Domain Science & Engineering

A New Facet of Informatics

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.

2. Domain Engineering

- We choose as our example domain that of transportation systems, $\delta{:}\Delta$.
  - From any such δ we can observe (obs_) a number of
    - simple entities Slides 7–45,
    - actions Slides 46–55,
    - events Slides 56–61,
    - and behaviours Slides 62–80.
  - 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.

2.1. Transport Simple Entities

1. There are five classes of simple entities in our example:
   (a) transportation nets — Slides 8–23,
   (b) people — Slides 24–28,
   (c) vehicles, — Slides 29–35,
   (d) time, — Slides 37–39, and
   (e) timetables, — Slides 40–45.

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

2.1.1. Transportation Nets

2.1.1.1. 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).

2.1.1.2. 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. γetH: N → HI $\xrightarrow{\sim}$ H
7. γetH(n)(hi) ≡
7.   **if** hi ∈ χtrHIs(n)
7.   **then let** h:H • mer_HI(h)=hi **in** h **end**
7.   **else chaos end**
8. γetL: N → LI $\xrightarrow{\sim}$ L
8. γetL(n)(li) ≡
8.   **if** li ∈ χtrLIs(n)
8.   **then let** l:L • mer_LI(l)=li **in** l **end**
8.   **else chaos end**

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

2.1.1.3. 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 $\rightarrow$ LI-**set**
10. mer_HIs: L $\rightarrow$ HI-**set**

axiom

9. $\forall$ n:N,h:H,l:L•h $\in$ obs_Hs(n)$\wedge$l $\in$ obs_Ls(n) $\Rightarrow$
10. **card** mer_HIs(l)=2
11. $\wedge$ $\forall$ li:LI • li $\in$ mer_LIs(h) $\Rightarrow$ li $\in$ χtrLIs(n)
12. $\wedge$ $\forall$ hi:HI • hi $\in$ mer_HIs(l) $\Rightarrow$ hi $\in$ χtrHIs(n)

2.1.1.4. 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. $\quad$ M = HI $\overrightarrow{m}$ (LI $\overrightarrow{m}$ HI)

axiom

13(a). $\quad \forall$ m:M,h_i:HI · h_i ∈ **dom** m ⇒

13(b). $\quad\quad$ **let** [l_ij↦h_j] = m(h_i) **in**

13(c). $\quad\quad$ h_j ∈ **dom** m ∧ m(h_j)=[l_ij↦h_i]

13(a). $\quad\quad$ **end**

14. From a net one can extract its map.

value

14. χtrM: N $\rightarrow$ M
14. χtrM(n) $\equiv$
14. [hi$\mapsto$[lij$\mapsto$hj|
14. lij:LI $\cdot$ lij $\in$ mer_LIs(γetH(n)(hi))
14. $\wedge$ hj = γetL(n)(lij)$\backslash$\{hi\}] |
14. hi:HI$\cdot$hi $\in$ χtrHIs(n)]

2.1.1.5. 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
15. $R' = (LI|HI)^*$
15. $R = \{|r:R'\bullet\exists\ n:N\bullet wf\_R(r)(n)|\}$
value
15. wf_R: $R' \to N \to$ **Bool**
15. wf_R(r)(n) $\equiv$ proper_adjacency(r) $\wedge$ embedded_route(r)(n)

15. proper_adjacency: $R' \to$ **Bool**
15. proper_adjacency(r) $\equiv$
15. $\forall$ i:**Nat**$\bullet$\{i,i+1\}$\subseteq$**inds** r$\Rightarrow$is_LI(r(i))$\wedge$is_HI(r(i+1))$\vee$is_HI(r(i))$\wedge$is_LI(r(i+1))

15. embedded_route: $R' \to N \to$ **Bool**
15. embedded_route(r)(n) $\equiv$
15. $\forall$ i:**Nat**$\bullet$\{i,i+1\}$\subseteq$**inds** r $\Rightarrow$
15. is_LI(r(i)) $\to$ r(i+1) $\in$ mer_HIs(γetL(r(i))(n)),
15. is_HI(r(i)) $\to$ r(i+1) $\in$ 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 ⟨li⟩ is a route of the net;
    (b) if hi is an identifier of a hub of a net then ⟨hi⟩ 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^tl r′ is a route of the net.
    (d) Only such routes which can be constructed by applying rules 16(a)–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

