Domain Science & Engineering

APSEC 2012 Tutorial, Hong Kong, 4 Dec. 2012

Dines Bjørner

DTU Informatics

November 16, 2012: 10:15

Schedule

- 09:00–09:10: Summary 1–8
- 09:10–09:50: A Domain Description Example 9–62
- 10:00–10:30: Domain Descriptions: Endurants I 63–89
- 11:00–11:40: Domain Descriptions: Endurants II 90–123
- 11:50–12:20: Domain Descriptions: Perdurants 124–187
- 12:20–12:30: Conclusion 188–194

Summary

- By a **domain entity** we shall understand
  - *a manifest phenomenon, something that can be pointed to,*
  - *which has a form of*
    - *either permanent character, that is, an endurant,*
    - *or "fleeting" character, that is, a perdurant.*

Example: 1 Domain Entities. Example domain entities are

- a street segment (an endurant),
- insertion of a link (a perdurant).

- By a **quality** we shall understand
  - *a property, expressible as a proposition,*
  - *which is either satisfied by an entity or is not satisfied.*

Example: 2 Entity Qualities. Example entity qualities are

- length of a street segment (links): *has_length*,
- location of a street segments (links): *has_location*,
- identities of the street intersections connected to a link: *has_mereology*,
- type of arguments of the insertion function: *has_argument*,
- type of result of the insertion function: *has_result*.

- By a **domain** we shall understand
  - *a set of domain entities,*
  - *a set of qualities and*
  - *a mapping of domain entities into qualities.*

- By **domain analysis** we shall understand
  - *the principled use of a set of techniques and tools*
  - *for identifying*
    - *domain entities,*
    - *qualities and*
    - *mappings between them.*

- By **domain description** we shall understand
  - *the principled use of a set of techniques and tools*
  - *for describing, informally and formally*
    - *domain entities,*
    - *qualities and*
    - *mappings between them.*

- In this seminar
  - we shall cover, in respective sections,
  - the techniques and tools of domain analysis and
  - the techniques and tools of domain description.

1. An Example

- The main example presents a terse narrative and formalisation of a road traffic domain.
  - Since the example description conceptually covers also major aspects of
    - railroad nets,
    - shipping nets, and
    - air traffic nets,
  - we shall use such terms as hubs and links to stand for
    - road (or street) intersection and road (or street) segments,
    - train stations and rail lines,
    - harbours and shipping lanes, and
    - airports and air lanes.

1.1. Parts
1.1.1. Root Sorts

- The domain,
  - the stepwise unfolding of
  - whose description is
  - to be exemplified,

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

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

type
1. Δ
1a. N
1b. F
1c. M

value
1a. **obs_**N: Δ → N
1b. **obs_**F: Δ → F
1c. **obs_**M: Δ → M

1.1.2. Sub-domain Sorts and Types

2. From the road net we can observe
   a. a composite part, HS, of road (i.e., street) intersections (hubs) and
   b. an composite part, LS, of road (i.e., street) segments (links).

type
2. HS, LS
value
2a. **obs_**HS: N → HS
2b. **obs_**LS: N → LS

3. From the fleet sub-domain, F, we observe a composite part, VS, of vehicles

type
3. VS
value
3. **obs_**VS: F → VS

4. From the composite sub-domain VS we observe
   a the composite part Vs, which we concretise as a set of vehicles
   b where vehicles, V, are considered atomic.

type
4a. Vs = V-**set**
4b. V
value
4a. **obs_**Vs: VS → V-**set**

- The "monitor" is considered atomic; it is an abstraction of the fact that
  - we can speak of the positions of each and every vehicle on the net
  - without assuming that we can indeed pin point these positions
  - by means of for example sensors.

1.1.3. Further Sub-domain Sorts and Types

- We now analyse the sub-domains of HS and LS.

5. From the hubs aggregate we decide to observe
   a the concrete type of a set of hubs,
   b where hubs are considered atomic; and
6. from the links aggregate we decide to observe
   a the concrete type of a set of links,
   b where links are considered atomic;

type

5a. Hs = H-**set**

6a. Ls = L-**set**

5b. H

6b. L

value

5. **obs_**Hs: HS → H-**set**

6. **obs_**Ls: LS → L-**set**

- We have no composite parts left to further analyse into parts
  - whether they be again composite
  - or atomic.
- That is,
  - at various, what we shall refer to as, **domain indexes**
  - we have discovered the following part types:
    - $\langle\Delta\rangle$: N, F, M
    - $\langle\Delta, N\rangle$: HS, LS
    - $\langle\Delta, F\rangle$: VS
    - $\langle\Delta, HS\rangle$: Hs, H
    - $\langle\Delta, LS\rangle$: Ls, L
    - $\langle\Delta, VS\rangle$: Vs, V
  - Thus we have ended up with atomic parts.

1.2. Properties

- Parts are distinguished by their properties:
  - the types and
  - the values

  of these.
- We consider three kinds of properties:
  - unique identifiers,
  - mereology and
  - attributes.

1.2.1. Unique Identifications

7. We decide the following:
   a each hub has a unique hub identifier,
   b each link has a unique link identifier and
   c each vehicle has a unique vehicle identifier.

type
7a. HI
7b. LI
7c. VI
value
7a. **uid**_H: H → HI
7b. **uid**_L: L → LI
7c. **uid**_V: V → VI

1.2.2. Mereology
1.2.2.1 Road Net Mereology

- By *mereology* we mean the study, knowledge and practice of understanding parts and part relations.

8. Each link is connected to exactly two hubs, that is,
   a from each link we can observe its mereology, that is, the identities of these two distinct hubs,
   b and these hubs must be of the net of the link;
9. and each hub is connected to zero, one or more links, that is,
   a from each hub we can observe its mereology, that is, the identities of these links,
   b and these links must be of the net of the hub.

value

8a. **mereo_**L: L → HI-**set**, **axiom** ∀ l:L·**card** **mereo_**L(l)=2

axiom

8b. ∀ n:N,l:L,hi:HI · l ∈ **obs_**Ls(**obs_**LS(n)) ∧ hi ∈ **mereo_**L(l)

8b. ⇒ ∃ h:H·h ∈ **obs_**Hs(**obs_**HS(n))∧**uid_**H(h)=hi

value

9a. **mereo_**H: H → LI-**set**

axiom

9b. ∀ n:N,h:H,li:LI · h ∈ **obs_**Hs(**obs_**HS(n)) ∧ li ∈ **mereo_**H(h)

9b. ⇒ ∃ l:L·l ∈ **obs_**Ls(**obs_**LS(n))∧**uid_**L(l)=li

1.2.2.2 Fleet of Vehicles Mereology

- In the traffic system that we are building up
  - there are no relations to be expressed between vehicles,
  - only between vehicles and the (single and only) monitor.
- Thus there is no mereology needed for vehicles.

1.2.3. Attributes

- We shall model attributes of
  - links,
  - hubs and
  - vehicles.
- The composite parts,
  - aggregations of hubs, HS and Hs,
  - aggregations of links, LS and Ls and
  - aggregations of vehicles, VS and Vs,

  also have attributes, but we shall omit modelling them here.

1.2.3.1 Attributes of Links

10. The following are attributes of links.
    a Link states, lσ:LΣ, which we model as possibly empty sets of pairs of distinct identifiers of the connected hubs.
        • A link state expresses the directions that are open to traffic across a link.
    b Link state spaces, lω:LΩ which we model as the set of link states.
        • A link state space expresses the states that a link may attain across time.
    c Further link attributes are length, location, etcetera.

- Link states are usually dynamic attributes
- whereas
    - link state spaces,
    - link length and
    - link location (usually some curvature rendition)

  are considered static attributes.

type
10a. $L\Sigma = (HI \times HI)$**-set**
axiom
10a. $\forall\ l\sigma{:}L\Sigma \cdot 0 \leq$ **card** $l\sigma \leq 2$
value
10a. **attr_**$L\Sigma$: $L \rightarrow L\Sigma$
axiom
10a. $\forall\ l{:}L \cdot$ **let** $\{hi,hi'\}=$**mereo_**$L(l)$ **in** **attr_**$L\Sigma(l) \subseteq \{(hi,hi'),(hi',hi)\}$ **end**
type
10b. $L\Omega = L\Sigma$**-set**
value
10b. **attr_**$L\Omega$: $L \rightarrow L\Omega$
axiom
10b. $\forall\ l{:}L \cdot$ **let** $\{hi,hi'\}=$**mereo_**$L(l)$ **in** **attr_**$L\Sigma(l) \in$ **attr_**$L\Omega(l)$ **end**
type
10c. LOC, LEN, ...
value
10c. **attr_**LOC: $L \rightarrow$ LOC, **attr_**LEN: $L \rightarrow$ LEN, ...

1.2.3.2 Attributes of Hubs

11. The following are attributes of hubs:
    a Hub states, $\mathsf{h}\sigma{:}\mathsf{H}\Sigma$, which we model as possibly empty sets of pairs of identifiers of the connected links.
        • A hub state expresses the directions that are open to traffic across a hub.
    b Hub state spaces, $\mathsf{h}\omega{:}\mathsf{H}\Omega$ which we model as the set of hub states.
        • A hub state space expresses the states that a hub may attain across time.
    c Further hub attributes are location, etcetera.

- Hub states are usually dynamic attributes
- whereas
    - hub state spaces and
    - hub location

  are considered static attributes.

type
11a. HΣ = (LI × LI)**-set**
value
11a. **attr_**HΣ: H → HΣ
axiom
11a. ∀ h:H · **attr_**HΣ(h)⊆{(li,li′)|li,li′:LI·{li,li′}⊆**mereo_**H(h)}
type
11b. HΩ = HΣ**-set**
value
11b. **attr_**HΩ: H → HΩ
axiom
11b. ∀ h:H · **attr_**HΣ(h) ∈ **attr_**HΩ(h)
type
11c. LOC, ...
value
11c. **attr_**LOC: L → LOC, ...

1.2.3.3 Attributes of Vehicles

12. Dynamic attributes of vehicles include
    a position
        i. at a hub (about to enter the hub — referred to by the link it is coming from, the hub it is at and the link it is going to, all referred to by their unique identifiers or
        ii. some fraction "down" a link (moving in the direction from a from hub to a to hub — referred to by their unique identifiers)
        iii. where we model fraction as a real between 0 and 1 included.
    b velocity, acceleration, etcetera.
13. All these vehicle attributes can be observed.

type

92a. VP = atH | onL

92(a)i. atH :: fli:LI × hi:HI × tli:LI

92(a)ii. onL :: fhi:HI × li:LI × frac:FRAC × thi:HI

92(a)iii. FRAC = **Real**, **axiom** ∀ frac:FRAC · 0 ≤ frac ≤ 1

92b. VEL, ACC, ...

value

13. **attr_**VP:V→VP, **attr_**onL:V→onL, **attr_**atH:V→atH

13. **attr_**VEL:V→VEL, **attr_**ACC:V→ACC

1.2.3.4 Vehicle Positions

14. Given a net, n:N, we can define the possibly infinite set of potential vehicle positions on that net, vps(n).
    a vps(n) is expressed in terms of the links and hubs of the net.
    b vps(n) is the
    c union of two sets:
        i. the potentially[1] infinite set of "on link" positions
        ii. for all links of the net
        and
        i. the finite set of "at hub" positions
        ii. for all hubs in the net.

