6 June 2016, 16:50 : Submitted to Formal Aspects of Computing

From Domain Descriptions to Requirements Prescriptions:

A Different Initial Approach to Requirements Engineering

Dines Bjørner

Fredsvej 11, DK-2840 Holte, Denmark.
DTU Compute, Technical University of Denmark, DK-2800 Kgs. Lyngby, Denmark.
E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/˜dibj

In [Bjø16b, *Manifest Domains: Analysis & Description*] we introduced a method for analysing and describing manifest domains. In this paper we show how to systematically, but, of course, not automatically, "derive" *initial* requirements prescriptions from domain descriptions. There are, as we see it, three kinds of requirements: (i) *domain requirements*, (ii) *interface requirements* and (iii) *machine requirements*. The *machine* is the hardware and software to be developed from the requirements. (i) **Domain requirements** are those requirements which can be expressed 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. We show principles, techniques and tools for "deriving" domain requirements. The domain requirements development focus on (i.1) *projection*, (i.2) *instantiation*, (i.3) *determination*, (i.4) *extension* and (i.5) *fitting*. We briefly review principles, techniques and tools for "deriving" *interface requirements* based on sharing domain (ii.1) endurants, and (ii.2) perdurants (i.e., actions, events and behaviours) with their machine correspondants. The unfolding of interface requirements lead to a number of machine concepts in terms of which the interface requirements are expressed. These machine concepts, both hardware and software, make possible the expression of a set of — what we shall call — ***derived requirements***. The paper explores this concept briefly. We do not cover machine requirements in this paper. The reason is that we find, cf. [Bjø06, Sect. 19.6], that when the individual machine requirements are expressed then references to domain phenomena are, in fact, abstract references, that is, they do not refer to the semantics of what they name. This paper claims only to structure the quest for requirements conception. Instead of "discovering" requirements 'ab initio', for example, through interviews with stake-holders, we suggest to "derive" the requirements based on domain descriptions. Instead of letting the individual requirements arise out of initial stake-holder interviews, we suggest to structure these (i) around the structures of domain descriptions, and (ii) around the structures emerging from domain, interface and machine requirements. We shall refer to the requirements emerging from (i+ii) as the initial requirements. To these we add the *derived requirements* merging from interview with stakeholders: We are strongly of the opinion that the techniques and tools of, for example, [DvLF93, Jac01, ZH04, JHJ07, OD08, van09] can be smoothly integrated with those of this paper. We think that there is some clarification to be gained. We claim that our approach contributes to a restructuring of the field of requirements engineering and its very many diverse concerns, a structuring that is logically motivated and is based on viewing software specifications as mathematical objects.

Keywords: Requirements engineering, domain description, design assumptions, design requirements

Correspondence and offprint requests to: Dines Bjørner, Fredsvej 11, DK-2840 Holte, Denmark. e-mail: bjorner@gmail.com

1. Introduction

In [Bjø16b, *Manifest Domains: Analysis & Description*] we introduced a method for analysing and describing manifest domains. In this paper we show how to systematically, but, of course, not automatically, "derive" requirements prescriptions from domain descriptions.

1.1. The Triptych Dogma of Software Development

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

1.2. Software As Mathematical Objects

Our base view is that *computer program*s are *mathematical object*s. That is, the text that makes up a computer program can be reasoned about. This view entails that computer program specifications can be reasoned about. And that the *requirements prescription*s upon which these specifications are based can be reasoned about. This base view entails, therefore, that specifications, whether *software design specification*s, or *requirements prescription*s, or *domain description*s, must [also] be *formal specification*s. This is in contrast to considering *software design specification*s being artifacts of sociological, or even of psychological "nature".

1.3. The Contribution of This Paper

We claim that the present paper content contributes to our understanding and practice of *software engineering* as follows: (1) it shows how the new phase of engineering, domain engineering, as introduced in [Bjø16b], forms a prerequisite for requirements engineering; (2) it endows the "classical" form of requirements engineering with a structured set of development stages and steps: (a) first a domain requirements stage, (b) to be followed by an interface requirements stages, and (c) to be concluded by a machine requirements stage; (3) it further structures and gives a reasonably precise contents to the stage of domain requirements: (i) first a projection step, (ii) then an instantiation step, (iii) then a determination step, (iv) then an extension step, and (v) finally a fitting step — with these five steps possibly being iterated; and (4) it also structures and gives a reasonably precise contents to the stage of interface requirements based on a notion of shared entities, Each of the steps (i–v) open for the possibility of *simplification*s. Steps (a–c) and (i-v), we claim, are new. They reflect a serious contribution, we claim, to a logical structuring of the field of requirements engineering and its very many otherwise seemingly diverse concerns.

1.4. Some Comments on the Paper Content

By ***methodology*** we understand the study and knowledge of one or more methods ⊙[1] By a ***method*** understand the study and knowledge of the principles, techniques and tools for constructing some artifact, here (primarily) software ⊙ This paper is, perhaps, unusual in the following respects: (i) It is a methodology paper, hence there are no "neat" theories about development, no succinctly expressed propositions, lemmas nor theorems, and hence no proofs[2]. (ii) As a consequence the paper is borne by many, and by extensive examples. (iii) The examples of this paper are all focused on a generic road transport net. (iv) To reasonably fully exemplify the requirements approach, illustrating how our method copes with a seeming complexity of interrelated method aspects, the full example of this paper embodies very many description and prescription elements: hundreds of concepts (types, axioms, functions). (v) This methodology paper covers a "grand" area of software engineering: Many textbooks and papers are written on *Requirements Engineering*. We postulate, in contrast to all such books (and papers), that *requirements engineering* should be founded

[1] The ⊙ marks the end of definitions.

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

on *domain engineering*. Hence we must, somehow, show that our approach relates to major elements of what the *Requirements Engineering* books put forward. (vi) As a result, this paper is long.

1.5. Structure of Paper

The structure of the paper is as follows: Section 2 provides a fair-sized, hence realistic example. Sections 3–5 covers our approach to requirements development. Section 3 overviews the issue of 'requirements'; relates our approach (i.e., Sects. 4–5) to *systems*, *user and external equipment* and *functional requirements*; and Sect. 3 also introduces the concepts of the *machine* to be requirements prescribed, the *domain*, the *interface* and the *machine requirements*. Section 4 covers the *domain requirements* stages of *projection* (Sect. 4.1), *instantiation* (Sect. 4.2), *determination* (Sect. 4.3), *extension* (Sect. 4.4) and *fitting* (Sect. 4.5). Section 5 covers key features of *interface requirements*: *shared phenomena* (Sect. 5.1.1), *shared endurants* (Sect. 5.1.2) and *shared action*s, *shared event*sand *shared behaviours* (Sect. 5.1.3). Section 5.1.3 further introduces the notion of *derived requirements*. Section 7 concludes the paper.

2. An Example Domain: Transport

In order to exemplify the various stages and steps of requirements development we first bring a domain description example. The example follows the steps of an idealised domain description. First we describe the endurants, then we describe the perdurants. Endurant description initially focus on the composite and atomic parts. Then on their "internal" qualities: unique identifications, mereologies, and attributes. The descriptions alternate between enumerated, i.e., labeled narrative sentences and correspondingly "numbered" formalisations. The narrative labels cum formula numbers will be referred to, frequently in the various steps of domain requirements development.

2.1. Endurants

Since we have chosen a manifest domain, that is, a domain whose endurants can be pointed at, seen, touched, we shall follow the analysis & description process as outlined in [Bjø16b] and formalised in [Bjø14b]. That is, we first identify, analyse and describe (manifest) parts, composite and atomic, abstract (Sect. 2.1.1) or concrete (Sect. 2.1.2). Then we identify, analyse and describe their unique identifiers (Sect. 2.1.3), mereologies (Sect. 2.1.4), and attributes (Sects. 2.1.5–2.1.5).

The example fragments will be presented in a small type-font.

2.1.1. Domain, Net, Fleet and Monitor

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

The root domain, Δ, is that of a composite traffic system (1.a..) with a road net, (1.b..) with a fleet of vehicles and (1.c..) of whose individual position on the road net we can speak, that is, monitor.[3]

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

type
1. Δ
1.a. N
1.b. F
1.c. M

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

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

[3] The monitor can be thought of, i.e., conceptualised. It is not necessarily a physically manifest phenomenon.

2. The road net consists of two composite parts,
   a. an aggregation of hubs and
   b. an aggregation of links.

```
type
2.a.  HA
2.b.  LA
```

```
value
2.a.  obs_part_HA: N → HA
2.b.  obs_part_LA: N → LA
```

2.1.2. Hubs and Links

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

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
value
3.  obs_part_HS: HA → HS
4.  obs_part_LS: LA → LS
5.  obs_part_VS: F → VS
```

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

```
value
6.a.  links: Δ → L-set
6.a.  links(δ) ≡ obs_part_LS(obs_part_LA(obs_part_N(δ)))
6.b.  hubs: Δ → H-set
6.b.  hubs(δ) ≡ obs_part_HS(obs_part_HA(obs_part_N(δ)))
```

2.1.3. Unique Identifiers

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

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
7.a.  NI, HAI, LAI, HI, LI, FI, VI, MI
value
7.c.  uid_NI:  N → NI
7.c.  uid_HAI: HA → HAI
7.c.  uid_LAI: LA → LAI
7.c.  uid_HI:  H → HI
7.c.  uid_LI:  L → LI
7.c.  uid_FI: F → FI
7.c.  uid_VI: V → VI
7.c.  uid_MI: M → MI
axiom
7.b.  NI⋂HAI=Ø, NI⋂LAI=Ø, NI⋂HI=Ø, etc.
```

where axiom 7.b.. 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` [Bjø16b, 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_lis(δ) ∧
17.       ∀ l:L•l ∈ ls•obs_mereo_L(l)⊆xtr_his(δ) ∧
18.a.       let f:F•f=obs_part_F(δ) ⇒
18.a.           let m:M•m=obs_part_M(δ),
18.a.               vs:VS•vs=obs_part_VS(f) in
18.a.           ∀ v:V•v ∈ vs⇒uid_V(v) ∈ obs_mereo_M(m)
18.b.         ∧ obs_mereo_M(m) = {uid_V(v)|v:V•v ∈ vs}
18.b.       end end
```

2.1.5. Attributes, I

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

- **Hubs:** *location*s[4] are considered static, *hub state*s and *hub state space*s are considered programmable;
- **Links:** *length*s and *location*s are considered static, *link state*s and *link state space*s are considered programmable;
- **Vehicles:** *manufacturer name*, *engine type* (whether diesel, gasoline or electric) and *engine power* (kW/horse power) are considered static; *velocity* and *acceleration* may be considered reactive (i.e., a function of gas pedal position, etc.), *global position* (informed via a `GNSS: Global Navigation Satellite System`) and *local position* (calculated from a global position) are considered biddable □

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

First hubs.

19. Hubs
    a. have geodetic locations, GeoH,
    b. have *hub states* which are sets of pairs of identifiers of links connected to the hub[5],
    c. and have *hub state spaces* which are sets of hub states[6].
20. For every net,
    a. link identifiers of a hub state must designate links of that net.
    b. Every hub state of a net must be in the hub state space of that hub.
21. We introduce an auxiliary function: xtr_lis extracts all link identifiers of a hub state.

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

Then links.

22. Links have lengths.
23. Links have geodetic location.
24. Links have states and state spaces:
    a. States modeled here as pairs, (hi', hi''), of identifiers the hubs with which the links are connected and indicating directions (from hub h' to hub h''.) A link state can thus have 0, 1, 2, 3 or 4 such pairs.
    b. State spaces are the set of all the link states that a link may enjoy.

```
type
22.   LEN
23.    GeoL
```

[4] By location we mean a geodetic position.

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

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

```
24.a.  LΣ = (HI×HI)-set
24.b.  LΩ = LΣ-set
value
22.     attr_LEN: L → LEN
23.     attr_GeoL: L → GeoL
24.a.  attr_LΣ: L → LΣ
24.b.  attr_LΩ: L → LΩ
axiom
24.  ∀ n:N •
```

```
24.     let ls = xtr−links(n), hs = xtr_hubs(n) in
24.     ∀ l:L•l ∈ ls ⇒
24.a.       let lσ = attr_LΣ(l) in
24.a.       0≤card lσ≤4
24.a.      ∧ ∀ (hi′,hi″):(HI×HI)•(hi′,hi″) ∈ lσ
24.a.           ⇒ {hi′,hi″}=obs_mereo_L(l)
24.b.      ∧ attr_LΣ(l) ∈ attr_LΩ(l)
24.     end end
```

Then vehicles.

25. Every vehicle of a traffic system has a position which is either ‘on a link’ or ‘at a hub’.
    a. An ‘on a link’ position has four elements: a unique link identifier which must designate a link of that traffic system and a pair of unique hub identifiers which must be those of the mereology of that link.
    b. The ‘on a link’ position real is the fraction, thus properly between 0 (zero) and 1 (one) of the length from the first identified hub “down the link” to the second identifier hub.
    c. An ‘at a hub’ position has three elements: a unique hub identifier and a pair of unique link identifiers — which must be in the hub state.

```
type
25.     VPos = onL | atH
25.a.   onL :: LI HI HI R
25.b.   R = Real      axiom ∀ r:R • 0≤r≤1
25.c.   atH :: HI LI LI
value
25.     attr_VPos: V → VPos
```

```
axiom
25.a.   ∀ n:N, onL(li,fhi,thi,r):VPos •
25.a.      ∃ l:L•l ∈obs_part_LS(obs_part_N(n))
25.a.          ⇒ li=uid_L(l)∧{fhi,thi}=obs_mereo_L(l),
25.c.   ∀ n:N, atH(hi,fli,tli):VPos •
25.c.      ∃ h:H•h ∈obs_part_HS(obs_part_N(n))
25.c.          ⇒ hi=uid_H(h)∧(fli,tli) ∈ attr_LΣ(h)
```

26. We introduce an auxiliary function `distribute`.
    a. `distribute` takes a net and a set of vehicles and
    b. generates a map from vehicles to distinct vehicle positions on the net.
    c. We sketch a “formal” `distribute` function, but, for simplicity we omit the technical details that secures distinctness — and leave that to an axiom !

27. We define two auxiliary functions:
    a. xtr_links extracts all links of a net and
    b. xtr_hub extracts all hubs of a net.

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

```
27.a.   xtr_links: N → L-set
27.a.   xtr_links(n)≡
27.a.      obs_part_LS(obs_part_LA(n))
27.b.   xtr_hubs: N → H-set
27.a.   xtr_hubs(n)≡
27.a.      obs_part_H(obs_part_HA_Δ(n))
```

And finally monitors. We consider only one monitor attribute.

28. The monitor has a vehicle traffic attribute.
    a. For every vehicle of the road transport system the vehicle traffic attribute records a possibly empty list of time marked vehicle positions.
    b. These vehicle positions are alternate sequences of ‘on link’ and ‘at hub’ positions
        i such that any sub-sequence of ‘on link’ positions record the same link identifier, the same pair of ‘’to’ and ‘from’ hub identifiers and increasing fractions,
        ii such that any sub-segment of ‘at hub’ positions are identical,
        iii such that vehicle transition from a link to a hub is commensurate with the link and hub mereologies, and
        iv such that vehicle transition from a hub to a link is commensurate with the hub and link mereologies.