16. χtrRs: N $\rightarrow$ R-**set**
16. χtrRs(n) $\equiv$ **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**
16. rs **end**

17. χtrRs: M $\rightarrow$ 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). $\mathrm{IM}' = \mathbf{Nat} \xrightarrow[m]{} \mathbf{Nat}$
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). $\quad \wedge \forall\ \mathrm{i}:\mathbf{Nat} \cdot \{\mathrm{i},\mathrm{i}+1\} \subseteq \mathbf{dom}\ \mathrm{im} \Rightarrow \mathrm{im}(\mathrm{i}) \leq \mathrm{im}(\mathrm{i}+1)$

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

2.1.1.6. 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. $H\Sigma = (LI \times LI)$**-set**
20. $L\Sigma = (HI \times HI)$**-set**
21. $H\Omega = H\Sigma$**-set**
22. $L\Omega = L\Sigma$**-set**

value

19. $\alpha\tau r H\Sigma$: $H \rightarrow H\Sigma$
20. $\alpha\tau r L\Sigma$: $L \rightarrow L\Sigma$
21. $\alpha\tau r H\Omega$: $H \rightarrow H\Omega$
22. $\alpha\tau r L\Omega$: $L \rightarrow L\Omega$

axiom

$\forall$ n:N, h:H, l:L • h $\in$ obs_Hs(n)$\wedge$l $\in$ obs_Ls(n) $\Rightarrow$

19. **let** $h\sigma = \alpha\tau r H\Sigma(h)$,
20. $l\sigma = \alpha\tau r L\Sigma(l)$ **in**
19. $\forall$ (li,li′):(LI$\times$LI)•(li,li′)$\in$ h$\sigma$$\Rightarrow$$\{$li,li′$\}$$\subseteq$$\chi$trLIs(n)
20. $\wedge$ $\forall$ (hi,hi′):(HI$\times$HI)•(hi,hi′)$\in$ l$\sigma$$\Rightarrow$$\{$hi,hi′$\}$$\subseteq$$\chi$trHIs(n)
21. $\wedge$ $h\sigma \in \alpha\tau r H\Omega(h)$
22. $\wedge$ $l\sigma \in \alpha\tau r L\Omega(l)$ **end**

2.1.2. 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
24. PI

value

23. obs_Ps: C → P-**set**
24. $\alpha\tau r$PI: P → PI

axiom

25. $\forall$ p,p′:P • p≠p′ ⇒ $\alpha\tau r$PI(p) ≠ $\alpha\tau r$PI(p′)

type

26. PNm, PAn, PGd, PAg, PHe, PWe, ...

value

26(a). $\alpha\tau r$PNm: P → PNm
26(b). $\alpha\tau r$PAn: P → PAn
26(c). $\alpha\tau r$PGd: P → PGd
26(d). $\alpha\tau r$PAg: P → PAg
26(e). $\alpha\tau r$PHe: P → PHe
26(f). $\alpha\tau r$PWe: P → PWe

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

value

27. χtrPIs: P-**set** → PI-**set**

27. χtrPIs(ps) ≡ {obs_PI(p)|p:P•p ∈ ps}

axiom

27. ∀ ps:P-**set** · **card** ps = **card** χtrPIs(ps)

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

2.1.4. 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≠v′ ⇒ obs_VI(v)≠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. $B \subset V$
value
32. is_B: $V \rightarrow$ **Bool**
33. is_InTF: $V \rightarrow$ **Bool**
34. χtrVIs: V-**set** $\rightarrow$ VI-**set**
34. χtrVIs(vs) $\equiv$ {obs_VI(v)|v:V•v $\in$ vs}
axiom
34. $\forall$ vs:V-**set** • **card** vs = **card** χtrVIs(vs)

2.1.5. 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 $\stackrel{\sim}{\to}$ PI
35. **pre** $\alpha\tau r$Driver(v): is_InTF(v)
36. $\alpha\tau r$Pass: V $\to$ 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 $\to$ PI
37. $\alpha\tau r$Own: P $\to$ 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.

