A Rôle for Domain Engineering in Software Developemnt

Why Current Requirements Engineering Seems Flawed !

Dines Bjørner, DTU Informatics, Denmark

April 19, 2012

1. Opening

- Before
  - we can design software, the how,
  - we must understand its requirements, the what.
- Before
  - we can formulate requirements,
  - we must understand the [application] domain.

- Examples of domains are:
  - air traffic,
  - airports,
  - container lines,
  - banks,
  - hospitals,
  - pipelines,
  - railways,
  - stock exchanges,
  - “the market”,

  etcetera.

- Thus we “divide” the process of developing software into three major phases:
  - $\mathbb{D}$omain engineering,
  - $\mathbb{R}$equirements engineering, and
  - $\mathbb{S}$oftware design.
- and pursue these phases such that $\mathbb{D}, \mathbb{S} \models \mathbb{R}$,
- that is, such that we can
  - prove the correctness of the $\mathbb{S}$oftware design
  - with respect to the $\mathbb{R}$equirements presecription
  - in the context of the $\mathbb{D}$omain description,
    - $*$ that is, under assumptions about the domain.

- So let's take a look at
  - what such a domain description might look like
  - and how we might "derive" a [domain] requirements prescription from a domain description.
- We shall not go into a methodology of constructing domain descriptions.

2. Structure of Talk

Contents

3. Domain Engineering

- We choose as our example domain that of transportation systems, δ:Δ.
  - From any such δ we can observe (obs_) a number of

∗ simple entities	Slides 8–46,	∗ events	Slides 57–62,
∗ actions	Slides 47–56,	∗ and behaviours	Slides 63–81.

  - This section will therefore be structured accordingly.
- Thus domains are composed from one or more
  - simple entities,
  - actions,
  - events and
  - behaviours;
- and it is the job of the domain analyser to "discover" these
  - entities,
  - their composition,
  - use and
  - other properties.

3.1. Transport Simple Entities

1. There are five classes of simple entities in our example:

(a) transportation nets	Slides 9–24,
(b) people	Slides 25–29,
(c) vehicles,	Slides 30–36,
(d) time,	Slides 38–40, and
(e) timetables,	Slides 41–46.

type

1(a). N
1(b). C
1(c). F
1(d). T
1(e). TT

value

1(a). obs_N: $\Delta \rightarrow$ N
1(b). obs_C: $\Delta \rightarrow$ C
1(c). obs_F: $\Delta \rightarrow$ F
1(d). obs_T: $\Delta \rightarrow$ T
1(e). obs_TT: $\Delta \rightarrow$ TT

3.1.1. Transportation Nets

3.1.1.1. Nets, Hubs and Links

2. Nets are composite simple entities from which one can observe
   (a) sets: hs:HS, of zero, one or more hubs and
   (b) sets: ls:LS, of zero, one or more links.

type
2. H, L
value
2(a). obs_HS: N → HS[1]
2(a). obs_Hs: HS → H-**set**
2(b). obs_LS: N → LS
2(b). obs_Ls: LS → L-**set**

[1] The prefix obs_ can be pronounced: 'observe' (obs_erve).

3.1.1.2. Hub and Link Identifiers

3. Hubs and links are uniquely identified.

4. Hub and link identifiers are all distinct.

```
type
3.  HI, LI
value
3.  mer_HI: H → HI
3.  mer_LI: L → LI
axiom
4.  ∀ n:N, h,h′:H, l,l′:L •
4.      {h,h′}⊆obs_Hs(n) ∧ {l,l′}⊆obs_Ls(n) ⇒
4.          h≠h′⇒mer_HI(h)≠mer_HI(h′) ∧
4.          l≠l′⇒mer_LI(l)≠mer_LI(l′)
```

[1]mer_HI reads: “the HI ‘mereology’ contribution from the argument (here H); that is, the prefix mer_ can be pronounced ‘mereology’ (mer_eology).

5. From a net one can extract (χtr[2]) the hub identifiers of all its hubs.

6. From a net one can extract the link identifiers of all its links.

value

5. χtrHIs: N $\rightarrow$ HI-**set**
5. χtrHIs(n) $\equiv$ {mer_HI(h)|h:H•h $\in$ obs_Hs(n)}
6. χtrLIs: N $\rightarrow$ LI-**set**
6. χtrLIs(n) $\equiv$ {mer_LI(l)|l:L•l $\in$ obs_Ls(n)}

[2] The prefix χtr can be pronounced 'extract' (χtract).

7. Given a net and an identifier of a hub of the net one can get (γet[3]) that hub from the net.
8. Given a net and an identifier of a link of the net one can get that link from the net.

```
value
7.  $\gamma$etH: N $\rightarrow$ HI $\overset{\sim}{\rightarrow}$ H
7.  $\gamma$etH(n)(hi) $\equiv$
7.      if hi $\in$ $\chi$trHIs(n)
7.          then let h:H $\cdot$ mer_HI(h)=hi in h end
7.          else chaos end
8.  $\gamma$etL: N $\rightarrow$ LI $\overset{\sim}{\rightarrow}$ L
8.  $\gamma$etL(n)(li) $\equiv$
8.      if li $\in$ $\chi$trLIs(n)
8.          then let l:L $\cdot$ mer_LI(l)=li in l end
8.          else chaos end
```

[3]The prefix γet can be pronounced ‘get’.

3.1.1.3. Mereology

9. From a hub one can observe the identifiers of all the (zero or more) links incident upon (or emanating from), i.e., connected to the hub.
10. From a link one can observe the distinct identifiers of the two distinct hubs the link connects.
11. The link identifiers observable from a hub must be identifiers of links of the net.
12. The hub identifiers observable from a link must be identifiers of hubs of the net.

```
value
9.   mer_LIs: H → LI-set
10.  mer_HIs: L → HI-set
axiom
9.  ∀ n:N,h:H,l:L•h ∈ obs_Hs(n)∧l ∈ obs_Ls(n) ⇒
10.       card mer_HIs(l)=2
11.     ∧ ∀ li:LI • li ∈ mer_LIs(h) ⇒ li ∈ χtrLIs(n)
12.     ∧ ∀ hi:HI • hi ∈ mer_HIs(l) ⇒ hi ∈ χtrHIs(n)
```

3.1.1.4. Maps

- Maps, **m:M**, are abstractions of nets.
- We shall model maps as follows:

13. hub identifiers map into singleton maps from link identifiers to hub identifiers, such that
    (a) if, in m, h_i
    (b) maps into $[l_{ij} \mapsto h_j]$,
    (c) then h_j maps into $[l_{ij} \mapsto h_i]$ in m, for all such h_i.

```
type
13.    M = HI ⟶ₘ (LI ⟶ₘ HI)
axiom
13(a).  ∀ m:M,h_i:HI • h_i ∈ dom m ⇒
13(b).      let [l_ij↦h_j] = m(h_i) in
13(c).      h_j ∈ dom m ∧ m(h_j)=[l_ij↦h_i]
13(a).      end
```

14. From a net one can extract its map.

```
value
14.  $\chi$trM: N $\rightarrow$ M
14.  $\chi$trM(n) $\equiv$
14.        [ hi$\mapsto$[ lij$\mapsto$hj|
14.                   lij:LI $\cdot$ lij $\in$ mer_LIs($\gamma$etH(n)(hi))
14.                 $\wedge$ hj = $\gamma$etL(n)(lij)$\backslash$$\{$hi$\}$ ] |
14.               hi:HI$\cdot$hi $\in$ $\chi$trHIs(n) ]
```

3.1.1.5. Routes

15. By a route of a net we shall here understand a non-zero sequence of alternative hub and link identifiers such that
    (a) adjacent elements of the list are hub and link identifiers of hubs, respectively links of the net, and such that
    (b) a link identifier identifies a link one of whose adjacent hubs are indeed identified by the "next" hub identifier of the route, respectively such that
    (c) a hub identifier identifies a hub one of whose connected links are indeed identified by the "next" link identifier of the route.

```
type
93.  R′ = (LI|HI)*
93.  R = {|r:R′•∃ n:N•wf_R(r)(n)|}
value
93.  wf_R: R′ → N → Bool
93.  wf_R(r)(n) ≡ proper_adjacency(r) ∧ embedded_route(r)(n)

93.  proper_adjacency: R′ → Bool
93.  proper_adjacency(r) ≡
93.     ∀ i:Nat•{i,i+1}⊆inds r⇒is_LI(r(i))∧is_HI(r(i+1))∨is_HI(r(i))∧is_LI(r(i+1))

93.  embedded_route: R′ → N → Bool
93.  embedded_route(r)(n) ≡
93.     ∀ i:Nat•{i,i+1}⊆inds r ⇒
93.        is_LI(r(i)) → r(i+1) ∈ mer_HIs(γetL(r(i))(n)),
93.        is_HI(r(i)) → r(i+1) ∈ mer_LIs(γetL(r(i))(n))
```

[3] is_LI and is_LI are specification language "built-in" functions, one for each type name. In general is_K(e), where K is a type name, expresses whether the simple entity e is of type K (or not).

16. Given a net one can calculate the possibly infinite set of all, possibly cyclic but finite length routes:
    (a) if li is an identifier of a link of a net then $\langle$li$\rangle$ is a route of the net;
    (b) if hi is an identifier of a hub of a net then $\langle$hi$\rangle$ is a route of the net;
    (c) if r and r′ are routes of a net n and if the last identifier of r is the same as the first identifier of r′ then r$\widehat{\ }$tl r′ is a route of the net.
    (d) Only such routes which can be constructed by applying rules 96–16(c) a finite[4] number of times are proper routes of the net.
17. Similarly one can extract routes from maps.

[4]If applied infinitely many times we include infinite length routes.

```
value
94.    χtrRs: N → R-set
94.    χtrRs(n) ≡ in
16(b).     let rs={⟨li⟩|li:LI·li ∈ χtrLIs(n)}∪{⟨hi⟩|hi:HI·hi ∈ χtrHIs(n)}
16(b).             ∪ {⟨hi,li⟩ | hi:HI,li:LI · ⟨hi⟩ ∈ rs
16(b).               ∧ li ∈ χtrLIs(n)∧li ∈ mer_LIs(γetH(n)(hi))}
16(b).             ∪ {⟨li,hi⟩ | li:LI,hi:HI · ⟨li⟩ ∈ rs
16(b).               ∧ hi ∈ χtrHIs(n)∧hi ∈ mer_HIs(γetL(n)(li))}
16(c).             ∪ {r^tl r′|r,r′:R·{r,r′}⊆rs ∧ r(len rl)=hd r′} in
94.        rs end

17.    χtrRs: M → R-set
17.    χtrRs(m) as rs
17.      pre  ∃ n:N · m = χtrM(n)
17.      post ∃ n:N · m = χtrM(n) ∧ rs = routes(n)
```

- For later use we define a concept of a 'stuttered sampling' of a route r.
  - The sequence ℓ is said to be a 'sampling' of a route r
    - if zero or more elements of r are not in ℓ;
  - and the sequence ℓ is said to be a 'stuttering' of a route r
    - if zero or more elements of r are repeated in ℓ —
  - while, in both cases ('sampling' an 'stuttering') the elements of r in ℓ follow their order in r.

18. A sequence, ℓ, of link and hub identifiers (in any order) is a 'stuttered sampling' of a route, r, of a net
    (a) if there exists a mapping, mi, from indices of the former into ascending and distinct indices of the latter
    (b) such that for all indexes, i, in ℓ, we have that $\ell(\mathsf{i}) = \mathsf{r}(\mathsf{mi}(\mathsf{i})) \wedge \mathsf{i} \leq \mathsf{mi}(\mathsf{i})$.

type

18(a). $\mathrm{IM}' = \mathbf{Nat} \underset{m}{\rightarrow} \mathbf{Nat}$

18(a). $\mathrm{IM} = \{|\mathrm{im}:\mathrm{IM}' \cdot \mathrm{wf\_IM}(\mathrm{im})|\}$

value

18(a). $\mathrm{wf\_IM}: \mathrm{IM}' \rightarrow \mathbf{Bool}$

18(a). $\mathrm{wf\_IM}(\mathrm{im}) \equiv$

18(a). $\quad \mathbf{dom}\ \mathrm{im} = \{1..\mathbf{max}\,\mathbf{dom}\ \mathrm{im}\}$

18(a). $\wedge\ \forall\ \mathrm{i}:\mathbf{Nat} \cdot \{\mathrm{i},\mathrm{i}+1\} \subseteq \mathbf{dom}\ \mathrm{im} \Rightarrow \mathrm{im}(\mathrm{i}) \leq \mathrm{im}(\mathrm{i}+1)$

18. $\mathrm{is\_stuttered\_sampling}: (\mathrm{LI}|\mathrm{HI})^* \times \mathrm{R} \rightarrow \mathbf{Bool}$

18. $\mathrm{is\_stuttered\_sampling}(\ell,\mathrm{r}) \equiv$

18(a). $\quad \exists\ \mathrm{im}:\mathrm{IM} \cdot \mathbf{dom}\ \mathrm{im} = \mathbf{inds}\ \ell \wedge \mathbf{rng}\ \mathrm{im} \subseteq \mathbf{inds}\ \mathrm{r} \Rightarrow$

18(b). $\qquad \forall\ \mathrm{i}:\mathbf{Nat} \cdot \mathrm{i} \in \mathbf{dom}\ \mathrm{im} \Rightarrow \ell(\mathrm{i}) = \mathrm{r}(\mathrm{mi}(\mathrm{i}))$

3.1.1.6. Hub and Link States

- A state of a hub (a link) indicates which are the permissible flows of traffic.