```
type
28.  Traffic = VI $\overrightarrow{m}$ (T × VPos)*
value
28.  attr_Traffic: M → Traffic
```

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

2.2. Perdurants

Our presentation of example perdurants is not as systematic as that of example endurants. Give the simple basis of endurants covered above there is now a huge variety of perdurants, so we just select one example from each of the three classes of perdurants (as outline in [Bjø16b]): a simple hub insertion *action* (Sect. 2.2.1), a simple link disappearance *event* (Sect. 2.2.2) and a not quite so simple *behaviour*, that of road traffic (Sect. 2.2.3).

2.2.1. Hub Insertion Action

29. Initially inserted hubs, h, are characterised
    a. by their unique identifier which not one of any hub in the net, n, into which the hub is being inserted,
    b. by a mereology, $\{\}$, of zero link identifiers, and
    c. by — whatever — attributes, $attrs$, are needed.
30. The result of such a hub insertion is a net, n',
    a. whose links are those of n, and
    b. whose hubs are those of n augmented with h.

```
value
29.  insert_hub: H → N → N
30.  insert_hub(h)(n) as n′
29.a.   pre: uid_H(h) ∉ xtr_his(n)
29.b.     ∧ obs_mereo_H= {}
29.c.     ∧ ...
30.a.   post: obs_part_Ls(n) = obs_part_Ls(n′)
30.b.       ∧ obs_part_Hs(n) ∪ {h} = obs_part_Hs(n′)
```

2.2.2. Link Disappearance Event

We formalise aspects of the link disappearance event:

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

```
value
31.  link_diss_event: N × N′ × Bool
31.  link_diss_event(n,n′) as tf
32.    pre:  obs_part_Ls(obs_part_LS(n))≠{}
33.    post: ∃ l:L•l ∈ obs_part_Ls(obs_part_LS(n)) ⇒
34.            l ∉ obs_part_Ls(obs_part_LS(n′))
34.            ∧ 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*.

Global Values: There is given some globally observable parts.

35. besides the domain, δ:Δ,
36. a net, n:N,
37. a set of vehicles, vs:V-**set**,
38. a monitor, m:M, and
39. a clock, clock, behaviour.
40. From the net and vehicles we generate an initial distribution of positions of vehicles.

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

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

Channels:

41. We additionally declare a set of vehicle-to-monitor-channels indexed
    a. by the unique identifiers of vehicles
    b. and the (single) monitor identifier.[7]

    and communicating vehicle positions.

```
channel
41.  {v_m_ch[vi,mi]|vi:VI•vi ∈ vis}:VPos
```

Behaviour Signatures:

42. The road traffic system behaviour, rts, takes no arguments (hence the first **Unit**)[8]; and "behaves", that is, continues forever (hence the last **Unit**).
43. The vehicle behaviour
    a. is indexed by the unique identifier, uid_V(v):VI,
    b. the vehicle mereology, in this case the single monitor identifier mi:MI,
    c. the vehicle attributes, obs__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 (on channel vm_ch[vi]); and behaves "forever".
44. The monitor behaviour takes
    a. the monitor identifier,
    b. the monitor mereology,
    c. the monitor attributes,
    d. and — factoring out one of the vehicle attributes — the discrete road traffic, drtf:dRTF, being repeatedly "updated" as the result of **in**put communications from (all) vehicles;
    e. the behaviour otherwise behaves forever.

```
value
42.  rts: Unit → Unit
43.  veh_{vi:VI}: mi:MI → vp:VPos → out vm_ch[vi,mi]  Unit
44.  mon_{mi:MI}: vis:VI-set → RTF → in {v_m_ch[vi,mi]|vi:VI•vi ∈ vis},clk_ch  Unit
```

The Road Traffic System Behaviour:

45. Thus we shall consider our **road traffic system**, rts, as
    a. the concurrent behaviour of a number of vehicles and, to "observe", or, as we shall call it, to monitor their movements,
    b. the monitor behaviour.

```
value
45.  rts() =
45.a.    ‖ {veh_{uid_VI(v)}(mi)(vm(uid_VI(v)))|v:V•v ∈ vs}
45.b.    ‖ 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 (⟨⟩, of so far "no road traffic"!).

In order for the monitor behaviour to assess the vehicle positions these vehicles communicate their positions to the monitor via a vehicle to monitor channel. In order for the monitor to time-stamp these positions it must be able to "read" a clock.

46. We describe here an abstraction of the vehicle behaviour **at** a Hub (hi).
    a. Either the vehicle remains at that hub informing the monitor of its position,
    b. or, internally non-deterministically,
       i moves onto a link, tli, whose "next" hub, identified by thi, is obtained from the mereology of the link identified by tli;

[7] Technically speaking: we could omit the monitor identifier.
[8] The **Unit** designator is an `RSL` technicality.

ii informs the monitor, on channel vm[vi,mi], that it is now at the very beginning (0) of the link identified by tli, whereupon the vehicle resumes the vehicle behaviour positioned at the very beginning of that link,

c. or, again internally non-deterministically, the vehicle "disappears — off the radar" !

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

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

   a. the vehicle remains at that link position informing the monitor of its position,

   b. or, internally non-deterministically, if the vehicle's position on the link has not yet reached the hub,

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

      ii else,

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

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

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

```
47.      veh$_{vi}$(mi)(vp:onL(li,fhi,thi,r)) ≡
47.a.      v_m_ch[vi,mi]!vp ; veh$_{vi}$(mi,va)(vp)
47.b.      ⊓ if r + $\ell_\varepsilon$≤1
47.b.i        then
47.b.i          v_m_ch[vi,mi]!onL(li,fhi,thi,r+$\ell_\varepsilon$) ;
47.b.i          veh$_{vi}$(mi)(onL(li,fhi,thi,r+$\ell_\varepsilon$))
47.b.ii       else
47.b.iiA          let li′:LI•li′ ∈ obs_mereo_H(get_hub(thi)(n)) in
47.b.iiB           v_m_ch[vi,mi]!atH(li,thi,li′);
47.b.iiB           veh$_{vi}$(mi)(atH(li,thi,li′)) end end
47.c.      ⊓ stop
```

The Monitor Behaviour

48. The monitor behaviour evolves around

   a. the monitor identifier,

   b. the monitor mereology,

   c. and the attributes, ma:ATTR

   d. — where we have factored out as a separate arguments — a table of traces of time-stamped vehicle positions,

   e. while accepting messages

      i about time

      ii and about vehicle positions

   f. and otherwise progressing "in[de]finitely".

49. Either the monitor "does own work"

50. or, internally non-deterministically accepts messages from vehicles.

   a. A vehicle position message, vp, may arrive from the vehicle identified by vi.

   b. That message is appended to that vehicle's movement trace – prefixed by time (obtained from the time channel),

   c. whereupon the monitor resumes its behaviour —

   d. where the communicating vehicles range over all identified vehicles.