38. **axiom** $\forall$ δ:Δ,ps:P-**set**,vs:V-**set** · ps=obs_Ps(δ)$\wedge$vs=obs_Vs(δ) $\Rightarrow$
38(a). $\forall$ v:V · v $\in$ vs $\Rightarrow$ $\alpha\tau r$Owner(v) $\in$ χtrPIs(ps)
38(b). $\wedge$ $\forall$ p:P · p $\in$ ps $\Rightarrow$ $\alpha\tau r$Own(p) $\subseteq$ χtrVIs(vs)

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$\cdot$p $\in$ ps$\}$

40. γetP: P-**set** $\rightarrow$ PI $\xrightarrow{\sim}$ P
40. γetP(ps)(pi) $\equiv$ **let** p:P$\cdot$ 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$\cdot$v $\in$ vs$\}$

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

2.1.6. Community & Fleet States

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

- communities and
- fleets.

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

2.1.7. 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. $\mathsf{obs\_TI}$: 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** $\forall -(\mathrm{t},\mathrm{ti}) \cdot \mathsf{obs\_TI}(\mathrm{t})\geq\mathrm{ti}$

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**

2.1.8. 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.

2.1.8.1. 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. S = atH | onL

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

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

48. Frac = **Real** **axiom** $\forall$ f:F$\cdot$0<f$\ll$1

2.1.8.2. 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 **proj**ection 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,_) ⟩
```

2.1.8.3. 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. $TT' = M \times TBL$
53(b). $TBL = BLid \ \overrightarrow{m} \ SUB\_TT$
53((b))i. $SUB\_TT = BSId \ \overrightarrow{m} \ BS$
53. $TT = \{|tt:TT' \cdot wf\_TT(tt)|\}$

value

53. $wf\_TT: TT' \to \mathbf{Bool}$
53. $wf\_TT(m,tbl) \equiv$
53(a). $\forall\ bsm,bsm':(BSId \overrightarrow{m} BS) \cdot \{bsm,bsm'\} \subseteq \mathbf{rng}\ tbl \Rightarrow \mathbf{dom}\ bsm \cap \mathbf{dom}\ bsm' = \{\}$
53(b). $\wedge\ \forall\ (r,bvl):BS \cdot (r,bvl) \in \mathbf{rng}\ bsm \Rightarrow r \in routes(m)$

2.2. 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.

2.2.1. 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. insertH: H → N ∼→ N
54. insertH(h)(n) as n′
54.   pre h∉obs_Hs(n)
54.   post obs_Hs(n)=obs_Hs(n′) ∪ {h} ∧
54.     obs_Ls(n)=obs_Ls(n′)

55. removeH: HI → N ∼→ N
55. removeH(hi)(n) as n′
55.   pre hi ∈ χtrHIs(n)
55.   post obs_LIs(get_HI(hi)(n))={} ∧
55.     obs_Hs(n′)=obs_Hs(n)\{get_HI(hi)(n)}

56. insertL: L → N ∼→ N
56. insertL(l)(n) as n′
56.   pre l∉obs_Ls(n)
56.   post obs_Ls(n′)=obs_Ls(n)∪{l}
56.     let {hi,hi′}=obs_HIs(l) in
56.     let (h,h′)=(γetH(hi)(n),γetH(hi′)(n)),
56.       (nh,nh′)=(γetH(hi)(n′),γetH(hi′)(n′)) in
56.     obs_LIs(nh)=obs_LIs(h)∪{obs_LI(l)},
56.     obs_LIs(nh′)=obs_LIs(h′)∪{obs_LI(l)} end

57. removeL: LI → N ∼→ N
57. removeL(li)(n) as n′
57.   pre li ∈ χtrLIs(n)
57.   post obs_Ls(n)=obs_Ls(n′)\{l}
57.     let {hi,hi′}=obs_HIs(get_L(li)(n)) in
57.     let (h,h′)=(get_H(hi)(n),get_H(hi′)(n)),
57.       (nh,nh′)=(get_H(hi)(n′),get_H(hi′)(n′)) in
57.     obs_LIs(nh)=obs_LIs(h)\{li},
57.     obs_LIs(nh′)=obs_LIs(h′)\{li} end end
```

2.2.2. 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 33).