19. The state of a hub is a set of pairs of link identifiers where these are the identifiers of links connected to the hub.
20. The state of a link is a set of pairs of distinct hub identifiers where these are the identifiers of the two hubs connected to the link.
21. The state space of a hub is a set of hub states.
22. The state space of a link is a set of link states.

We say that states and state spaces are $\alpha\tau r$ibutes of hubs and links.

```
type
19.  $\mathrm{H\Sigma} = (\mathrm{LI} \times \mathrm{LI})$-set
20.  $\mathrm{L\Sigma} = (\mathrm{HI} \times \mathrm{HI})$-set
21.  $\mathrm{H\Omega} = \mathrm{H\Sigma}$-set
22.  $\mathrm{L\Omega} = \mathrm{L\Sigma}$-set
value
19.  $\alpha\tau r\mathrm{H\Sigma}$: $\mathrm{H} \rightarrow \mathrm{H\Sigma}$
20.  $\alpha\tau r\mathrm{L\Sigma}$: $\mathrm{L} \rightarrow \mathrm{L\Sigma}$
21.  $\alpha\tau r\mathrm{H\Omega}$: $\mathrm{H} \rightarrow \mathrm{H\Omega}$
22.  $\alpha\tau r\mathrm{L\Omega}$: $\mathrm{L} \rightarrow \mathrm{L\Omega}$
axiom
     $\forall$ n:N, h:H, l:L $\cdot$ h $\in$ obs_Hs(n)$\wedge$l $\in$ obs_Ls(n) $\Rightarrow$
19.        let h$\sigma$ = $\alpha\tau r\mathrm{H\Sigma}$(h),
20.              l$\sigma$ = $\alpha\tau r\mathrm{L\Sigma}$(l) in
19.        $\forall$ (li,li'):(LI$\times$LI)$\cdot$(li,li')$\in$ h$\sigma$$\Rightarrow$$\{$li,li'$\}$$\subseteq$$\chi$trLIs(n)
20.    $\wedge$ $\forall$ (hi,hi'):(HI$\times$HI)$\cdot$(hi,hi')$\in$ l$\sigma$$\Rightarrow$$\{$hi,hi'$\}$$\subseteq$$\chi$trHIs(n)
21.    $\wedge$ h$\sigma$ $\in$ $\alpha\tau r\mathrm{H\Omega}$(h)
22.    $\wedge$ l$\sigma$ $\in$ $\alpha\tau r\mathrm{L\Omega}$(l) end
```

3.1.2. Communities and People

23. A community is a community of people here considered an unordered set.
24. As simple entities we consider people (persons) to be uniquely identifier atomic dynamic inert entities.

    We shall later view such people as a main state component of people as behaviours.
25. No two persons have the same unique identifier.
26. Essential attributes of persons are:

    (a) name,
    (b) ancestry,
    (c) gender,
    (d) age,
    (e) height,
    (f) weight,

    and others.

Additional attributes will be brought forward in the next section (Vehicles).

type
23. P
91. PI
value
23. obs_Ps: C $\rightarrow$ P-**set**
91. $\alpha\tau r$PI: P $\rightarrow$ PI
axiom
92. $\forall$ p,p′:P • p$\neq$p′ $\Rightarrow$ $\alpha\tau r$PI(p) $\neq$ $\alpha\tau r$PI(p′)
type
26. PNm, PAn, PGd, PAg, PHe, PWe, ...
value
26(a). $\alpha\tau r$PNm: P $\rightarrow$ PNm
26(b). $\alpha\tau r$PAn: P $\rightarrow$ PAn
26(c). $\alpha\tau r$PGd: P $\rightarrow$ PGd
26(d). $\alpha\tau r$PAg: P $\rightarrow$ PAg
26(e). $\alpha\tau r$PHe: P $\rightarrow$ PHe
26(f). $\alpha\tau r$PWe: P $\rightarrow$ PWe

27. From any set of persons one can extract its corresponding set of unique person identifiers.

value
27. χtrPIs: P-**set** $\rightarrow$ PI-**set**
27. χtrPIs(ps) $\equiv$ $\{$obs_PI(p)$|$p:P$\cdot$p $\in$ ps$\}$
axiom
27. $\forall$ ps:P-**set** $\cdot$ **card** ps $=$ **card** χtrPIs(ps)

3.1.3. An Aside on Simple Entity Equality Modulo an Attribute

- Attributes have names and values.
  - (Not just people,
  - but also the simple entities of nets,, hubs and links,
  - as well as of other simple entities to be introduced later.)
- Some attributes are dynamic, that is, their values may change.
- We wish to be able to express that a simple entity, p,
  - some of whose attribute values may change,
  - is “still, basically, that same” simple entity,
  - that is, that $p = p'$ —
  - where we assume that the only thing which does not change is some notion of a unique simple entity identifier.

28. The attribute observers of people are those of observing names, ancestry, gender, age, height, weight, and others.

    Let $\mathsf{SE}\alpha\tau r\mathsf{set}$ stand for the set of attribute functions of the simple entity whose class (type) is SE.

29. Then to express that a simple entity of type SE in invariant modulo some observer function $\alpha\tau r\mathsf{A}$, specifically, in this case, that a person is invariant wrt. height, we write as is shown in formula 29. below, where p and p' is the ("before", "after") person that is claimed to be "the same", i.e. invariant modulo $\alpha\tau r\mathsf{A}$.

type

28. $\mathrm{P}\alpha\tau r\mathrm{set} = \{|\ \alpha\tau r\mathrm{PNm},\ \alpha\tau r\mathrm{An},\ \alpha\tau r\mathrm{Gd},\ \alpha\tau r\mathrm{Ag},\ \alpha\tau r\mathrm{He},\ \alpha\tau r\mathrm{We},\ ...|\}$

axiom

29. $\forall\ \ \alpha\tau r\mathcal{F}{:}\mathrm{P}\alpha\tau r\mathrm{set}\cdot\ \ \alpha\tau r\mathcal{F} \in \mathrm{P}\alpha\tau r\mathrm{set}\backslash\{\alpha\tau r\mathrm{H}\} \Rightarrow \alpha\tau r\mathcal{F}(\mathrm{p}){=}\alpha\tau r\mathcal{F}(\mathrm{p}')$

3.1.4. Fleets and Vehicles

30. A fleet is a composite simple entity.
31. From a fleet one can observe its atomic simple sub-entities of vehicle.
    (a) Vehicles, in addition to their unique vehicle identity,
    (b) may enjoy some static attributes: weight, size, etc., and dynamic attributes: directed velocity, directed acceleration,
    (c) position on the net:
    (d) at a hub or on a link, etc.

type
30. F
31. V
31(a). VI
31(b). We, Sz, ..., DV, DA, ...
value
31. obs_Vs: F $\rightarrow$ V-**set**
31(a). obs_VI: V $\rightarrow$ VI
31(b). $\alpha\tau r$We: V $\rightarrow$ We, ..., $\alpha\tau r$DV: V $\rightarrow$ DV, ...
31(c). $\alpha\tau r$VP: V $\rightarrow$ VP
type
31(d). VP == atH(hi) | onL(fhi,li,f:**Real**,thi) **axiom** $0<f\ll1$
axiom
31(a). $\forall$ v,v′:V•v$\neq$v′ $\Rightarrow$ obs_VI(v)$\neq$obs_VI(v′)

32. Buses are vehicles, but not all vehicles are buses.
33. Vehicles are either in the traffic (to be defined later) or are not.
34. From any set of vehicles one can extract its corresponding set of unique vehicle identifiers.

type
32. $\mathrm{B} \subset \mathrm{V}$
value
32. $\mathsf{is}\_\mathrm{B}: \mathrm{V} \rightarrow \mathbf{Bool}$
33. $\mathsf{is}\_\mathrm{InTF}: \mathrm{V} \rightarrow \mathbf{Bool}$
34. $\chi\mathrm{trVIs}:\ \mathrm{V}\text{-}\mathbf{set} \rightarrow \mathrm{VI}\text{-}\mathbf{set}$
34. $\chi\mathrm{trVIs}(\mathrm{vs}) \equiv \{\mathsf{obs}\_\mathrm{VI}(\mathrm{v})|\mathrm{v}:\mathrm{V}\cdot\mathrm{v} \in \mathrm{vs}\}$
axiom
34. $\forall\ \mathrm{vs}:\mathrm{V}\text{-}\mathbf{set} \cdot \mathbf{card}\ \mathrm{vs} = \mathbf{card}\ \chi\mathrm{trVIs}(\mathrm{vs})$

3.1.5. Vehicles and People

35. Vehicles in traffic have a driver who is a person, and distinct vehicles have distinct drivers.

36. Vehicles in traffic have zero, one or more passengers – who are persons different from the driver.

37. Vehicles have one owner (who is a person) and persons own zero or more vehicles.

35. $\alpha\tau r$Driver: V $\overset{\sim}{\rightarrow}$ PI
35.   **pre** $\alpha\tau r$Driver(v): is_InTF(v)
36. $\alpha\tau r$Pass: V $\rightarrow$ PI-**set**
36.   **pre** $\alpha\tau r$Pass(v): is_InTF(v) $\Rightarrow$ $\alpha\tau r$Driver(v)$\notin$$\alpha\tau r$Ps(v)
37. $\alpha\tau r$Owner: V $\rightarrow$ PI
37. $\alpha\tau r$Own: P $\rightarrow$ VI-**set**

38. In the (domain state) context of the set of persons, ps, and the set of vehicles, vs, in the domain (δ:Δ), we have the following constraints:
    (a) the person, p, identified by pi, as the owner of a vehicle, v, in vs, is in ps; and
    (b) the vehicle, v, identified by vi, as being owned be a person, p, in ps, is in vs.

$$
\begin{array}{ll}
38. & \textbf{axiom}\ \forall\ \delta{:}\Delta,\mathrm{ps}{:}\mathrm{P}\textbf{-set},\mathrm{vs}{:}\mathrm{V}\textbf{-set} \cdot \mathrm{ps}{=}\mathsf{obs\_}\mathrm{Ps}(\delta)\wedge\mathrm{vs}{=}\mathsf{obs\_}\mathrm{Vs}(\delta) \Rightarrow \\
38(a). & \quad \forall\ \mathrm{v}{:}\mathrm{V} \cdot \mathrm{v} \in \mathrm{vs} \Rightarrow \alpha\tau r\mathrm{Owner}(\mathrm{v}) \in \chi\mathrm{trPIs}(\mathrm{ps}) \\
38(b). & \quad \wedge\ \forall\ \mathrm{p}{:}\mathrm{P} \cdot \mathrm{p} \in \mathrm{ps} \Rightarrow \alpha\tau r\mathrm{Own}(\mathrm{p}) \subseteq \chi\mathrm{trVIs}(\mathrm{vs})
\end{array}
$$

39. Given a set of persons one can extract the set of the unique person identifiers of these persons.
40. Given a set of persons one can extract the set of the unique vehicle identifiers of vehicles owned by these persons.
41. Given a set of persons and a unique person identifier (of one of these persons) one can get that person.
42. Given a set of vehicles one can extract the set of the unique vehicles identifiers of these vehicles.

value

39. χtrPIs: P-**set** $\rightarrow$ PI-**set**
39. χtrPIs(ps) $\equiv$ $\{\alpha\tau r$PI(p)|p:P·p $\in$ ps$\}$

40. γetP: P-**set** $\rightarrow$ PI $\overset{\sim}{\rightarrow}$ P
40. γetP(ps)(pi) $\equiv$ **let** p:P· p $\in$ ps $\wedge$ pi=$\alpha\tau r$PI(p) **in** p **end**
40. **pre** pi $\in$ χtrPIs(ps)

41. χtrVIs: V-**set** $\rightarrow$ VI-**set**
41. χtrVIs(vs) $\equiv$ $\{\alpha\tau r$VI(v)|v:V·v $\in$ vs$\}$

42. γetV: V-**set** $\rightarrow$ VI $\overset{\sim}{\rightarrow}$ V
42. γetV(vs)(vi) $\equiv$ **let** v:V· v $\in$ vs $\wedge$ vi=$\alpha\tau r$VI(v) **in** v **end**
42. **pre** vi $\in$ χtrVIs(vs)

3.1.6. Community & Fleet States

43. We shall later need to refer to a state consisting of pairs of

- communities and
- fleets.

43. $\mathrm{CF\Sigma} = \mathrm{C} \times \mathrm{F}$

3.1.7. Time

- Time is an elusive "quantity", ripe, always, for philosophical discourses, for example:
  - J. M. E. McTaggart:
    The Unreality of Time (1908),
  - Wayne D. Blizard:
    A Formal Theory of Objects, Space and Time (1990) and
  - Johan van Benthem:
    The Logic of Time (1991).
- Here we shall take a somewhat more mundane view of time.

44. Time is here considered a dense, enumerable set of points.
45. A time interval is the numerical distance between two such points.
46. There is a time starting point and thus we can speak of the time interval since then!
    (a) One can compare two times and one can compare two time intervals.
    (b) One can add a time and an interval to obtain a time.
    (c) One can subtract a time interval from a time to obtain, conditionally, a time.
    (d) One can subtract a time from a time to obtain, conditionally, a time interval.
    (e) One can multiply a time interval with a real to obtain a time interval.
    (f) One can divide one time interval by another to obtain a real.

type

44. T

45. TI

value

46. obs_TI: T $\rightarrow$ TI

46(a). $<,\leq,=,>,\geq$: $((\mathrm{T}\times\mathrm{T})|(\mathrm{TI}\times\mathrm{TI})) \rightarrow$ **Bool**

46(b). $+$: $\mathrm{T}\times\mathrm{TI} \rightarrow \mathrm{T}$

46(c). $-$: $\mathrm{T}\times\mathrm{TI} \xrightarrow{\sim} \mathrm{T}$ **axiom** $\forall\ -(\mathrm{t},\mathrm{ti}) \cdot$ obs_TI(t)$\geq$ti

46(d). $-$: $((\mathrm{T}\times\mathrm{T})|(\mathrm{TI}\times\mathrm{TI})) \xrightarrow{\sim} \mathrm{TI}$ **axiom** $\forall\ -(\tau,\tau') \cdot \tau'\leq\tau$

46(e). $*$: $\mathrm{TI}\times$**Real** $\rightarrow$ TI

46(f). $/$: $\mathrm{TI}\times\mathrm{TI} \rightarrow$ **Real**

3.1.8. Timetables

- By a timetable we shall here understand a transport timetable: a listing of the times that public transport services, say a bus, arrive and depart specified locations.
- We shall model a concept of timetables in four "easy"
  - steps by first defining bus stops,
  - then bus schedules
  - and finally timetables.