```
48.   mon$_{mi}$(vis)(trf) ≡
49.        mon$_{mi}$(vis)(trf)
50.     ⊓
50.a.      []{let tvp = (clk_ch?,v_m_ch[vi,mi]?) in
50.b.          let trf′ = trf † [vi ↦ trf(vi)$^\frown$<tvp>] in
50.c.          mon$_{mi}$(vis)(trf′)
50.d.          end end | vi:VI • vi ∈ vis}
```

We are about to complete a long, i.e., a 6.3 page example (!). We can now comment on the full example: The domain, $\delta : \Delta$ is a manifest part. The road net, $n : N$ is also a manifest part. The fleet, $f : F$, of vehicles, $vs : VS$, likewise, is a manifest part. But the monitor, $m : M$, is a concept. One does not have to think of it as a manifest "observer". The vehicles are on — or off — the road (i.e., links and hubs). We know that from a few observations and generalise to all vehicles. They either move or stand still. We also, similarly, know that. Vehicles move. Yes, we know that. Based on all these repeated observations and generalisations we introduce the concept of vehicle traffic. Unless positioned high above a road net — and with good binoculars — a single person cannot really observe the traffic. There are simply too many links, hubs, vehicles, vehicle positions and times. Thus we conclude that, even in a richly manifest domain, we can also "speak of", that is, describe concepts over manifest phenomena, including time !

2.3. Domain Facets

The example of this section, i.e., Sect. 2, focuses on the *domain facet* [Bjø10a, 2008] of (i) *instrinsics*. It does not reflect the other *domain facet*s: (ii) domain support technologies, (iii) domain rules, regulations & scripts, (iv) organisation &

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

3. Requirements

This and the next three sections, Sects. 4.–5., are the main sections of this paper. Section 4. is the most detailed and systematic section. It covers the *domain requirements* operations of *projection*, *instantiation*, *determination*, *extension* and, less detailed, *fitting*. Section 5. surveys the *interface requirements* issues of *shared phenomena*: *shared endurant*s, *shared action*s, *shared event*s and *shared behaviour*, and "completes" the exemplification of the detailed *domain extension* of our requirements into a *road pricing system*. Section 5. also covers the notion of *derived requirements*. This, the initial, section captures main concepts and principles of requirements. Sections 4.–5. covers *initial requirements*. By ***initial requirements*** we shall, "operationally" speaking, understand the requirements that are derived from the general principles outlined in these sections ⊙ In contrast to these are the further requirements that are typically derived either from the *domain facet description*s of *intrinsic*, the *support technology*, the *rules & regulations*, the *organisation & management*, and the *human behaviour facet*s [Bjø10a] — not covered in this paper, (and/)or by more conventional means [DvLF93, Jac01, ZH04, Lau02, JHJ07, OD08, van09].

•••

Definition 1. **Requirements (I):** By a ***requirements*** we understand (cf., [IEE90, IEEE Standard 610.12]): *"A condition or capability needed by a user to solve a problem or achieve an objective"* ⊙

The objective of requirements engineering is to create a *requirements prescription*: A ***requirements prescription*** specifies observable properties of endurants and perdurants of ***the machine*** such as the requirements stake-holders wish them to be ⊙ The ***machine*** is what is required: that is, the *hardware* and *software* that is to be designed and which are to satisfy the requirements ⊙ A requirements prescription thus (putatively) expresses what there should be. A requirements prescription expresses nothing about the design of the possibly desired (required) software. But as the requirements prescription is presented in the form of a model, one can base the design on that model. We shall show how a major part of a requirements prescription can be "derived" from "its" prerequisite domain description.

Rule 1. The "Golden Rule" of Requirements Engineering: Prescribe only those requirements that can be objectively shown to hold for the designed software ⊗[9] "Objectively shown" means that the designed software can either be tested, or be model checked, or be proved (verified), to satisfy the requirements. Caveat: Since we do not illustrate formal tests, model checking nor theorem proving, we shall, alas, not illustrate adherence to this rule.

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 paper. The rule is clear. It is a question for proper management to see that it is adhered to. See the "Caveat" above !

Rule 3. Requirements Adequacy: Make sure that requirements cover what users expect ⊗ That is, do not express a requirement for which you have no users, but make sure that all users' requirements are represented or somehow accommodated. In other words: the requirements gathering process needs to be like an extremely "fine-meshed net": One must make sure that all possible stake-holders have been involved in the requirements acquisition process, and that possible conflicts and other inconsistencies have been obviated.

Rule 4. Requirements Implementability: Make sure that requirements are implementable ⊗ That is, do not express a requirement for which you have no assurance that it can be implemented. In other words, although the requirements phase is not a design phase, one must tacitly assume, perhaps even indicate, somehow, that an implementation is possible. But the requirements in and by themselves, may stay short of expressing such designs. Caveat: The domain and requirements specifications are, in our approach, model-oriented. That helps expressing 'implementability'.

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)

[9] ⊗ marks the end of a rule.

offer of functions and of behaviours while also expressing which events the machine shall "handle" ⊙ By a machine that "maintains" endurants we shall mean: a machine which, "between" users' use of that machine, "keeps" the data that represents these entities. From earlier we repeat:

Definition 3: **Machine:** By *machine* we shall understand a, or the, combination of hardware and software that is the target for, or result of the required computing systems development ⊙ So this, then, is a main objective of requirements development: to start towards the design of the hardware + software for the computing system.

Definition 4: **Requirements (III):** To specify the machine ⊙ When we express requirements and wish to "convert" such requirements to a realisation, i.e., an implementation, then we find that some requirements (parts) imply certain properties to hold of the hardware on which the software to be developed is to "run", and, obviously, that remaining — probably the larger parts of the — requirements imply certain properties to hold of that software.

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

3.1. Some Requirements Aspects

We shall unravel requirements in two stages — (i) the first stage is sketchy (and thus informal) (ii) while the last stage is systematic and both informal and formal. The sketchy stage consists of (i.1) a narrative *problem/objective sketch*, (i.2) a narrative *system requirements sketch*, and (i.3) a narrative *user & external equipment requirements sketch*. (ii) The narrative and formal stage consists of *design assumptions* and *design requirements*. It is systematic, and mandates both strict narrative and formal prescriptions. And it is "derivable" from the domain description. In a sense stage (i) is made superfluous once stage (ii) has been completed. The formal, engineering design work is to based on stage (ii). The purpose of the two stages (i–ii) is twofold: to gently lead the requirements engineer and the reader into the requirements problems while leading the requirements engineer and reader to focus on the very requirements essentials.

3.1.1. Requirements Sketches

Problem, Solution and Objective Sketch

Definition 5. Problem, Solution and Objective Sketch: By a problem, solution and objective sketch we understand a narrative which emphasises what the *problem* to be solved is, outlines a possible *solution* and sketches an *objective* of the solution ⊙

Example 1. The Problem/Objective Requirements: A Sketch: The *problem* is that of traffic congestion. The chosen *solution* is to [build and] operate a toll-road system integrated into a road net and charge toll-road users a usage fee. The *objective* is therefore to create a **road-pricing product.** By a road-pricing product we shall understand an information technology-based system containing computers and communications equipment and software that enables the recording of *vehicle* movements within the *toll-road* and thus enables the *owner* of the road net to charge the *owner* of the vehicles *fees* for the usage of that toll-road □

Systems Requirements

Definition 6. System Requirements: By a ***system requirements narrative*** we understand a narrative which emphasises the overall assumed and/or required hardware and software system equipment ⊙

Example 2. The Road-pricing System Requirements: A Narrative: The requirements are based on the following constellation of system equipment: (i) there is assumed a GNSS: a GLOBAL NAVIGATION SATELLITE SYSTEM; (ii) there are *vehicles* equipped with GNSS receivers; (iii) there is a well-delineated road net called a *toll-road* net with specially equipped *toll-gate*s with *vehicle identification sensor*s, *exit barrier*s which afford (only specially equipped) vehicles to exit[10] from the toll-road net; and (iv) 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: (a) The GNSS is assumed to continuously offer vehicles information about their global position; (b) *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; (c) *toll-gates* shall register the identity of vehicles passing the toll-road and offer this information to the calculator; and (d) the *calculator* shall accept all messages from vehicles and gates and use

[10] We omit consideration of entry barriers.

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** [1] the `GNSS` satellite and telecommunications equipment, [2] the vehicle `GNSS receiver` equipment, [3] the vehicle handling of `GNSS` input and forwarding, to the road pricing system, of its interpretation of `GNSS` input, [4] the toll-gate sensor equipment, [5] the toll-gate barrier equipment, [6] the toll-gate handling of entry, vehicle identification and exit sensors and the forwarding of vehicle identification to the road pricing calculator, and [7] 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" □

User and External Equipment Requirements

Definition 7. User and External Equipment Requirements: By a ***user and external equipment requirements narrative*** we understand a narrative which emphasises assumptions about the human user and external equipment interfaces to the system components ⊙

The user and external equipment requirements detail, and thus make explicit, the assumptions listed in Example 2.

Example 3. The Road-pricing User and External Equipment Requirements: Narrative: The human users of the road-pricing system are: (a) *vehicle drivers,* (b) toll-gate sensor, actuator and barrier *service staff,* and (c) the road-pricing calculator *service staff.* The external equipment are: (1) firstly, the `GNSS` satellites and the telecommunications equipment which enables *communication* between (i) the `GNSS` satellites and vehicles, (ii) vehicles and the road-pricing calculator and (iii) toll-gates and the road-pricing calculator. Moreover, the external *equipment* are (2) the toll-gates with their sensors: entry, vehicle identity, and exit, and the barrier actuator. The external *equipment* are, finally, (3), the vehicles ! □

That is, although we do indeed exemplify domain and requirements aspects of users and external equipment, we do not expect to machine, i.e., to hardware or software design these elements; *they are assumed already implemented !*

3.1.2. *The Narrative and Formal Requirements Stage*

Assumption and Design Requirements

Definition 8. Assumption and Design Requirements: By ***assumption and design requirements*** we understand precise prescriptions of the endurants and perdurants of the (to be designed) system components and the assumptions which that design must rely upon ⊙

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

3.2. The Three Phases of Requirements Engineering

There are, as we see it, three kinds of design assumptions and requirements: (i) *domain requirements*, (ii) *interface requirements* and (iii) *machine requirements*. (i) **Domain requirements** are those requirements which can be expressed sôlely using terms of the domain ⊙ (ii) **Interface requirements** are those requirements which can be expressed only using technical terms of both the domain and the machine ⊙ (iii) **Machine requirements** are those requirements which, in principle, can be expressed sôlely using terms of the machine ⊙

Definition 9. Verification Paradigm: Some preliminary designations: let $\mathscr{D}$ designate the the domain description; let $\mathscr{R}$ designate the requirements prescription, 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 $\mathscr{R}$equirements prescription in the context of the $\mathscr{D}$omain description ⊙

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. We refer to [GGJZ00, Gunter, Jackson and Zave, 2000] for an analysis of a varieties of forms in which $\models$ relate to variants of $\mathscr{D}$, $\mathscr{R}$ and $\mathscr{S}$.

3.3. Order of Presentation of Requirements Prescriptions

The *domain requirements development* stage — as we shall see — can be sub-staged into: *projection*, *instantiation*, *determination*, *extension* and *fitting*. The *interface requirements development* stage — can be sub-staged into *shared: endurant, action, event and behaviour* developments, where "sharedness" pertains to phenomena shared between, i.e., "present" in, both the domain (concretely, manifestly) and the machine (abstractly, conceptually). These development stages need not be pursued in the order of the three stages and their sub-stages. We emphasize that one thing is the stages and steps of development, as for example these: projection, instantiation, determination, extension, fitting, shared endurants, shared actions, shared events, shared behaviours, etcetera, another thing is the requirements prescription that results from these development stages and steps. The further software development, after and on the basis of the requirements prescription starts only when all stages and steps of the requirements prescription have been fully developed. The domain engineer is now free to rearrange the final prescription, irrespective of the order in which the various sections were developed, in such a way as to give a most pleasing, pedagogic and 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 this paper are for the design (ii) of that part of the vehicle software which interfaces the GNSS receiver and the road pricing calculator (cf. Items 129.–132.), (iii) of that part of the toll-gate software which interfaces the toll-gate and the road pricing calculator (cf. Items 137.–139.) and (i) of the road pricing calculator (cf. Items 168.–181.) □

Example 5. **Road Pricing System — Design Assumptions**: The design assumptions for the road pricing calculator include: (i) that *vehicles* behave as prescribed in Items 128.–132., (ii) that the GNSS regularly offers vehicles correct information as to their global position (cf. Item 129.), (iii) that *toll-gates* behave as prescribed in Items 134.–139., and (iv) 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 this paper are for the design of software for the toll-gate and its interfaces to the road pricing system, i.e., Items 133.–134. ⊙

Example 7. **Toll-Gate System — Design Assumptions**: The design assumptions for the toll-gate system include (i) that the vehicles behave as per Items 128.–132., and (ii) that the road pricing calculator behave as per Items 168.–181. ⊙

3.5. Derived Requirements

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

4. Domain Requirements

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

Definition 10. **Domain Requirements Prescription:** A ***domain requirements prescription*** is that subset of the requirements prescription whose technical terms are defined in a domain description ⊙

To determine a relevant subset all we need is collaboration with requirements, cum domain stake-holders. Experimental evidence, in the form of example developments of requirements prescriptions from domain descriptions, appears to show that one can formulate techniques for such developments around a few domain-description-to-requirements-prescription operations. We suggest these: *projection*, *instantiation*, *determination*, *extension* and *fitting*. In Sect. 3.3 we mentioned that the order in which one performs these domain-description-to-domain-requirements-prescription operations is not necessarily the order in which we have listed them here, but, with notable exceptions, one is well-served in starting out requirements development by following this order.

4.1. Domain Projection

Definition 11. Domain Projection: By a ***domain projection*** we mean *a subset of the domain description, one which projects out all those endurants: parts, materials and components, as well as perdurants: actions, events and behaviours that the stake-holders do not wish represented or relied upon by the machine* ⊙

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

4.1.1. Domain Projection — Narrative

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

Example 8. Domain Requirements. Projection: A Narrative Sketch: We require that the road pricing system shall [at most] relate to the following domain entities – and only to these[11]: 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. 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[12].

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. $\Delta_{\mathscr{P}}$
1.a. $\mathrm{N}_{\mathscr{P}}$
1.b. $\mathrm{F}_{\mathscr{P}}$
value
1.a. **obs_part_**$\mathrm{N}_{\mathscr{P}}$: $\Delta_{\mathscr{P}} \to \mathrm{N}_{\mathscr{P}}$
1.b. **obs_part_**$\mathrm{F}_{\mathscr{P}}$: $\Delta_{\mathscr{P}} \to \mathrm{F}_{\mathscr{P}}$

type
2.a. $\mathrm{HA}_{\mathscr{P}}$
2.b. $\mathrm{LA}_{\mathscr{P}}$
value
2.a. **obs_part_**HA: $\mathrm{N}_{\mathscr{P}} \to \mathrm{HA}$
2.b. **obs_part_**LA: $\mathrm{N}_{\mathscr{P}} \to \mathrm{LA}$ □

Concrete Types

type
3. $\mathrm{H}_{\mathscr{P}}, \mathrm{HS}_{\mathscr{P}} = \mathrm{H}_{\mathscr{P}}$**-set**
4. $\mathrm{L}_{\mathscr{P}}, \mathrm{LS}_{\mathscr{P}} = \mathrm{L}_{\mathscr{P}}$**-set**
5. $\mathrm{V}_{\mathscr{P}}, \mathrm{VS}_{\mathscr{P}} = \mathrm{V}_{\mathscr{P}}$**-set**
value

3. **obs_part_**$\mathrm{HS}_{\mathscr{P}}$: $\mathrm{HA}_{\mathscr{P}} \to \mathrm{HS}_{\mathscr{P}}$
4. **obs_part_**$\mathrm{LS}_{\mathscr{P}}$: $\mathrm{LA}_{\mathscr{P}} \to \mathrm{LS}_{\mathscr{P}}$
5. **obs_part_**$\mathrm{VS}_{\mathscr{P}}$: $\mathrm{F}_{\mathscr{P}} \to \mathrm{VS}_{\mathscr{P}}$
6.a. links: $\Delta_{\mathscr{P}} \to$ L**-set**
6.a. links$(\delta_{\mathscr{P}}) \equiv$ **obs_part_**$\mathrm{LS}_{\mathscr{R}}$(**obs_part_**$\mathrm{LA}_{\mathscr{R}}(\delta_{\mathscr{R}})$)

[11] By 'relate to . . . these' we mean that the required system does not rely on domain phenomena that have been "projected away".
[12] Restrictions of the net to the toll road nets, hinted at earlier, will follow in the next domain requirements steps.

6.b. hubs: $\Delta_{\wp} \rightarrow$ H-**set**

6.b. hubs($\delta_{\wp}$) ≡ **obs_part**_HS$_{\wp}$(**obs_part**_HA$_{\wp}$($\delta_{\wp}$)) □

Unique Identifiers

type
7.a. HI, LI, VI, MI
value
7.c. **uid**_HI: H$_{\wp}$ → HI
7.c. **uid**_LI: L$_{\wp}$ → LI

7.c. **uid**_VI: V$_{\wp}$ → VI
7.c. **uid**_MI: M$_{\wp}$ → MI
axiom
7.b. HI∩LI=∅, HI∩VI=∅, HI∩MI=∅,
7.b. LI∩VI=∅, LI∩MI=∅, VI∩MI=∅ □

Mereology

value
12. **obs_mereo**_H$_{\wp}$: H$_{\wp}$ → LI-**set**
13. **obs_mereo**_L$_{\wp}$: L$_{\wp}$ → HI-**set**
13. **axiom** ∀ l:L$_{\wp}$ • **card** **obs_mereo**_L$_{\wp}$(l)=2
14. **obs_mereo**_V$_{\wp}$: V$_{\wp}$ → MI
15. **obs_mereo**_M$_{\wp}$: M$_{\wp}$ → VI-**set**
axiom
16. ∀ $\delta_{\wp}$:$\Delta_{\wp}$, hs:HS•hs=hubs(δ), ls:LS•ls=links($\delta_{\wp}$) ⇒
16. ∀ h:H$_{\wp}$•h ∈ hs ⇒
16. **obs_mereo**_H$_{\wp}$(h)⊆xtr_his($\delta_{\wp}$) ∧

17. ∀ l:L$_{\wp}$•l ∈ ls •
16. **obs_mereo**_L$_{\wp}$(l)⊆xtr_lis($\delta_{\wp}$) ∧
18.a. **let** f:F$_{\wp}$•f=**obs_part**_F$_{\wp}$($\delta_{\wp}$) ⇒
18.a. vs:VS$_{\wp}$•vs=**obs_part**_VS$_{\wp}$(f) **in**
18.a. ∀ v:V$_{\wp}$•v ∈ vs ⇒
18.a. **uid**_V$_{\wp}$(v) ∈ **obs_mereo**_M$_{\wp}$(m) ∧
18.b. **obs_mereo**_M$_{\wp}$(m)
18.b. = {**uid**_V$_{\wp}$(v)|v:V•v ∈ vs}
18.b. **end** □

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

First **hubs:**

type
19.a. GeoH
19.b. H$\Sigma_{\wp}$ = (LI×LI)-**set**t
19.c. H$\Omega_{\wp}$ = H$\Sigma_{\wp}$-**set**
value
19.b. **attr**_H$\Sigma_{\wp}$: H$_{\wp}$ → H$\Sigma_{\wp}$
19.c. **attr**_H$\Omega_{\wp}$: H$_{\wp}$ → H$\Omega_{\wp}$

axiom
20. ∀ $\delta_{\wp}$:$\Delta_{\wp}$,
20. **let** hs = hubs($\delta_{\wp}$) **in**
20. ∀ h:H$_{\wp}$ • h ∈ hs •
20.a. xtr_lis(h)⊆xtr_lis($\delta_{\wp}$)
20.b. ∧ **attr**_$\Sigma_{\wp}$(h) ∈ **attr**_$\Omega_{\wp}$(h)
20. **end** □

Then **links:**

type
23. GeoL
24.a. L$\Sigma_{\wp}$ = (HI×HI)-**set**
24.b. L$\Omega_{\wp}$ = L$\Sigma_{\wp}$-**set**

value
23. **attr**_GeoL: L → GeoL
24.a. **attr**_L$\Sigma_{\wp}$: L$_{\wp}$ → L$\Sigma_{\wp}$
24.b. **attr**_L$\Omega_{\wp}$: L$_{\wp}$ → L$\Omega_{\wp}$
axiom
24.a.– 24.b. on Page 5.

Finally **vehicles:** For 'road pricing' we need vehicle positions. But, for "technical reasons", we must abstain from the detailed description given in Items 25.–25.c.[13] We therefore *simplify* vehicle positions.

51. A simplified vehicle position designates
    a. either a link
    b. or a hub,

type
51. SVPos = SonL | SatH

51.a. SonL :: LI
51.b. SatH :: HI
axiom
25.a.' ∀ n:N, SonL(li):SVPos •
25.a.' ∃ l:L•l ∈**obs_part**_LS(**obs_part**_N(n)) ⇒ li=**uid**_L(l)
25.c.' ∀ n:N, SatH(hi):SVPos •
25.c.' ∃ h:H•h ∈**obs_part**_HS(**obs_part**_N(n)) ⇒ hi=**uid**_H(h)

Global Values

value
35. $\delta_{\wp}$:$\Delta_{\wp}$,
36. n:N$_{\wp}$ = **obs_part**_N$_{\wp}$($\delta_{\wp}$),
36. ls:L$_{\wp}$-**set** = links($\delta_{\wp}$),

36. hs:H$_{\wp}$-**set** = hubs($\delta_{\wp}$),
36. lis:LI-**set** = xtr_lis($\delta_{\wp}$),
36. his:HI-**set** = xtr_his($\delta_{\wp}$)

Behaviour Signatures: We omit the monitor behaviour.

[13] 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*).

52. We leave the vehicle behaviours' attribute argument undefined.

```
type
52.  ATTR
value
42.  trs$_{\mathscr{P}}$: Unit → Unit
43.  veh$_{\mathscr{P}}$: VI×MI×ATTR → ... Unit
```

The System Behaviour: We omit the monitor behaviour.

```
value
45.a.  trs$_{\mathscr{P}}$()=‖{veh$_{\mathscr{P}}$(uid_VI(v),obs_mereo_V(v),_) | v:V$_{\mathscr{P}}$•v ∈ vs}
```

The Vehicle Behaviour: Given the simplification of vehicle positions we *simplify* the vehicle behaviour given in Items 46.–47.