[1] The 'potentiality' arises from the nature of FRAC. If fractions are chosen as, for example, 1/5'th, 2/5'th, ..., 4/5'th, then there are only a finite number of "on link" vehicle positions. If instead fraction are arbitrary infinitesimal quantities, then there are infinitely many such.

value

14. vps: N → VP**-infset**
14b. vps(n) ≡
14a. **let** ls=**obs_**Ls(**obs_**LS(n)), hs=**obs_**Hs(**obs_**HS(n)) **in**
14(c)i. { onL(fhi,uid(l),f,thi) | fhi,thi:HI,l:L,f:FRAC ·
14(c)ii. l ∈ ls ∧ {fhi,thi}=**mereo_**L(l) }
14c. ∪
14(c)i. { atH(fli,**uid_**H(h),tli) | fli,tli:LI,h:H ·
14(c)ii. h ∈ hs ∧ {fli,tli}⊆**mereo_**H(h) }
14a. **end**

- Given a net and a finite set of vehicles
  - we can distribute these over the net, i.e., assign initial vehicle positions,
  - so that no two vehicles "occupy" the same position, i.e., are "crashed" !
- Let us call the non-deterministic assignment function, i.e., a relation, for vpr.

15. vpm:VPM is a bijective map from vehicle identifiers to (distinct) vehicle positions.
16. vpr has the obvious signature.
17. vpr(vs)(n) is defined in terms of
18. a non-deterministic selection, vpa, of vehicle positions, and
19. a non-deterministic assignment of these vehicle positions to vehicle identifiers —
20. being the resulting distribution.

type

15. VPM′ = VI $\overrightarrow{m}$ VP
15. VPM = {| vpm:VPM′ · **card dom** vpm = **card rng** vpm |}

value

16. vpr: V**-set** × N → VMP
17. vpr(vs)(n) ≡
18. **let** vpa:VP**-set** · vpa ⊆ vps(vs)(n) ∧ **card** vpa = vard vs **in**
19. **let** vpm:VPM · **dom** vpm = vps ∧ **rng** vpm = vpa **in**
20. vpm **end end**

1.3. Definitions of Auxiliary Functions

21. From a net we can extract all its link identifiers.
22. From a net we can extract all its hub identifiers.

value

21. xtr_LIs: N → LI-**set**
21. xtr_LIs(n) ≡ {**uid**_L(l)|l:L·l ∈ **obs**_Ls(**obs**_LS(n))}
22. xtr_HIs: N → HI-**set**
22. xtr_HIs(n) ≡ {**uid**_H(l)|h:H·h ∈ **obs**_Hs(**obs**_HS(n))}

23. Given a link identifier and a net get the link with that identifier in the net.
24. Given a hub identifier and a net get the hub with that identifier in the net.

value

??. get_H: HI → N $\xrightarrow{\sim}$ H
??. get_H(hi)(n) ≡ ι h:H·h ∈ **obs**_Hs(**obs**_HS(n))∧**uid**_H(h)=hi
??. **pre**: hi ∈ xtr_HIs(n)
??. get_L: LI → N $\xrightarrow{\sim}$ L
??. get_L(li)(n) ≡ ι l:L·l ∈ **obs**_Ls(**obs**_LS(n))∧**uid**_L(l)=li
??. **pre**: hl ∈ xtr_LIs(n)

- The ι a:A·$\mathcal{P}$(a) expression
  - yields the unique value a:A
  - which satisfies the predicate $\mathcal{P}$(a).
  - If none, or more than one exists then the function is undefined.

1.4. States

- There are different notions of state. In our example these are some of the states:
  - the road net composition of hubs and links;
  - the state of a link, or a hub; and
  - the vehicle position.

1.5. Actions

- An action is what happens when a function invocation changes, or potentially changes a state.
- Examples of traffic system actions are:
  - insertion of hubs,
  - insertion of links,
  - removal of hubs,
  - removal of links,
  - setting of hub state ($\mathsf{h}\sigma$),
  - setting of link state ($\mathsf{l}\sigma$),
  - moving a vehicle along a link,
  - moving a vehicle from a link to a hub and
  - moving a vehicle from a hub to a link.

25. The **insert** action applies to a net and a hub and conditionally yields an updated net.
    a The condition is that there must not be a hub in the "argument" net with the same unique hub identifier as that of the hub to be inserted and
    b the hub to be inserted does not initially designate links with which it is to be connected.
    c The updated net contains all the hubs of the initial net "plus" the new hub.
    d and the same links.

value

67. ins_H: N → H $\xrightarrow{\sim}$ N
67. ins_H(n)(h) **as** n′, **pre**: pre_ins_H(n)(h), **post**: post_ins_H(n)(h)

67a. pre_ins_H(n)(h) ≡
67a. ∼∃ h′:H · h′ ∈ **obs**_Hs(n) ∧ **uid**_HI(h)=**uid**_HI(h′)
67b. ∧ **mereo**_H(h) = {}

67c. post_ins_H(n)(h)(n′) ≡
67c. **obs**_Hs(n) ∪ {h} = **obs**_Hs(n′)
67d. ∧ **obs**_Ls(n) = **obs**_Ls(n′)

1.6. Events

- By an **event** we understand
  - a state change
  - resulting indirectly from an unexpected application of a function,
  - that is, that function was performed "surreptitiously".
- Events can be characterised by a pair of (before and after) states, a predicate over these and, optionally, a **time** or **time interval**.
- Events are thus like actions:
  - change states,
  - but are usually
    - either caused by "previous" actions,
    - or caused by "an outside action".

26. Link disappearance is expressed as a predicate on the "before" and "after" states of the net. The predicate identifies the "missing" ℓink (!).
27. Before the disappearance of link ℓ in net n
    a the hubs h' and h'' connected to link ℓ
    b were connected to links identified by $\{l'_1, l'_2, \ldots, l'_p\}$ respectively $\{l''_1, l''_2, \ldots, l''_q\}$
    c where, for example, l'_i, l''_j are the same and equal to uid_$\Pi(\ell)$.

```
68.  link_dis: N × N → Bool
68.  link_dis(n,n′) ≡
68.      ∃ ℓ:L · pre_link_dis(n,ℓ) ⇒ post_link_dis(n,ℓ,n′)
69.  pre_link_dis: N × L → Bool
69.  pre_link_dis(n,ℓ) ≡ ℓ ∈ obs_Ls(n)
```

28. After link ℓ disappearance there are instead
    a two separate links, ℓ_i and ℓ_j, "truncations" of ℓ
    b and two new hubs h''' and h''''
    c such that ℓ_i connects h' and h''' and
    d ℓ_j connects h'' and h'''';
    e Existing hubs h' and h'' now have mereology
        i. $\{l'_1, l'_2, \ldots, l'_p\} \setminus \{\mathsf{uid\_}\Pi(\ell)\} \cup \{\mathsf{uid\_}\Pi(\ell_i)\}$ respectively
        ii. $\{l''_1, l''_2, \ldots, l''_q\} \setminus \{\mathsf{uid\_}\Pi(\ell)\} \cup \{\mathsf{uid\_}\Pi(\ell_j)\}$
29. All other hubs and links of n are unaffected.

30. We shall "explain" *link disappearance* as the combined, instantaneous effect of
    a. first a **remove link** "event" where the **removed link** connected **hub**s hi_j and hi_k;
    b. then the **insert**ion of two new, "fresh" **hub**s, h_α and h_β;
    c. "followed" by the **insert**ion of two new, "fresh" links $\mathsf{l}_{j\alpha}$ and $\mathsf{l}_{k\beta}$ such that
        i. $\mathsf{l}_{j\alpha}$ connects hi_j and h_α and
        ii. $\mathsf{l}_{k\beta}$ connects hi_k and $\mathsf{h}_{k\beta}$

value

72. post_link_dis(n,ℓ,n$'$) $\equiv$
72. **let** h_a,h_b:H ·
72. **let** {li_a,li_b}=**mereo_**L(ℓ) **in**
72. (get_H(li_a)(n),get_H(li_b)(n)) **end in**
72a. **let** n$''$ = rem_L(n)(**uid_**L(ℓ)) **in**
72b. **let** h$_\alpha$,h$_\beta$:H · {h$_\alpha$,h$_\beta$}∩**obs_**Hs(n)={} **in**
72b. **let** n$'''$ = ins_H(n$''$)(h$_\alpha$) **in**
72b. **let** n$''''$ = ins_H(n$'''$)(h$_\beta$) **in**
72c. **let** l$_{j\alpha}$,l$_{k\beta}$:L · {l$_{j\alpha}$,l$_{k\beta}$}∩**obs_**Ls(n)={}
72c. ∧ **mereo_**L(l$_{j\alpha}$) = {**uid_**H(h_a),**uid_**H(h$_\alpha$)}
72c. ∧ **mereo_**L(l$_{k\beta}$) = {**uid_**H(h_b),**uid_**H(h$_\beta$)} **in**
72(c)i. **let** n$'''''$ = ins_L(n$''''$)(l$_{j\alpha}$) **in**
72(c)ii. n$'$ = ins_L(n$'''''$)(l$_{k\beta}$) **end end end end end end end**

1.7. Behaviours
1.7.1. Traffic
1.7.1.1 Continuous Traffic

- For the road traffic system
  - perhaps the most significant example of a behaviour
  - is that of its traffic
    31. the continuous time varying discrete positions of vehicles, vp:VP[2],
    32. where time is taken as a dense set of points.

type

74. $\mathbb{cT}$
73. $\mathrm{cRTF} = \mathbb{cT} \rightarrow (\mathrm{V} \xrightarrow[m]{} \mathrm{VP})$

[2] For VP see Item 92a on Slide 169.

1.7.1.2 Discrete Traffic

- We shall model, not continuous time varying traffic, but

33. discrete time varying discrete positions of vehicles,
34. where time can be considered a set of linearly ordered points.