3.1.8.1. Bus Stops

- To properly define a timetable we thus need to introduce the notion of ‘specified locations’.

47. By a bus location (that is, a bus stop), we shall understand a location
    (a) either *at a hub*
    (b) or *down a fraction of the distance between two hubs (a from and a to hub) along a link*.
48. The fraction is a real close to 0 and certainly much less than 1.

type

47. $\quad \mathrm{S} = \mathrm{atH} \mid \mathrm{onL}$

47(a). $\mathrm{atH} == \mu\alpha\kappa\mathrm{AtH}(\mathrm{hi{:}HI})$

47(b). $\mathrm{onL} == \mu\alpha\kappa\mathrm{OnL}(\mathrm{fhi{:}HI,li{:}LI,f{:}Frac,thi{:}HI})$

48. $\quad \mathrm{Frac} = \textbf{Real}$ $\quad$ **axiom** $\forall\ \mathrm{f{:}F}\cdot 0<\mathrm{f}\ll 1$

3.1.8.2. Bus Schedules

49. A bus stop visit is modelled as a triple: an arrival time, a bus stop location and a departure time — such that the latter is larger than (i.e., "after") the former.
50. A bus schedule is a pair: a route and a list of two or more "consecutive" bus stop visits where "consecutiveness" has two parts:
    (a) the **projection** of the list of bus stop visits onto just a list of its "at Hub" and "on Link" identifiers must form a stuttered sampling of the route,
    (b) departure times of the "former" bus stop visit must be "before" the arrival time of the latter, and
    (c) if two or more consecutive stops along the same link, then a former stop must be a fraction down the link less than a latter stop.

```
type
49.     BV   = T × S × T  axiom ∀ (at,bs,dt):S • at<dt
50.     BS′  = R × BVL, BVL  = BV*
50.     BS   = {|bs•wf_BS(bs)|}
value
50.     wf_BS(r,l) ≡
50(b).       is_stuttered_sampling(proj(l),r)
50(b).     ∧ ∀ i:Nat•{i,i+1}<inds l ⇒
50(b).         case (l(i),l(i+1)) of
50(b).           ((_,atH(hi),dt),(at,atH(hi′),_)) → dt<at,
50(b).           ((_,atH(hi),dt),(at,onL(fi,li,f,ti),_)) → dt<at,
50(b).           ((_,onL(fi,li,f,ti),dt),(at,atH(hi),_)) → dt<at,
50(b).           ((_,onL(fi,li,f,ti),_),(at,onL(fi′,li′,f′,ti′),_)) → dt<at
50(c).                ∧ fi=fi′∧li=li′∧ti=ti′ ⇒ f<f′ end
50(a).      proj: BV* → (HI|LI)*
50(a).      proj(bvl) ≡
50(a).        ⟨ case bs of atH(hi) → hi, onL(_,li,_,_) → li end
50(a).          | i:Nat,bv:BV: i ∈ inds bvl ∧ bv=bvl(i)=(_,bs,_) ⟩
```

3.1.8.3. Bus Transport Timetables

51. Bus schedules are grouped into bus lines
52. and bus schedules have distinct identifiers.
53. A timetable is now a pair of
    (a) a transport map and
    (b) a table which
        i. to each bus line associates a sub-timetable
            - which to each bus schedule identifier
            - associates a bus schedule,

    such that

    (a) no bus schedule identifier appears twice in the timetable and
    (b) each bus schedule is commensurate with the transport map.

```
type
51.        BLId
52.        BSId
53.        TT′ = M × TBL
53(b).     TBL = BLid →m SUB_TT
53((b))i.  SUB_TT = BSId →m BS
53.        TT = {|tt:TT′•wf_TT(tt)|}
value
53.        wf_TT: TT′ → Bool
53.        wf_TT(m,tbl) ≡
53(a).         ∀ bsm,bsm′:(BSId →m BS)•{bsm,bsm′}⊆rng tbl⇒dom bsm ∩ dom bsm′={}
53(b).         ∧ ∀ (r,bvl):BS • (r,bvl) ∈ rng bsm ⇒ r ∈ routes(m)
```

3.2. Transport Actions

- We consider each of four of the these three kinds of transport simple entities as being “the center” of events:
  - the net,
  - people and vehicles and
  - timetables.

3.2.1. Transport Net Actions

54. One can insert hubs into a net to obtain an updated net. The inserted hub has no 'connected link identifiers'.
55. One can remove a hub from a net to obtain an updated net. The removed hub must have no 'connected link identifiers'.
56. One can insert a link into a net to obtain an updated net. The inserted link must have two existing 'connecting hub identifiers' and their hubs (cannot have contained the link identifier of the inserted link) must now record that link identifier as the only change to their attributes.
57. One can remove a link from a net to obtain an updated net. The hubs identified by the removed links' 'connecting hubs' must have their 'connected link identifiers' no longer reflecting the removed link — as their only change.

```
value
54. $\text{insertH}: H \rightarrow N \xrightarrow{\sim} N$
54. $\text{insertH}(h)(n)$ **as** $n'$
54.   **pre** $h \notin \mathsf{obs\_}\text{Hs}(n)$
54.   **post** $\mathsf{obs\_}\text{Hs}(n) = \mathsf{obs\_}\text{Hs}(n') \cup \{h\} \wedge$
54.     $\mathsf{obs\_}\text{Ls}(n) = \mathsf{obs\_}\text{Ls}(n')$

55. $\text{removeH}: HI \rightarrow N \xrightarrow{\sim} N$
55. $\text{removeH}(hi)(n)$ **as** $n'$
55.   **pre** $hi \in \chi\text{trHIs}(n)$
55.   **post** $\mathsf{obs\_}\text{LIs}(\text{get\_HI}(hi)(n)) = \{\} \wedge$
55.     $\mathsf{obs\_}\text{Hs}(n') = \mathsf{obs\_}\text{Hs}(n) \backslash \{\text{get\_HI}(hi)(n)\}$

56. $\text{insertL}: L \rightarrow N \xrightarrow{\sim} N$
56. $\text{insertL}(l)(n)$ **as** $n'$
56.   **pre** $l \notin \mathsf{obs\_}\text{Ls}(n)$
56.   **post** $\mathsf{obs\_}\text{Ls}(n') = \mathsf{obs\_}\text{Ls}(n) \cup \{l\}$
56.     **let** $\{hi,hi'\} = \mathsf{obs\_}\text{HIs}(l)$ **in**
56.     **let** $(h,h') = (\gamma\text{etH}(hi)(n), \gamma\text{etH}(hi')(n)),$
56.       $(nh,nh') = (\gamma\text{etH}(hi)(n'), \gamma\text{etH}(hi')(n'))$ **in**
56.     $\mathsf{obs\_}\text{LIs}(nh) = \mathsf{obs\_}\text{LIs}(h) \cup \{\mathsf{obs\_}\text{LI}(l)\},$
56.     $\mathsf{obs\_}\text{LIs}(nh') = \mathsf{obs\_}\text{LIs}(h') \cup \{\mathsf{obs\_}\text{LI}(l)\}$ **end**

57. $\text{removeL}: LI \rightarrow N \xrightarrow{\sim} N$
57. $\text{removeL}(li)(n)$ **as** $n'$
57.   **pre** $li \in \chi\text{trLIs}(n)$
57.   **post** $\mathsf{obs\_}\text{Ls}(n) = \mathsf{obs\_}\text{Ls}(n') \backslash \{l\}$
57.     **let** $\{hi,hi'\} = \mathsf{obs\_}\text{HIs}(\text{get\_L}(li)(n))$ **in**
57.     **let** $(h,h') = (\text{get\_H}(hi)(n), \text{get\_H}(hi')(n)),$
57.       $(nh,nh') = (\text{get\_H}(hi)(n'), \text{get\_H}(hi')(n'))$ **in**
57.     $\mathsf{obs\_}\text{LIs}(nh) = \mathsf{obs\_}\text{LIs}(h) \backslash \{li\},$
57.     $\mathsf{obs\_}\text{LIs}(nh') = \mathsf{obs\_}\text{LIs}(h') \backslash \{li\}$ **end end**
```

3.2.2. People and Vehicle Actions

58. We shall only consider actions on people and vehicles in the (state) context of the community and fleet of a transport system, cf. Item 38 (Slide 34).

59. People can transfer (xfer) ownership of vehicles (being transferred vi,v,v′) one-at-a-time, from one person (fpi,fp – selling) to another person (tpi,tp buying).

```
value
58.    $\text{xfer\_V: PI}\times\text{VI}\times\text{PI} \rightarrow (\text{C}\times\text{F}) \rightarrow (\text{C}\times\text{F})$
58.    $\text{xfer\_V(fpi,vi,tpi)(c,f)}$ as $\text{(c}',\text{f}')$
58.         pre ...
58.         post $\text{xfer\_V(fpi,vi,tpi)(obs\_Ps(c),obs\_Vs(f))} = (\text{ps}',\text{vs}')$
58.            $\wedge\ \forall\ \mathcal{F}_C{:}\alpha\tau r\text{Cs(c)}\cdot\mathcal{F}_C\text{(c)}=\mathcal{F}_C(\text{c}')$
58.            $\wedge\ \forall\ \mathcal{F}_F{:}\alpha\tau r\text{Fs(f)}\cdot\mathcal{F}_F\text{(f)}=\mathcal{F}_F(\text{f}')$
58.    $\text{xfer\_V: PI}\times\text{VI}\times\text{PI} \rightarrow (\text{P-}\mathbf{set}\times\text{V-}\mathbf{set}) \rightarrow (\text{P-}\mathbf{set}\times\text{V-}\mathbf{set})$
59.    $\text{xfer\_V(fpi,vi,tpi)(ps,vs)}$ as $(\text{ps}',\text{vs}')$
60(a).      pre  $\text{fpi}\neq\text{tpi}\wedge\{\text{fpi,tpi}\}\subseteq\chi\text{trPIs(ps)}\wedge\text{vi} \in \chi\text{trVIs(vs)}$
60(b).      post let $\text{(fp,tp)}=(\gamma\text{etP(fpi)(ps)},\gamma\text{etP(tpi)(ps)}),$
60(c).                $(\text{fp}',\text{tp}')=(\gamma\text{etP(fpi)(ps}'),\gamma\text{etP(tpi)(ps}')),$
60(d).                $(\text{v},\text{v}')=(\gamma\text{etV(vi)(vs)},\gamma\text{etP(vi)(vs}'))$ in
60(e).              $\text{ps}\backslash\{\text{fp,tp}\} = \text{ps}'\backslash\{\text{fp}',\text{tp}'\} \wedge \text{vs}\backslash\{\text{v}\} = \text{vs}\backslash\{\text{v}'\}$
60(f).           $\wedge\ \text{fp}' = \text{sell(fp,vi)} \wedge \text{tp}' = \text{buy(tp,vi)} \wedge \text{v}' = \text{xfer\_Owner(vi,fp,tp}$
```

60. We explain the above pre/post conditions:
    (a) The from and to persons must be distinct and they and the identified vehicle must be in the current domain state.
    (b) We need to be able to refer to the from and to persons before
    (c) and after the transfer vehicle ownership action,
    (d) as well as to the vehicle changing ownership.
    (e) Except for the persons and vehicle involved in the transfer operation no changes occur to the persons and vehicles of the current domain state.
    (f) Simultaneously the from person sells the vehicle, the to person buys that same vehicle and the vehicle changes owner.

value

61. sell: $\mathrm{P} \times \mathrm{VI} \rightarrow \mathrm{P}$
61. sell(p,vi) **as** p′
61(a). obs_PI(p)=obs_PI(p′)
61(b). $\wedge$ vi $\in \alpha\tau r$Own(p) $\wedge$ vi $\notin \alpha\tau r$Own(p′)
61(c). $\wedge$ $\forall$ F:P$\alpha\tau r$set$\setminus\{\alpha\tau r$VI$\}$•F(p)=F(p′)
62. buy: $\mathrm{P} \times \mathrm{VI} \rightarrow \mathrm{P}$
62. buy(p,vi) **as** p′
62(a). obs_PI(p)=obs_PI(p′)
62(b). $\wedge$ vi $\notin \alpha\tau r$Own(p) $\wedge$ vi $\in \alpha\tau r$Own(p′)
62(c). $\wedge$ $\forall$ F:P$\alpha\tau r$set$\setminus\{\alpha\tau r$VI$\}$•F(p)=F(p′)
63. xfer_Owner: $\mathrm{PI} \times \mathrm{V} \times \mathrm{PI} \rightarrow \mathrm{V}$
63. xfer_Owner(fpi,v,tpi) **as** v′
63(a). obs_VI(v)=obs_VI(v′)
63(b). $\wedge$ fpi=$\alpha\tau r$Owner(v) $\wedge$ tpi$\neq\alpha\tau r$Owner(v)
63(c). $\wedge$ fpi$\neq\alpha\tau r$Owner(v′) $\wedge$ tpi=$\alpha\tau r$Owner(v′)
63(d). $\wedge$ $\forall$ F:P$\alpha\tau r$set$\setminus\{\alpha\tau r$VI$\}$•F(p)=F(p′)

61. The buyer function:
    (a) The seller identity is unchanged.
    (b) The vehicle was owned by the seller before, but not after the transfer.
    (c) All other seller attributes are unchanged.
62. The seller function:
    (a) The buyer identity is unchanged.
    (b) The vehicle was not owned by the buyer before, but is owned by the buyer after the transfer.
    (c) All other buyer attributes are unchanged.
63. The vehicle ownership change function:
    (a) The vehicle identity is unchanged.
    (b) The seller identity is noted in the vehicle before the transfer but is not noted after the transfer.
    (c) The buyer identity is not noted in the vehicle before the transfer but is noted after the transfer.
    (d) All other vehicle attributes are unchanged.