```
46.′     veh$_{vi}$(mi)(vp:SatH(hi)) ≡
46.a.′         v_m_ch[vi,mi]!SatH(hi) ; veh$_{vi}$(mi)(SatH(hi))
46.b.i'     ⌈⌉ let li:LI•li ∈ obs_mereo_H(get_hub(hi)(n)) in
46.b.ii′        v_m_ch[vi,mi]!SonL(li) ; veh$_{vi}$(mi)(SonL(li)) end
46.c.′      ⌈⌉ stop
```

```
47.′     veh$_{vi}$(mi)(vp:SonL(li)) ≡
47.a.′         v_m_ch[vi,mi]!SonL(li) ; veh$_{vi}$(mi)(SonL(li))
47.b.iiA′   ⌈⌉ let hi:HI•hi ∈ obs_mereo_L(get_link(li)(n)) in
47.b.iiB′       v_m_ch[vi,mi]!SatH(hi) ; veh$_{vi}$(mi)(atH(hi)) end
47.c.′      ⌈⌉ stop
```

We can simplify Items 46.′–47.c.′ further.

```
53.  veh$_{vi}$(mi)(vp) ≡
54.      v_m_ch[vi,mi]!vp ; veh$_{vi}$(mi)(vp)
55.    ⌈⌉ case vp of
55.        SatH(hi) →
56.          let li:LI•li ∈ obs_mereo_H(get_hub(hi)(n)) in
57.          v_m_ch[vi,mi]!SonL(li) ; veh$_{vi}$(mi)(SonL(li)) end,
55.        SonL(li) →
58.          let hi:HI•hi ∈ obs_mereo_L(get_link(li)(n)) in
59.          v_m_ch[vi,mi]!SatH(hi) ; veh$_{vi}$(mi)(atH(hi)) end end
60.    ⌈⌉ stop
```

53. This line coalesces Items 46.′ and 47.′.
54. Coalescing Items 46.a.′ and 47.′.
55. Captures the distinct parameters of Items 46.′ and 47.′.
56. Item 46.b.i′.
57. Item 46.b.ii′.
58. Item 47.b.iiA′.
59. Item 47.b.iiB′.
60. Coalescing Items 46.c.′ and 47.c.′.

The above vehicle behaviour definition will be transformed (i.e., further "refined") in Sect. 5.1.3's Example 18; cf. Items 128.– 132. on Page 24 □

4.1.3. *Discussion*

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

4.2. Domain Instantiation

Definition 12. Instantiation: By ***domain instantiation*** we mean a ***refinement*** *of the partial domain requirements prescription (resulting from the projection step) in which the refinements aim at rendering the endurants: parts, materials and components, as well as the perdurants: actions, events and behaviours of the domain requirements prescription more concrete, more specific* ⊙ Instantiations usually render these concepts less general.

Properties that hold of the projected domain shall also hold of the (therefrom) instantiated domain.

Refinement of endurants can be expressed (i) either in the form of concrete types, (ii) or of further "delineating" axioms over sorts, (iii) 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 facing page, 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 next page shown by white filled circles (i.e., hubs). These toll-road plaza hubs are proper hubs of the 'ordinary' road net, n$'_{\mathscr{P}}$.

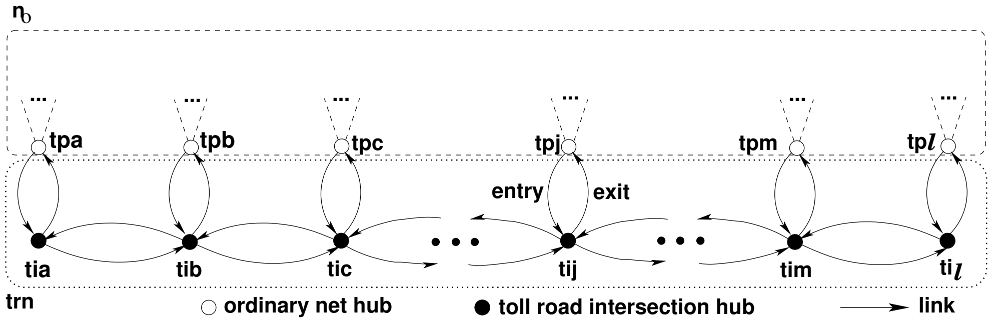

Fig. 1. A simple, linear toll-road net trn. tp_j: *t*oll *p*laza j, ti_j: *t*oll road *i*ntersection j.
Upper dashed sub-figure hint at an ordinary road net n_o.
Lower dotted sub-figure hint at a toll-road net trn.
Dash-dotted (- - -) "V"-images above tp_js hint at links to remaining "parts" of n_o.

61. The instantiated domain, $\delta_{\mathscr{I}}$:$\Delta_{\mathscr{I}}$ has just the net, $\mathsf{n}_{\mathscr{I}}$:$\mathsf{N}_{\mathscr{I}}$ being instantiated.
62. The road net consists of two "sub-nets"
    a. an "ordinary" road net, n_o:$\mathsf{N}_{\mathscr{P}'}$ and
    b. a toll-road net proper, trn:$\mathsf{TRN}_{\mathscr{I}}$ —
    c. "connected" by an interface hil:HIL:
        i That interface consists of a number of toll-road plazas (i.e., hubs), modeled as a list of hub identifiers, hil:HI*.
        ii The toll-road plaza interface to the toll-road net, trn:$\mathsf{TRN}_{\mathscr{I}}$[14], has each plaza, hil[i], connected to a pair of toll-road links: an entry and an exit link: $(l_e\text{:}L, l_x\text{:}L)$.
        iii The toll-road plaza interface to the 'ordinary' net, n_o:$\mathsf{N}_{\mathscr{P}'}$, has each plaza, i.e., the hub designated by the hub identifier hil[i], connected to one or more ordinary net links, $\{l_{i_1}, l_{i_2}, \cdots, l_{i_k}\}$.

62.b. The toll-road net, trn:$\mathsf{TRN}_{\mathscr{I}}$, 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 ("*u*p" 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).

ℓ is the number of toll plazas, hence also the number of toll-road intersection hubs and therefore a number one larger than the number of pairs of main toll-road ("*u*p" and "*d*own") links □

type
61. $\Delta_{\mathscr{I}}$
62. $\mathsf{N}_{\mathscr{I}} = \mathsf{N}_{\mathscr{P}'} \times \mathrm{HIL} \times \mathrm{TRN}$
62.a. $\mathsf{N}_{\mathscr{P}'}$
62.b. $\mathrm{TRN}_{\mathscr{I}} = (\mathrm{L}\times\mathrm{L})^* \times \mathrm{H}^* \times (\mathrm{L}\times\mathrm{L})^*$
62.c. $\mathrm{HIL} = \mathrm{HI}^*$

axiom
62.d. $\forall$ $\mathrm{n}_{\mathscr{I}}$:$\mathrm{N}_{\mathscr{I}}$ •
62.d.   **let** (n_Δ,hil,(exll,hl,lll)) = $\mathrm{n}_{\mathscr{I}}$ **in**
62.d.   **len** hil = **len** exll = **len** hl = **len** lll + 1
62.d.   **end** □

We have named the "ordinary" net sort (primed) $\mathsf{N}_{\mathscr{P}'}$. It is "almost" like (unprimed) $\mathsf{N}_{\mathscr{P}}$ — except that the interface hubs are also connected to the toll-road net entry and exit links.

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

63. The toll-road intersection hubs all[15] have distinct identifiers.

63. wf_dist_toll_road_isect_hub_ids: H^*→**Bool**
63. wf_dist_toll_road_isect_hub_ids(hl) ≡
63.   **len** hl = **card** xtr_his(hl)

64. The toll-road links all have distinct identifiers.

64. wf_dist_toll_road_u_d_link_ids: $(\mathrm{L}\times\mathrm{L})^*$→**Bool**
64. wf_dist_toll_road_u_d_link_ids(lll) ≡
64.   2 × **len** lll = **card** xtr_lis(lll)

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

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

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

```
65. wf_dist_e_x_link_ids: (L×L)*→Bool
```

```
65. wf_dist_e_x_link_ids(exll) ≡
65.    2 × len exll = card xtr_lis(exll)
```

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

```
66. wf_isoltd_toll_road_isect_hubs: HI*×H*→N_𝒥→Bool
```

```
66. wf_isoltd_toll_road_isect_hubs(hil,hl)(n_𝒥) ≡
66.    let ls=xtr_links(n_𝒥) in
66.    let his = ∪ {obs_mereo_L(l)|l:L•l ∈ ls} in
66.    his ∩ xtr_his(hl) = {} end end
```

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

```
67. wf_p_hubs_pt_of_ord_net: HI*→N′_Δ→Bool
67. wf_p_hubs_pt_of_ord_net(hil)(n'_Δ) ≡
67.    elems hil ⊆ xtr_his(n′_Δ)
```

68. The plaza hub mereologies must each,
   a. besides identifying at least one hub of the ordinary net,
   b. also identify the two entry/exit links with which they are supposed to be connected.

```
68. wf_p_hub_interf: N′_Δ→Bool
```

```
68. wf_p_hub_interf(n_o,hil,(exll,_,_)) ≡
68.    ∀ i:Nat • i ∈ inds exll ⇒
68.       let h = get_H(hil(i))(n′_Δ) in
68.       let lis = obs_mereo_H(h) in
68.       let lis′ = lis \ xtr_lis(n′) in
68.       lis′ = xtr_lis(exll(i)) end end end
```

69. The mereology of each toll-road intersection hub must identify
   a. the entry/exit links
   b. and exactly the toll-road 'up' and 'down' links
   c. with which they are supposed to be connected.

```
69. wf_toll_road_isect_hub_iface: N_𝒥→Bool
69. wf_toll_road_isect_hub_iface(_,_,(exll,hl,lll)) ≡
```

```
69.     ∀ i:Nat • i ∈ inds hl ⇒
69.        obs_mereo_H(hl(i)) =
69.a.        xtr_lis(exll(i)) ∪
69.        case i of
69.b.           1 → xtr_lis(lll(1)),
69.b.           len hl → xtr_lis(lll(len hl−1))
69.b.           _ → xtr_lis(lll(i)) ∪ xtr_lis(lll(i−1))
69.        end
```

70. The mereology of the entry/exit links must identify exactly the
   a. interface hubs and the
   b. toll-road intersection hubs
   c. with which they are supposed to be connected.

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

71. The mereology of the toll-road 'up' and 'down' links must
   a. identify exactly the toll-road intersection hubs
   b. with which they are supposed to be connected.

```
71. wf_u_d_links: (L×L)*×H*→Bool
```

```
71. wf_u_d_links(lll,hl) ≡
71.    ∀ i:Nat • i ∈ inds lll ⇒
71.       let (ul,dl)  = lll(i) in
71.       obs_mereo_L(ul) = obs_mereo_L(dl) =
71.a.      uid_H(hl(i)) ∪ uid_H(hl(i+1)) end
71.    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}
```

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

```
72. wf_instantiated_net: N_𝒥 → Bool
72. wf_instantiated_net(n′_Δ,hil,(exll,hl,lll))
63.    wf_dist_toll_road_isect_hub_ids(hl)
64.   ∧ wf_dist_toll_road_u_d_link_ids(lll)
65.   ∧ wf_dist_e_e_link_ids(exll)
66.   ∧ wf_isolated_toll_road_isect_hubs(hil,hl)(n′)
67.   ∧ wf_p_hubs_pt_of_ord_net(hil)(n′)
68.   ∧ wf_p_hub_interf(n′_Δ,hil,(exll,_,_))
69.   ∧ wf_toll_road_isect_hub_iface(_,_,(exll,hl,lll))
70.   ∧ wf_exll(exll,hil,hl)
71.   ∧ 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, $n_{\mathscr{J}}$:$N_{\mathscr{J}}$. We need to show the refinement relation:

- abstraction($n_{\mathscr{J}}$) = $n_{\mathscr{P}}$.

```
value
73.  abstraction: $N_{\mathscr{I}} \to N_{\mathscr{P}}$
74.  abstraction($n'_\Delta$,hil,(exll,hl,lll)) ≡
75.    let $n_{\mathscr{P}}:N_{\mathscr{P}}$ •
75.      let hs = obs_part_HS$_{\mathscr{P}}$(obs_part_HA$_{\mathscr{P}}$($n'_{\mathscr{P}}$)),
75.          ls = obs_part_LS$_{\mathscr{P}}$(obs_part_LA$_{\mathscr{P}}$($n'_{\mathscr{P}}$)),
75.          ths = elems hl,
75.          eells = xtr_links(eell), llls = xtr_links(lll) in
76.          hs∪ths=obs_part_HS$_{\mathscr{P}}$(obs_part_HA$_{\mathscr{P}}$($n_{\mathscr{P}}$))
77.        ∧ ls∪eells∪llls=obs_part_LS$_{\mathscr{P}}$(obs_part_LA$_{\mathscr{P}}$($n_{\mathscr{P}}$))
78.    $n_{\mathscr{P}}$ end end
```

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

4.2.3. ***Discussion***

Domain descriptions, such as illustrated in [Bjø16b, *Manifest Domains: Analysis & Description*] and in this paper, model families of concrete, i.e., specifically occurring domains. Domain instantiation, as exemplified in this section (i.e., Sect. 4.2), "narrow down" these families. Domain instantiation, such as it is defined, cf. Definition 12 on Page 16, allows the requirements engineer to instantiate to a concrete instance of a very specific domain, that, for example, of the toll-road between *Bolzano Nord* and *Trento Sud* in Italy (i.e., *n*=7)[16].

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

79. *The entry/exit and toll-road links*
    a. are always all one way links,
    b. as indicated by the arrows of Fig. 1 on Page 17,
    c. such that each pair allows traffic in opposite directions.

```
79.     opposite_traffics: (L×L)* × (L×L)* → Bool
79.     opposite_traffics(exll,lll) ≡
79.       ∀ (lt,lf):(L×L) • (lt,lf) ∈ elems exll⌢lll ⇒
79.a.       let (ltσ,lfσ) = (attr_LΣ(lt),attr_LΣ(lf)) in
79.a.′.      attr_LΩ(lt)={ltσ}∧attr_LΩ(ft)={ftσ}
79.a.″.    ∧ card ltσ = 1 = card lfσ
79.       ∧ let ({(hi,hi′)},{(hi″,hi‴)}) = (ltσ,lfσ) in
79.c.       hi=hi‴ ∧ hi′=hi″
79.       end end
```

Predicates 79.a.′. and 79.a.″. express the same property.