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. xfer_V: PI×VI×PI → (C×F) → (C×F)
58. xfer_V(fpi,vi,tpi)(c,f) **as** (c′,f′)
58. **pre** ...
58. **post** xfer_V(fpi,vi,tpi)(**obs**_Ps(c),**obs**_Vs(f)) = (ps′,vs′)
58. $\wedge \forall \mathcal{F}_C : \alpha\tau r$Cs(c)$\cdot\mathcal{F}_C$(c)=$\mathcal{F}_C$(c′)
58. $\wedge \forall \mathcal{F}_F : \alpha\tau r$Fs(f)$\cdot\mathcal{F}_F$(f)=$\mathcal{F}_F$(f′)
58. xfer_V: PI×VI×PI → (P-**set**×V-**set**) → (P-**set**×V-**set**)
59. xfer_V(fpi,vi,tpi)(ps,vs) **as** (ps′,vs′)

60(a). **pre** fpi≠tpi∧{fpi,tpi}⊆χtrPIs(ps)∧vi ∈ χtrVIs(vs)
60(b). **post let** (fp,tp)=(γetP(fpi)(ps),γetP(tpi)(ps)),
60(c). (fp′,tp′)=(γetP(fpi)(ps′),γetP(tpi)(ps′)),
60(d). (v,v′)=(γetV(vi)(vs),γetP(vi)(vs′)) **in**
60(e). ps\{fp,tp} = ps′\{fp′,tp′} ∧ vs\{v} = vs\{v′}
60(f). ∧ fp′ = sell(fp,vi) ∧ tp′ = buy(tp,vi) ∧ v′ = 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: P × VI → 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\\{$\alpha\tau r$VI}•F(p)=F(p′)
62. buy: P × VI → 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\\{$\alpha\tau r$VI}•F(p)=F(p′)
63. xfer_Owner: PI × V × PI → 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\\{$\alpha\tau r$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.

2.2.3. 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 $\stackrel{\sim}{\to}$ 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 $\stackrel{\sim}{\to}$ 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}]

2.3. Transport Events

2.3.1. 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. $\mathrm{link\_disappearance}: \mathrm{N} \times \mathrm{N} \to \mathrm{T} \to \mathbf{Bool}$

71. $\mathrm{link\_disappearance}(n,n')(t) \equiv$

71. $\quad \mathbf{let}\ (hs,ls)=(\mathsf{obs\_}Hs,\mathsf{obs\_}Ls)(n),\ (hs',ls')=(\mathsf{obs\_}Hs,\mathsf{obs\_}Ls)(n')\ \mathbf{in}$

71(a). $\quad \exists\ h',h'':H \bullet \{h,h'\} \subseteq hs \cap hs'$

71(a). $\wedge\ \mathbf{let}\ (hi',hi'')=(\mathsf{obs\_}HI(h'),\mathsf{obs\_}HI(h''))\ \mathbf{in}$

71(a). $\quad \mathbf{let}\ (nh',nh'')=(get\_H(hi')(n'),get\_H(hi'')(n'))\ \mathbf{in}$

71(b). $\quad \exists\ h''',h'''':H \bullet \{h''',h''''\}=hs' \setminus hs$

71(c). $\wedge\ \exists\ l':L \bullet \{l'\}=\mathsf{obs\_}Ls(n) \cap \mathsf{obs\_}Ls(n') \wedge \exists\ l'',l''':L \bullet \{l'',l'''\}=\mathsf{obs\_}Ls(n') \setminus \mathsf{obs\_}Ls(n')$

71(d). $\wedge\ \alpha\tau r HIs(l'')=\{hi',\mathsf{obs\_}HI(h''')\} \wedge \alpha\tau r HIs(l''')=\{hi'',\mathsf{obs\_}HI(h'''')\}$

71(e). $\wedge\ \alpha\tau r LIs(h')=\alpha\tau r LIs(nh') \setminus \{\mathsf{obs\_}LI(l')\} \cup \mathsf{obs\_}LI(l'') \wedge \alpha\tau r LIs(h'')=\alpha\tau r LIs(nh'') \setminus \{\mathsf{obs\_}$

71(f). $\wedge\ \alpha\tau r HIs(l')=\{\mathsf{obs\_}HI(nh'),\mathsf{obs\_}HI(h''')\}$