76. $\mathrm{d}\mathbb{T}$
75. $\mathrm{dRTF} = \mathrm{d}\mathbb{T} \xrightarrow{m} (\mathrm{V} \xrightarrow{m} \mathrm{VP})$

35. The road traffic that we shall model is, however, of vehicles referred to by their unique identifiers.

type

77. $\mathrm{RTF} = \mathrm{d}\mathbb{T} \xrightarrow{m} (\mathrm{VI} \xrightarrow{m} \mathrm{VP})$

1.7.1.3 Time: An Aside

- We shall take a rather simplistic view of time
  [wayne.d.blizard.90,mctaggart-t0,prior68,J.van.Benthem.Log

36. We consider $\mathbf{d}\mathbb{T}$, or just $\mathbb{T}$, to stand for a totally ordered set of time points.
37. And we consider $\mathbb{TI}$ to stand for time intervals based on $\mathbb{T}$.
38. We postulate an infinitesimal small time interval δ.
39. $\mathbb{T}$, in our presentation, has lower and upper bounds.
40. We can compare times and we can compare time intervals.
41. And there are a number of "arithmetics-like" operations on times and time intervals.

type

78. $\mathbb{T}$
79. $\mathbb{TI}$

value

80. δ:$\mathbb{TI}$
81. $\mathbb{MIN}, \mathbb{MAX}$: $\mathbb{T} \to \mathbb{T}$
81. $<,\leq,=,\geq,>$: $(\mathbb{T}\times\mathbb{T})|(\mathbb{TI}\times\mathbb{TI}) \to$ **Bool**
82. $-$: $\mathbb{T}\times\mathbb{T} \to \mathbb{TI}$
83. $+$: $\mathbb{T}\times\mathbb{TI},\mathbb{TI}\times\mathbb{T} \to \mathbb{T}$
83. $-,+$: $\mathbb{TI}\times\mathbb{TI} \to \mathbb{TI}$
83. $*$: $\mathbb{TI}\times$**Real** $\to \mathbb{TI}$
83. $/$: $\mathbb{TI}\times\mathbb{TI} \to$ **Real**

42. We postulate a global clock behaviour which offers the current time.

43. We declare a channel clk_ch.

value

84. clock: $\mathbb{T} \rightarrow$ **out** clk_ch **Unit**

84. clock(t) $\equiv$ ... clk_ch!t ... clock(t $\lceil\rceil$ t+δ)

channnel

85. clk_ch:$\mathbb{T}$

1.7.2. Globally Observable Parts

- There is given

44. a net, **n:N**,
45. a set of vehicles, **vs:V-set**, and
46. a monitor, **m:M**.

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

value

87. n:N = **obs_**N(Δ)
87. ls:L**-set** = **obs_**Ls(**obs_**LS(n)), hs:H**-set** = **obs_**Hs(**obs_**HS(n)),
87. lis:LI**-set** = {**uid_**L(l)|l:L·l ∈ ls}, his:HI**-set** = {**uid_**H(h)|h:H·h ∈ hs}
88. vs:V**-set** = **obs_**Vs(**obs_**VS(**obs_**F(Δ))), vis:V**-set** = {**uid_**V(v)|v:V·v
89. m:**obs_**M(Δ)

1.7.3. Road Traffic System Behaviours

47. Thus we shall consider our road traffic system, **rts**, as
    a the concurrent behaviour of a number of vehicles and,
      to "observe", or, as we shall call it, to monitor their movements,
    b the **mon**itor behaviour, based on
    c the monitor and its unique identifier,
    d an initial vehicle position map, and
    e an initial starting time.

value

47c. mi:MI = **uid_**(m)

47d. vpm:VPM = vpr(vs)(n)

47e. t_0:T = clk_ch?

86. rts() =

86a. ‖ {veh(**uid_**V(v))(v)(vpm(**uid_**V(v)))|v:V·v ∈ vs}

86b. ‖ mon(mi)(m)([t_0 ↦ vpm])

- where the "extra" monitor argument
  - records the discrete road traffic, RTF,
  - initially set to the singleton map from an initial start time, t_0 to the initial assignment of vehicle positions.

1.7.4. Channels

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

48. Thus we declare a set of channels indexed by the unique identifiers of vehicles and communicating vehicle positions.

channel

90. {vm_ch[mi,vi]|vi:VI•vi ∈ vis}:VP

1.7.5. Behaviour Signatures

49. The road traffic system behaviour, **rts**, takes no arguments; and "behaves", that is, continues forever.
50. The vehicle behaviours are indexed by the unique identifier, **uid_V(v):VI**, the vehicle part, **v:V** and the vehicle position; offers communication to the monitor behaviour; and behaves "forever".
51. The monitor behaviour takes monitor part, **m:M**, as argument and also the discrete road traffic, **drtf:dRTF**; the behaviour otherwise runs forever.

value

93. rts: **Unit** → **Unit**
94. veh: vi:VI → v:V → VP → **out** vm_ch[vi],mi:MI **Unit**
95. mon: mi:MI → m:M → dRTF → **in** {vm_ch[mi,vi]|vi:VI·vi ∈ vis},clk_c

1.7.6. The Vehicle Behaviour

52. A vehicle process
    - is indexed by the unique vehicle identifier vi:VI,
    - the vehicle "as such", v:V and
    - the vehicle position, vp:VPos.

    The vehicle process communicates
    - with the monitor process on channel vm[vi]
    - (sends, but receives no messages), and
    - otherwise evolves "in[de]finitely" (hence **Unit**).

53. We describe here an abstraction of the vehicle behaviour **at** a **Hub** (**hi**).
    a Either the vehicle remains at that hub informing the monitor,
    b or, internally non-deterministically,
        i. moves onto a link, **tli**, whose "next" hub, identified by **thi**, is obtained from the mereology of the link identified by **tli**;
        ii. informs the monitor, on channel **vm[vi]**, that it is now on the link identified by **tli**,
        iii. whereupon the vehicle resumes the vehicle behaviour positioned at the very beginning (**0**) of that link,
    c or, again internally non-deterministically,
    d the vehicle "disappears — off the radar" !

97. veh(vi)(v)(vp:atH(fli,hi,tli)) ≡
97a. vm_ch[mi,vi]!vp ; veh(vi)(v)(vp)
97b. ⌈⌉
97(b)i. **let** {hi′,thi}=**mereo**_L(get_L(tli)(n)) **in assert:** hi′=hi
97(b)ii. vm_ch[mi,vi]!onL(tli,hi,0,thi) ;
97(b)iii. veh(vi)(v)(onL(tli,hi,0,thi)) **end**
97c. ⌈⌉
97d. **stop**

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

96. veh(vi)(v)(vp:onL(fhi,li,f,thi)) ≡
98a. vm_ch[mi,vi]!vp ; veh(vi)(v)(vp)
98b. ⌈⌉
98c. **if** f + δ<1
98(c)i. **then** vm_ch[mi,vi]!onL(fhi,li,f+δ,thi) ;
98(c)i. veh(vi)(v)(onL(fhi,li,f+δ,thi))
98(c)ii. **else let** li′:LI·li′ ∈ **mereo**_H(get_H(thi)(n)) **in**
98(c)iiA. vm_ch[mi,vi]!atH(li,thi,li′);
98(c)iiB. veh(vi)(v)(atH(li,thi,li′)) **end end**
99. ⌈⌉
100. **stop**

1.7.7. The Monitor Behaviour

57. The monitor behaviour evolves around the attributes of an own "state", m:M, a table of traces of vehicle positions, while accepting messages about vehicle positions and otherwise progressing "in[de]finitely".
58. Either the monitor "does own work"
59. or, internally non-deterministically accepts messages from vehicles.
    a. A vehicle position message, vp, may arrive from the vehicle identified by vi.
    b. That message is appended to that vehicle's movement trace,
    c. whereupon the monitor resumes its behaviour —
    d. where the communicating vehicles range over all identified vehicles.

101. mon(mi)(m)(rtf) ≡
102. mon(mi)(own_mon_work(m))(rtf)
103. ⌈⌉
103a. [] { **let** ((vi,vp),t) = (vm_ch[mi,vi]?,clk_ch?) **in**
103b. **let** rtf′ = rtf † [t ↦ rtf(max **dom** rtf) † [vi ↦ vp]] **in**
103c. mon(mi)(m)(rtf′) **end**
103d. **end** | vi:VI · vi ∈ vis }

102. own_mon_work: M → dRTF → M

- We do not describe the clock behaviour by other than stating that it continually offers the current time on channel clkm_ch. ■

2. Domain Descriptions — Endurants

2.1. What is a Part?

- By a part we mean an observable manifest endurant.

2.1.1. Classes of "Same Kind" Parts

- We repeat:
  - the domain describer does not describe instances of parts,
  - but seeks to describe classes of parts of the same kind.
- Instead of the term 'same kind' we shall use either the terms
  - part sort or
  - part type.
- By a same kind class of parts, that is a part sort or part type we shall mean
  - a class all of whose members, i.e., parts,
  - enjoy "exactly" the same properties
  - where a property is expressed as a proposition.

Example: 3 Part Properties.

- Examples of part properties are:
  - *has unique identity*,
  - *has mereology*,
  - *has length*,
  - *has location*,
  - *has traffic movement restriction*,
  - *has position*,
  - *has velocity* and
  - *has acceleration*.

2.1.2. Concept Analysis as a Basis for Part Typing

- The domain analyser examines collections of parts.
  - In doing so the domain analyser discovers and thus identifies and lists a number of properties.
  - Each of the parts examined usually satisfies only a subset of these properties.
  - The domain analyser now groups parts into collections
    - such that each collection have its parts satisfy the same set of properties,
    - such that no two distinct collections are indexed, as it were, by the same set of properties, and
    - such that all parts are put in some collection.
  - The domain analyser now
    - assigns distinct type names (same as sort names)
    - to distinct collections.
- That is how we assign types to parts.
- We shall return later to a proper treatment of formal concept analysis [Wille:ConceptualAnalysis1999].

2.2. Atomic and Composite Parts

- Parts may be analysed into disjoint sets of
  - atomic parts and
  - composite parts.
- **Atomic parts** are those which,
  - in a given context,
  - are deemed *not* to consist of meaningful, separately observable proper sub-parts.
- **Composite parts** are those which,
  - in a given context,
  - are deemed to *indeed* consist of meaningful, separately observable proper sub-parts.
- A sub-part is a part.

Example: 4 Atomic and/or Composite Parts. To one person a part may be atomic; to another person the same part may be composite.

- It is the domain describer who decides the outcome of this aspect of domain analysis.
  - In some domain analysis a 'person' may be considered an atomic part.
    - For the domain of ferrying cars with passengers
    - persons are considered parts.
  - In some other domain analysis a 'person' may be considered a composite part.
    - For the domain of medical surgery
    - persons may be considered composite parts. ■

Example: 5 Container Lines.

- We shall presently consider containers (as used in container line shipping) to be atomic parts.
- And we shall consider a container vessel to be a composite part consisting of
  - an indexed set of container bays
  - where each container bay consists of indexed set of container rows
  - where each container row consists of indexed set of container stacks
  - where each container stack consists of a linearly indexed sequence of containers.
- Thus container vessels, container bays, container rows and container stacks are composite parts. ■

2.2.1. Atomic Parts

- When we observe
  - what we have decided, i.e., analysed, to be an endurant,
  - more specifically an **atomic part**, of a domain,
  - we are observing an **instance** of an atomic part.
- When we describe those instances
  - we describe, not their **value**s, i.e., the instances,
  - but their
    - **type** and
    - **properties**.