All Toll-road Hubs are Free-flow

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

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

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

```
80.   free_flow_toll_road_hubs(exl,ll) ≡
80.      ∀ i:Nat•i ∈ inds hl ⇒
80.          attr_HΣ(hl(i)) =
80.a.              hσ_ex_ls(exl(i))
80.b.              ∪ hσ_et_ls(exl(i),(i,ll))
80.c.              ∪ hσ_tx_ls(exl(i),(i,ll))
80.d.              ∪ hσ_tt_ls(i,ll)
```

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

```
80.a.  hσ_ex_ls: (L×L)→LΣ
80.a.  hσ_ex_ls(e,x) ≡ {(uid_LI(e),uid_LI(x))}
```

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

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

The *em* and *in* in the toll-road link list (em:L×in:L)* designate selectors for *em*anating, respectively *in*cident links. 80.c.: from incident *t*oll-road links to e*x*it *l*ink*s*:

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

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

```
80.d.  hσ_tt_ls: Nat×(em:L×in:L)*→LΣ
80.d.  hσ_tt_ls(i,ll) ≡
80.d.     case i of
80.d.        2       → {(uid_LI(in(ll(1))),uid_LI(em(ll(1))))},
80.d.        len ll+1 → {(uid_LI(in(ll(len ll))),uid_LI(em(ll(len ll))))},
80.d.        _        → {(uid_LI(in(ll(i−1))),uid_LI(em(ll(i−1)))),
80.d.                    (uid_LI(in(ll(i))),uid_LI(em(ll(i))))}
80.d.     end  □
```

The example above illustrated ‘domain determination’ with respect to endurants. Typically “endurant determination” is expressed in terms of axioms that limit state spaces — where “endurant instantiation” typically “limited” the mereology of endurants: how parts are related to one another. We shall not exemplify domain determination with respect to perdurants.

4.3.2. ***Discussion***

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

4.4. Domain Extension

Definition 14. Extension: By ***domain extension*** we understand *the introduction of endurants (see Sect. 4.4.1) and perdurants (see Sect. 5.2) that were not feasible in the original domain, but for which, with computing and communication, and with new, emerging technologies, for example, sensors, actuators and satellites, there is the possibility of feasible implementations, hence the requirements, that what is introduced becomes part of the unfolding requirements prescription* ⊙

4.4.1. ***Endurant Extensions***

Definition 15. Endurant Extension: By an ***endurant extension*** we understand the introduction of one or more endurants into the projected, instantiated and determined domain $\mathscr{D}_{\mathscr{R}}$ resulting in domain $\mathscr{D}_{\mathscr{R}}'$, such that these form a *conservative extension* of the theory, $\mathscr{T}_{\mathscr{D}_{\mathscr{R}}}$ denoted by the domain requirements $\mathscr{D}_{\mathscr{R}}$ (i.e., “before” the extension), that is: every theorem of $\mathscr{T}_{\mathscr{D}_{\mathscr{R}}}$ is still a theorem of $\mathscr{T}_{\mathscr{D}_{\mathscr{R}}'}$.

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

Example 13. Domain Requirements — Endurant Extension: We present the extensions in several steps. Some of them will be developed in this section. Development of the remaining will be deferred to Sect. 5.1.3. 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”.

[a] Vehicle Extension:

81. There is a domain, $\delta_{\mathscr{E}}$:$\Delta_{\mathscr{E}}$, which contains
82. a fleet, f$_{\mathscr{E}}$:F$_{\mathscr{E}}$, that is,
83. a set, vs$_{\mathscr{E}}$:VS$_{\mathscr{E}}$, of
84. extended vehicles, v$_{\mathscr{E}}$:V$_{\mathscr{E}}$ — their extension amounting to
85. a dynamic reactive attribute, whose value, ti-gpos:TiGpos, at any time, reflects that vehicle's *time-stamped global position.*[17]
86. The vehicle's GNSS receiver calculates, *loc_pos*, its local position, lpos:LPos, based on these signals.
87. Vehicles access these *external attribute*s via the *external attribute* channel, attr_TiGPos_ch.

type
81. $\Delta_{\mathscr{E}}$
82. $\mathrm{F}_{\mathscr{E}}$
83. $\mathrm{VS}_{\mathscr{E}} = \mathrm{V}_{\mathscr{E}}$**-set**
84. $\mathrm{V}_{\mathscr{E}}$
85. $\mathrm{TiGPos} = \mathbb{T} \times \mathrm{GPos}$
86. GPos, LPos

value
81. $\delta_{\mathscr{E}}$:$\Delta_{\mathscr{E}}$
82. **obs_part_**$\mathrm{F}_{\mathscr{E}}$: $\Delta_{\mathscr{E}} \rightarrow \mathrm{F}_{\mathscr{E}}$
82. f = **obs_part_**$\mathrm{F}_{\mathscr{E}}(\delta_{\mathscr{E}})$
83. **obs_part_**$\mathrm{VS}_{\mathscr{E}}$: $\mathrm{F}_{\mathscr{E}} \rightarrow \mathrm{VS}_{\mathscr{E}}$
83. vs = **obs_part_**$\mathrm{VS}_{\mathscr{E}}$(f)
83. vis = xtr_vis(vs)
85. attr_TiGPos_ch[vi]?
86. loc_pos: GPos → LPos

channel
86. {attr_TiGPos_ch[vi]|vi:VI•vi ∈ vis}:TiGPos

We define two auxiliary functions,

88. xtr_vs, which given a domain, or a fleet, extracts its set of vehicles, and
89. xtr_vis which given a set of vehicles generates their unique identifiers.

value
88. xtr_vs: $(\Delta_{\mathscr{E}}|\mathrm{F}_{\mathscr{E}}|\mathrm{VS}_{\mathscr{E}}) \rightarrow \mathrm{V}_{\mathscr{E}}$**-set**
88. xtr_vs(arg) ≡
88. **is_**$\Delta_{\mathscr{E}}$(arg) → **obs_part_**$\mathrm{VS}_{\mathscr{E}}$(**obs_part_**$\mathrm{F}_{\mathscr{E}}$(arg)),
88. **is_**$\mathrm{F}_{\mathscr{E}}$(arg) → **obs_part_**$\mathrm{VS}_{\mathscr{E}}$(arg),
88. **is_**$\mathrm{VS}_{\mathscr{E}}$(arg) → arg
89. xtr_vis: $(\Delta_{\mathscr{E}}|\mathrm{F}_{\mathscr{E}}|\mathrm{VS}_{\mathscr{E}}) \rightarrow$ VI**-set**
89. xtr_vis(arg) ≡ {**uid_**VI(v)|v ∈ xtr_vs(arg)}

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

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

type
90. C, CI

value
90. **obs_part_**C: $\Delta_{\mathscr{E}} \rightarrow$ C
90. **uid_**CI: C → CI
90. c = **obs_part_**C($\delta_{\mathscr{E}}$)
90. ci = **uid_**CI(c)

[c] Vehicle to Road Pricing Calculator Channel:

91. Vehicles can, on their own volition, offer the timed local position, viti-lpos:VITiLPos
92. to the pricing calculator, c:C$_{\mathscr{E}}$ along a vehicles-to-calculator channel, v_c_ch.

type
91. VITiLPos = VI × ($\mathbb{T}$ × LPos)

channel
92. {v_c_ch[vi,ci]|vi:VI,ci:CI•vi∈vis∧ci=**uid_**C(c)}:VITiLPos

[d] Toll-gate Sorts and Dynamic Types: We extend the domain with toll-gates for vehicles entering and exiting the toll-road entry and exit links. Figure 2 illustrates the idea of gates.

[17] We refer to literature on GNSS, *global navigation satellite systems*. The simple vehicle position, vp:SVPos, is determined from three to four time-stamped signals received from a like number of GNSS satellites [ESA].

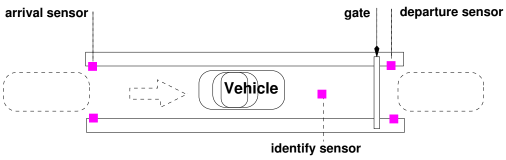

Fig. 2. A toll plaza gate

Figure 2 is intended to illustrate a vehicle entering (or exiting) a toll-road arrival link. The toll-gate is equipped with three sensors: an arrival sensor, a vehicle identification sensor and an departure sensor. The arrival sensor serves to prepare the vehicle identification sensor. The departure sensor serves to prepare the gate for closing when a vehicle has passed. The vehicle identify sensor identifies the vehicle and "delivers" a pair: the current time and the vehicle identifier. Once the vehicle identification sensor has identified a vehicle the gate opens and a message is sent to the road pricing calculator as to the passing vehicle's identity and the identity of the link associated with the toll-gate (see Items 109.- 110. on the next page).

93. The domain contains the extended net, n:$N_{\mathscr{E}}$,
94. with the net extension amounting to the toll-road net, TRN$_{\mathscr{E}}$, that is, the instantiated toll-road net, trn:TRN$_{\mathscr{I}}$, 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

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

```
type
93.     N_ℰ
94.     TRN_ℰ = (EG×XG)* × TRN_ℐ
95.     GI
value
93.     obs_part_N_ℰ: Δ_ℰ → N_ℰ
94.     obs_part_TRN_ℰ: N_ℰ → TRN_ℰ
95.     uid_G: (EG|XG) → GI
```

```
96.     obs_mereo_G: (EG|XG) → (LI×CI)
94.     trn:TRN_ℰ = obs_part_TRN_ℰ(δ_ℰ)
channel
97.     {attr_entry_ch[gi]|gi:GI•xtr_eGIds(trn)} "enter"
97.     {attr_exit_ch[gi]|gi:GI•xtr_xGIds(trn)} "exit"
98.     {attr_identity_ch[gi]|gi:GI•xtr_GIds(trn)} TIVI
type
98.     TIVI = 𝕋 × VI
```

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

99. xtr_eGℓ extracts the ℓist of entry gates,
100. xtr_xGℓ extracts the ℓist of exit gates,
101. xtr_eGIds extracts the $s$et of entry gate identifiers,
102. xtr_xGIds extracts the $s$et of exit gate identifiers,
103. xtr_Gs extracts the $s$et of all gates, and
104. xtr_GIds extracts the $s$et of all gate identifiers.

```
value
99.   xtr_eGℓ: TRN_ℰ → EG*
99.   xtr_eGℓ(pgl,_) ≡ {eg|(eg,xg):(EG,XG)•(eg,xg)∈ elems pgl}
100.  xtr_xGℓ: TRN_ℰ → XG*
100.  xtr_xGℓ(pgl,_) ≡ {xg|(eg,xg):(EG,XG)•(eg,xg)∈ elems pgl}
101.  xtr_eGIds: TRN_ℰ → GI-set
101.  xtr_eGIds(pgl,_) ≡ {uid_GI(g)|g:EG•g ∈ xtr_eGs(pgl,_)}
```

```
102.  xtr_xGIds: TRN_ℰ → GI-set
102.  xtr_xGIds(pgl,_) ≡ {uid_GI(g)|g:EG•g ∈ xtr_xGs(pgl,_)}
103.  xtr_Gs: TRN_ℰ → G-set
103.  xtr_Gs(pgl,_) ≡ xtr_eGs(pgl,_) ∪ xtr_xGs(pgl,_)
104.  xtr_GIds: TRN_ℰ → GI-set
104.  xtr_GIds(pgl,_) ≡ xtr_eGIds(pgl,_) ∪ xtr_xGIds(pgl,_)
```

105. A **well-formedness condition** expresses
    a. that there are as many entry end exit gate pairs as there are toll-plazas,
    b. that all gates are uniquely identified, and
    c. that each entry [exit] gate is paired with an entry [exit] link and has that link's unique identifier as one element of its mereology, the other elements being the calculator identifier and the vehicle identifiers.

The well-formedness relies on awareness of

106. the unique identifier, ci:CI, of the road pricing calculator, c:C, and
107. the unique identifiers, vis:VI-**set**, of the fleet vehicles.

```
axiom
105.    ∀ n:N_ℛ3, trn:TRN_ℛ3 •
105.       let (exgl,(exl,hl,lll)) = obs_part_TRN_ℛ3(n) in
105.a.       len exgl = len exl = len hl = len lll + 1
105.b.     ∧ card xtr_GIds(exgl) = 2 ∗ len exgl
```

```
105.c.    ∧ ∀ i:Nat•i ∈ inds exgl•
105.c.         let ((eg,xg),(el,xl)) = (exgl(i),exl(i)) in
105.c.         obs_mereo_G(eg) = (uid_U(el),ci,vis)
105.c.       ∧ obs_mereo_G(xg) = (uid_U(xl),ci,vis)
105.      end end
```

[e] Toll-gate to Calculator Channels:

108. We distinguish between *entry* and *exit* gates.
109. Toll road entry and exit gates offers the road pricing calculator a pair: whether it is an entry or an exit gates, and pair of the passing vehicle's identity and the time-stamped identity of the link associated with the toll-gate
110. to the road pricing calculator via a (gate to calculator) channel.

type
108. $\mathrm{EE} = ''\mathtt{entry}''|''\mathtt{exit}''$
109. $\mathrm{EEVITiLI} = \mathrm{EE}\times(\mathrm{VI}\times(\mathbb{T}\times\mathrm{SonL}))$
channel
110. $\{\mathrm{g\_c\_ch}[\mathrm{gi,ci}]|\mathrm{gi{:}GI}\bullet\mathrm{gi} \in \mathrm{gis}\}\mathrm{:EETiVILI}$

[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, $\mathrm{n}_\mathscr{E}$:$\mathrm{N}_\mathscr{E}$, including the mereology and the geodetic attributes of links and hubs.

type
111. $\mathrm{TRM} = \mathrm{VI} \underset{m}{\rightarrow} (\mathbb{T} \times \mathrm{SVPos})^*$
112. $\mathrm{VPLF} = \mathrm{GRM} \rightarrow \mathrm{LPos} \rightarrow (\mathrm{SVPos} \mid ''\mathtt{off\_N}'')$
113. GRM

value
111. $\textbf{attr}\_\mathrm{TRM}\colon \mathrm{C}_\mathscr{E} \rightarrow \mathrm{TRM}$
112. $\textbf{attr}\_\mathrm{VPLF}\colon \mathrm{C}_\mathscr{E} \rightarrow \mathrm{VPLF}$

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

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

19.a. 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. $\mathrm{GRM} = (\mathrm{GeoL} \underset{m}{\rightarrow} \mathrm{LI}) \bigcup (\mathrm{GeoH} \underset{m}{\rightarrow} \mathrm{HI})$
value
115. $\mathrm{geo\_GRM}\colon \mathrm{N} \rightarrow \mathrm{GRM}$
115. $\mathrm{geo\_GRM(n)} \equiv$

115. $\textbf{let}\ \mathrm{ls} = \mathrm{xtr\_links(n)},\ \mathrm{hs} = \mathrm{xtr\_hubs(n)}\ \textbf{in}$
115. $[\textbf{attr}\_\mathrm{GeoL(l)}\mapsto\textbf{uid}\_\mathrm{LI(l)}|\mathrm{l{:}L}\bullet\mathrm{l} \in \mathrm{ls}]$
115. $\cup$
115. $[\textbf{attr}\_\mathrm{GeoH(h)}\mapsto\textbf{uid}\_\mathrm{HI(h)}|\mathrm{h{:}H}\bullet\mathrm{h} \in \mathrm{hs}]\ \textbf{end}$

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. $\mathrm{obtain\_SVPos}\colon \mathrm{GRM} \rightarrow \mathrm{LPos} \rightarrow \mathrm{SVPos}$

116. $\mathrm{obtain\_SVPos(grm)(lpos)}\ \textbf{as}\ \mathrm{svpos}$
116. $\textbf{post}\colon \textbf{case}\ \mathrm{svpos}\ \textbf{of}$
116. $\quad\quad \mathrm{SatH(hi)} \rightarrow \mathrm{within(lpos,grm(hi))},$
116. $\quad\quad \mathrm{SonL(li)} \rightarrow \mathrm{within(lpos,grm(li))},$
116. $\quad\quad ''\mathtt{off\_N}'' \rightarrow \textbf{true end}$

where *within* is a predicate which holds if its first argument, a local position calculated from a GNSS-generated global position, falls within the point set representation of the geodetic locations of a link or a hub. The design of the *obtain_SVPos* represents an interesting challenge.