71(g). $\wedge\ \alpha\tau r HIs(l'')=\{\mathsf{obs\_}HI(nh''),\mathsf{obs\_}HI(h'''')\}$

end end end

2.3.2. People Events

72. People are born and people pass away.

value

72. birth: P-**set** $\times$ P-**set** $\rightarrow$ T $\rightarrow$ **Bool**

72. birth(ps,ps′)(t) $\equiv \exists$ p:P $\cdot$ p $\notin$ ps $\wedge$ p $\in$ ps′ $\wedge$ ps′=ps $\cup \{$p$\}$

72. death: P-**set** $\times$ P-**set** $\rightarrow$ T $\rightarrow$ **Bool**

72. death(ps,ps′)(t) $\equiv \exists$ p:P $\cdot$ p $\in$ ps $\wedge$ p $\notin$ ps′ $\wedge$ ps′=ps$\setminus\{$p$\}$

2.3.3. Vehicle Events

73. Vehicles are manufactured and vehicles are scrapped.

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

value

73. mfgd: V-**set** $\times$ V-**set** $\rightarrow$ T $\rightarrow$ **Bool**
73. mfgd(vs,vs′)(t) $\equiv \exists$ v:V $\cdot$ v $\notin$ vs $\wedge$ v $\in$ vs′ $\wedge$ vs′=vs $\cup$\{v\}
73. scrpd: V-**set** $\times$ V-**set** $\rightarrow$ T $\rightarrow$ **Bool**
73. scrpd(vs,vs′)(t) $\equiv \exists$ v:V $\cdot$ v $\in$ vs $\wedge$ v $\notin$ ps′ $\wedge$ vs′=vs\\{v\}
74. coll: V-**set** $\times$ V-**set** $\rightarrow$ T $\rightarrow$ **Bool**
74. coll(vs,vs′)(t) $\equiv$ χtrVIs(vs)=χtrVIs(vs′)
74. $\wedge$ $\exists$ vs″:V-**set** $\cdot$ **card** vs″$\geq$2 $\wedge$ vs″$\subset$vs′
74. $\wedge$ $\forall$ v,v′:V-**set**$\cdot$v$\neq$b′ $\wedge$ \{v,v′\}$\subseteq$vs″ $\wedge$ samePos(v,v′)
74. samePos: V $\times$ V $\rightarrow$ T $\rightarrow$ **Bool**
74. samePos(v,v′)(t) $\equiv$
74. **case** ($\alpha\tau r$VP,$\alpha\tau r$VP) **of** (onL(fhi,li,f,thi),onL(fhi,li,f,thi)) $\rightarrow$ **true**, _ $\rightarrow$ **false end**

2.3.4. 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.

2.4. 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 64–72
  - then similarly of Vehicle Behaviours 73–80
  - of Timetable Behaviours
  - before tackling the more composite Net Behaviours.

2.4.1. 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.

2.4.1.1. 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 persons.
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")

2.4.1.2. 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())
```

2.4.1.3. 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**

2.4.1.4. 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**

2.4.2. Fleet and Vehicle Behaviours

- We describe the concepts of
  - a **fleet** of a dynamically varying number of vehicles
  - and individual **vehicle**s
- 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 66.
- 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.

2.4.2.1. 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")

2.4.2.2. 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())
```

2.4.2.3. 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
```

2.4.2.4. 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
```