3.2.3. Time Table Actions

- Timetables are dynamic inert simple entities.
  - They do not change their value by own volition.
  - Their value is changed only by some external action upon them.

64. One can create an empty timetable.
65. One can inquire whether a timetable is empty.
66. One can inquire as to the set of bus line identifies of a timetable.
67. One can inquire as to the set of all bus lines' unique bus schedules identifiers.
68. For every bus line identity one can inquire as to the set of unique bus schedule identifiers.
69. One can insert a bus schedule with an appropriate new bus schedule identifier into a timetable.
70. One can delete an appropriately identified bus schedule from a non-empty timetable.

```
value
64. emptyTT: Unit → TT
64. emptyTT() as tt axiom is_empty(tt)
65. is_emptyTT: TT → Bool
65. is_emptyTT(_,tbl) ≡ case m of (_,[ bli↦bsm ]∪ tbl′)→false,_→true end
66. χtrBLIds: TT → BLId-set
66. χtrBLIds(_,tbl) ≡ dom tbl
67. χtrBSIds: TT → BSid-set
67. χtrBSIds(_,tbl) ≡ ∪{tbl(bli)|bli:BLid•bli ∈ dom tbl}
68. χtrBSIds: TT × BLid → BSid-set
68. χtrBSIds((_,tbl),bli) ≡ dom tbl(bli)
69. insert_BS: (BLid × (BSid × BS)) → TT ~→ TT
69. insert_BS(bli,(bsi,bs))(m,tbl) as (m′,tbl′)
69.    pre  wf_TT(m,tbl) ∧ bsi ∉ χtrBSids(m,tbl)
69.    post wf_TT(m′,tbl′) ∧ m=m′
69.       ∧ bli∉ dom tbl ⇒ tbl′ = tbl ∪ [ bli↦[ bsi↦bs ] ]
69.       ∧ bli ∈ dom tbl ⇒ tbl′ = tbl † [ bli↦tbl(bli)∪[ bsi↦bs ] ]
70. delete_BS: (BLid × (BSid × BS)) → TT ~→ TT
70. delete_BS(bli,(bsi,bs))(m,tbl) as (m′,tbl′)
70.    pre  wf_TT(m,tbl) ∧ bli ∈ dom tbl ∧ bsi ∈ dom(tbl(bli))
70.    post wf_TT(m′,tbl′) ∧ m=m′ ∧ tbl′ = tbl † [ bli↦tbl(bli)\{bsi} ]
```

3.3. Transport Events

3.3.1. Transport Net Events

- Events are characterisable by a predicate over before/after state pairs and times.
- The event of a mudslide "removing" the linkage between two hubs can be modelled as follows:
  - first the removal of the affected link (ℓ, connecting hubs h' and h''),
  - then the insertion of two fresh hubs (h''' and h''''), and
  - finally the insertion of new links (ℓ' and ℓ'' between h' and h''', respectively h'' and h'''').
- With these "actions" as the only actions at or during the event we have that:

71. A link_disappearance predicate can be defined as follows:
   (a) there exists h' and h'' in net n with these hubs becoming nh' and nh'' in net n', and
   (b) there exists exactly and only h''' and h'''' in the new net n' which were not in the old net n,
   (c) exactly one link, ℓ', has disappeared from net n (that is: was in n but is not in n'), and exactly two links, ℓ'', ℓ''', (which were not in n) have appeared in net n',
   (d) the two new links, ℓ'' and ℓ''', are linking h' with h''', respectively h'' with h'''',
   (e) hub h' (h'') is no longer connected to ℓ' (ℓ'), but includes ℓ'' (ℓ'''),
   (f) hub h''' (h'''') connects to only ℓ'' (ℓ'''), and
   (g) link ℓ' (ℓ'') connects $\{h', h'''\}$ ($\{h', h'''\}$).

The event predicate *link_disappearance* is between the nets before and after the event – and some arbitrary time.

```
type
   T
value
71. link_disappearance: N × N → T → Bool
71. link_disappearance(n,n′)(t) ≡
71.       let (hs,ls)=(obs_Hs,obs_Ls)(n), (hs′,ls′)=(obs_Hs,obs_Ls)(n′) in
71(a).       ∃ h′,h″:H•{h,h′}⊆hs ∩ hs′
71(a).   ∧ let (hi′,hi″)=(obs_HI(h′),obs_HI(h″)) in
71(a).       let (nh′,nh″)=(get_H(hi′)(n′),get_H(hi″)(n′)) in
71(b).       ∃ h‴,h⁗:H•{h‴,h⁗}=hs′\hs
71(c).   ∧ ∃ l′:L•{l′}=obs_Ls(n) ∩ obs_Ls(n′) ∧ ∃ l″,l‴:L•{l″,l‴}=obs_Ls(n′)\obs_Ls(n′)
71(d).   ∧ $\alpha\tau r$HIs(l″)={hi′,obs_HI(h‴)}∧$\alpha\tau r$HIs(l‴)={hi″,obs_HI(h⁗)}
71(e).   ∧ $\alpha\tau r$LIs(h′)=$\alpha\tau r$LIs(nh′)\{obs_LI(l′)}∪ obs_LI(l″) ∧ $\alpha\tau r$LIs(h″)=$\alpha\tau r$LIs(nh″)\{obs_
71(f).   ∧ $\alpha\tau r$HIs(l′)={obs_HI(nh′),obs_HI(h‴)}
71(g).   ∧ $\alpha\tau r$HIs(l″)={obs_HI(nh″),obs_HI(h⁗)}
      end end end
```

3.3.2. People Events

72. People are born and people pass away.

value

72. $\text{birth: P-}\mathbf{set} \times \text{P-}\mathbf{set} \to \text{T} \to \mathbf{Bool}$
72. $\text{birth(ps,ps}')(\text{t}) \equiv \exists\ \text{p:P} \cdot \text{p} \notin \text{ps} \wedge \text{p} \in \text{ps}' \wedge \text{ps}'\text{=ps} \cup \{\text{p}\}$
72. $\text{death: P-}\mathbf{set} \times \text{P-}\mathbf{set} \to \text{T} \to \mathbf{Bool}$
72. $\text{death(ps,ps}')(\text{t}) \equiv \exists\ \text{p:P} \cdot \text{p} \in \text{ps} \wedge \text{p} \notin \text{ps}' \wedge \text{ps}'\text{=ps} \backslash \{\text{p}\}$

3.3.3. Vehicle Events

73. Vehicles are manufactured and vehicles are scrapped.

74. Two or more vehicles end up in a mass collision.

value

73. $\text{mfgd: V-}\mathbf{set} \times \text{V-}\mathbf{set} \rightarrow \text{T} \rightarrow \mathbf{Bool}$

73. $\text{mfgd(vs,vs}')\text{(t)} \equiv \exists\ \text{v:V} \cdot \text{v} \notin \text{vs} \wedge \text{v} \in \text{vs}' \wedge \text{vs}'\text{=vs} \cup \{\text{v}\}$

73. $\text{scrpd: V-}\mathbf{set} \times \text{V-}\mathbf{set} \rightarrow \text{T} \rightarrow \mathbf{Bool}$

73. $\text{scrpd(vs,vs}')\text{(t)} \equiv \exists\ \text{v:V} \cdot \text{v} \in \text{vs} \wedge \text{v} \notin \text{ps}' \wedge \text{vs}'\text{=vs}\setminus\{\text{v}\}$

74. $\text{coll: V-}\mathbf{set} \times \text{V-}\mathbf{set} \rightarrow \text{T} \rightarrow \mathbf{Bool}$

74. $\text{coll(vs,vs}')\text{(t)} \equiv \chi\text{trVIs(vs)=}\chi\text{trVIs(vs}')$

74. $\quad \wedge\ \exists\ \text{vs}'':\text{V-}\mathbf{set} \cdot \mathbf{card}\ \text{vs}''\geq 2 \wedge \text{vs}''\subset\text{vs}'$

74. $\qquad \wedge\ \forall\ \text{v,v}':\text{V-}\mathbf{set}\cdot\text{v}\neq\text{b}' \wedge \{\text{v,v}'\}\subseteq\text{vs}'' \wedge \text{samePos(v,v}')$

74. $\text{samePos: V} \times \text{V} \rightarrow \text{T} \rightarrow \mathbf{Bool}$

74. $\text{samePos(v,v}')\text{(t)} \equiv$

74. $\qquad \mathbf{case}\ (\alpha\tau r\text{VP},\alpha\tau r\text{VP})\ \mathbf{of}\ (\text{onL(fhi,li,f,thi),onL(fhi,li,f,thi))} \rightarrow \mathbf{true},\ \_ \rightarrow \mathbf{false\ end}$

3.3.4. Timetable Events

- Timetables are considered to be concepts.
- They may be recorded on paper, electronically or on billboards.
- Somehow they, i.e., the timetable for some specific form of vehicles and for some specific net, are all copies of one another.
- They somehow do not disappear.
- So we decide not to conjure an image, or images, of timetable events and then "model" it, or them.

3.4. Transport Behaviours

- One thing is a simple entity, or a constellation of simple entities;
- another thing is a behaviour "centered around" that, or those, simple entities:
  - a net,
  - a person,
  - a vehicle,

  or other such simple entities as behaviours.
- As we shall soon see,
  - we model behaviours as processes
  - with a notion of a state
  - which significantly includes
    - a simple net entity,
    - a simple person entity,
    - a simple vehicle entity.
- Colloquially we can thus speak of some phenomenon, both
  - by referring to it as a simple entity and
  - by referring to it as a behaviour.

- The complexity of transport behaviours is such that we “stepwise” refine a sketch of transport behaviours;
  - first we sketch some aspects of People Behaviours 65–73
  - then similarly of Vehicle Behaviours 74–81
  - of Timetable Behaviours
  - before tackling the more composite Net Behaviours

3.4.1. Community and Person Behaviours

- We make a distinction between describing
  - the dynamically varying number of people of our domain, $\delta:\Delta$ — modelled as the behaviour **community** — and
  - the individual person, modelled as the behaviours **nascent** and **person**.
- We need to model each individual person behaviour and do so as a `CSP` process.
- We also need to model the dynamically varying number of person behaviours. But `CSP` cannot model that "easily".
  - So we use some technical tricks — of which we are not "proud".

- The model, with one **community** and an indefinite number of **nascent** and **person** behaviours, is not really a proper model of the domain of people.
  - The model of the birth of persons —
    - reflected in the **community** and **nascent/person** behaviours —
  - and the decease of persons —
    - reflected in the same behaviours —
  - is not a very good model.
  - The problem is that we know of no formal specification language which handles the dynamic creation and demise of processes.[5]

[5]The π-Calculus is a mathematical system (a notation etc.) for investigating mobile processes and for giving semantics to the kind of formal specification language which handles the dynamic creation and demise of processes.

3.4.1.1. A Community System Behaviour

75. The concurrent constellation of

- one **community** and
- an indefinite number of pairs of **nascent** and **person**

behaviours will be referred to as the **people_system** behaviour.

76. The **people_system** behaviour is refers to a global (constant) value **pids**: an indefinite set of the unique identifiers of *nascent* (as yet unborn) and **person**s.
77. Each individual of the indefinite number of **nascent** behaviours is initialised with its (future) unique person identity.
78. The **community** behaviour models the birth of persons and kicks off the identified **nascent** behaviour by communicating a person (i.e., a "baby") to the **nascent** behaviour.
79. The identity of a `"deceased"` **person** behaviour is communicated to the **community** behaviour.

80. The communications mentioned in Items 78–79 are modelled by CSP output/inputs over a set of unique person identified community_to_nascent channels, CtN(pi), and person_to_community channels, NtC(pi) channels.