[g] "Total" System State: Global values:

117. There is a given domain, $\delta_\mathscr{E}$:$\Delta_\mathscr{E}$;
118. there is the net, $\mathrm{n}_\mathscr{E}$:$\mathrm{N}_\mathscr{E}$, of that domain;
119. there is toll-road net, $\mathrm{trn}_\mathscr{E}$:$\mathrm{TRN}_\mathscr{E}$, of that net;
120. there is a set, $\mathrm{egs}_\mathscr{E}$:$\mathrm{EG}_\mathscr{E}$**-set**, of entry gates;
121. there is a set, $\mathrm{xgs}_\mathscr{E}$:$\mathrm{XG}_\mathscr{E}$**-set**, of exit gates;
122. there is a set, $\mathrm{gis}_\mathscr{E}$:$\mathrm{GI}_\mathscr{E}$**-set**, ofgate identifiers;
123. there is a set, $\mathrm{vs}_\mathscr{E}$:$\mathrm{V}_\mathscr{E}$**-set**, of vehicles;
124. there is a set, $\mathrm{vis}_\mathscr{E}$:$\mathrm{VI}_\mathscr{E}$**-set**, of vehicle identifiers;
125. there is the road-pricing calculator, $\mathrm{c}_\mathscr{E}$:$\mathrm{C}_\mathscr{E}$ and
126. there is its unique identifier, $\mathrm{ci}_\mathscr{E}$:CI.

value
117. $\delta_\mathscr{E}{:}\Delta_\mathscr{E}$
118. $\mathrm{n}_\mathscr{E}{:}\mathrm{N}_\mathscr{E} = \textbf{obs\_part}\_\mathrm{N}_\mathscr{E}(\delta_\mathscr{E})$
119. $\mathrm{trn}_\mathscr{E}{:}\mathrm{TRN}_\mathscr{E} = \textbf{obs\_part}\_\mathrm{TRN}_\mathscr{E}(\mathrm{n}_\mathscr{E})$
120. $\mathrm{egs}_\mathscr{E}{:}\mathrm{EG}\textbf{-set} = \mathrm{xtr\_egs}(\mathrm{trn}_\mathscr{E})$
121. $\mathrm{xgs}_\mathscr{E}{:}\mathrm{XG}\textbf{-set} = \mathrm{xtr\_xgs}(\mathrm{trn}_\mathscr{E})$

122. $\mathrm{gis}_\mathscr{E}{:}\mathrm{XG}\textbf{-set} = \mathrm{xtr\_gis}(\mathrm{trn}_\mathscr{E})$
123. $\mathrm{vs}_\mathscr{E}{:}\mathrm{V}_\mathscr{E}\textbf{-set} = \textbf{obs\_part}\_\mathrm{VS}(\textbf{obs\_part}\_\mathrm{F}_\mathscr{E}(\delta_\mathscr{E}))$
124. $\mathrm{vis}_\mathscr{E}{:}\mathrm{VI}\textbf{-set} = \{\textbf{uid}\_\mathrm{VI}(\mathrm{v}_\mathscr{E})|\mathrm{v}_\mathscr{E}{:}\mathrm{V}_\mathscr{E}\bullet\mathrm{v}_\mathscr{E} \in \mathrm{vs}_\mathscr{E}\}$
125. $\mathrm{c}_\mathscr{E}{:}\mathrm{C}_\mathscr{E} = \textbf{obs\_part}\_\mathrm{C}_\mathscr{E}(\delta_\mathscr{E})$
126. $\mathrm{ci}_\mathscr{E}{:}\mathrm{CI}_\mathscr{E} = \textbf{uid}\_\mathrm{CI}(\mathrm{c}_\mathscr{E})$

In the following we shall omit the cumbersome $_\mathscr{E}$ subscripts.

[h] “Total” System Behaviour: The signature and definition of the system behaviour is sketched as are the signatures of the vehicle, toll-gate and road pricing calculator. We shall model the behaviour of the road pricing system as follows: we shall not model behaviours nets, hubs and links; thus we shall model only the behaviour of vehicles, veh, the behaviour of toll-gates, gate, and the behaviour of the road-pricing calculator, calc, The behaviours of vehicles and toll-gates are presented here. But the behaviour of the road-pricing calculator is “deferred” till Sect. 5.1.3 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,
    b. with the parallel composition of the parallel (likewise distributed) composition of the behaviours of all entry gates,
    c. with the parallel composition of the parallel (likewise distributed) composition of the behaviours of all exit gates,
    d. with the parallel composition of the behaviour of the road-pricing calculator,

```
value
127.   sys: Unit → Unit
127.   sys() ≡
127.a.       ‖ {veh_uid_V(v)(obs_mereo_V(v))|v:V•v ∈ vs}
127.b.     ‖ ‖ {gate_uid_EG(eg)(obs_mereo_G(eg),”entry”)|eg:EG•eg ∈ egs}
127.c.     ‖ ‖ {gate_uid_XG(xg)(obs_mereo_G(xg),”exit”)|xg:XG•xg ∈ xgs}
127.d.       ‖  calc_uid_C(c)(vis,gis)(rlf)(trm)

128.   veh_vi: (ci:CI×gis:GI-set) → in attr_TiGPos[vi] out v_c_ch[vi,ci] Unit
134.   gate_gi: (ci:CI×VI-set×LI)×ee:EE → in attr_entry_ch[gi,ci],attr_id_ch[gi,ci],attr_exit_ch[gi,ci] out attr_barrier_ch[gi],g_c_ch[gi,ci]  Unit
168.   calc_ci: (vis:VI-set×gis:GI-set)×VPLF→TRM→ in {v_c_ch[vi,ci]|vi:VI•vi ∈ vis},{g_c_ch[gi,ci]|gi:GI•gi ∈ gis}  Unit
```

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

Vehicle Behaviour: We refer to the vehicle behaviour, in the domain, described in Sect. 2’s The Road Traffic System Behaviour Items 46. and Items 47., Page 9 and, projected, Page 16.

128. Instead of moving around by explicitly expressed internal non-determinism[18] 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, $(\tau$,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.

```
value
128.   veh_vi: (ci:CI×gis:GI-set) →
128.        in attr_TiGPos_ch[vi] out v_c_ch[vi,ci] Unit
128.   veh_vi(ci,gis) ≡
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) end end
128.       pre vi ∈ vis
```

The *vehicle* signature has *attr_TiGPos_ch*[*vi*] model an external vehicle attribute and *v_c_ch*[*vi,ci*] the *embedded attribute sharing* [Bjø16b, Sect. 4.1.1 and 4.5.2] between vehicles (their position) and the price calculator’s road map. The above behaviour represents an assumption about the behaviour of vehicles. If we were to design software for the monitoring and control of vehicles then the above vehicle behaviour would have to be refined in order to serve as a proper interface requirements. The refinement would include handling concerns about the drivers’ behaviour when entering, passing and exiting toll-gates, about the proper function of the GNSS equipment, and about the safe communication with the road price calculator. The above concerns would already have been addressed in a model of *domain facets* such as *human behaviour*, *technology support*, proper tele-communications *scripts*, etcetera. We refer to [Bjø10a].

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

133. We parameterise the toll-gate behaviour as either an entry or an exit gate.
134. Toll-gates operate autonomously and cyclically.
135. The attr_enter_ch event “triggers” the behaviour specified in formula line Item 136.–138. starting with a ”Raise” barrier action.
136. The time-of-passing and the identity of the passing vehicle is sensed by attr_passing_ch channel events.
137. Then the road pricing calculator is informed of time-of-passing and of the vehicle identity vi and the link li associated with the gate – and with a ”Lower” barrier action.
138. And finally, after that vehicle has left the entry or exit gate the barrier is again ”Lower”ered and
139. that toll-gate’s behaviour is resumed.

type

[18] We refer to Items 46.b., 46.c. on Page 9 and 47.b., 47.b.ii, 47.c. on Page 9

```
133.  EE = ”enter” | ”exit”
value
134.  gate_gi: (ci:CI×VI-set×LI)×ee:EE →
134.      in attr_enter_ch[gi],attr_passing_ch[gi],attr_leave_ch[gi]
134.      out attr_barrier_ch[gi],g_c_ch[gi,ci] Unit
134.  gate_gi((ci,vis,li),ee) ≡
135.      attr_enter_ch[gi] ? ; attr_barrier_ch[gi] ! ”Lower”
136.      let (τ,vi) = attr_passing_ch[gi] ? in assert vi ∈ vis
137.      (attr_barrier_ch[gi] ! ”Raise”
137.       ‖ g_c_ch[gi,ci] ! (ee,(vi,(τ,SonL(li))))) ;
138.      attr_leave_ch[gi] ? ; attr_barrier_ch[gi] ! ”Lower”
139.      gate_gi((ci,vis,li),ee)
134.      end
134.      pre li ∈ lis
```

The *gate* signature's *attr_enter_ch*[*gi*], *attr_passing_ch*[*gi*], *attr_barrier_ch*[*gi*] and *attr_leave_ch*[*gi*] model respective *external attributes* [Bjø16b, Sect. 4.1.1 and 4.5.2] (the *attr_barrier_ch*[*gi*] models reactive (i.e., output) attribute), while *g_c_ch*[*gi,ci*] models the *embedded attribute sharing* between gates (their identification of vehicle positions) and the calculator road map. The above behaviour represents an assumption about the behaviour of toll-gates. If we were to design software for the monitoring and control of toll-gates then the above gate behaviour would have to be refined in order to serve as a proper interface requirements. The refinement would include handling concerns about the drivers' behaviour when entering, passing and exiting toll-gates, about the proper function of the entry, passing and exit sensors, about the proper function of the gate barrier (opening and closing), and about the safe communication with the road price calculator. The above concerns would already have been addressed in a model of *domain facets* such as *human behaviour*, *technology support*, proper tele-communications *scripts*, etcetera. We refer to [Bjø10a] □

We shall define the *cal*culator behaviour in Sect. 5.1.3 on Page 29. The reason for this deferral is that it exemplifies *interface requirements*.

4.4.2. Discussion

The requirements assumptions expressed in the specifications of the vehicle and gate behaviours assume that these behave in an orderly fashion. But they seldom do ! The attr_TiGPos_ch sensor may fail. And so may the attr_enter_ch, attr_passing_ch, and attr_leave_ch sensors and the attr_barrier_ch actuator. These attributes represent *support technology* facets. They can fail. To secure fault tolerance one must prescribe very carefully what counter-measures are to be taken and/or the safety assumptions. We refer to [ZH04, JHJ07, OD08]. They cover three alternative approaches to the handling of fault tolerance. Either of the approaches can be made to fit with our approach. First one can pursue our approach to where we stand now. Then we join the approaches of either of [ZH04, JHJ07, OD08]. [JHJ07] likewise decompose the requirements prescription as is suggested here.

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, $d_{r_1}, d_{r_2}, \ldots, d_{r_n}$, being considered, and that these pertain to the same domain — and can hence be assumed covered by a same domain description.

Definition 16. **Requirements Fitting:** By ***requirements fitting*** we mean a *harmonisation* of $n > 1$ domain requirements that have overlapping (shared) not always consistent parts and which results in n *partial domain requirements*', $p_{d_{r_1}}, p_{d_{r_2}}, \ldots, p_{d_{r_n}}$, 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 17. **Requirements Harmonisation:** By ***requirements harmonisation*** we mean a number of alternative and/or co-ordinated prescription actions, one set for each of the domain requirements actions: *Projection*, *Instantiation*, *Determination* and *Extension*. They are – we assume n separate software product requirements: *Projection:* If the n product requirements do not have the same projections, then identify a common projection which they all share, and refer to it as the common projection. Then develop, for each of the n product requirements, if required, a *specific projection* of the common one. Let there be m such specific projections, $m \leq n$. *Instantiation:* First instantiate the common projection, if any instantiation is needed. Then for each of the m specific projections instantiate these, if required. *Determination:* Likewise, if required, "perform" "determination" of the possibly instantiated common projection, and, similarly, if required, "perform" "determination" of the up to m possibly instantiated projections. *Extension:* Finally "perform extension" likewise: First, if required, of the common projection (etc.), then, if required, on the up m specific projections (etc.). These harmonization developments may possibly interact and may need to be iterated ⊙

By a ***partial domain requirement***s we mean a domain requirements which is short of (that is, is missing) some prescription parts: text and formula ⊙ By a ***shared domain requirement***s we mean a domain requirements ⊙ By ***requirements fitting*** m shared $d$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. 2 nor in this section (i.e., Sect. 4) nor in the following two sections. We refer to [Bjø10a].

Example 14. **Domain Requirements — Fitting**: Example 13 hints at three possible sets of interface requirements: (i) for a road pricing [sub-]system, as will be illustrated in Sect. 5.1.3; (ii) for a vehicle monitoring and control [sub-]system, and (iii) for a toll-gate monitoring and control [sub-]system. The vehicle monitoring and control [sub-]system would focus on implementing the vehicle behaviour, see Items 128.- 132. on Page 24. The toll-gate monitoring and control [sub-]system would focus on implementing the calculator behaviour, see Items 134.- 139. on Page 24. The fitting amounts to (a) making precise the (narrative and formal) texts that are specific to each of of the three (i–iii) separate sub-system requirements are kept separate; (b) ensuring that (meaning-wise) shared texts that have different names for (meaning-wise) identical entities have these names renamed appropriately; (c) that these texts are subject to commensurate and ameliorated further requirements development; etcetera □

5. Interface and Derived Requirements

We remind the reader that ***interface requirements*** can be expressed only using terms from both the domain and the machine ⊙ Users are not part of the machine. So no reference can be made to users, such as *"the system must be user friendly"*, and the like ![19] By ***interface requirements*** we [also] mean *requirements prescriptions which refines and extends the domain requirements by considering those requirements of the domain requirements whose endurants (parts, materials) and perdurants (actions, events and behaviours) are **"shared"** between the domain and the machine (being requirements prescribed)* ⊙ The two *interface requirements* definitions above go hand–in–hand, i.e., complement one-another.