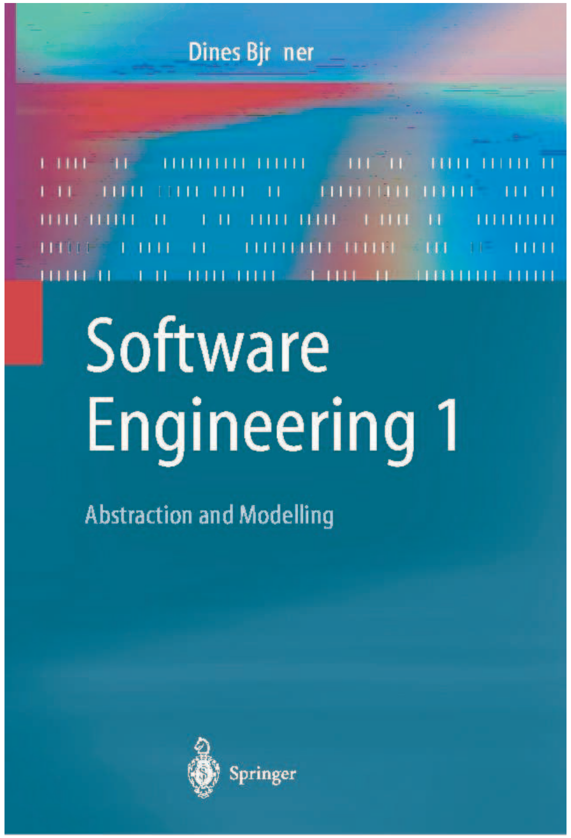
2.5. Discussion of Domain Engineering

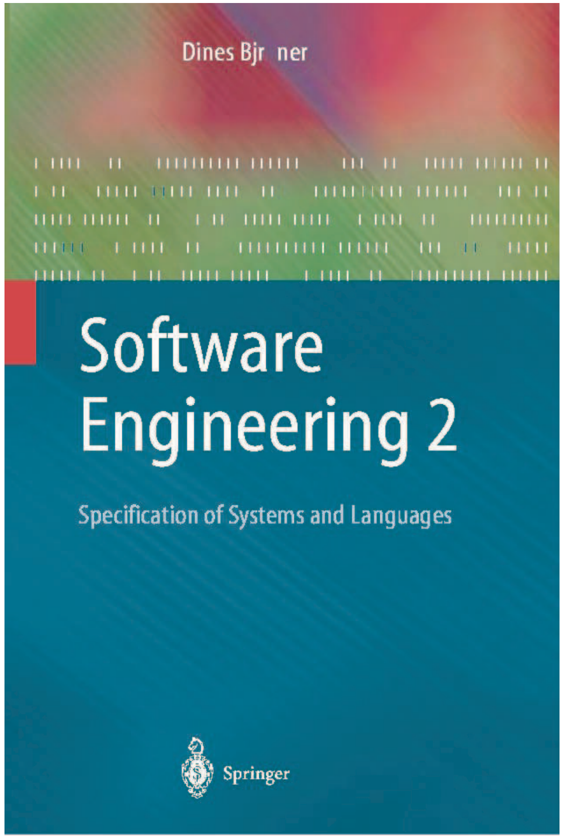
- We have just touched a few issues of a methodology for domain engineering.
- Thus we have not dealt with principles and techniques of describing domain facets:
  - intrinsics,
  - support technologies,
  - rules and regulations,
  - scripts,
  - management and organisation, and
  - human behaviour.
- Each of these, and other methodological topics have an own set of principles and techniques and an emerging underlying theory.

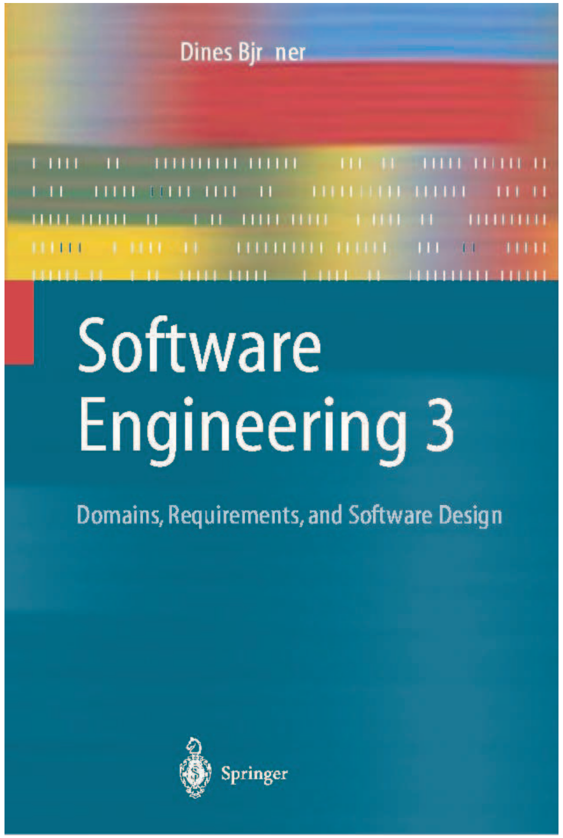
2.6. From Domains to Requirements

- We shall not illustrate
  - how sizable parts of computing systems requirements prescriptions
  - can be systematically 'derived" from domain descriptions.
- But we shall just mention that it can be done through theory-based (algebraic) operation techniques such as
  - projection,
  - instantiation,
  - determination,
  - extension and
  - fitting.