- In this section on **endurant entities**
  we shall unfold what these properties might be.
- But, for now, we focus on the **type** of the observed **atomic part**.
- So the situation is that we are observing a number of atomic parts
  - and we have furthermore decided that
  - they are all of *"the same kind"*.

- What does it mean for a number of atomic parts to be of *"the same kind"* ?
  - It means
    - that we have decided,
    - for any pair of **part**s considered of the same kind,
    - that the kinds of properties,
      - for such two **part**s,
    - are *"the same"*,
      - that is, of the same **type**, but possibly of different **value**s,
    - and that a number of different, other "facets",
    - are not taken into consideration.

- That is,
  - we **abstract** a collection of atomic parts
  - to be of the same kind,
  - thereby "dividing the domain of endurants" into possibly two distinct sets
    - those that are of the analysed kind, and
    - those that are not.

- It is now our description choice to associate with a set of **atomic parts** of *"the same kind"*
  - a **part type** (by suggesting a name for that **type**, for example, T) and
  - a set of **properties** (of its values):
    - **unique identifier**,
    - **mereology** and
    - **attributes**.

- Later we shall introduce **discrete perdurants**
  (actions, events and behaviours)
  whose **signatures** involves (possibly amongst others) type **T**.
- Now we can characterise *"of the same kind"* **atomic part facets**[3]
  - being of the same, named **part type**,
  - having the same **unique identifier type**,
  - having the same **mereology**
    (but not necessarily the same **mereology values**), and
  - having the same set of **attributes**
    (but not necessarily of the same **attribute values**),
- The *"same kind"* criteria apply equally well to **composite part facets**.

[3] as well as "of the same kind" **composite part facets**.

Example: 6 Transport Nets: Atomic Parts (I).

- The types of atomic transportation net parts are:
  - hubs, say of type H, and
  - links, say of type L.
- The chosen mereology associates with every hub and link a
  - distinct unique identifiers
  - (of types HI and LI respectively), and, *vice versa*,
  - how hubs and links are connected:
    - hubs to any number of links and
    - links to exactly two distinct hubs.

- The chosen attributes of
  - hubs include
    - hub location,
    - hub design[4],
    - hub traffic state[5],
    - hub traffic state space[6], etc.;
  - and of links include
    - link location,
    - link length,
    - link traffic state[7],
    - link traffic state space[8], etc.
- With these mereologies and attributes we see that we can consider hubs and links as different kinds of atomic parts. ■

[4] Design: simple crossing, freeway "cloverleaf" interchange, etc.

[5] A hub traffic state is (for example) a set of pairs of link identifiers where each such pair designates that traffic can move from the first designated link to the second.

[6] A hub state space is (for example) the set of all hub traffic states that a hub may range over.

[7] A link traffic state is (for example) a set of zero to two distinct pairs of the hub identifiers of the link mereology.

[8] A link traffic state space is (for example) the set of all link traffic states that a link may range over.

Observers for Atomic Parts

- Let the domain describer decide
  - that a **type**, A (or Δ), is **atomic**,
  - hence that it does not consists of sub-parts.
- Hence there are no **observer** to be associated with A (or Δ).

2.2.2. Composite Parts

- The domain describer has chosen to consider
  - a part (i.e., a **part type**)
  - to be a composite part (i.e., a **composite part type**).
- Now the domain describer has to analyse the types of the sub-parts of the composite part.
  - There may be just one *"kind of"* sub-part of a composite part[9],
  - or there may be more than one *"kind of"*[10].
- For each such **sub-part type**
  - the domain describer decides on
  - an appropriate, distinct **type name** and
  - a **sub-part observer** (i.e., a **function signature**).

[9] that is, only one sub-part type
[10] that is, more than one sub-part type

Example: 7 Container Vessels: Composite Parts. We bring pairs of informal, narrative description texts and formalisations.

- For a container vessel, say of type V, we have
  - *Narrative:*
    - A container vessel, v:V, consists of container bays, bs:BS.
    - A container bay, b:B, consists of container rows, rs:RS.
    - A container row, r:R, consists of container stacks, ss:SS.
    - A container stack, s:S, consists of a linearly indexed sequence of containers.
  - *Formalisation:*

```
type V,BS, value obs_BS: V→BS,
type B,RS, value obs_RS: B→RS,
type R,SS, value obs_CS: R→SS,
type SS,S, value obs_S: SS→S,
type S = C*.
```

2.2.3. Abstract Types, Sorts, and Concrete Types

- By an **abstract type**, or a **sort**, we shall understand a type
  - which has been given a name
  - but is otherwise undefined, that is,
    - is a space of undefined mathematical quantities,
      - where these are given properties
      - which we may express in terms of **axiom**s over sort (including **property**) values.

- By a **concrete type** we shall understand a type, T,
  - which has been given both a name
  - and a defining type expression of, for example the form
    - T = A-**set**,
    - T = A-**infset**,
    - T = A×B×···×C,
    - T = A*,
    - T = A^{ω},
    - T = A $\xrightarrow[m]{}$ B,
    - T = A→B,
    - T = A$\xrightarrow{\sim}$B, or
    - T = A|B|···|C.
  - where A, B, ..., C are **type names** or **type expressions**.

Example: 8 Container Bays. We continue Example 7 on Slide 79.

type Bs = BId $\overrightarrow{m}$ B,
value obs_Bs: BS→Bs,

type Rs = RId $\overrightarrow{m}$ R,
value obs_Rs: B→Rs,

type Ss = SId $\overrightarrow{m}$ S,
value obs_Ss: R→Ss,

type S = C*.

Observers for Composite Parts I/II

- Let the domain describer decide
  - that a type, A (or Δ), is composite
  - and that it consists of sub-parts of types B, C, ..., D.
- We can initially consider these types B, C, ..., D, as **abstract types**, or **sorts**, as we shall mostly call them.
- That means that there are the following formalisations:
  - **type** A, B, C, ..., D;
  - **value** obs_B: A→B, obs_C: A→C, ..., obs_D: A→D.

Observers for Composite Parts II/II

- We can also consider the types B, C, ..., D, as concrete types,
  - **type** Bc = TypBex, Cc = TypCex, ..., Dc = TypDex;
  - **value** obs_Bc: B→Bc, obs_Cc: C→Cc, ..., obs_Dc: D→Dc,
  - where TypBex, TypCex, ..., TypDex are type expressions as, for example, hinted at above.
- The prefix obs_ distinguishes part observers
  - from mereology observers (uid_, mereo_) and
  - attribute observers (attr_).

2.3. Properties

- Endurants have **properties**.
  - Properties are
    - what makes up a parts (and materials) and,
    - with **property values** distinguishes
      one part from another part and
      one material from another material.
  - We name properties.
    - **Properties** of **parts** and **materials** can be given distinct names.
    - We let these names also be the **property type name**.
    - Hence two parts (materials) of the same **part type** (**material type**)
      have the same set of **property type names**.

- Properties are all that distinguishes parts (and materials).
  - The part types (material types)
    in themselves do not express properties.
  - They express a class of parts (respectively materials).
  - All parts (materials) of the same type
  - have the same property types.
  - Parts (materials) of the different types
    have different sets of property types,

- For pragmatic reasons we distinguish between three kinds of properties:
  - ⟐ **unique identifiers**, ⟐ **mereology**, and ⟐ **attributes**.
- If you "remove" a property from a part
  - ⟐ it "looses" its (former) part type,
  - ⟐ to, in a sense, attain another part type:
    - ⟐ perhaps of another, existing one,
    - ⟐ or a new "created" one.
- *But we do not know* how to model *removal of a property* from an endurant value![11]

[11] And we see no need for describing such type-changes. Crude oil does not "morph" into fuel oil, diesel oil, kerosene and petroleum. Crude oil is consumed and the fractions result from distillation, for example, in an oil refinery.

Example: 9 Atomic Part Property Kinds.

- We distinguish between two kinds of persons:
  - ‘living persons’ and ‘deceased persons’;
  - they could be modelled by two different part types:
    - LP: living person, with a set of properties,
    - DP: deceased person, with a, most likely, different set of properties.
- All persons have been born, hence have a birth date (static attributes).
- Only deceased persons have a (well-defined) death date.

- All persons also have height and weight profiles (i.e., with dated values, i.e., **dynamic attributes**).
- One can always associate a **unique identifier** with each person.
- Persons are related, family-wise:
  - have parents (living or deceased),
  - (up to four known) grandparents, etc.,
  - may have brothers and sisters (zero or more),
  - may have children (zero or more), etc.
  - These family-relations can be considered the **mereology** for living persons. ■

2.3.1. Unique Identification

- We can assume that all parts
  - of the same part type
  - can be uniquely distinguished,
  - hence can be given unique identifications.

Unique Identification

- With every **part**, whether **atomic** or **composite** we shall associate a **unique part identifier**, of just **unique identifier**.
- Thus we shall associate with **part type** T
  - the **unique part type identifier type** TI,
  - and a **unique part identifier observer** function, **uid_TI**: T→TI.
- These associations (TI and **uid_TI**) are, however,
  - usually expressed explicitly,
  - whether they are ("subsequently") needed!

- The **unique identifier** of a part
  - can not be changed;
  - hence we can say that
    - no matter what a given **part**'s **property values** may take on,
    - that **part** cannot be confused with any other part.
- Since we can talk about this concept of **unique identification**,
  - we can **abstract**ly describe it —
    - and do not have to bother about any representation,
    - that is, whether we can humanly observe **unique identifiers**.

2.3.2. Mereology

- Mereology [CasatiVarzi1999][12] (from the Greek $\mu\epsilon\rho o\varsigma$ 'part') is
  - *the theory of part-hood relations:*
  - *of the relations of part to whole and*
  - *the relations of part to part within a whole.*

[12] Achille Varzi: Mereology, http://plato.stanford.edu/entries/mereology/

- For pragmatic reasons we choose to model the mereology of a domain in either of two ways
  - either by defining a **concrete type** as a model of the composite type,
  - or by endowing the sub-parts of the composite part with structures of **unique part identifier**s.

  or by suitable combinations of these.

Example: 10 Container Bays, Etcetera: Mereology. First we show how to model indexed set of container bays, rows and stacks for the previous example.

- *Narrative:*
  - (i) An indexed set, bs:BS, of bays is a bijective map from unique bay identifiers, bid:BId, to bays, b:B.
  - (ii) An indexed set, rs:RS, of rows is a bijective map from unique row identifiers, rid:RId, to rows, r:R.
  - (iii) An indexed set, ss:SS, of stacks is a bijective map from unique stack identifiers, sid:SId, to stacks, s:S.
  - (iv) A stack is a linear indexed sequence of containers, c:C.