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

5.1. Interface Requirements

5.1.1. Shared Phenomena

By ***sharing*** we mean (a) 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 (b) 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 (c) 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 (d) 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

[19] So how do we cope with the statement: *"the system must be user friendly"* ? We refer to Sect. 5.3.2 on Page 31 for a discussion of this issue.

machine; and ***behaviour sharing*** shall lead to interface requirements for action and event dialogues between the machine and its environment.

Environment–Machine Interface: Domain requirements extension, Sect. 4.4, usually introduce new endurants into (i.e., 'extend' the) domain. Some of these endurants may become elements of the domain requirements. Others are to be projected "away". Those that are let into the domain requirements either have their endurants represented, somehow, also in the machine, or have (some of) their properties, usually some attributes, accessed by the machine. Similarly for perdurants. Usually the machine representation of shared perdurants access (some of) their properties, usually some attributes. The interface requirements must spell out which domain extensions are shared. Thus domain extensions may necessitate a review of domain projection, instantiations and determination. In general, there may be several of the projection–eliminated parts (etc.) whose dynamic attributes need be accessed in the usual way, i.e., by means of attr_XYZ_ch channel communications (where XYZ is a projection–eliminated part attribute).

Example 15. **Interface Requirements — Projected Extensions**: We refer to Fig. 2 on Page 22.We do not represent the GNSS system in the machine: only its "effect": the ability to record global positions by accessing the GNSS attribute (channel):

channel
87. {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:

(97.) the vehicle entry sensors (leftmost ■s),
(97.) the vehicle identity sensor (center ■), and
(98.) the vehicle exit sensors (rightmost ■s)

by channels — we refer to Example 13 (Sect. 5.1.3, Page 22):

channel
97. {attr_entry_ch[gi]|gi:GI•xtr_eGIds(trn)} ″enter″
97. {attr_exit_ch[gi]|gi:GI•xtr_xGIds(trn)} ″exit″
98. {attr_identity_ch[gi]|gi:GI•xtr_GIds(trn)} TIVI □

5.1.2. ***Shared Endurants***

Example 16. **Interface Requirements. Shared Endurants**: The main shared endurants are the vehicles, the net (hubs, links, toll-gates) and the price calculator. As domain endurants hubs and links undergo changes, all the time, with respect to the values of several attributes: *length, geodetic information, names, wear and tear* (where-ever applicable), *last/next scheduled maintenance* (where-ever applicable), *state* and *state space*, and many others. Similarly for vehicles: their position, velocity and acceleration, and many other attributes. We then come up with something like hubs and links are to be represented as tuples of relations; each net will be represented by a pair of relations a hubs relation and a links relation; each hub and each link may or will be represented by several tuples; etcetera. In this database modeling effort it must be secured that "standard" operations on nets, hubs and links can be supported by the chosen relational database system □

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 reader 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., $(ui,me,attrs)$.

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

5.1.3. *Shared Perdurants*

We can expect that for every part in the domain that is shared with the machine and for which there is a corresponding behaviour of the domain there might be a corresponding process of the machine. If a projected, instantiated, 'determinated' and possibly extended domain part is dynamic, then it is definitely a candidate for being shared and having an associated machine process. We now illustrate the concept of shared perdurants via the domain requirements extension example of Sect. 4.4, i.e. Example 13 Pages 20–25.

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
```

The *calc*ulator signature's *v_c_ch*[*ci,vi*] and *g_c_ch*[*ci,gi*] model the *embedded attribute sharing* between vehicles (their position), respectively gates (their vehicle identfication) and the calculator road map [Bjø16b, Sect. 4.1.1 and 4.5.2].

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.

```
170.  react_to_gates(ci,(vis,gis),vplf)(trm) ≡
170.     let (ee,(τ,(vi,li))) = ⎕ {g_c_ch[ci,gi]?|gi:GI•gi∈ gis} in
176.     case ee of
177.        "Enter" →
178.            calc(ci,(vis,gis),vplf)(trm∪[vi↦⟨(τ,SonL(li))⟩]),
179.         "Exit"  →
180.            billing(vi,trm(vi)⌢⟨(τ,SonL(li))⟩);
181.            calc(ci,(vis,gis),vplf)(trm\{vi}) end end
```

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

5.2. Derived Requirements

Definition 18. Derived Perdurant: By a ***derived perdurant*** we shall understand a perdurant which is not shared with the domain, but which focus on exploiting facilities of the software or hardware of the machine ⊙

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

5.2.1. *Derived Actions*

Definition 19. Derived Action: By a ***derived action*** we shall understand (a) a conceptual action (b) that calculates a usually non-Boolean valued property from, and possibly changes to (c) a machine behaviour state (d) as instigated by some actor ⊙

Example 19. Domain Requirements. Derived Action: Tracing Vehicles: The example is based on the *Road Pricing Calculator Behaviour* of Example 18 on the previous page. The “external” actor, i.e., a user of the *Road Pricing Calculator* system wishes to trace specific vehicles “cruising” the toll-road. That user (a *Road Pricing Calculator* staff), issues a command to the *Road Pricing Calculator* system, with the identity of a vehicle not already being traced. As a result the *Road Pricing Calculator* system augments a possibly void trace of the timed toll-road positions of vehicles. We augment the definition of the *calc*ulator definition Items 168.–181., Pages 29–29.

182. Traces are modeled by a pair of dynamic attributes:
    a. as a programmable attribute, *tra:TRA*, of the set of identifiers of vehicles being traced, and
    b. as a reactive attribute, *vdu:VDU*[20], that maps vehicle identifiers into time-stamped sequences of simple vehicle positions, i.e., as a subset of the *trm:TRM* programmable attribute.
183. The actor-to-calculator *begin* or *end* trace command, *cmd:Cmd*, is modeled as an autonomous dynamic attribute of the *calc*ulator.
184. The *calc*ulator signature is furthermore augmented with the three attributes mentioned above.
185. The occurrence and handling of an actor trace command is modeled as a non-deterministic external choice and a *react_to_trace_cmd* behaviour.
186. The reactive attribute value (`attr_vdu_ch ?`) is that subset of the traffic map (*trm*) which records just the time-stamped sequences of simple vehicle positions being traced (*tra*).

```
type
182.a.  TRA = VI-set
182.b.  VDU = TRM
183.    Cmd = BTr | ETr
183.    BTr :: VI
183.    ETr :: VI

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

The *185.,186. attr_cmd_ch,attr_vdu_ch* of the *calc*ulator signature models the *calc*ulator’s external *command* and *visual display unit* attributes.

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

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

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

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

5.2.2. *Derived Events*

Definition 20. Derived Event: By a ***derived event*** we shall understand (a) a conceptual event, (b) that calculates a property or some non-Boolean value (c) from a machine behaviour state change ⊙

[20] VDU: visual display unit

Example 20. Domain Requirements. Derived Event: Current Maximum Flow: The example is based on the *Road Pricing Calculator Behaviour* of Examples 19 and 18 on Page 29. By "the current maximum flow" we understand a time-stamped natural number, the number representing the highest number of vehicles which at the time-stamped moment cruised or now cruises around the toll-road net. We augment the definition of the *cal*culator definition Items 168.–193., Pages 29–30.

194. We augment *cal*culator signature with
195. a time-stamped natural number valued dynamic programmable attribute, *(t:𝕋,max:Max)*.
196. Whenever a vehicle enters the toll-road net, through one of its [entry] gates,
    a. it is checked whether the resulting number of vehicles recorded in the *road traffic map* is higher than the hitherto *max*imum recorded number.
    b. If so, that programmable attribute has its number element "upped" by one.
    c. Otherwise not.
197. No changes are to be made to the react_to_gates behaviour (Items 170.–181. Page 29) when a vehicle exits the toll-road net.

```
type
195.   MAX = 𝕋 × NAT
value
184.,194.  calc: ci:CI×(vis:VI-set×gis:GI-set) → RLF → TRM → TRA → MAX
169.,170.      in {v_c_ch[ci,vi]|vi:VI•vi ∈ vis}, {g_c_ch[ci,gi]|gi:GI•gi ∈ gis}, attr_cmd_ch,attr_vdu_ch  Unit
170.   react_to_gates(ci,(vis,gis))(vplf)(trm)(tra)(t,m) ≡
170.     let (ee,(τ,(vi,li))) = ⌷{g_c_ch[ci,gi]|gi:GI•gi∈ gis} in
176.     case ee of
196.       "Enter" → calc(ci,(vis,gis))(vplf)(trm∪[vi↦⟨(τ,SonL(li))⟩])(tra)(τ,if card dom trm = m then m+1 else m end),
197.       "Exit" → billing(vi,trm(vi)⌢⟨(τ,SonL(li))⟩); calc(ci,(vis,gis))(vplf)(trm\{vi})(tra)(t,m) end
176.     end
```

The above behaviour, Items 168. on Page 29 through 196.c., is the one for which we are to design software □

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

5.2.3. *No Derived Behaviours*

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

5.3. Discussion

5.3.1. *Derived Requirements*

Formulation of derived actions or derived events usually involves technical terms not only from the domain but typically from such conceptual 'domains' as mathematics, economics, engineering or their visualisation. Derived requirements may, for some requirements developments, constitute "sizable" requirements compared to "all the other" requirements. For their analysis and prescription it makes good sense to first having developed "the other" requirements: domain, interface and machine requirements. The treatment of the present paper does not offer special techniques and tools for the conception, &c., of derived requirements. Instead we refer to the seminal works of [DvLF93, Lau02, van09].

5.3.2. *Introspective Requirements*

Humans, including human users are, in this paper, considered to never be part of the domain for which a requirements prescription is being developed. If it is necessary to involve humans in the domain description or the requirements prescription then their prescription is to reflect assumptions upon whose behaviour the machine rely. It is therefore that we, above, have stated, in passing, that we cannot accept requirements of the kind: *"the machine must be user friendly"*, because, in reality, it means *"the user must rely upon the machine being 'friendly'"* whatever that may mean. We are not requirements prescribing humans, nor their sentiments !

6. Machine Requirements

Other than listing a sizable number of *machine requirement facet*s we shall not cover machine requirements in this paper. The reason for this is as follows. We find, cf. [Bjø06, Sect. 19.6], that when the individual machine requirements are expressed then references to domain phenomena are, in fact, abstract references, that is, they do not refer to the semantics of what they name. Hence *machine requirement*s "fall" outside the scope of this paper — with that scope being *"derivation" of requirements from domain specifications* with emphasis on derivation techniques that relate to various aspects of the domain.

(A) There are the *technology requirements* of (1) *performance* and (2) *dependability*. Within *dependability requirements* there are (a) *accessibility*, (b) *availability*, (c) *integrity*, (d) *reliability*, (e) *safety*, (f) *security* and (g) *robustness* requirements. A proper treatment of dependability requirements need a careful definition of such terms as *failure, error, fault,* and, from these *dependability.* (B) And there are the *development requirements* of (i) *process*, (ii) *maintenance*, (iii) *platform*, (iv) *management* and (v) *documentation* requirements. Within *maintenance requirements* there are (ii.1) *adaptive*, (ii.2) *corrective*, (ii.3) *perfective*, (ii.4) *preventive*, and (ii.5) *externsional* requirements. Within *platform requirements* there are (iii.1) *development*, (iii.2) *execution*, (iii.3) *maintenance*, and (iii.4) *demonstration* platform requirements. We refer to [Bjø06, Sect. 19.6] for an early treatment of *machine requirements*.

7. Conclusion

Conventional requirements engineering considers the domain only rather implicitly. Requirements gathering ('acquisition') is not structured by any pre-existing knowledge of the domain, instead it is "structured" by a number of relevant techniques and tools [Jac01, van09, Jac10] which, when applied, "fragment-by-fragment" "discovers" such elements of the domain that are immediately relevant to the requirements. The present paper turns this requirements prescription process "up-side-down". Now the process is guided ("steered", "controlled") almost exclusively by the domain description which is assumed to be existing before the requirements development starts. In conventional requirements engineering many of the relevant techniques and tools can be said to take into account *sociological* and *psychological* facets of gathering the requirements and *linguistic* facets of expressing these requirements. That is, the focus is rather much on the *process.* In the present paper's requirements "derivation" from domain descriptions the focus is all the time on the descriptions and prescriptions, in particular on their formal expressions and the "transformation" of these. That is (descriptions and) prescriptions are considered formal, *mathematical* objects. That is, the focus is rather much on the *objects.*

•••

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

What has been Achieved ? We have shown how to systematically "derive" initial aspects of requirements prescriptions from domain descriptions. The stages[21] and steps[22] of this "derivation"[23] are new. We claim that current requirements engineering approaches, although they may refer to a or the 'domain', are not really 'serious' about this: they do not describe the domain, and they do not base their techniques and tools on a reasoned understanding of the domain. In contrast we have identified, we claim, a logically motivated decomposition of requirements into three phases, cf. Footnote 21., of domain requirements into five steps, cf. Footnote 22., and of interface requirements, based on a concept of shared entities, tentatively into (α) shared endurants, (β) shared actions, (γ) shared events, and (δ) shared behaviours (with more research into the (α-δ) techniques needed).

Present Shortcomings and Research Challenges: We see three shortcomings: (1) The "derivation" techniques have yet to consider "extracting" requirements from *domain facet descriptions*. Only by including *domain facet descriptions* can we, in "deriving" *requirements prescriptions*, include failures of, for example, support technologies and humans, in the design of tault-tolerant software. (2) The "derivation" principles, techniques and tools should be given a formal treatment. (3) There is a serious need for relating the approach of the present paper to that

[21] (a) domain, (b) interface and (c) machine requirements

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

[23] We use double quotation marks: "..." to indicate that the derivation is not automatable.

of the seminal text book of [van09, Axel van Lamsweerde]. [van09] is not being “replaced” by the present work. It tackles a different set of problems. We refer to the penultimate paragraph before the **Acknowledgement** closing.

Comparison to “Classical” Requirements Engineering: Except for a few, represented by two, we are not going to compare the contributions of the present paper with published journal or conference papers on the subject of requirements engineering. The reason for this is the following. The present paper, rather completely, we claim, reformulates requirements engineering, giving it a ‘foundation’, in *domain engineering*, and then developing *requirements engineering* from there, viewing requirements prescriptions as “derived” from domain descriptions. We do not see any of the papers, except those reviewed below [JHJ07] and [DvLF93], referring in any technical sense to ‘domains’ such as we understand them.

