing System 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 ■

6.2. Dependability Requirements

- Dependability is a complex notion.

6.2.1. 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 20 *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 21 *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 22 *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 23 ***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 24 ***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.

- Having already discussed the "threats" aspect,
- we shall therefore discuss the "means" aspect of the *dependability tree*.

- Attributes:
  - Accessibility
  - Availability
  - Integrity
  - Reliability
  - Safety
  - Security
- Means:
  - Procurement
    - Fault prevention
    - Fault tolerance
  - Validation
    - Fault removal
    - Fault forecasting
- Threats:
  - Faults
  - Errors
  - Failures

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

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

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

- The crucial term above is “satisfies”.
- The issue is: To what “degree”?
- As we shall see — in a later later lecture — to cope properly
  - with dependability requirements and
  - their resolution

  requires that we deploy
  - mathematical formulation techniques,
  - including analysis and simulation,

  from statistics (stochastics, etc.).

6.2.2. 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 26 ***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 20 **Machine Requirements. Road Pricing System 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.

6.2.3. 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 27 ***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 21 **Machine Requirements. Road Pricing System 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 ■

6.2.4. Integrity

Definition 28 ***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 22 **Machine Requirements. Road Pricing System Integrity**:

- We do not require an explicit formulation of integrity.
- We instead refer to the
  - reliability,
  - safety,
  - security and
  - robustness

  measures (below) ■

6.2.5. Reliability

Definition 29 ***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 23 **Machine Requirements. Road Pricing System Reliability**:

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

6.2.6. Safety

Definition 30 ***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 24 **Machine Requirements. Road Pricing System 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 ■

6.2.7. 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 31 ***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 25 **Machine Requirements. Road Pricing System Security**:

- We omit consideration of road pricing system security ■

6.2.8. Robustness

Definition 32 ***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 26 Machine Requirements. Road Pricing System 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

6.3. Maintenance Requirements

Definition 33 ***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 45 on Slide 281.

6.3.1. Adaptive Maintenance

Definition 34 ***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 27 **Machine Requirements. Road Pricing System Adaptive Maintenance**:

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

6.3.2. Corrective Maintenance

Definition 35 ***Corrective Maintenance:*** By *corrective maintenance* we understand such maintenance which

- corrects a software error ■

Example 28 **Machine Requirements. Road Pricing System Corrective Maintenance**:

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

6.3.3. Perfective Maintenance

Definition 36 ***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 29 **Machine Requirements. Road Pricing System Perfective Maintenance**:

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

6.3.4. Preventive Maintenance

Definition 37 ***Preventive Maintenance:*** By *preventive maintenance* we understand such maintenance which

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

Example 30 **Machine Requirements. Road Pricing System Preventive Maintenance**:

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

6.3.5. Extensional Maintenance

Definition 38 ***Extensional Maintenance:*** By *extensional maintenance* we understand such maintenance which adds new functionalities to the software, i.e., which implements additional requirements ■

Example 31 **Machine Requirements. Road Pricing System Extensional Maintenance**:

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

6.4. Platform Requirements

6.4.1. Delineation and Facets of Platform Requirements

Definition 39 ***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 40 ***Platform Requirements:*** By *platform requirements* we mean a combination of the following:

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

6.4.2. Execution Platform

Definition 41 ***Execution Platform Requirements:*** By *execution platform requirements* we shall understand such machine requirements which

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

6.4.3. Demonstration Platform

Definition 42 ***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 ■

6.4.4. Development Platform

Definition 43 ***Development Platform Requirements:*** By *development platform requirements* we shall understand such machine requirements which

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

6.4.5. Maintenance Platform

Definition 44 ***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 32 Machine Requirements. Road Pricing System Platform Requirements:

- The road pricing system platform requirements are: the system shall
  - executed and demonstrated on to be detailed and
  - developed and maintained on to be detailed ■
  -

6.5. Documentation Requirements

Definition 45 ***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 46 ***Documentation Requirements:*** By *documentation requirements*

- we mean requirements
- of any of the software documents
- that together make up
  - software and
  - hardware[9]

Example 33 **Machine Requirements — Documentation**:

- The road pricing system documentation requirements shall include
  - all of the software documents
  - implied by Definition 45 on Slide 281 above

[9] — we omit a definition of what we mean by hardware such as the one we gave for software, cf. Definition 45 on Slide 281.

6.6. Discussion

TO BE TYPED

7. Conclusion

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

7.1. What has been Achieved ?

- We have put forward a “new approach” to requirements engineering.
  - The “newness” comes from its reliance on there being a reasonably “complete” domain description.
  - The “approach” is manifested in separating the concerns of
    - domain requirements, i.e., design assumptions,
    - from interface and machine requirements, i.e., design requirements.
-

7.2. Present Shortcomings

- ◈
  - ◈
  - ◈
  - ◈

7.3. Comparison to “Classical” Requirements Engineering

- ◈ [van09]
  - ◈ [Lau02]
  - ◈
  - ◈

7.4. Future Work: Research Challenges

- We have outlined three major stages of requirements development, and, within these, a number of steps.
- They can be used, we claim, to advantage, already now — as they have indeed been used over the years in projects with which we have been associated.
- But more experimental research and path-finder projects has to be absolved.
- A number of research issues need be studied. We list a few.
  - **Domain Facets:**
  -
  -
  -
  -

7.5. Acknowledgments

- ◈ 5 LINES TO BE WRITTEN

8. Bibliography

TO BE WRITTEN

9. References

[Bjø10] 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.

[Bjø14a] 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.

[Bjø14b] Dines Bjørner. Manifest Domains: Analysis & Description. Research Report, 2014. Superceded by [Bjø16].

[Bjø16] Dines Bjørner. Manifest Domains: Analysis & Description. *Expected published by Formal Aspects of Computing*, 44 pages. 2016.

[Lau02] Søren Lauesen. *Software Requirements - Styles and Techniques*. Addison-Wesley, UK, 2002.

[van09] Axel van Lamsweerde. *Requirements Engineering: From System Goals to UML Models to Software Specifications*. Wiley, 2009.