- Applying these techniques
  - a domain description
  - is gradually "transformed" into
  - a requirements prescription
  - with each operational step entailing formal analysis to help ensure consistency and completeness.
- The specific issues dealt with in this talk namely domain science and engineering —
  - can, in the context of software,
  - be seen as part of the triptych:
    - ∗ domains (domain engineering),
    - ∗ requirements (requirements engineering) and
    - ∗ software (design).


Dines Bjr ner
Software
Engineering 1
Abstraction and Modelling
Springer


Dines Bjr ner
Software
Engineering 2
Specification of Systems and Languages
Springer


Dines Bjr ner
Software
Engineering 3
Domains, Requirements, and Software Design
Springer

2.7. Formal Description Languages

- The partial descriptions given were expressed in RSL, the Raise Specification Language.
- But other formal specification languages can be used:
  - Alloy,
  - B/Event-B,
  - VDM or
  - Z,
- as augmented by, for example,
  - Petri Nets,
  - MSC,
  - State Charts,
  - DC,
  - TLA+
  - or other.

3. Broader Aspects of Domain Science & Engineering

3.1. From Science to Technology

- Natural science researchers study "mother nature" to in order to understand it.
- Domain scientists study human-made infrastructures order to understand them.
- Engineers "walk the bridge" between science and technology
  - constructing technology based on scientific theories and
  - studying technologies to find new scientific facts.

3.2. Natural Science Engineering vs. Domain Engineering

- We cannot design software before we have a reasonable grasp of the requirements put to that software.
- We cannot express requirements before we have a reasonable grasp of the domain in which that software is to serve:

3.2.1. Some Examples

3.2.1.1. Automotive Engineering

- An automotive engineer,
  - when designing an automobile transmission system,
  - makes extensive use of basic laws of the theories of mechanics,
  - and would not be hired unless he had a certified, deep knowledge of the laws of mechanics.

3.2.1.2. Communications Engineering

- A radio communications engineer,
  - when designing a radio antenna,
  - makes extensive use of the theories relating to Maxwell's Equations,
  - and would not be hired unless she had a certified, deep knowledge of the laws of electromagnetic wave propagation.
- *Maxwell's Equations* are an example of mathematical modelling.

$$
\begin{aligned}
\nabla \times \overline{E} &= -\frac{\partial \overline{B}}{\partial t} \\
\nabla \times \overline{H} &= \overline{J} + \frac{\partial \overline{D}}{\partial t} \\
\nabla \bullet \overline{B} &= 0 \\
\nabla \bullet \overline{D} &= \rho \\
\nabla \bullet \overline{J} &= -\frac{\partial \rho}{\partial t}
\end{aligned}
$$

3.2.1.3. Building Engineering

- A civil engineer,
  - when designing, for example, a bridge,
  - makes extensive use of the theories of structural statics,
  - and would not be hired unless he had a certified, deep knowledge of the laws of structural statics.

3.2.1.4. Aeronautical Engineering

- An aeronautics engineer,
  - when designing, say, a supersonic aircraft,
  - makes extensive use of the theories of aerodynamics,
  - and would not be hired unless she had a certified, deep knowledge of the laws of aero-, thermo- and hydrodynamics.

3.2.1.5. Software Engineering

- A software engineer is, today,
  - often asked to develop software for such diverse fields as
    - transportation,
    - health-care,
    - financial services,
    - production,
    - the e-market,
    - pipelines,
    - etc.
  - without having any theories about
    - transportation,
    - health-care,
    - financial services,
    - production,
    - marketing & sales,
    - pipelines,
    - etc.

    to refer to.
  - Moreover, the software engineers are not expected to be knowledgeable about any such theories.

3.3. Constructing Domain Descriptions vs. Using Domain Models

- Domains are researched, that is,
  - analysed,
  - described and
  - theories established,

  by domain scientists and domain engineers.
- Domain models, i.e., domain descriptions, are used, that is,
  - studied,
  - adapted to 'subset' domains,
  - combined with other (such) domain descriptions,
  - transformed into specific requirements prescriptions,
  - etcetera,

  by domain and requirements engineers.