81. Once a nascent behaviour "comes alive" (i.e., a person is alive), communication related to "death" notification concerning that person is from that person's behaviour to the community behaviour via the appropriate person_to_community, PtC(pi) channel.

```
value
76.  pids:PI-set

75.  people_system: Unit → Unit
75.  people_system() ≡
76.      community()
77.      ∥ ∥{nascent(pi)|pi:PI•pi ∈ pids}

channel
80.  {CtN(pi)|pi:PI•pi ∈ pids}: mkBirth(pi:PI,p:P)
81.  {PtC(pi)|pi:PI•pi ∈ pids}: mkDeceased(pi:PI,"deceased")
```

3.4.1.2. A Community Behaviour

82. The community behaviour refers to a global (constant) value of the set of unique person identifiers — of unborn, living or ”deceased” persons.
83. We distinguish between two distinct sets of events:
    (a) persons being born (a singleton event) and
    (b) persons passing away (a singleton event).
84. A birth gives rise to a person, p, being communicated to its identified (obs_PI(p)) nascent behaviour.
85. A person behaviour informs the community behaviour of the decease of that person.

```
variable
    lps:P-set := {} [living persons]
value
82. community: Unit →
82.     out {CtN[i]|i:PI•i ∈ pids}
82.     in {PtC[i]|i:PI•i ∈ pids}  Unit
82. community() ≡
84.     (let p:P•p ∉ lps ∧ obs_PI(p) ∈ pids in
84.      (lps := lps ∪ {p} ∥ CtN(obs_PI(p))!mkBirth(obs_PI(p),p)) end
84.      community())
82.     ⌈⌉
85.     (let m = ⌈⌉{PtC(pi)?|pi:PI•pi ∈ pids} in
85.      assert: ∃ pi:PI•m = mkDeceased("deceased",pi) ;
85.      let mkDeceased("deceased",pi) = m in
85.      let p:P • p ∈ lps ∧ obs_PI(p)=pi in
85.      lps := lps \ {p} end end end
85.      community())
```

3.4.1.3. A Nascent Behaviour

86. A **nascent** behaviour

87. awaits a “birth” notification (in the form of a person identifier and a person) from the **community** behaviour and

88. becomes an appropriate **person** behaviour.

```
value
86. nascent: pi:PI → in CtN(pi) out ... Unit
86. nascent(pi) ≡
87.     let m = CtN(pi) ? in
88.     if m=mkMfgd(pi,p)
88.         then let mkBirth(pi,p) = m in person(pi)(p) end
88.         else chaos end end
```

3.4.1.4. A Person Behaviour

89. The person behaviour has as state-component the atomic simple person entity.
90. We distinguish between four distinct sets of pairs of events and actions:
    (a) death;
    (b) buying and
    (c) selling;
    (d) driver on and
    (e) driver off; and
    (f) passenger on and
    (g) passenger off.

```
type
90.  PAoE == death|buy|sell|start|stop|enter|leave
value
89.  person: pi:PI × P → in ... out PtPs(pi) ... Unit
90.  person(pi)(p) ≡
90.      let a = death⌈⌉buy⌈⌉sell⌈⌉start⌈⌉stop⌈⌉enter⌈⌉leave in
90.      let p′ = case a of
90(a).               death     → "deceased",
90(b).               buy       → buy_act(p),        90(c). sell     → sell_act(p),
90(d).               driv_on   → driv_on_act(p),    90(e). driv_off → driv_off_act(p),
90(f).               pass_on   → pass__act(p)       90(g). pass_off → pass_off_act(p)
89.             end in
89.      if p′="deceased"
89.        then PtoPs(pi)!mkDeceased("deceased") ; stop
89.        else person(pi)(p′)
89.      assert: pi=obs_PI(p)=obs_PI(p′) end end end
```

3.4.2. Fleet and Vehicle Behaviours

- We describe the concepts of
  - a fleet of a dynamically varying number of vehicles
  - and individual vehicles
- using identical modelling techniques as those used for the description of a community of persons.
- We shall therefore restart the numbering of the narrative and formalised items below as from Item 75 on page 67.
- The listener can then "verify" that the two models, that of a community of persons and that of a fleet of vehicles have rather identical behavioural structures.

3.4.2.1. A Vehicle System Behaviour

75. The concurrent constellation of

    - one **fleet** (of vehicles) and
    - an indefinite number of pairs of **latent** and **vehicle**

    behaviours will be referred to as the **vehicle_system** behaviour.

76. The **fleet** behaviour refers to a global constant value, **vids**: an indefinite set of the unique identifiers of *latent*, actual and `"scrapped"` **vehicles**.

77. Each individual of the indefinite number of **latent** behaviours is initialised with its (future) unique vehicle identity.

78. The **fleet** behaviour models the manufacturing of vehicles and kicks off the identified **latent** behaviour by communicating a properly identified vehicle to that **latent** behaviour.

79. The identity of of a ”scrapped” **vehicle** behaviour is communicated to the **fleet** behaviour.

80. The communications mentioned in Items 78–79 are modelled by `CSP` output/inputs over a set of unique vehicle identified fleet_to_latent vehicle channels, FtL(vi).

81. Once a latent vehicle behaviour "comes alive" (i.e., a vehicle has been manufactured and is operating), communication related to `"scrap"` notification concerning that vehicle is from that vehicle's behaviour to the fleet behaviour via the appropriate vehicle_to_fleet, VtF(pi) channel.

value

76. vids:VI-**set**

75. vehicle_system: **Unit** → **Unit**
75. vehicle_system() ≡
76.   fleet(vids)
77.   ∥ ∥{latent(vi)|vi:VI•vi ∈ vids}

channel

80. {FtL(pi)|vi:VI•vi ∈ vids}: mkMfgd(vi:VI,v:V)
81. {VtF(pi)|vi:VI•vi ∈ vids}: mkScrapped(vi:VI,″`scrapped`″)

3.4.2.2. A Vehicle Fleet Behaviour

82. The fleet behaviour refers to a global (constant) value, vids. the set of unique vehicle identifiers — of yet to be manufactured, manufactured and scrapped vehicles.
83. We distinguish between two distinct sets of events:
    (a) vehicles being manufactured (a singleton event) and
    (b) vehicles being scrapped (a singleton event).
84. Vehicle manufacturing gives rise to a vehicle, v, being communicated to its identified (obs_VI(v)) latent behaviour.
85. A vehicle behaviour informs the fleet behaviour of the scrapping of that vehicle.

```
variable
    avs:V-set := {} [active or scrapped vehicles]
value
82. fleet: Unit →
82.     out {FtL[vi]|vi:VI•i ∈ vids}
82.     in {CtF[vi]|vi:VI•i ∈ vids}  Unit
82. fleet() ≡
84.    (let v:V•v ∉ avs ∧ obs_VI(v) ∈ vids in
84.     (avs := avs ∪ {v} ‖ FtL(obs_VI(v))!mkMfgd(obs_VI(v),v)) end
84.     fleet())
82.    ⌈⌉
85.    (let m = ⌈⌉{VtF(vi)?|vi:VI•vi ∈ vids} in
85.     assert:∃ vi:VI • m = mkScrapped(vi,″scrapped″) ;
85.     let mkScrapped(vi,″scrapped″) = m in
85.     let v:V • v ∈ avs ∧ obs_VI(v)=vi in
85.     avs := avs \ {v} end end end
85.     fleet())
```

3.4.2.3. A Latent Behaviour

86. A **latent** behaviour

87. awaits a manufactured notification (including a vehicle) from the **fleet** behaviour and

88. becomes an appropriate **vehicle** behaviour.

```
value
86. latent: vi:VI → in VtL(vi) out ... Unit
86. latent(vi) ≡
87.     let m = PstN(vi) ? in
88.     if m=mkMfgd(″manufactured″,v) assert: vi=obs_VI(v)
88.         then let mkMfgd(_,v) = m in vehicle(vi)(v) end
88.         else chaos end end
```

3.4.2.4. A Vehicle Behaviour

89. The **vehicle** behaviour has as state-component the atomic simple vehicle entity.

90. We distinguish between one event and four distinct sets of pairs or triples of actions:

    (a) scrap (event);

    (b) buying

    (c) and selling;

    (d) driver on

    (e) and driver off;

    (f) passenger on,

    (g) and passenger off;

    (h) and entering the net,

    (i) driving on the net,

    (j) and leaving the net.

```
type
90. VAoE == scrap|buy|sell|driv_on|driv_off|pass_on||pass_off|enter|drive|leave
value
89. vehicle: vi:VI → V → in ... out VtF(pi) ... Unit
90. vehicle(vi)(v) ≡
90.     let a = scrap⌈⌉buy⌈⌉sell⌈⌉driv_on⌈⌉driv_off⌈⌉pass_on⌈⌉pass_off⌈⌉enter⌈⌉drive⌈⌉leave in
90.     let v′ = case a of
90(a).              scrap    → "scrapped",
90(b).              buy      → buy_act(v),        90(c).  sell      → sell_act(v),
90(d).              driv_on  → driv_on_act(v),    90(e).  driv_off  → driv_off_act(v),
90(f).              pass_on  → pass_on_act(v),    90(g).  pass_off  → pass_off_act(v),
90(h).              enter    → enter_act(v),      90(i).  drive     → drive_act(v),
90(j).              leave    → leave_act(v),
89.             end in
89.     if v′="scrapped"
89.       then VtF(vi)!mkScrapped(vi,"scrapped") ; stop
89.       else vehicle(vi)(v′)
89.     assert: vi=obs_VI(v)=obs_VI(v′) end end end
```

3.5. Discussion of Domain Engineering

- We have just touched a few issues of a methodology for domain engineering.
- Thus we have not dealt with principles and techniques of describing domain facets:
  - intrinsics,
  - support technologies,
  - rules and regulations,
  - scripts,
  - management and organisation, and
  - human behaviour.
- Each of these, and other methodological topics have an own set of principles and techniques and an emerging underlying theory.
- One will be touched upon in tomorrow's 10:30 am colloquium.

4. Requirements Engineering

4.1. Preliminaries

4.1.1. The Machine = Hardware + Software

- By 'the machine' we shall understand the
  - software to be developed and
  - hardware (equipment + base software) to be configured
- for the domain application.

4.1.2. Requirements Prescription

- The core part of the requirements engineering of a computing application is the requirements prescription.
  - A requirements prescription tells us which parts of the domain are to be supported by ‘the machine’.
  - A requirements is to satisfy some goals.
  - Usually the goals cannot be prescribed in such a manner that they can served directly as a basis for software design.
  - Instead we derive the requirements from the domain descriptions and then argue (incl. prove) that the goals satisfy the requirements.
  - In this talk we shall not show the latter but shall show the former.

4.1.3. A Suitable Decomposition of the Requirements Prescription

- We consider three forms of requirements prescription:
  - the domain requirements,
  - the interface requirements and
  - the machine requirements.
- Recall that the machine is the hardware and software (to be required).
  - Domain requirements are those whose technical terms are from the domain only.
  - Machine requirements are those whose technical terms are from the machine only.
  - Interface requirements are those whose technical terms are from both.

4.1.4. An Aside on Our Example

- We shall continue our “ongoing” example.
- Our requirements is for a toll-road system.
- The goals of having a toll-road system are:
  - to decrease transport times between selected hubs of a general net; and
  - to decrease traffic accidents and fatalities while moving on the toll-road net as compared to comparable movements on the general net.

- The toll-road net, however, must be paid for by its users.
  - Therefore toll-road net entries and exits occur at toll-road plazas
  - with these plazas containing entry and exit toll-booths
  - where tickets can be issued, respectively collected and travel paid for.
- We shall very briefly touch upon these toll-booths, in the Extension part (as from Slide 102) below.
- So all the other parts of the next section (Sect. on page 89) serve to build up to the Extension part.

4.2. Business Process Re-engineering (BPR)

- Before embarking on the detailed elaboration of requirements
  - it is advised that a thorough, rough-sketching of the re-engineering of the business processes take place.
    - ∗ A toll-road system is a special net consisting of a linear sequence of toll-road links separated by toll-road hubs.
    - ∗ Vehicles gain access to these hubs and links by entering (and leaving) the toll-road net at toll plazas, through entry (respectively exit) booths connected to the toll-road hubs by plaza to toll-road hub hubs.
    - ∗ Vehicles collect tickets upon entering the toll-road net.
    - ∗ Vehicles move around the toll-road hubs and links.
    - ∗ And vehicles return tickets and pay for using the toll-road net upon leaving that net.

4.3. Domain Requirements

- Domain requirements cover all those aspects of the domain —
  - simple entities,
  - actions,
  - events and
  - behaviours —
- which are to be supported by 'the machine'.

- Thus domain requirements are developed by systematically “revising” cum “editing” the domain description:
  - which parts are to be **projected:** left in or out;
  - which general descriptions are to be **instantiated** into more specific ones;
  - which non-deterministic properties are to be made more **determinate**; and
  - which parts are to be **extended** with such computable domain description parts which are not feasible without IT.

- Projection, instantiation, determination and extension are the basic engineering tasks of domain requirements engineering.
- An example may best illustrate what is at stake.
- The example is that of a toll-way system — in contrast to the general nets covered by description Items 1(a)–22 (Slides 8–23).
- See Fig. 1.