- *Formalisation:*
  - (i) **type** BS, B, BId,
    Bs=BId $\overrightarrow{m}$ B,
    value obs_Bs: BS→Bs
    (or obs_Bs: BS→(BId $\overrightarrow{m}$ B));
  - (ii) **type** RS, R, RId,
    Rs=RId $\overrightarrow{m}$ R,
    value obs_Rs: RS→Rs
    (or obs_Rs: RS→(RId $\overrightarrow{m}$ R));
  - (iii) **type** SS, S, SId,
    Ss=SId $\overrightarrow{m}$ S;
  - (iv) **type** C,
    S=C*.

Example: 11 Transport Nets: Mereology.

- We show how to model a mereology
  - ⟐ for a transport net of links and hubs.
- *Narrative:*

(i) Hubs and links are endowed with unique hub, respectively link identifiers.

(ii) Each hub is furthermore endowed with a hub mereology which lists the unique link identifiers of all the links attached to the hub.

(iii) Each link is furthermore endowed with a link mereology which lists the set of the two unique hub identifiers of the hubs attached to the link.

(iv) Link identifiers of hubs and hub identifiers of links must designate hubs, respectively links of the net.

- *Formalisation:*

(i) **type** H, HI, L, LI;

value

(ii) uid_HI:H→HI, uid_LI:L→LI,

mereo_H:H→**LI-set**, mereo_L:L→**HI-set**,

axiom

(iii) $\forall$ l:L · **card** mereo_L(l) = 2

(iv) $\forall$ n:N, l:L, h:H · l ∈ obs_Ls(obs_LS(n)) ∧ h ∈ obs__Hs(obs_HS(n))
$\forall$ hi:HI · hi ∈ mereo_L(l) ⇒
$\exists$ h′:H·h′ ∈ obs_Hs(obs_HS(n)) ∧ uid_HI(h)=hi
∧ $\forall$ li:LI · li ∈ mereo_H(h) ⇒
$\exists$ l′:L·l′ ∈ obs_Ls(obs_LS(n)) ∧ uid_LI(l)=li

Concrete Models of Mereology

The concrete mereology example models above illustrated maps and sequences as such models.

- In general we can model mereologies in terms of
  - (i) sets: **A-set**,
  - (ii) Cartesians: $\mathsf{A}_1 \times \mathsf{A}_2 \times ... \times \mathsf{A}_m$,
  - (iii) lists: A^*, and
  - (iv) maps: $\mathsf{A} \xrightarrow[m]{} \mathsf{B}$,

  where A, A_1, A_2,...,A_m and B are types [we assume that they are type names] and where the A_1, A_2,...,A_m type names need not be distinct.
- Additional concrete types, say D, can be defined by concrete type definitions, D=E, where E is either of the type expressions (i–iv) given above or (v) $\mathsf{E}_i|\mathsf{E}_j$, or (vi) (E_i). where E_k (for suitable k) are either of (i–vi).
- Finally it may be necessary to express well-formedness predicates for concretely modelled mereologies.

Abstract Models of Mereology

Abstractly modelling mereology of parts, to us, means the following.

- With part types P_1, P_2, ..., P_n
  - is associated the unique part identifier types, Π_1, Π_2, ..., Π_n,
  - that is $\mathsf{uid\_}\Pi_i$: $\mathsf{P}_i \to \Pi_i$ for $i \in \{1..n\}$,
- and with each part type, P_i,
  - is then associated a **mereology** observer,
  - $\mathsf{mereo\_P}_i$: $\mathsf{P}_i \to \Pi_j$**-set**$\times\Pi_k$**-set**$\times...\times\Pi_\ell$**-set**,
- such that for all p:Pi we have that
  - if $\mathsf{mereo\_P}_i(\mathsf{p}) = (\{..., \pi_{j_a}, ...\},\{..., \pi_{k_b}, ...\},...,\{..., \pi_{\ell_c}, ...\})$
  - for $i, j, k, ...\ell \in \{1..n\}$
  - then part $\mathsf{p}:\mathsf{P}_i$ is connected (related) to the parts identified by ..., π_{j_a}, ... π_{k_b}, ..., π_{ℓ_c}, ...
- Finally it may be necessary to express axioms for abstractly modelled mereologies.

- How parts are related to other parts
  - is really a modelling choice, made by the domain describer.
  - It is not necessarily something
    that is obvious
    from observing the parts.

Example: 12 Pipelines: A Physical Mereology.

- Let pipes of a pipe line be composed with valves, pumps, forks and joins of that pipe line.
- Pipes, valves, pumps, forks and joins (i.e., pipe line units) are given unique pipe, valve, pump, fork and join identifiers.
- A mereology for the pipe line could now endow pipes, valves and pumps with
  - one input unique identifier, that of the predecessor successor unit, and
  - one output unique identifier, that of the successor unit.
- Forks would then be endowed with
  - two input unique identifiers, and
  - one out put unique identifier;
- and joins "the other way around".

Example: 13 Documents: A Conceptual Mereology.

- The mereology of, for example, this document,
  - that is, of the tutorial slides,

  is determined by the author.
- There unfolds, while writing the document,
  - a set of unique identifiers
  - for section, subsection, sub-subsection, paragraph, etc., units. and
  - between texts of a "paper version" of the document and slides of a "slides version" of the document.

- This occurs as the author necessarily
  - inserts cross-references,
    - in unit texts to other units, and
    - from unit texts to other documents (i.e., 'citations');
  - and while inserting "page" shifts for the slides.
- From those inserted references
  there emerges what we could call the document mereology. ■

- So the determination of a, or the, mereology of composite parts
  - is either given by physical considerations,
  - or are given by (more-or-less) logical (or other) considerations,
  - or by combinations of these.
- The "design" of mereologies improves with experience.

Example: 14 Pipelines: Mereology.

- We divert from our line of examples centered around
  - transport nets and, to some degree,
  - container transport,
- to bring a second, in a series of examples
  - on pipelines
  - (for liquid or gaseous material flow).

60. A pipeline consists of connected units, u:U.
61. Units have unique identifiers.
62. And units have mereologies, ui:UI:
    a pump, pu:Pu, pipe, pi:Pi, and valve, va:Va, units have one input connector and one output connector;
    b fork, fo:Fo, [join, jo:Jo] units have one [two] input connector[s] and two [one] output connector[s];
    c well, we:We, [sink, si:Si] units have zero [one] input connector and one [zero] output connector.
    d Connectors of a unit are designated by the unit identifier of the connected unit.
    e The auxiliary sel_UIs_in selector funtion selects the unique identifiers of pipeline units providing input to a unit;
    f sel_UIs_out selects unique identifiers of output recipients.

type
60. U = Pu | Pi | Va | Fo | Jo | Si | We
61. UI

value
61. uid_U: U → UI
62. mereo_U: U → UI**-set** × UI**-set**
62. wf_mereo_U: U → **Bool**
62. wf_mereo_U(u) ≡
62a. is_(Pu|Pi|Va)(u) → **card** iusi = 1 = **card** ouis,
62b. is_Fo(u) → **card** iuis = 1 ∧ **card** ouis = 2,
62b. is_Jo(u) → **card** iuis = 2 ∧ **card** ouis = 1,
62c. is_We(u) → **card** iuis = 0 ∧ **card** ouis = 1,
62d. is_Si(u) → **card** iuis = 1 ∧ **card** ouis = 0

62e. sel_UIs_in
62e. sel_UIs_in(u) ≡ **let** (iuis,_)=mereo_U(u) **in** iuis **end**
62f. sel_out: U → UI**-set**
62f. sel_UIs_out(u) ≡ **let** (_,ouis)=mereo_U(u) **in** ouis **end**

2.3.3. Attributes

- By an **attribute** of a part, **p**:P, we shall understand
  - some **observable property**, some **phenomenon**,
  - that is not a **sub-part** of **p**
  - but which characterises **p**
  - such that all parts of type P have that attribute and
  - such that "removing" that attribute from **p**
    (if such was possible)
    "renders" the type of **p** undefined.
- We ascribe types to attributes — not, therefore, to be confused with types of (their) parts.

Example: 15 Attributes.

- Example attributes of links of a transport net are:
  - length LEN,
  - location LOC,
  - state LΣ and
  - state space LΩ,
- Example attributes of a person could be:
  - name NAM,
  - birth date BID,
  - gender GDR,
  - weight WGT,
  - height HGT and
  - address ADR.

- Example attributes of a transport net could be:
  - name of the net,
  - legal owner of the net,
  - a map of the net,
  - etc.
- Example attributes of a container vessel could be:
  - name of container vessel,
  - vessel dimensions,
  - vessel tonnage (TEU),
  - vessel owner,
  - current stowage plan,
  - current voyage plan, etc.

2.3.3.1 Static and Dynamic Attributes

- By a **static attribute** we mean an attribute (of a part) whose value remains fixed.
- By a **dynamic attribute** we mean an attribute (of a part) whose value may vary.

Example: 16 Static and Dynamic Attributes.

- The length and location attributes of links are static.
- The state and state space attributes of links and hubs are dynamic.
- The birth-date attribute of a person is considered static.
- The height and weight attributes of a person are dynamic.
- The map of a transport net may be considered dynamic.
- The current stowage and the current voyage plans of a vessel should be considered dynamic. ■

Attribute Types and Observers, I/II

- Let the domain describer decide that parts of type P
- have attributes of types A_1, A_2, ..., A_t.
- This means that the following two formal clauses arise:
  - P, A_1, A_2, ..., A_t and
  - $\mathsf{attr\_A}_1$:P→A_1, $\mathsf{attr\_A}_2$:P→A_2, ..., $\mathsf{attr\_A}_t$:P→A_t

Attribute Types and Observers, II/II

- We may wish to annotate the list of **attribute type name**s as to whether they are static or dynamic, that is,
  - whether **value**s of some **attribute type**
  - vary or
  - remain fixed.
- The prefix **attr_** distinguishes **attribute observer**s from **part observer**s (**obs_**) and **mereology observer**s (**uid_**, **mereo_**).

2.4. Shared Attributes and Properties

- Shared attributes and shared properties
  - play an important rôle in understanding domains.

2.4.1. Attribute Naming

- We now *impose a restriction* on the naming of part attributes.
  - If attributes
    - of two different parts
    - of different part types
    - are identically named
    - then attributes must be somehow related, over time!
  - The "somehow" relationship must be described.

Example: 17 Shared Bus Time Tables.

- Let our domain include that of *bus time tables* for *busses* on a *bus transport net* as described in many examples in this seminar.
- We can then imagine a *bus transport net* as containing the following parts:
  - a *net*,
  - a *management system*,
  - a set of *busses*.
- For the sake of argument we consider a *bus time table* to be an attribute of the *bus management system*.
- And we also consider *bus time tables* to be attributes of *busses*.