[JHJ07, Deriving Specifications for Systems That Are Connected to the Physical World] The paper that comes closest to the present paper in its serious treatment of the [problem] domain as a precursor for requirements development is that of [JHJ07, Jones, Hayes & Jackson]. A purpose of [JHJ07] (Sect. 1.1, Page 367, last §) is to see “how little can one say” (about the problem domain) when expressing assumptions about requirements. This is seen by [JHJ07] (earlier in the same paragraph) as in contrast to our form of domain modeling. [JHJ07] reveals assumptions about the domain when expressing *rely guarantee*s in tight conjunction with expressing the *guarantee* (requirements). That is, analysing and expressing requirements, in [JHJ07], goes hand-in-hand with analysing and expressing fragments of the domain. The current paper takes the view that since, as demonstrated in [Bjø16b], it is possible to model sizable aspects of domains, then it would be interesting to study how one might “derive” — and which — requirements prescriptions from domain descriptions; and having demonstrated that (i.e., the “how much can be derived”) it seems of scientific interest to see how that new start (i.e., starting with a priori given domain descriptions or starting with first developing domain descriptions) can be combined with existing approaches, such as [JHJ07]. We do appreciate the “tight coupling” of rely–guarantees of [JHJ07]. But perhaps one looses understanding the domain due to its fragmented presentation. If the ‘relies’ are not outright, i.e., textually directly expressed in our domain descriptions, then they obviously must be provable properties of what our domain descriptions express. Our, i.e., the present, paper — with its background in [Bjø16b, Sect. 4.7] — develops — with a background in [Jac95, M.A. Jackson] — a set of principles and techniques for the access of attributes. The “discovery” of the `CM` and `SG` channels of [JHJ07] and of the type of their messages, seems, compared to our approach, less systematic. Also, it is not clear how the [JHJ07] case study “scales” up to a larger domain. The *sluice gate* of [JHJ07] is but part of a large (‘irrigation’) system of reservoirs (water sources), canals, sluice gates and the fields (water sinks) to be irrigated. We obviously would delineate such a larger system and research & develop an appropriate, both informal, a narrative, and formal domain description for such a class of irrigation systems based on assumptions of precipitation and evaporation. Then the users’ requirements, in [JHJ07], that the sluice gate, over suitable time intervals, is open 20% of the time and otherwise closed, could now be expressed more pertinently, in terms of the fields being appropriately irrigated.

[DvLF93, Goal-directed Requirements Acquisition] outlines an approach to requirements acquisition that starts with fragments of domain description. The domain description is captured in terms of predicates over *actors, actions, events, entities* and (their) *relations*. Our approach to domain modeling differs from that of [DvLF93] as follows: Agents, actions, entities and relations are, in [DvLF93], seen as specialisations of a concept of *objects*. The nearest analogy to relations, in [Bjø16b], as well as in this paper, is the signatures of perdurants. Our ‘agents’ relate to discrete endurants, i.e., parts, and are the behaviours that evolve around these parts: one agent per part ! [DvLF93] otherwise include describing parts, relations between parts, actions and events much like [Bjø16b] and this paper does. [DvLF93] then introduces a notion of *goal*. *A* ***goal****, in [DvLF93], is defined as ″a nonoperational objective to be achieved by the desired system. Nonoperational means that the objective is not formulated in terms of objects and actions “available” to some agent of the system* ⊙[24]″ [DvLF93] then goes on to exemplify goals. In this, the current paper, we are not considering *goal*s, also a major theme of [van09].[25] Typically the expression of goals of

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

[25] An example of a goal — for the road pricing system — could be that of *shortening travel times of motorists, reducing gasoline consumption and air pollution, while recouping investments on toll-road construction.* We consider techniques for ensuring the above kind of goals “outside” the realm of computer & computing science but “inside” the realm of operations research (`OR`) — while securing that the `OR` models are commensurate with our domain models.

[DvLF93, van09], are “within” computer & computing science and involve the use of temporal logic.[26] *"Constraints are operational objectives to be achieved by the desired (i.e., required) system, ..., formulated in terms of objects and actions “available” to some agents of the system. ... Goals are made operational through constraints. ... A constraint operationalising a goal amounts to some abstract “implementation” of this goal"* [DvLF93]. [DvLF93] then goes on to express goals and constraints operationalising these. [DvLF93] is a fascinating paper[27] as it shows how to build goals and constraints on domain description fragments.

•••

These papers, [JHJ07] and [DvLF93], as well as the current paper, together with such seminal monographs as [ZH04, OD08, van09], clearly shows that there are many diverse ways in which to achieve precise requirements prescriptions. The [ZH04, OD08] monographs primarily study the $\mathscr{D},\mathscr{S} \models \mathscr{R}$ specification and proof techniques from the point of view of the specific tools of their specification languages[28]. Physics, as a natural science, and its many engineering ‘renditions’, are manifested in many separate sub-fields: Electricity, mechanics, statics, fluid dynamics — each with further sub-fields. It seems, to this author, that there is a need to study the [ZH04, OD08, van09] approaches and the approach taken in this paper in the light of identifying sub-fields of requirements engineering. The title of the present paper suggests one such sub-field.

Acknowledgments: This paper has been many years underway. Earlier versions have been the basis for (“innumerable”) PhD lectures and seminars around the world — after I retired, in 2007, from The Technical University of Denmark. I thank the many organisers of those events for their willingness to “hear me out”: Jin Song Dong, NUS, Singapore; Kokichi Futatsugi and Kazuhiro Ogata, JAIST, Japan; Dominique Méry, Univ. of Nancy, France; Franz Wotawa and Bernhard K. Aichernig, Techn. Univ. of Graz, Austria; Wolfgang J. Paul, Univ. of Saarland, Germany; Alan Bundy, University of Edinburgh, Scotland; Tetsuo Tamai, then at Tokyo Univ., now at Hosei Univ., Tokyo, Japan; Jens Knoop, Techn. Univ. of Vienna, Austria; Dömölki Balint and Kozma László, Eötös Loránt Univ., Budapest, Hungary; Lars-Henrik Ericsson, Univ. of Uppsala, Sweden; Peter D. Mosses, Univ. of Swansea, Wales; Magne Haveraaen, Univ. of Bergen, Norway; Sun Meng, Peking Univ., China; He JiFeng and Zhu HuiBiao, East China Normal Univ., Shanghai, China; Zhou Chaochen, Lin Huimin and Zhan Naijun, Inst. of Softw., CAS, Beijing, China; Victor P. Ivannikov, Inst. of Sys. Prgr., RAS, Moscow, Russia; Luís Soares Barbosa and Jose Nuno Oliveira, Univ. of Minho, Portugal; and Steeve McKeever and Andreas Hamfeldt, Uppsala University.

7.1. Bibliographical Notes

I have thought about domain engineering for more than 20 years. But serious, focused writing only started to appear since [Bjø06, Part IV] — with [Bjø03, Bjø97] being exceptions: [Bjø07] suggests a number of domain science and engineering research topics; [Bjø10a] covers the concept of domain facets; [BE10] explores compositionality and Galois connections. [Bjø08, Bjø10c] show how to systematically, but, of course, not automatically, “derive” requirements prescriptions from domain descriptions; [Bjø11a] takes the triptych software development as a basis for outlining principles for believable software management; [Bjø09, Bjø14a] presents a model for Stanisław Leśniewski’s [CV99] concept of mereology; [Bjø10b, Bjø11b] present an extensive example and is otherwise a precursor for the present paper; [Bjø11c] presents, based on the `TripTych` view of software development as ideally proceeding from domain description via requirements prescription to software design, concepts such as software demos and simulators; [Bjø13] analyses the `TripTych`, especially its domain engineering approach, with respect to [Mas43, Mas54, Maslow]’s and [PS04, Peterson’s and Seligman’s]’s notions of humanity: how can computing relate to notions of humanity; the first part of [Bjø14b] is a precursor for [Bjø16b] with the second part of [Bjø14b] presenting a first formal model of the elicitation process of analysis and description based on the prompts more definitively presented in the current paper; and with [Bjø14c] focus on domain safety criticality. The present paper, [Bjø16a], marks, for me, a high point, with [Bjø16b] now constituting the base introduction to domain science & engineering.

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

[27] — that might, however, warrant a complete rewrite.

[28] The Duration Calculus [`DC`], respectively `DC, Timed Automata` and `Z`

8. References

[BE10] Dines Bjørner and Asger Eir. Compositionality: Ontology and Mereology of Domains. Some Clarifying Observations in the Context of Software Engineering in July 2008, eds. Martin Steffen, Dennis Dams and Ulrich Hannemann. In *Festschrift for Prof. Willem Paul de Roever Concurrency, Compositionality, and Correctness*, volume 5930 of *Lecture Notes in Computer Science*, pages 22–59, Heidelberg, July 2010. Springer.

[Bjø97] Dines Bjørner. Michael Jackson's Problem Frames: Domains, Requirements and Design. In Li ShaoYang and Michael Hinchley, editors, *ICFEM'97: International Conference on Formal Engineering Methods*, Los Alamitos, November 12–14 1997. IEEE Computer Society. Final Version.

[Bjø03] Dines Bjørner. Domain Engineering: A "Radical Innovation" for Systems and Software Engineering ? In *Verification: Theory and Practice*, volume 2772 of *Lecture Notes in Computer Science*, Heidelberg, October 7–11 2003. Springer–Verlag. The Zohar Manna International Conference, Taormina, Sicily 29 June – 4 July 2003. .

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

[Bjø07] Dines Bjørner. Domain Theory: Practice and Theories, Discussion of Possible Research Topics. In *ICTAC'2007*, volume 4701 of *Lecture Notes in Computer Science (eds. J.C.P. Woodcock et al.)*, pages 1–17, Heidelberg, September 2007. Springer.

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

[Bjø09] Dines Bjørner. On Mereologies in Computing Science. In *Festschrift: Reflections on the Work of C.A.R. Hoare*, History of Computing (eds. Cliff B. Jones, A.W. Roscoe and Kenneth R. Wood), pages 47–70, London, UK, 2009. Springer.

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

[Bjø10b] Dines Bjørner. Domain Science & Engineering – From Computer Science to The Sciences of Informatics, Part I of II: The Engineering Part. *Kibernetika i sistemny analiz*, (4):100–116, May 2010.

[Bjø10c] Dines Bjørner. The Rôle of Domain Engineering in Software Development. Why Current Requirements Engineering Seems Flawed! In *Perspectives of Systems Informatics*, volume 5947 of *Lecture Notes in Computer Science*, pages 2–34, Heidelberg, Wednesday, January 27, 2010. Springer.

[Bjø11a] Dines Bjørner. Believable Software Management. *Encyclopedia of Software Engineering*, 1(1):1–32, 2011.

[Bjø11b] Dines Bjørner. Domain Science & Engineering – From Computer Science to The Sciences of Informatics Part II of II: The Science Part. *Kibernetika i sistemny analiz*, (2):100–120, May 2011.

[Bjø11c] Dines Bjørner. Domains: Their Simulation, Monitoring and Control – A Divertimento of Ideas and Suggestions. In *Rainbow of Computer Science, Festschrift for Hermann Maurer on the Occasion of His 70th Anniversary.*, Festschrift (eds. C. Calude, G. Rozenberg and A. Saloma), pages 167–183. Springer, Heidelberg, Germany, January 2011.

[Bjø13] Dines Bjørner. *Domain Science and Engineering as a Foundation for Computation for Humanity*, chapter 7, pages 159–177. Computational Analysis, Synthesis, and Design of Dynamic Systems. CRC [Francis & Taylor], 2013. (eds.: Justyna Zander and Pieter J. Mosterman).

[Bjø14a] Dines Bjørner. *A Rôle for Mereology in Domain Science and Engineering*. Synthese Library (eds. Claudio Calosi and Pierluigi Graziani). Springer, Amsterdam, The Netherlands, October 2014.

[Bjø14b] Dines Bjørner. Domain Analysis: Endurants – An Analysis & Description Process Model. In Shusaku Iida, José Meseguer, and Kazuhiro Ogata, editors, *Specification, Algebra, and Software: A Festschrift Symposium in Honor of Kokichi Futatsugi*. Springer, May 2014.

[Bjø14c] Dines Bjørner. Domain Engineering – A Basis for Safety Critical Software. Invited Keynote, ASSC2014: Australian System Safety Conference, Melbourne, 26–28 May, December 2014.

[Bjø16a] Dines Bjørner. From Domain Descriptions to Requirements Prescriptions – A Different Approach to Requirements Engineering. *Submitted for consideration by Formal Aspects of Computing*, 2016.

[Bjø16b] Dines Bjørner. Manifest Domains: Analysis & Description. *Expected published by Formal Aspects of Computing*, 2016.

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

[DvLF93] Anne Dardenne, Axel van Lamsweerde, and Stephen Fickas. Goal-directed requirements acquisition. *Sci. Comput. Program.*, 20(1-2):3–50, April 1993.

[ESA] ESA. Global Navigation Satellite Systems. Web, European Space Agency. http://en.wikipedia.org/wiki/Satellite_navigation.

[GGJZ00] Carl A. Gunter, Elsa L. Gunter, Michael A. Jackson, and Pamela Zave. A Reference Model for Requirements and Specifications. *IEEE Software*, 17(3):37–43, May–June 2000.

[IEE90] IEEE Computer Society. IEEE–STD 610.12-1990: Standard Glossary of Software Engineering Terminology. Technical report, IEEE, IEEE Headquarters Office, 1730 Massachusetts Avenue, N.W., Washington, DC 20036-1992, USA. Phone: +1-202-371-0101, FAX: +1-202-728-9614, 1990.

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

[Jac01] Michael A. Jackson. *Problem Frames — Analyzing and Structuring Software Development Problems*. ACM Press, Pearson Education. Addison-Wesley, England, 2001.

[Jac10] Michael A. Jackson. Program Verification and System Dependability. In Paul Boca and Jonathan Bowen, editors, *Formal Methods: State of the Art and New Directions*, pages 43–78, London, UK, 2010. Springer.

[JHJ07] Cliff B. Jones, Ian Hayes, and Michael A. Jackson. Deriving Specfications for Systems That Are Connected to the Physical World. In Cliff Jones, Zhiming Liu, and James Woodcock, editors, *Formal Methods and Hybrid Real-Time Systems: Essays in Honour of Dines Bjørner and Zhou Chaochen on the Occasion of Their 70th Birthdays*, volume 4700 of *Lecture Notes in Computer Science*, pages 364–390. Springer, 2007.

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

[Mas43] Abraham Maslow. A Theory of Human Motivation. *Psychological Review*, 50(4):370–96, 1943. http://psychclassics.yorku.ca/-Maslow/motivation.htm.

[Mas54] Abraham Maslow. *Motivation and Personality*. Harper and Row Publishers, 3rd ed., 1954.

[OD08] Ernst-Rüdiger Olderog and Henning Dierks. *Real-Time Systems: Formal Specification and Automatic Verification*. Cambridge University Press, UK, 2008.

[PS04] Christopher Peterson and Martin E.P. Seligman. *Character strengths and virtues: A handbook and classification*. Oxford University Press, 2004.

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

[ZH04] Chao Chen Zhou and Michael R. Hansen. *Duration Calculus: A Formal Approach to Real–time Systems*. Monographs in Theoretical Computer Science. An EATCS Series. Springer–Verlag, 2004.