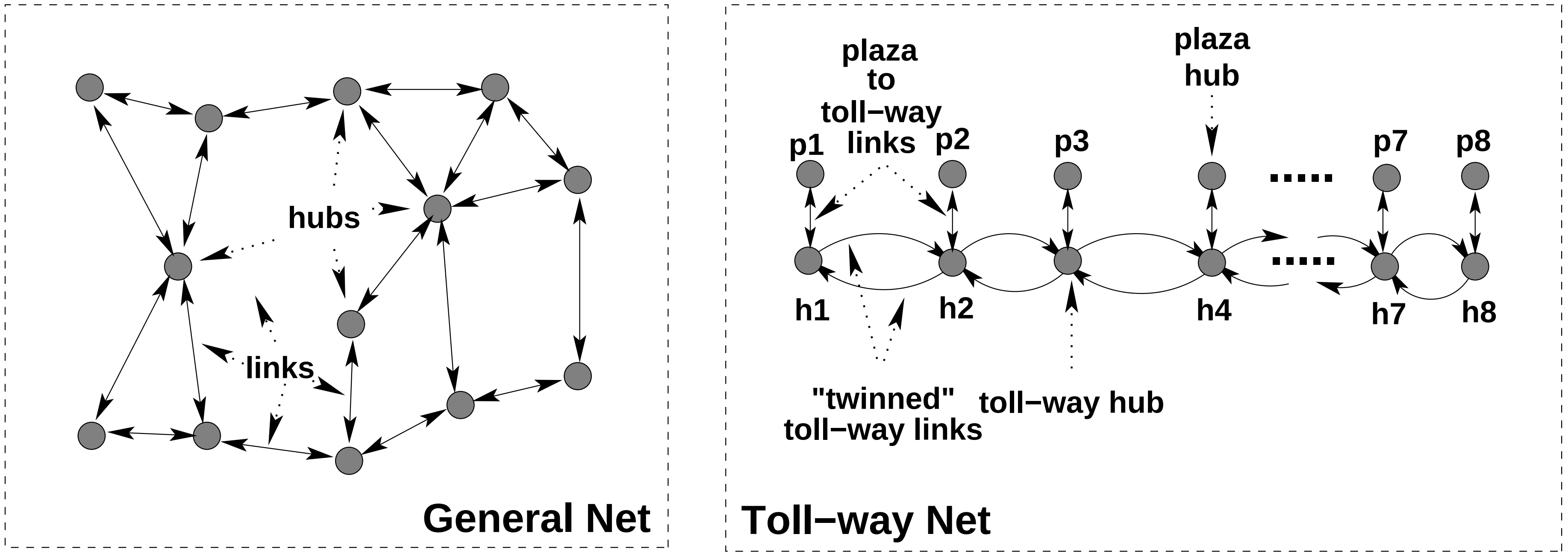


Figure 1: General and Toll-way Nets

4.3.1. Projection

We keep what is needed to prescribe the toll-road system and leave out the rest.

91. We keep the description, narrative and formalisation,
    (a) nets, hubs, links,
    (b) hub and link identifiers,
    (c) hub and link states,
92. as well as related observer functions.

type
91(a). N, H, L
91(b). HI, LI
91(c). HΣ, LΣ
value
92. obs_Hs,obs_Ls,obs_HI,obs_LI,
92. obs_HIs,obs_LIs,obs_HΣ,obs_L Σ

4.3.2. Instantiation

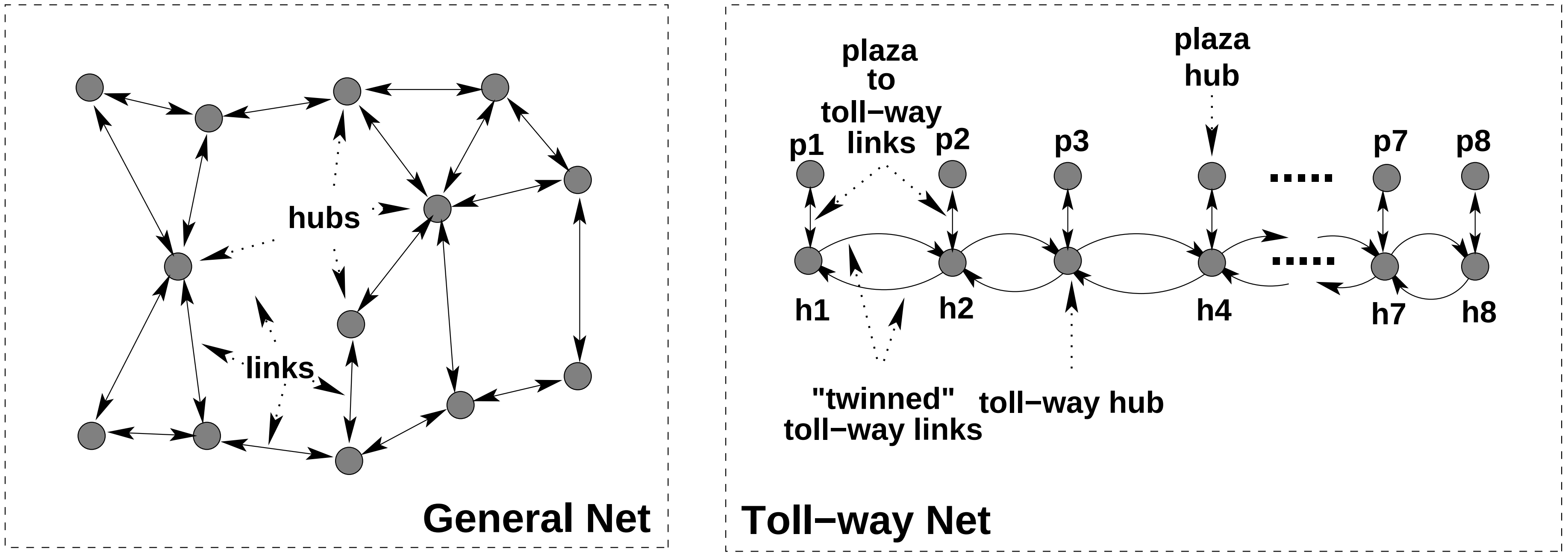


Figure 2: General and Toll-way Nets

- From the general net model of earlier formalisations
- we instantiate the toll-way net model now described.

93. The net is now concretely modelled as a pair of sequences.
94. One sequence models the plaza hubs, their plaza-to-toll-way link and the connected toll-way hub.
95. The other sequence models the pairs of "twinned" toll-way links.
96. From plaza hubs one can observe their hubs and the identifiers of these hubs.
97. The former sequence is of m such plaza "complexes" where $m \geq 2$; the latter sequence is of $m - 1$ "twinned" links.
98. From a toll-way net one can abstract a proper net.
99. One can show that the posited abstraction function yields well-formed nets, i.e., nets which satisfy previously stated axioms.

type

93. $\mathsf{TWN} = \mathsf{PC}^* \times \mathsf{TL}^*$
94. $\mathsf{PC} = \mathsf{PH} \times \mathsf{L} \times \mathsf{H}$
95. $\mathsf{TL} = \mathsf{L} \times \mathsf{L}$

value

94. $\mathsf{obs\_H}: \mathsf{PH} \rightarrow \mathsf{H}, \mathsf{obs\_HI}: \mathsf{PH} \rightarrow \mathsf{HI}$

axiom

97. $\forall\ (\mathsf{pcl},\mathsf{tll}):\mathsf{TWN} \cdot$
97. $\quad 2 \leq \mathbf{len}\ \mathsf{pcl} \wedge \mathbf{len}\ \mathsf{pcl} = \mathbf{len}\ \mathsf{tll}+1$

value

98. $\mathsf{abs\_N}: \mathsf{TWN} \rightarrow \mathsf{N}$
98. $\mathsf{abs\_N}(\mathsf{pcl},\mathsf{tll})\ \mathbf{as}\ \mathsf{n}$
98. $\quad \mathbf{pre}\text{:}\ \mathsf{wf\_TWN}(\mathsf{pcl},\mathsf{tll})$
98. $\quad \mathbf{post}\text{:}$
98. $\qquad \mathsf{obs\_Hs}(\mathsf{n}) =$
98. $\qquad\quad \{\mathsf{h},\mathsf{h}'|(\mathsf{h},\_,\mathsf{h}'):\mathsf{PC}\cdot(\mathsf{h},\_,\mathsf{h}') \in \mathbf{elems}\ \mathsf{pcl}\} \wedge$
98. $\qquad \mathsf{obs\_Ls}(\mathsf{n}) =$
98. $\qquad\quad \{\mathsf{l}|(\_,\mathsf{l},\_):\mathsf{PC}\cdot(\_,\mathsf{l},\_) \in \mathbf{elems}\ \mathsf{pcl}\} \cup$
98. $\qquad\quad \{\mathsf{l},\mathsf{l}'|(\mathsf{l},\mathsf{l}'):\mathsf{TL}\cdot(\mathsf{l},\mathsf{l}') \in \mathbf{elems}\ \mathsf{tll}\}$

theorem:

99. $\forall\ \mathsf{twn}:\mathsf{TWN} \cdot \mathsf{wf\_TWN}(\mathsf{twn}) \Rightarrow \mathsf{wf\_N}(\mathsf{abs\_N}(\mathsf{twn}))$

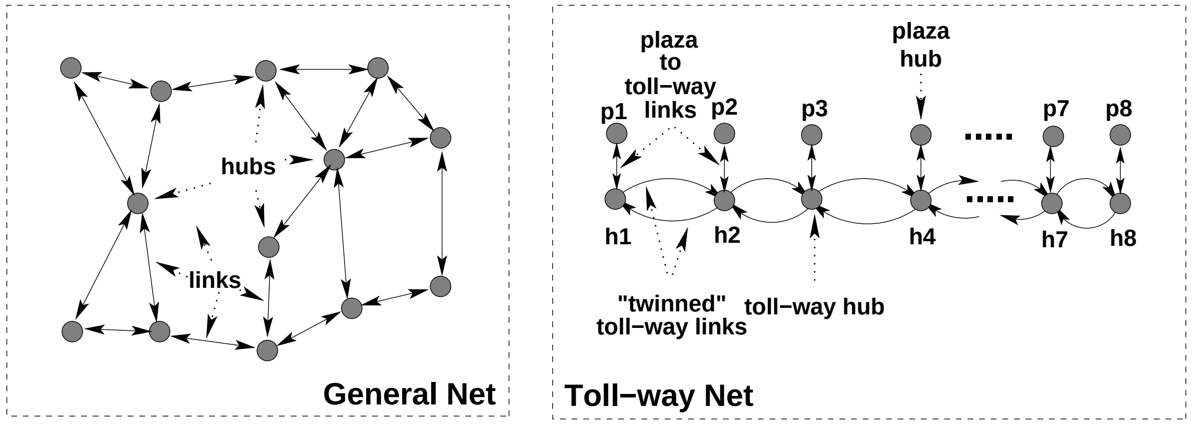


Figure 3: General and Toll-way Nets

4.3.2.1. Model Well-formedness wrt. Instantiation

- Instantiation restricts general nets to toll-way nets.
- Well-formedness deals with proper mereology: that observed identifier references are proper.
- The well-formedness of instantiation of the toll-way system model can be defined as follows:

100. The i'plaza complex, (p_i, l_i, h_i), is instantiation-well-formed if
    (a) link l_i identifies hubs p_i and h_i, and
    (b) hub p_i and hub h_i both identifies link l_i; and if

101. the i'th pair of twinned links, tl_i, tl'_i,
    (a) has these links identify the toll-way hubs of the i'th and $i{+}1$'st plaza complexes $((p_i, l_i, h_i)$ respectively $(p_{i+1}, l_{i+1}, h_{i_1}))$.

```
value
  Instantiation_wf_TWN: TWN → Bool
  Instantiation_wf_TWN(pcl,tll) ≡
100.     ∀ i:Nat • i ∈ inds pcl⇒
100.       let (pi,li,hi)=pcl(i) in
100(a).      obs_LIs(li)={obs_HI(pi),obs_HI(hi)}
100(b).      ∧ obs_LI(li)∈ obs_LIs(pi)∩ obs_LIs(hi)
101.       ∧ let (li′,li″) = tll(i) in
101.         i < len pcl ⇒
101.           let (pi′,li‴,hi′) = pcl(i+1) in
101(a).          obs_HIs(li) = obs_HIs(li′) = {obs_HI(hi),obs_HI(hi′)}
             end end end
```

4.3.3. Determination

- Determination, in this example, fixes states of hubs and links.
- The state sets contain only one set.
  - Twinned toll-way links allow traffic only in opposite directions.
  - Plaza to toll-way hubs allow traffic in both directions.
  - Toll-way hubs allow traffic to flow freely from
    - ∗ plaza to toll-way links
    - ∗ and from incoming toll-way links
    - ∗ to outgoing toll-way links
    - ∗ and toll-way to plaza links.
- We omit formalisation.
- The determination-well-formedness of the toll-way system model can be defined as follows[6]:

[6] i ranges over the length of the sequences of twinned toll-way links, that is, one less than the length of the sequences of plaza complexes. This "discrepancy" is reflected in out having to basically repeat formalisation of both Items 103(a) and 103(b).

4.3.3.1. Model Well-formedness wrt. Determination

- We need define well-formedness wrt. determination.
- Please study Fig. 4.