- We think of the *bus time table* of a *bus*
  - to be that subset of the *bus management system bus time table*
  - which corresponds to the *bus*' *line number*.
- By saying that *bus time tables*
  - "corresponds" to well-defined subsets of
  - the *bus management system bus time table*

  we mean the following
  - The value of the *bus bus time table*
  - must at every time
  - be equal to the corresponding *bus line entry* in the *bus management system bus time table*. ■

2.4.2. Attribute Sharing

- We say that two **parts**,
  - of no matter what **part type**,
  - *share* an **attribute**,
  - if the following is the case:
    - the corresponding **part types** (and hence the **parts**)
    - have **identically named attributes**.
    - We say that **identically named attributes** designate **shared attributes**.
  - We do not present the corresponding invariants over **parts** with **identically named attributes**.

2.5. Shared Properties

- We say that two parts,
  - of no matter what part type,
  - *share* a property,
  - if either of the following is the case:
    - (i) either the corresponding part types (and hence the parts) have shared attributes;
    - (ii) or the unique identifier type of one of the parts potentially is in the mereology type of the other part;
    - (iii) or both.
  - We do not present the corresponding invariants over parts with shared properties.

2.6. Summary of Discrete Endurants

- We have introduced the endurant notions of atomic parts and composite parts:
  - part types,
  - part observers (**obs_**),
    - sort observers, and
    - concrete type observers;
  - part properties:
    - unique identifiers:
      - unique part identifier observers (**uid_**),
      - unique part identifier types,
    - mereology:
      - part mereologies,
      - part mereology observers (**mereo_**);

      and
    - attributes:
      - attribute observers (**attr_**) and
      - attribute types.

- The unique identifier property cannot necessarily be observed:
  - it is an abstract concept and
  - can be objectively "assigned".

  That is: unique identifiers are not required to be manifest.
- The mereology property also cannot usually be observed:
  - it is also an abstract concept,
  - but can be deduced from careful analysis.

  That is: mereology is not required to be manifest.
- The attributes can be observed:
  - usually by simple physical measurements,
  - or by deduction from (conceptual) facts,

  That is: attributes are usually only "indirectly" manifest.

Discrete Endurant Modelling I/II

Faced with a **phenomenon** the **domain analyser** has to decide

- whether that **phenomenon** is an **entity** or not, that is, whether
  - an **endurant** or
  - a **perdurant** or
  - neither.
- If **endurant** and if **discrete**, then whether it is
  - an atomic part or
  - a composite part.
- Then the **domain analyser** must decide on its type,
  - whether an **abstract type** (a **sort**)
  - or a **concrete type**, and, if so, which concrete form.

Discrete Endurant Modelling II/II

- Next the **unique identifier** and the **mereology** of the **part type** (e.g., P) must be dealt with:
  - **type name** (e.g., PI) for and, hence, **unique identifier observer name** (uid_PI) of **unique identifiers** and the
  - **part mereology types** and **mereology observer name** (mereo_P).
- Finally the designer must decide on the **part type attributes** for parts p:P:
  - for each such a suitable **attribute type name**, for example, A_i for suitable i,
  - a corresponding **attribute observer signature**, attr_A_i:P→A_i,
  - and whether an attribute is considered **static** or **dynamic**.

3. Domain Descriptions — Perdurants
3.1. States
3.1.1. General

- The characterisation of the concept of perdurant
  - mentioned time,
  - but implied a concept that we shall call state.
- In this version of this seminar
  - we shall not cover the modelling of time phenomena —
  - but we shall model that some actions occur before others.

- By a **state** we shall understand a collection of parts
  - such that each of these parts have dynamic attributes.
- We can characterise the state
  - by giving it a type,
  - for example, Σ, where the **state type definition**
  - $\Sigma = \mathsf{S}_1 \times \mathsf{S}_2 \times \cdots \times \mathsf{S}_s$
  - assembles the types of the parts making up the state —
  - where we assume that types S_1, S_2, ..., S_s
    - are types of parts
    - such that no S_i is a sub-part (of a subpart, ...) of some S_j,
    - and such that each part has **dynamic attributes**.

Example: 18 Net and Vessel States.

- We may consider a transport net, n:N, to represent a state (subject to the actions of maintaining a net: adding or removing a hub, adding or removing a link, etc.).
- We may also consider a hub, h:H, to represent a state (subject to the changing of a hub traffic signal: from red to green, etc., for specific directions through the hub).
- We may consider a container vessel to represent a state (subject to adding or removing containers from, respectively onto the top of stacks). ■

Thus the context determines how wide a scope the domain designer chooses for the state concept.

3.1.2. State Invariants

- States are subject to invariants.

Example: 19 State Invariants: Transport Nets.

- Net hubs and links may be inserted into and removed from nets.
- Thus is also introduced changes to the net mereology.
- Yet, the axioms, as illustrated in Example 11, must remain invariant.
- Likewise changes to dynamic attributes may well be subject to the holding of certain well-formedness constraints.
- We will illustrate this claim.

With each hub we associate a hub [link] state and a hub [link] state space.

63. A hub [link] state models the permissible routes from hub input links to (same) hub output links [respectively through a link].
64. A hub [link] state space models the possible set of hub [link] states that a hub [link] is intended to "occupy".

type

63. HΣ = (LI × LI)**-set**, LΣ = HI**-set**
64. HΩ = HΣ**-set**, LΩ = LΣ**-set**

value

63. attr_HΣ: H → HΣ, attr_LΣ: L → LΣ
64. attr_HΩ: H → HΩ, attr_LΩ: L → LΩ

65. For any given hub, h, with links, $l_1, l_2, ..., l_n$ incident upon (i.e., also emanating from) that hub, each hub state in the hub state space
66. must only contain such pairs of (not necessarily distinct) link identifiers that are identifiers of $l_1, l_2, ..., l_n$.

value

65. wf_HΩ: H → **Bool**
65. wf_HΩ(h) ≡ ∀ hσ:HΣ · hσ ∈ attr_HΩ(h) ⇒ wf_HΣ(h)

65. wf_HΣ: H → **Bool**
65. wf_HΣ(h) ≡
66. ∀ (li,li′):(LI×LI)·(li,li′)∈ attr_HΣ(h) ⇒ {li,li′} ⊆ mereo_H(h)

- This well-formedness criterion is part of the state invariant over nets.
  - We never write down the full state invariant for nets.
  - It is tacitly assume to be the collection of all the axioms and well-formedness predicates over net parts.

3.2. A Final Note on Endurant Properties

- The properties of parts and materials are fully captured by
  - (i) the unique part identifiers,
  - (ii) the part mereology and
  - (iii) the full set ofpart attributes and material attributes
- We therefore postulate a property function
  - when when applied to a part or a material
  - yield this triplet, (i–iii), of properties
  - in a suitable structure.

type
  Props = {|PI|**nil**|} × {|(PI-**set**×...×PI-**set**)|**nil**|} × Attrs
value
  props: Part|Material → Props

- where
  - Part stands for a part type,
  - Material stands for a material type,
  - PI stand for unique part identifiers and
  - PI-set×...×PI-set for part mereologies.
- The {|...|} denotes a proper specification language sub-type and **nil** denotes the empty type.

4. Discrete Perdurants
4.1. General

- From Wikipedia:
  - *Perdurant: Also known as occurrent, accident or happening.*
  - *Perdurants are those entities for which only a fragment exists if we look at them at any given snapshot in time.*
  - *When we freeze time we can only see a fragment of the perdurant.*
  - *Perdurants are often what we know as processes, for example 'running'.*
  - *If we freeze time then we only see a fragment of the running, without any previous knowledge one might not even be able to determine the actual process as being a process of running.*
  - *Other examples include an activation, a kiss, or a procedure.*

- We shall consider actions and events
  - to occur instantaneously,
  - that is, in time, but taking no time
- Therefore we shall consider actions and events to be perdurants.

4.2. Discrete Actions

- By a **function** we understand
  - a thing
  - which when **applied** to a **value**, called its **argument**,
  - **yields** a **value**, called its **result**.
- An **action** is
  - a **function**
  - **invoked** on a **state value**
  - and is one that potentially changes that value.

Example: 20 Transport Net and Container Vessel Actions.

- *Inserting* and *removing* hubs and links in a net are considered actions.
- *Setting* the traffic signals for a hub (which has such signals) is considered an action.
- *Loading* and *unloading* containers from or unto the top of a container stack are considered actions.

4.2.1. Action Signatures

- By an action signature we understand a quadruple:
  - a function name,
  - a function definition set type expression,
  - a total or partial function designator ($\rightarrow$, respectively $\xrightarrow{\sim}$), and
  - a function image set type expression:

    fct_name: A $\rightarrow$ Σ $(\rightarrow|\xrightarrow{\sim})$ Σ [$\times$ R],

  where $(X \mid Y)$ means either X or Y, and $[Z]$ means optional Z.

Example: 21 Action Signatures: Nets and Vessels.

insert_Hub: N$\rightarrow$H$\xrightarrow{\sim}$N;
remove_Hub: N$\rightarrow$HI$\xrightarrow{\sim}$N;
set_Hub_Signal: N$\rightarrow$HI$\xrightarrow{\sim}$H$\Sigma\xrightarrow{\sim}$N
load_Container: V$\rightarrow$C$\rightarrow$StackId$\xrightarrow{\sim}$V; and
unload_Container: V$\rightarrow$StackId$\xrightarrow{\sim}$(V$\times$C).

4.2.2. Action Definitions

- There are a number of ways in which to characterise an action.
- One way is to characterise its underlying function by a pair of predicates:
  - precondition: a predicate over function arguments — which includes the state, and
  - postcondition: a predicate over function arguments, a proper argument state and the desired result state.
  - If the precondition holds, i.e., is **true**, then the arguments, including the argument state, forms a proper 'input' to the action.
  - If the postcondition holds, assuming that the precondition held, then the resulting state [and possibly a yielded, additional "result" (R)] is as they would be had the function been applied.

Example: 22 Transport Nets: Insert Hub Action.

67. The insert action applies to a net and a hub and conditionally yields an updated net.
    a The condition is that there must not be a hub in the "argument" net with the same unique hub identifier as that of the hub to be inserted and
    b the hub to be inserted does not initially designate links with which it is to be connected.
    c The updated net contains all the hubs of the initial net "plus" the new hub.
    d and the same links.

value

67. insert_H: N → H $\xrightarrow{\sim}$ N
67. insert_H(n)(h) **as** n′, **pre**: pre_insert_H(n)(h), **post**: post_insert_H(n)(h)

67a. pre_insert_H(n)(h) ≡
67a. ∼∃ h′:H · h′ ∈ obs_Hs(n) ∧ uid_HI(h)=uid_HI(h′)
67b. ∧ mereo_H(h) = {}

67c. post_insert_H(n)(h)(n′) ≡
67c. obs_Hs(n) ∪ {h} = obs_Hs(n′)
67d. ∧ obs_Ls(n) = obs_Ls(n′)

- We refer to the notes accompanying these lectures.
- There you will find definitions of insert_link, remove_hub and remove_link action functions.