- Domain scientists are like physicists,
  - able to create the equivalent of Maxwell's equations,
  - and thus typically at PhD level.
- Domain engineers
  - need not be able to "discover" the equivalent of Maxwell's equations,
  - but must be able to understand such models,
  - "massage" them:
    * edit,
    * combined,
    * projected,
    * instantiated,
    * determinated,
    * extended,
    * etc.

3.4. Domain Science Independent of Software Engineering

- But domain science is — potentially — a much wider field of study and knowledge than sketched here.
  - First we must recall that domain science is concerned with
    * what – of man-made infrastructure components – can be described,
    * how to describe and analyse that,
    * and with formal properties of domain description languages.
  - Thus domain science embodies or borders on topics of philosophy, for example:
    * mereology,
    * ontology and
    * epistemelogy.

- Domain engineering need not "be followed" by requirements engineering and software design.
  - One can create a domain description just in order to simply understand that domain.
  - And one can use domain models for
    - business process modelling and
    - business process re-engineering.
- In this talk we shall not elaborate these topics further.

3.5. Domain Science Transgressing Other Sciences

- Domain science and engineering is not "restricted" to computing science and software engineering.
  - Just like mathematics is practised: studied and applied across such disciplines as
    - life sciences,
    - social science and economics,
    - natural sciences and
    - engineering,

    with each discipline itself developing "an own mathematics",
  - so domain sciences can be practised across
    - air traffic,
    - banking,
    - container lines,
    - health care,
    - pipelines,
    - securities trading,
    - transportation,
    - etcetera.

    where each discipline will itself develop "an own mathematics".

4. Conclusion

4.1. Informatics: A New Universe

- We define 'informatics' as the confluence of
  - *the immaterial sciences and engineering of computing (software),*
  - *the material sciences and engineering of computers (IT: hardware),*
  - *the immaterial sciences and engineering of domain models, and*
  - *mathematics (including mathematical modelling).*
- Whereas IT is a universe of material quantity:
  - faster,
  - cheaper,
  - smaller,
  - large capacity, etc.
- informatics is primarily a universe of intellectual quality:
  - fit for purpose,
  - pleasing,
  - human[6],
  - fun, etc.

[6] Through the phase-, stage- and stepwise "refinement" of domain models via requirements into software while ensuring that that software reflects and only reflects proper domain concepts, one can ensure "user-fiendliness".

4.2. An Exact Sciences Motivation for Interdisciplinarity

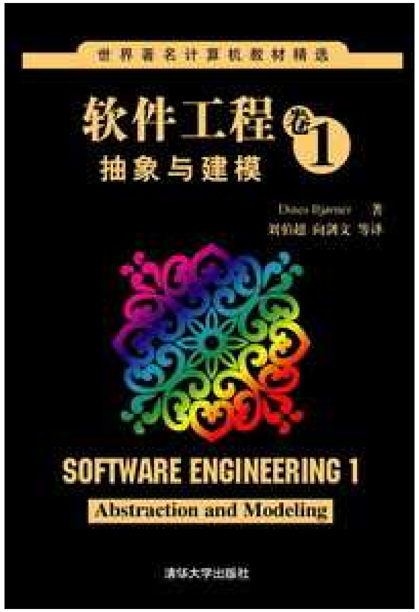
- Examples of interdisciplinary models:
  - gas or oil pipe line systems
    http://www2.imm.dtu.dk/~dibj/pipelines.pdf,
  - stock exchanges
    http://www2.imm.dtu.dk/~dibj/tse-1.pdf,
  - road transport systems
    http://www2.imm.dtu.dk/~dibj/comet/comet1.pdf,
  - railway systems
    http://www.railwaydomain.org/,
  - container line industry
    http://www2.imm.dtu.dk/~dibj/container-paper.pdf,
  - logistics
    http://www2.imm.dtu.dk/~dibj/logistics.pdf,
  - the market[7]
    http://www2.imm.dtu.dk/~dibj/themarket.pdf,

[7] consumers, retailers, wholesalers, producers

– etcetera.

4.3. Closing

- Thanks
- Questions ?



/home/db/2011/swansea/DEroleinSE.tex