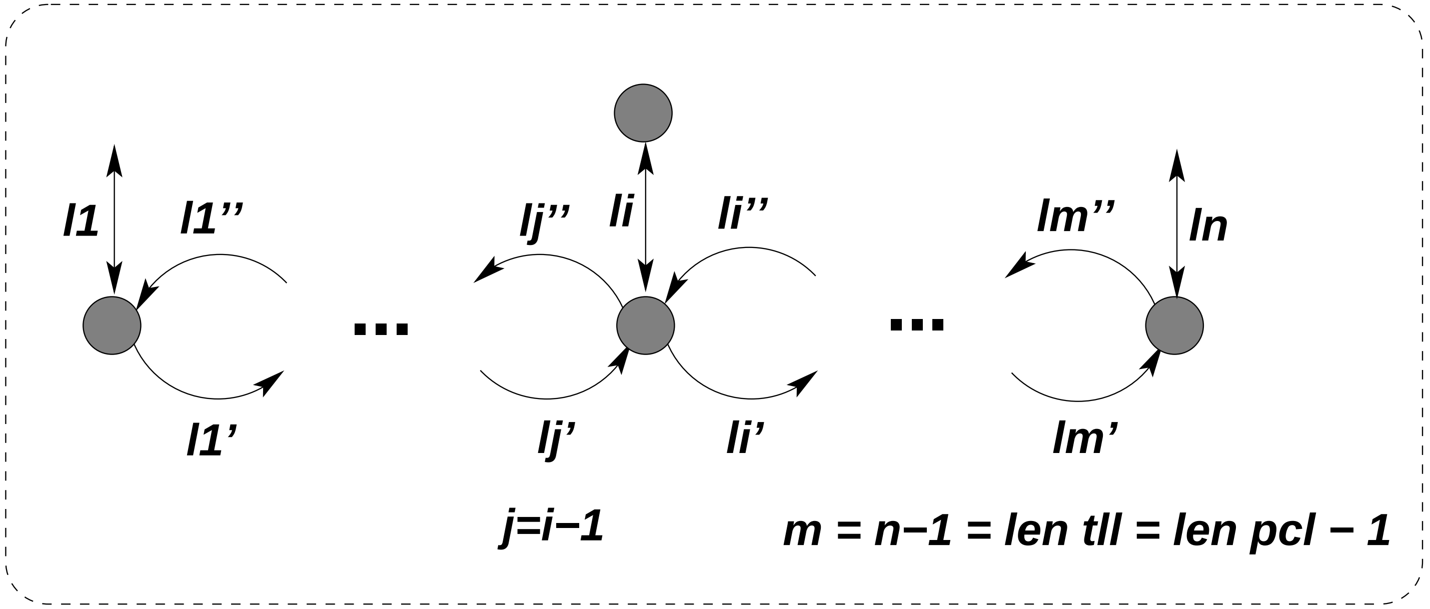


Figure 4: Hubs and Links

102. All hub and link state spaces contain just one hub, respectively link state.

103. The i'th plaza complex, pcl(i):(p_i, l_i, h_i) is determination-well-formed if

    (a) l_i is open for traffic in both directions and

    (b) p_i allows traffic from h_i to "revert"; and if

104. the i'th pair of twinned links (li', li'') (in the context of the i+1st plaza complex, pcl(i+1):$(p_{i+1}, l_{i+1}, h_{i+1})$) are determination-well-formed if

    (a) link l_i' is open only from h_i to h_{i+1} and

    (b) link l_i'' is open only from h_{i+1} to h_i; and if

105. the jth toll-way hub, h_j (for $1 \leq\ j \leq \mathbf{len}\,$pcl) is determination-well-formed if, depending on whether j is the first, or the last, or any "in-between" plaza complex positions,

    (a) [the first:] hub $i = 1$ allows traffic in from l_1 and l_1'', and onto l_1 and l_1'.

    (b) [the last:] hub $j = i + 1 = \mathbf{len}\,$pcl allows traffic in from $l_{\mathbf{len}\,\mathsf{tll}}$ and $l''_{\mathbf{len}\,\mathsf{tll}-1}$, and onto $l_{\mathbf{len}\,\mathsf{tll}}$ and $l'_{\mathbf{len}\,\mathsf{tll}-1}$.

    (c) [in-between:] hub $j = i$ allows traffic in from l_i, l_i'' and l_i' and onto l_i, l_{i-1}' and l_i''.

```
value
103.    Determination_wf_TWN: TWN → Bool
103.    Determination_wf_TWN(pcl,tll) ≡
103.      ∀ i:Nat• i ∈ inds tll ⇒
103.        let (pi,li,hi) = pcl(i),
103.            (npi,nli,nhi) = pcl(i+1), in
103.            (li′,li″) = tll(i) in
102.           obs_HΩ(pi)={obs_HΣ(pi)}∧obs_HΩ(hi)={obs_HΣ(hi)}
102.       ∧ obs_LΩ(li)={obs_LΣ(li)}∧obs_LΩ(li′)={obs_LΣ(li′)}
102.       ∧ obs_LΩ(li″)={obs_LΣ(li″)}
103(a).     ∧ obs_LΣ(li)
103(a).          = {(obs_HI(pi),obs_HI(hi)),(obs_HI(hi),obs_HI(pi))}
103(a).     ∧ obs_LΣ(nli)
103(a).          = {(obs_HI(npi),obs_HI(nhi)),(obs_HI(nhi),obs_HI(npi))}
103(b).     ∧ {(obs_LI(li),obs_LI(li))}⊆obs_HΣ(pi)
103(b).     ∧ {(obs_LI(nli),obs_LI(nli))}⊆obs_HΣ(npi)
104(a).     ∧ obs_LΣ(li′)={(obs_HI(hi),obs_HI(nhi))}
104(b).     ∧ obs_LΣ(li″)={(obs_HI(nhi),obs_HI(hi))}
105.       ∧ case i+1 of
105(a).         2 → obs_HΣ(h_1)=
105(a).                 {(obs_LΣ(l_1),obs_LΣ(l_1)),
105(a).                  (obs_LΣ(l_1),obs_LΣ(l_1″)),
105(a).                  (obs_LΣ(l″_1),obs_LΣ(l_1)),
105(a).                  (obs_LΣ(l″_1),obs_LΣ(l′_1))},
105(b).         len pcl → obs_HΣ(h_i+1)=
105(b).                 {(obs_LΣ(l_len pcl),obs_LΣ(l_len pcl)),
105(b).                  (obs_LΣ(l_len pcl),obs_LΣ(l′_len tll)),
105(b).                  (obs_LΣ(l″_len tll),obs_LΣ(l_len pcl)),
105(b).                  (obs_LΣ(l″_len tll),obs_LΣ(l′_len tll))},
105(c).         _ → obs_HΣ(h_i)=
105(c).                 {(obs_LΣ(l_i),obs_LΣ(l_i)),
105(c).                  (obs_LΣ(l_i),obs_LΣ(l′_i)),
105(c).                  (obs_LΣ(l_i),obs_LΣ(l″_i−1)),
105(c).                  (obs_LΣ(l″_i),obs_LΣ(l′_i)),
105(c).                  (obs_LΣ(l″_i),obs_LΣ(l′_i−1)),
105(c).                  (obs_LΣ(l″_i),obs_LΣ(l′_i))}
103.      end end
```

4.3.4. Extension

- For our example we choose to consider the toll plazas.
- A toll plaza,
  - in addition to its hub,
  - also contains vehicle
    - entry and
    - exit

    booths.
- We refer to Fig. 5 on the next page.

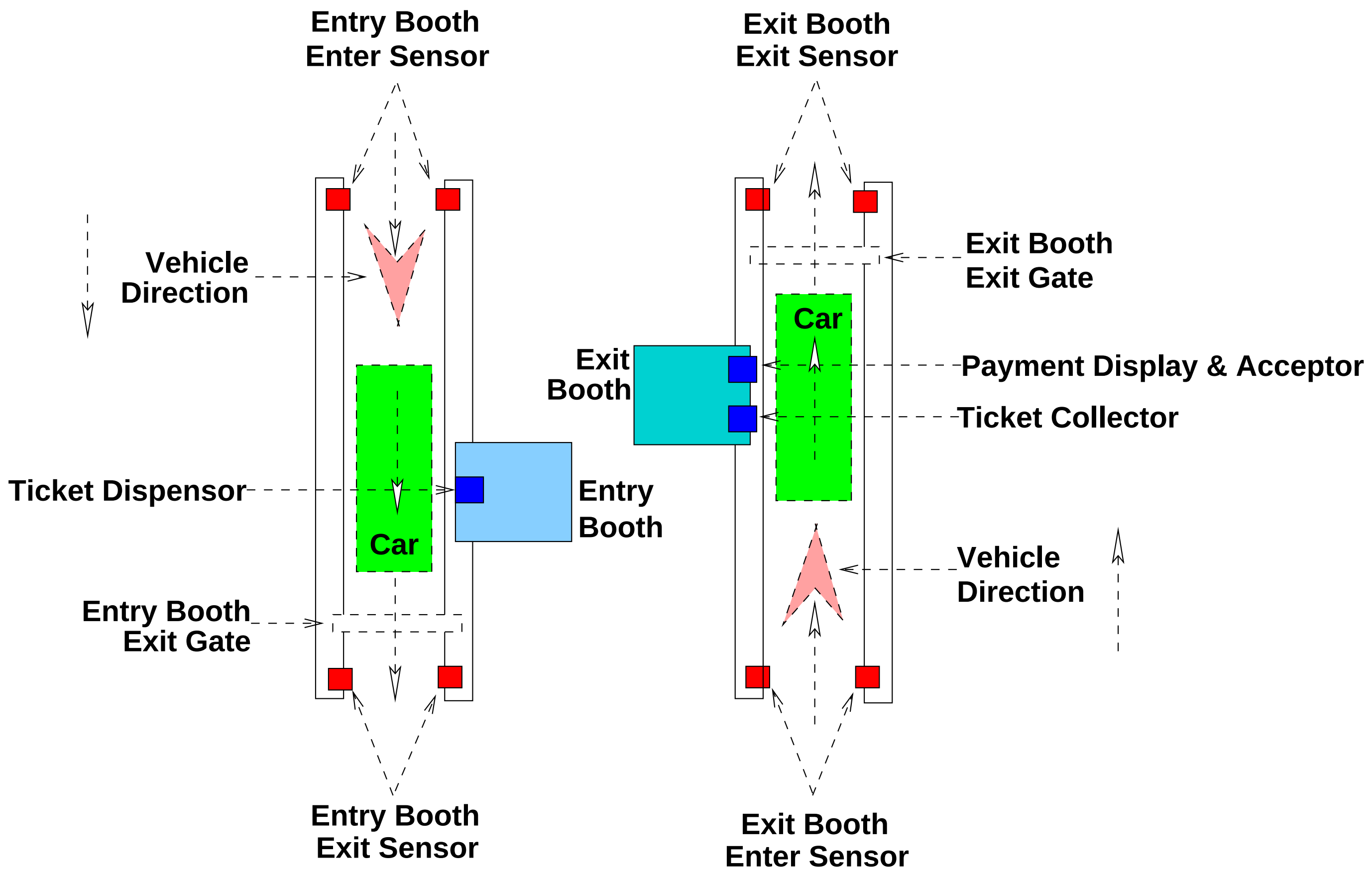


Figure 5: Entry and Exit Toll Boths

- The following is a prolonged example.
- It contains three kinds of formalisations:
  - a `RAISE/CSP` model,
  - a `Duration Calculus` model [zcc+mrh2002,olderogdirks2008] and
  - a `Timed Automata` model [AluDil:94,olderogdirks2008].

4.3.4.0.1. • *A* RAISE/CSP *Model* •

Without much ado:

106. A toll plaza consists of a one pair of an entry booth and and entry gate and one pair of an exit booth and an exit gate.
107. Entry booths consist of an entry sensor, a ticket dispenser and an exit sensor.
108. Exit booths consist of an entry sensor, a ticket collector, a payment display and a payment component.

type

106. $\mathrm{PZ} = (\mathrm{EB}\times\mathrm{G}) \times (\mathrm{XB}\times\mathrm{G})$

107. $\mathrm{EB} = \ldots$

108. $\mathrm{XB} = \ldots$

4.3.4.0.2. • *Cars* •

We summarize an earlier model of vehicles:

109. There are vehicles.

110. Vehicles have unique vehicle identifications.

type
109. V
110. VId
value
110. obs_VId: $V \rightarrow VId$
axiom
110. $\forall$ v,v′:V $\cdot$ $v \neq v' \Rightarrow obs\_VId(v) \neq obs\_VId(v')$

4.3.4.0.3. • *Entry Booths* •

The description now given is an idealisation. It assumes that everything works: that the vehicles behave as expected and that the electro-mechanics of booths and gates do likewise.

111. An entry_sensor registers whether a car is entering the entry booth or not,
    (a) that is, for the duration of the car passing the entry_sensor that sensor senses the car identification cid
    (b) otherwise it senses "nothing".

112. A ticket_dispenser
    (a) either holds a ticket or does not hold a ticket, i.e., no_ticket;
    (b) normally it does not hold a ticket;
    (c) the ticket_dispenser holds a ticket soon after a car has passed the entry_sensor;
    (d) the passing car collects the ticket –
    (e) after which the ticket_dispenser no longer holds a ticket.

113. An exit_sensor
    (a) registers the identification of a car leaving the toll booth
    (b) otherwise it senses "nothing".

⋆ *Gates* ⋆

As part of entry/exit booths:

114. A gate
   (a) is either closed or open;
   (b) it is normally closed;
   (c) if a car is entering it is secured set to close (as a security measure);
   (d) once a car has collected a ticket it is set to open;
   (e) and once a car has passed the exit_sensor it is again set to close.

⋆ *A Simple Formalisation* ⋆

```
type
  C, CI
  G = open | close
  TK == Ticket | no_ticket
value
  obs_CI: (C|Ticket) → CI
channel
  entry_sensor:CI
  ticket_dispenser:Ticket
  exit_sensor:CI
  gate_ch:G
value
  vs:V-set
  eb:EB,xb:XB,eg,xg:G
```

```
value
  eg,xg:G, eb:EB, xb:XB, vs:V-set

  system: G×EV×V-set×XB×G → Unit
  system(eg,eb,vs,xb,xg) ≡
     entry_gate(eg)
     ‖ entry_booth(eb)
     ‖ ‖{car(obs_CId(c),c)|ci:C,v:C•c ∈ cs}
     ‖ exit_booth(xb)
     ‖ exit_gate(xg)
```

```
car: CI × C → out entry_sensor,exit_sensor
          in ticket_dispenser  Unit
car(ci,c) ≡
   entry_sensor ! ci ;
   let ticket = ticket_dispenser ? assert: ticket ≠ no_ticket in
   ticket_dispenser ! no_ticket ;
   exit_sensor ! ci ;
   car(add(ticket,c)) end
```

```
entry_booth: Unit → in entry_sensor, exit_sensor
                  out ticket_dispenser
                  out gate_ch  Unit
entry_booth(b) ≡
   gate_ch ! close ;
   let ci = entry_sensor ? in
   gate_ch ! open ;
   ticket_dispenser ! make_ticket(cid) ;
   let res = ticket_dispenser ? assert: res = no_ticket ;
   let ci′ = exit_sensor ? assert: ci′ = ci ;
   gate_ch ! close ;
   entry_booth(add_ticket(ticket,b)) end end end
```

```
entry_gate: G → in gate Unit
entry_gate(g) ≡
    case gate_ch ? of
        close → exit_gate(close) assert: g = open,
        open → exit_gate(open) assert: g = close
    end
```

add_ticket: Ticket $\times$ C $\overset{\sim}{\to}$ C
  pre add_Ticket(t,c): ∼has_Ticket(c)
  post: add_Ticket(t,c): has_Ticket(c)
has_ticket: (C|B) $\to$ **Bool**
obs_ticket: (C|B) $\overset{\sim}{\to}$ Ticket
  pre obs_ticket(cb): has_Ticket(cb)
rem_ticket: (C $\overset{\sim}{\to}$ C) | (B $\overset{\sim}{\to}$ B)
  pre rem_ticket(cb): has_Ticket(cb)
  post rem_ticket(cb): ∼has_Ticket(cb)

In the next section, “A `Duration Calculus` Model” we shall start refining the descriptions given above. We do so in order to handle failures of vehicles to behave as expected and of the electro-mechanics of booths and gates.