Modelling Actions, I/III

- The domain describer has decided that an entity is a perdurant and is, or represents an action: was *"done by an agent and intentionally under some description"* [Davidson1980].
  - The domain describer has further decided that the observed action is of a class of actions — of the "same kind" — that need be described.
  - By actions of the 'same kind' is meant that these can be described by the same function signature and function definition.

Modelling Actions, II/III

- First the domain describer must decide on the underlying **function signature**.
  - The **argument type** and the **result type** of the signature are those of either previously identified
    - parts and/or materials,
    - unique part identifiers, and/or
    - attributes.

Modelling Actions, III/III

- Sooner or later the domain describer must decide on the **function definition**.
  - The form must be decided upon.
  - For pre/post-condition forms it appears to be convenient to have developed, "on the side", a **theory of mereology** for the part types involved in the function signature.

4.3. Discrete Events

- By an **event** we understand
  - a state change
  - resulting indirectly from an unexpected application of a function,
  - that is, that function was performed "surreptitiously".
- Events can be characterised by a pair of (before and after) states, a predicate over these and, optionally, a **time** or **time interval**.
- Events are thus like actions:
  - change states,
  - but are usually
    - either caused by "previous" actions,
    - or caused by "an outside action".

Example: 23 Events.

- *Container vessel:* A container falls overboard
  sometimes between times t and t'.
- *Financial service industry:* A bank goes bankrupt
  sometimes between times t and t'.
- *Health care:* A patient dies
  sometimes between times t and t'.
- *Pipeline system:* A pipe breaks
  sometimes between times t and t'.
- *Transportation:* A link "disappears"
  sometimes between times t and t'.

4.3.1. Event Signatures

- An event signature
  - is a predicate signature
  - having an event name,
  - a pair of state types $(\Sigma \times \Sigma)$,
  - a total function space operator $(\rightarrow)$
  - and a **Bool**ean type constant:
  - evt: $(\Sigma \times \Sigma) \rightarrow \mathbf{Bool}$.
- Sometimes there may be a good reason
  - for indicating the type, ET, of an event cause value,
  - if such a value can be identified:
  - evt: ET $\times (\Sigma \times \Sigma) \rightarrow \mathbf{Bool}$.

4.3.2. Event Definitions

- An event definition takes the form of a predicate definition:
  - A predicate name and argument list, usually just a state pair,
  - an existential quantification
    - over some part (of the state) or
    - over some dynamic attribute of some part (of the state)
    - or combinations of the above
  - a pre-condition expression over the input argument(s),
  - an implication symbol ($\Rightarrow$), and
  - a post-condition expression over the argument(s).
- $\mathsf{evt}(\sigma,\sigma') = \exists\ (\mathsf{ev{:}ET}) \bullet \mathsf{pre\_evt}(\mathsf{ev})(\sigma) \Rightarrow \mathsf{post\_evt}(\mathsf{ev})(\sigma,\sigma')$.
- There may be variations to the above form.

Example: 24 Narrative of Link Event. The disappearance of a link in a net, for example due to a mud slide, or a bridge falling down, or a fire in a road tunnel, can, for example be described as follows:

68. Link disappearance is expressed as a predicate on the "before" and "after" states of the net. The predicate identifies the "missing" ℓink (!).
69. Before the disappearance of link ℓ in net n
    a the hubs h' and h'' connected to link ℓ
    b were connected to links identified by $\{l'_1, l'_2, \ldots, l'_p\}$ respectively $\{l''_1, l''_2, \ldots, l''_q\}$
    c where, for example, l'_i, l''_j are the same and equal to $\mathsf{uid\_}\Pi(\ell)$.

70. After link ℓ disappearance there are instead
    a two separate links, ℓ_i and ℓ_j, "truncations" of ℓ
    b and two new hubs h''' and h''''
    c such that ℓ_i connects h' and h''' and
    d ℓ_j connects h'' and h'''';
    e Existing hubs h' and h'' now have mereology
        i. $\{l'_1, l'_2, \ldots, l'_p\} \setminus \{\mathsf{uid\_\Pi}(\ell)\} \cup \{\mathsf{uid\_\Pi}(\ell_i)\}$ respectively
        ii. $\{l''_1, l''_2, \ldots, l''_q\} \setminus \{\mathsf{uid\_\Pi}(\ell)\} \cup \{\mathsf{uid\_\Pi}(\ell_j)\}$
71. All other hubs and links of n are unaffected. ■

Example: 25 Formalisation of Link Event. Continuing

Example 24 above:

68. link_disappearance: N × N → **Bool**
68. link_disappearance(n,n′) ≡
68. ∃ ℓ:L · pre_link_dis(n,ℓ) ⇒ post_link_dis(n,ℓ,n′)

69. pre_link_dis: N × L → **Bool**
69. pre_link_dis(n,ℓ) ≡ ℓ ∈ obs_Ls(n)

72. We shall "explain" *link disappearance* as the combined, instantaneous effect of
    a. first a **remove link** "event" where the **removed link** connected **hub**s hi_j and hi_k;
    b. then the **insert**ion of two new, "fresh" **hub**s, h_α and h_β;
    c. "followed" by the **insert**ion of two new, "fresh" links $\mathsf{l}_{j\alpha}$ and $\mathsf{l}_{k\beta}$ such that
        i. $\mathsf{l}_{j\alpha}$ connects hi_j and h_α and
        ii. $\mathsf{l}_{k\beta}$ connects hi_k and $\mathsf{h}_{k\beta}$

value

72. post_link_dis(n,ℓ,n$'$) $\equiv$
72a. **let** n$''$ = remove_L(n)(uid_L(ℓ)) **in**
72b. **let** h$_\alpha$,h$_\beta$:H · {h$_\alpha$,h$_\beta$}$\cap$obs_Hs(n)={} **in**
72b. **let** n$'''$ = insert_H(n$''$)(h$_\alpha$) **in**
72b. **let** n$''''$ = insert_H(n$'''$)(h$_\beta$) **in**
72c. **let** l$_{j\alpha}$,l$_{k\beta}$:L · {l$_{j\alpha}$,l$_{k\beta}$}$\cap$obs_Ls(n)={} **in**
72(c)i. **let** n$'''''$ = insert_L(n$''''$)(l$_{j\alpha}$) **in**
72(c)ii. n$'$ = insert_L(n$'''''$)(l$_{k\beta}$) **end end end end end end**

- We refer to the notes accompanying these lectures.
- There you will find definitions of insert_link, remove_hub and remove_link action functions.

Modelling Events I/II

- The domain describer has decided that an entity is a perdurant and is, or represents an event: occurred surreptitiously, that is, was not an action that was *"done by an agent and intentionally under some description"* [Davidson1980].
  - The domain describer has further decided that the observed event is of a class of events — of the "same kind" — that need be described.
  - By events of the 'same kind' is meant that these can be described by the same predicate function signature and predicate function definition.

Modelling Events, II/II

- First the domain describer must decide on the underlying **predicate function signature**.
  - The **argument type** and the **result type** of the signature are those of either previously identified
    - parts,
    - unique part identifiers, or
    - attributes.
- Sooner or later the domain describer must decide on the **predicate function definition**.
  - For predicate function definitions it appears to be convenient to have developed, "on the side", a **theory of mereology** for the part types involved in the function signature.

4.4. Discrete Behaviours

- We shall distinguish between
  - discrete behaviours (this section) and
  - continuous behaviours (Sect. ??).
- Roughly discrete behaviours
  - proceed in discrete (time) steps —
  - where, in this seminar, we omit considerations of time.
  - Each step corresponds to an action or an event or a time interval between these.
  - Actions and events may take some (usually inconsiderable time),
  - but the domain analyser has decided that it is not of interest to understand what goes on in the domain during that time (interval).
  - Hence the behaviour is considered discrete.

- Continuous behaviours
  - are continuous in the sense of the calculus of mathematical;
  - to qualify as a continuous behaviour time must be an essential aspect of the behaviour.
  - We shall treat continuous behaviours in Sect. 9.
- Discrete behaviours can be modelled in many ways, for example using
  - CSP [Hoare85+2004].
  - MSC [MSCall],
  - Petri Nets [m:petri:wr09] and
  - Statechart [Harel87].
- We refer to Chaps. 12–14 of [TheSEBook2wo].
- In this seminar we shall use RSL/CSP.

4.4.1. What is Meant by 'Behaviour'?

- We give two characterisations of the concept of 'behaviour'.
  - a "loose" one and
  - a "slanted one.
- A loose characterisation runs as follows:
  - by a **behaviour** we understand
    - a set of sequences of
    - **actions**, **events** and **behaviours**.

- A "slanted" characterisation runs as follows:
  - by a **behaviour** we shall understand
    - either a **sequential behaviour** consisting of a possibly infinite sequence of zero or more actions and events;
    - or one or more **communicating behaviours** whose **output actions** of one behaviour may **synchronise** and **communicate** with **input actions** of another behaviour; and
    - or two or more **behaviours** acting either as **internal non-deterministic behaviours** ($\sqcap$) or as **external non-deterministic behaviours** ($\Box$).

- This latter characterisation of behaviours
  - is "slanted" in favour of a CSP, i.e., a communicating sequential behaviour, view of behaviours.
  - We could similarly choose to "slant" a behaviour characterisation in favour of
    - Petri Nets, or
    - MSCs, or
    - Statecharts, or other.

4.4.2. Behaviour Narratives

- Behaviour narratives may take many forms.
  - A behaviour may best be seen as composed from several interacting behaviours.
    - Instead of narrating each of these,
    - as will be done in Example **??**,
    - one may proceed by first narrating the interactions of these behaviours.
  - Or a behaviour may best be seen otherwise,
    - for which, therefore, another style of narration may be called for,
    - one that "traverses the landscape" differently.
  - Narration is an art.
  - Studying narrations – and practice – is a good way to learn effective narration.

Example: 26 A Road Traffic System. We continue our long line of examples around transport nets. The present example interprets these as road nets.

4.4.2.1 Continuous Traffic

- For the road traffic system
  - ⟴ perhaps the most significant example of a behaviour
  - ⟴ is that of its traffic
    73. the continuous time varying discrete positions of vehicles, vp:VP[13],
    74. where time is taken as a dense set of points.

type

74. $\mathbb{cT}$
73. $\mathrm{cRTF} = \mathbb{cT} \rightarrow (\mathrm{V} \xrightarrow[m]{} \mathrm{VP})$

[13] For VP see Item 92a on Slide 169.

4.4.2.2 Discrete Traffic

- We shall model, not continuous time varying traffic, but

75. discrete time varying discrete positions of vehicles,
76. where time can be considered a set of linearly ordered points.