4.3.4.0.6. • *A* `Duration Calculus` *Model* •

- We abstract the channels of the `RAISE/CSP` model to now be Boolean-valued variables.

type
  ES = **Bool** [**true**=passing, **false**=not_passing]
  TD = **Bool** [**true**=ticket, **false**=no_ticket]
  G = **Bool** [**true**=open, **false**=closing⌈⌉closed⌈⌉opening]
  XS = **Bool** [**true**=car_has_just_passed, **false**=car_passing⌈⌉no-one_passing
variable
  entry_sensor:ES := **false** ;
  ticket_dispenser:TD := **false** ;
  gate:G := **false** ;
  exit_sensor:XS := **false** ;

115. No matter its position, the **gate** must be **closed** within no more than δ_{eg} time units after the **entry_sensor** has registered that a car is entering the toll booth.
116. A ticket must be in the **ticket_dispenser** within δ_{et} time units after the **entry_sensor** has registered that a car is entering the toll booth.
117. The ticket is in the **ticket_dispenser** at most δ_{tdc} time units
118. The **gate** must be **open** within δ_{go} time units after a ticket has been collected.
119. The exit sensor is registering (i.e., is on) the identification of exiting cars and is not registering anything when no car is passing (i.e., is off).

115. $\sim(\lceil \text{entry\_sensor} \rceil \ ; \ (\ell = \delta_{eg} \wedge \lceil \text{gate} \rceil))$
116. $\sim(\lceil \text{entry\_sensor} \rceil \ ; \ (\ell = \delta_{et} \wedge \lceil \sim\text{ticket\_dispenser} \rceil))$
117. $\Box(\lceil \sim\text{ticket\_dispenser} \rceil \Rightarrow \ell < \delta_{tdc})$
118. $\sim(\lceil \text{ticket\_dispenser} \rceil \ ; \ (\lceil \sim\text{ticket\_dispenser} \wedge \sim\text{gate} \rceil \wedge \ell \geq \delta_{go}))$
119. $\Box(\lceil \text{gate=closing} \rceil \Rightarrow \lceil \sim \text{exit\_sensor} \rceil)$

4.3.4.0.7. • *A* Timed Automata *Model* •

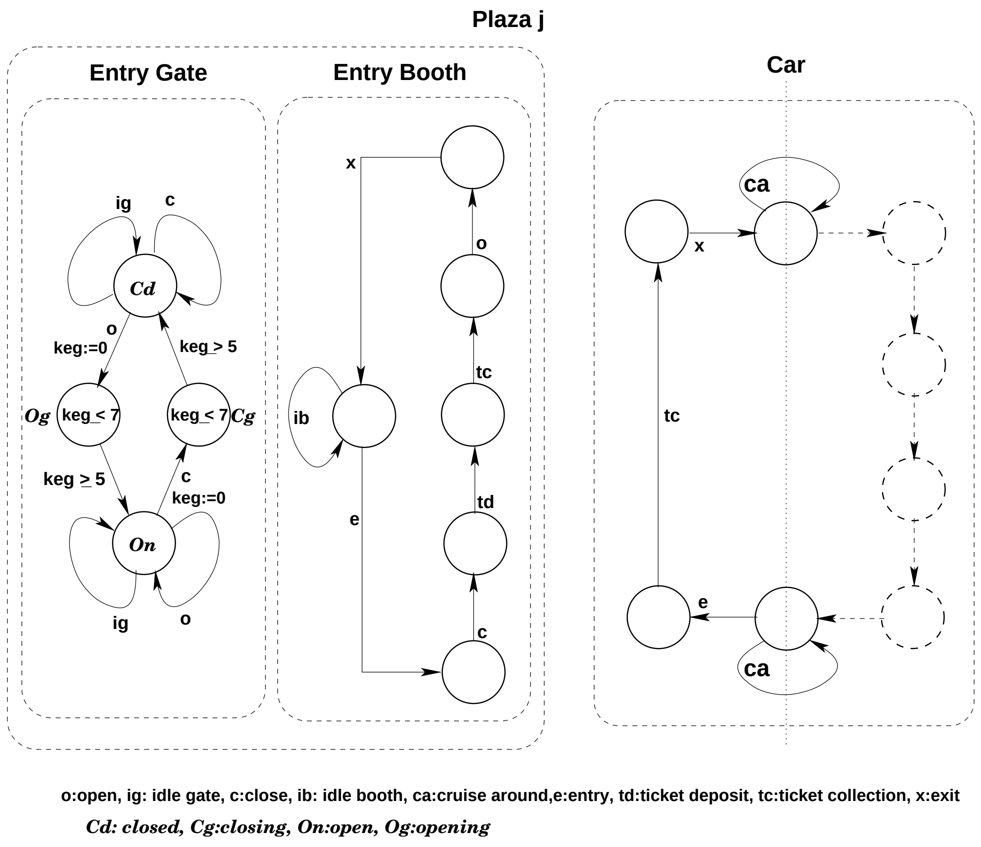
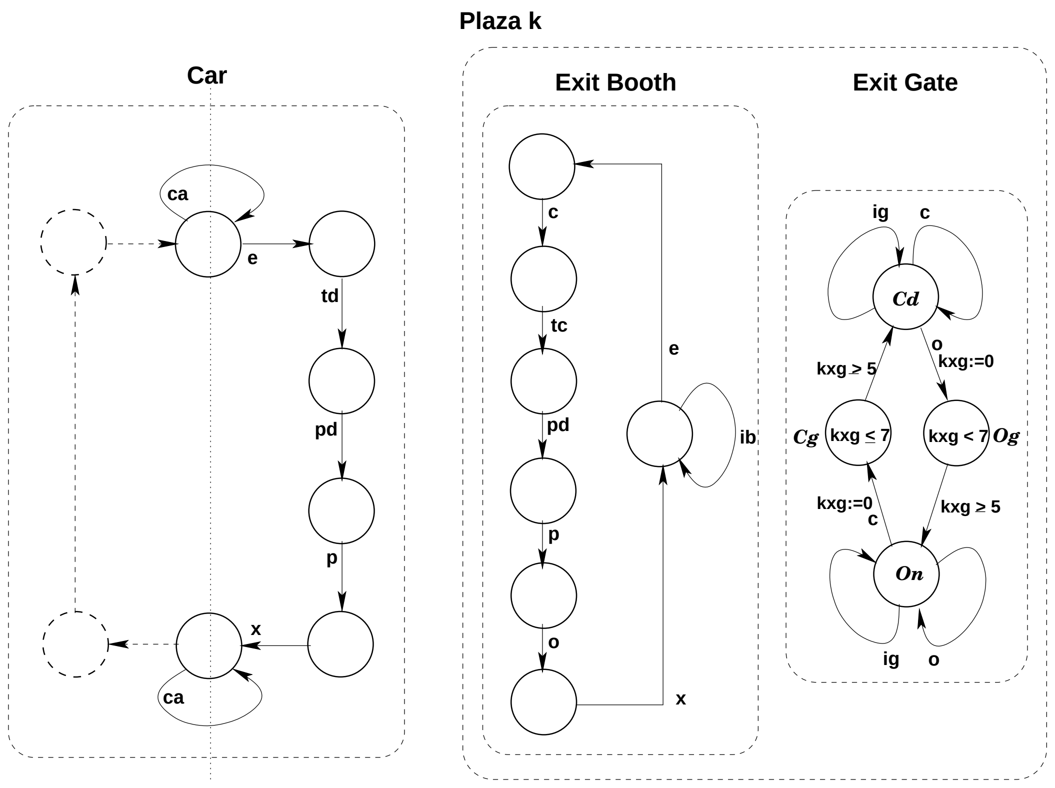

Plaza j
Entry Gate
Entry Booth
Car
ig
c
Cd
o
keg:=0
keg ≥ 5
Og
keg ≤ 7
keg ≤ 7
Cg
keg ≥ 5
c
keg:=0
On
ig
o
x
o
tc
ib
td
e
c
ca
x
tc
e
ca
o:open, ig: idle gate, c:close, ib: idle booth, ca:cruise around,e:entry, td:ticket deposit, tc:ticket collection, x:exit
Cd: closed, Cg:closing, On:open, Og:opening

Figure 6: A `timed automata` model of gate, entry booth and car interactions



ca:cruise around, ib:idle, e:entry, td:ticket deposit, pd:payment display, p: payment, x:exit, c:close, o:open, ig:idle gate

Figure 7: A `timed automata` model of car, exit booth and gate interactions

4.4. Interface and Machine Requirements

- Interface requirements take into consideration both
  - the domain description and
  - the machine:
    - the hardware + base systems software
    - upon which the software to be designed

    is to be implemented.
- So interface requirements are not exclusively “derived” from the narrated and formalised domain description.
- And the machine requirements
  - make hardly any concrete reference to the domain description.

4.5. Discussion of Requirements Engineering

- As was the case for our coverage of domain engineering, there is more to requirements engineering than shown in this talk !

5. Software Design

- We shall likewise omit any serious coverage of the software design process — except for these remarks:
  - As the domain description serves as a model for the development of the requirements development,
  - so do the requirements prescription serve as a model for software design; that is,
  - our whole software development is model-oriented.

6. Concluding Remarks

- We have over-viewed the TRIPTYCH approach to software development:
  - core aspects of domain engineering and
  - core aspects of requirements.
- The conclusions that one may be able to draw from the example —
  - or at least a reasonable small number of such examples —

  are

  - that domains can be described, informally as well as formally;
  - that domain requirements can be systematically (but not automatically) "derived" from domain descriptions; and
  - that **this approach puts requirements engineering in a rather new light.**

6.1. Domain Models as a Prerequisite for RE

- Domains are seldomly computable, requirements must always be.
- It has been suggested — and strongly so — that
  - requirements engineering be based on a domain description
  - covering at least the "area of requirements interest".
- We have not covered 'Interface Requirements',
  - those aspects of requirements which can be expressed using terms from both the domain and the machine — there shared entities,
  - but, again, a concise domain model would help significantly
  - we claim.
- And we have also not covered 'Machine Requirements',
  - those aspects of requirements which can be expressed using terms from just the machine,
  - so here domain models do not contribute much.
- So I suggest that we revise research into and practice of RE.

6.2. Oh Yes, Conventional RE Contains Elements of DE

- Indeed, most RE texts contain repeated references to the necessity of considering "the domain".
- But these "necessity references"
  - do not require that the requirements engineer separately model the domain,
  - do not really expect the requirements engineer to go well beyond the scope and span of the requirements when considering the domain, and
  - do not formally relate domains and requirements.
- Here, we are strongly suggesting that domains
  - be understood,
  - be described (informally and formally)
  - independent of requirements considerations.

6.3. Domain Engineering as a Free-standing Activity

- Aim is to just understand a domain:
  - *“What is a container line industry”*
  - *“What is a railway system”*
  - *“What is a hospital”*
  - *“What is a financial service industry”*
- Just like
  - a physicist try understand *“the big bang”*,
  - an economist try understand *a country’s national debt process*,
  - a biologist try understand some *aspects of evolution.*
- To create a domain model, to study it and make it ready for general use make take 10-20 years.
  - It took physicists many years before their theory of matter could be applied.
  - But that is no reason for not doing domain engineering and science.

6.4. Domain Theories

- By a domain theory we shall understand a theory about the model of the domain as described.
  - A domain description is a foundation for a theory.
  - The proof system of the formal specification language in which the domain description is expressed is another foundation.
  - Theorems derived from these two foundations contribute to the theory.

- An examples of a domain theorem for railways could be:
  - Assuming that train traffic is on time wrt. a train timetable we can expect the following to hold:
  - given that a train timetable is modulo some time interval,
  - then the # of trains arriving at a station
  - minus the # of trains ending their journey at that station
  - plus the # of trains starting their journey at that station
  - equals the # of trains leaving that station.
- Domain models should be aimed at establishing domain theories.

6.5. Domain Science

- By domain science
  - we understand the theoretical foundation
  - specific to the engineering of domain descriptions.
- Examples of issues of domain science are:
  - (i) a theory of a calculus of domain description constructors
    * such as illustrated in tomorrow's speculative talk;
  - (ii) a theory of mereology models,
    * cf. my April 2009 [TonyHoare75thBirthday] paper:
    * for every $\mathcal{M}$ereology there is "a correspondning CSP$_{\mathcal{M}}$ expression", and
    * for every CSP$_{\mathcal{M}}$ expression there is "a corresponding $\mathcal{M}$ereology".
  - (iii) etcetera.

Thanks !

Questions ?

/home/db/2011/swansea/DEflawedRE-s.tex