76. $\mathbb{dT}$
75. $\mathbb{dRTF} = \mathbb{dT} \xrightarrow{m} (\mathbb{V} \xrightarrow{m} \mathbb{VP})$

77. The road traffic that we shall model is, however, of vehicles referred to by their unique identifiers.

type

77. $\mathbb{RTF} = \mathbb{dT} \xrightarrow{m} (\mathbb{VI} \xrightarrow{m} \mathbb{VP})$

4.4.2.3 Time: An Aside

- We shall take a rather simplistic view of time [wayne.d.blizard.90,mctaggart-t0,prior68,J.van.Benthem.Log

78. We consider $\mathbf{d}\mathbb{T}$, or just $\mathbb{T}$, to stand for a totally ordered set of time points.
79. And we consider $\mathbb{TI}$ to stand for time intervals based on $\mathbb{T}$.
80. We postulate an infinitesimal small time interval δ.
81. $\mathbb{T}$, in our presentation, has lower and upper bounds.
82. We can compare times and we can compare time intervals.
83. And there are a number of "arithmetics-like" operations on times and time intervals.

type

78. $\mathbb{T}$
79. $\mathbb{TI}$

value

80. δ:$\mathbb{TI}$
81. $\mathbb{MIN}, \mathbb{MAX}$: $\mathbb{T} \rightarrow \mathbb{T}$
81. $<,\leq,=,\geq,>$: $(\mathbb{T}\times\mathbb{T})|(\mathbb{TI}\times\mathbb{TI}) \rightarrow$ **Bool**
82. $-$: $\mathbb{T}\times\mathbb{T} \rightarrow \mathbb{TI}$
83. $+$: $\mathbb{T}\times\mathbb{TI},\mathbb{TI}\times\mathbb{T} \rightarrow \mathbb{T}$
83. $-,+$: $\mathbb{TI}\times\mathbb{TI} \rightarrow \mathbb{TI}$
83. $*$: $\mathbb{TI}\times$**Real** $\rightarrow \mathbb{TI}$
83. $/$: $\mathbb{TI}\times\mathbb{TI} \rightarrow$ **Real**

84. We postulate a global clock behaviour which offers the current time.

85. We declare a channel clk_ch.

value

84. clock: $\mathbb{T} \rightarrow$ **out** clk_ch **Unit**
84. clock(t) $\equiv$ ... clk_ch!t ... clock(t $\lceil\rceil$ t+δ)

channnel

85. clk_ch:$\mathbb{T}$

4.4.2.4 Road Traffic System Behaviours

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

value

```
86.  trs() =
86a.     ‖ {veh(uid_V(v))(v)|v:V·v ∈ vs}
86b.     ‖ mon(m)([ ])
```

- where the "extra" monitor argument ([ ])
  - records the discrete road traffic, RTF,
  - initially set to the empty map (of, "so far no road traffic"!).

4.4.2.5 Globally Observable Parts

- There is given

87. a net, n:N,
88. a set of vehicles, vs:V-set, and
89. a monitor, m:M.

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

value

87. n:N = obs_N(Δ)
87. ls:L-set = obs_Ls(obs_LS(n)), hs:H-set = obs_Hs(obs_HS(n)),
87. lis:LI-set = {uid_L(l)|l:L·l ∈ ls}, his:HI-set = {uid_H(h)|h:H·h ∈ hs}
88. vs:V-set = obs_Vs(obs_VS(obs_F(Δ))), vis:V-set = {uid_V(v)|v:V·v ∈
89. m:obs_M(Δ)

4.4.2.6 Channels

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

90. Thus we declare a set of channels indexed by the unique identifiers of vehicles and communicating vehicle positions; and
91. a single clock to monitor channel.

channel

90. {vm_ch[vi]|vi:VI·vi ∈ vis}:VP
91. clkm_ch:dT

4.4.2.7 An Aside: Attributes of Vehicles

92. Dynamic attributes of vehicles include
    a position
        i. at a hub (about to enter the hub — referred to by the link it is coming from, the hub it is at and the link it is going to, all referred to by their unique identifiers or
        ii. some fraction "down" a link (moving in the direction from a from hub to a to hub — referred to by their unique identifiers)
        iii. where we model fraction as a real between 0 and 1 included.
    b velocity, acceleration, etcetera.

type

92a. VP = atH | onL
92(a)i. atH :: fli:LI × hi:HI × tli:LI
92(a)ii. onL :: fhi:HI × li:LI × frac:FRAC × thi:HI
92(a)iii. FRAC = **Real**, **axiom** ∀ frac:FRAC • 0 ≤ frac ≤ 1
92b. Vel, Acc, ...

4.4.2.8 Behaviour Signatures

93. The road traffic system behaviour, rts, takes no arguments; and "behaves", that is, continues forever.
94. The vehicle behaviours are indexed by the unique identifier, uid_V(v):VI, the vehicle part, v:V and the vehicle position; offers communication to the monitor behaviour; and behaves "forever".
95. The monitor behaviour takes monitor part, m:M, as argument and also the discrete road traffic, drtf:dRTF; the behaviour otherwise runs forever.

value

93. rts: **Unit** → **Unit**
94. veh: vi:VI → v:V → VP → **out** vm_ch[vi] **Unit**
95. mon: m:M → RTF → **in** {vm_ch[vi]|vi:VI·vi ∈ vis},clkm_ch **Unit**

4.4.2.9 The Vehicle Behaviour

96. A vehicle process
    - is indexed by the unique vehicle identifier vi:VI,
    - the vehicle "as such", v:V and
    - the vehicle position, vp:VP.

    The vehicle process communicates
    - with the monitor process on channel vm[vi]
    - (sends, but receives no messages), and
    - otherwise evolves "infinitely" (hence **Unit**).

97. We describe here an abstraction of the vehicle behaviour **at** a Hub (**hi**).
    a Either the vehicle remains at that hub informing the monitor,
    b or, internally non-deterministically,
        i. moves onto a link, **tli**, whose "next" hub, identified by **thi**, is obtained from the mereology of the link identified by **tli**;
        ii. informs the monitor, on channel **vm[vi]**, that it is now on the link identified by **tli**,
        iii. whereupon the vehicle resumes the vehicle behaviour positioned at the very beginning (**0**) of that link,
    c or, again internally non-deterministically,
    d the vehicle "disappears — off the radar" !

97. veh(vi)(v)(vp:atH(fli,hi,tli)) ≡
97a. vm_ch[vi]!vp ; veh(vi)(v)(vp)
97b. ⌈⌉
97(b)i. **let** {hi′,thi}=mereo_L(get_L(tli)(n)) **in assert:** hi′=hi
97(b)ii. vm_ch[vi]!onL(tli,hi,0,thi) ;
97(b)iii. veh(vi)(v)(onL(tli,hi,0,thi)) **end**
97c. ⌈⌉
97d. **stop**

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

96. veh(vi)(v)(vp:onL(fhi,li,f,thi)) ≡
98a. vm_ch[vi]!vp ; veh(vi)(v)(vp)
98b. ⌈⌉
98c. **if** f + δ<1
98(c)i. **then** vm_ch[vi]!onL(fhi,li,f+δ,thi) ;
98(c)i. veh(vi)(v)(onL(fhi,li,f+δ,thi))
98(c)ii. **else let** li′:LI·li′ ∈ mereo_H(get_H(thi)(n)) **in**
98(c)iiA. vm_ch[vi]!atH(li,thi,li′);
98(c)iiB. veh(vi)(v)(atH(li,thi,li′)) **end end**
99. ⌈⌉
100. **stop**

4.4.2.10 The Monitor Behaviour

101. The monitor behaviour evolves around the attributes of an own "state", m:M, a table of traces of vehicle positions, while accepting messages about vehicle positions and otherwise progressing "in[de]finitely".
102. Either the monitor "does own work"
103. or, internally non-deterministically accepts messages from vehicles.
    a A vehicle position message, vp, may arrive from the vehicle identified by vi.
    b That message is appended to that vehicle's movement trace,
    c whereupon the monitor resumes its behaviour —
    d where the communicating vehicles range over all identified vehicles.

101. mon(m)(rtf) ≡
102. mon(own_mon_work(m))(rtf)
103. ⌈
103a. ▯ { **let** ((vi,vp),t) = (vm_ch[vi]?,clkm_ch?), **in**
103b. **let** rtf′ = rtf † [t ↦ rtf(max **dom** rtf) † [vi ↦ vp]] **in**
103c. mon(m)(rtf′) **end**
103d. **end** | vi:VI · vi ∈ vis }

102. own_mon_work: M → TBL → M

- We do not describe the clock behaviour by other than stating that it continually offers the current time on channel clkm_ch. ■

Example: 27 A Pipeline System Behaviour.

- We consider pipeline system units to represent also the following behaviours:
  - For each kind of unit, cf. Example 14 on Slide 105, there are the unit processes:
    - unit,
    - well (Item 62c on Slide 106),
    - pipe (Item 62a),
    - pump (Item 62a),
    - valve (Item 62a),
    - fork (Item 62b),
    - join (Item 62b) and
    - sink (Item 62d on Slide 106).

channel
  { pls_u_ch[ui]:ui:UI•i ∈ UIs(pls) } MUPLS
  { u_u_ch[ui,uj]:ui,uj:UI•{ui,uj}⊆UIs(pls) } MUU
type
  MUPLS, MUU
value
  pipeline_system: PLS → **in**,**out** { pls_u_ch[ui]:ui:UI•i ∈ UIs(pls) } **Unit**
  pipeline_system(pls) ≡ ‖ { unit(u)|u:U•u ∈ obs_Us(pls) }
  unit: U → **Unit**
  unit(u) ≡
62c. is_We(u) → well(uid_U(u))(u),
62a. is_Pu(u) → pump(uid_U(u))(u),
62a. is_Pi(u) → pipe(uid_U(u))(u),
62a. is_Va(u) → valve(uid_U(u))(u),
62b. is_Fo(u) → fork(uid_U(u))(u),
62b. is_Jo(u) → join(uid_U(u))(u),
62d. is_Si(u) → sink(uid_U(u))(u)

- We illustrate essentials of just one of these behaviours.

62b. fork: ui:UI → u:U → **out**,**in** pls_u_ch[ui],
62b. **in** { u_u_ch[iui,ui] | iui:UI · iui ∈ sel_UIs_in(u) }
62b. **out** { u_u_ch[ui,oui] | iui:UI · oui ∈ sel_UIs_out(u) } **Unit**
62b. fork(ui)(u) ≡
62b. **let** u′ = core_fork_behaviour(ui)(u) **in**
62b. fork(ui)(u′) **end**

- The core_fork_behaviour(ui)(u) distributes
  - what oil (or gas) in receives,
    - on the one input sel_UIs_in(u) = {iui},
    - along channel u_u_ch[iui]
  - to its two outlets
    - sel_UIs_out(u) = {oui_1,oui_2},
    - along channels u_u_ch[oui_1], u_u_ch[oui_2].

- The core_fork_behaviour(ui)(u) also communicates with the pipeline_system behaviour.
  - What we have in mind here is to model a traditional supervisory control and data acquisition, SCADA system.

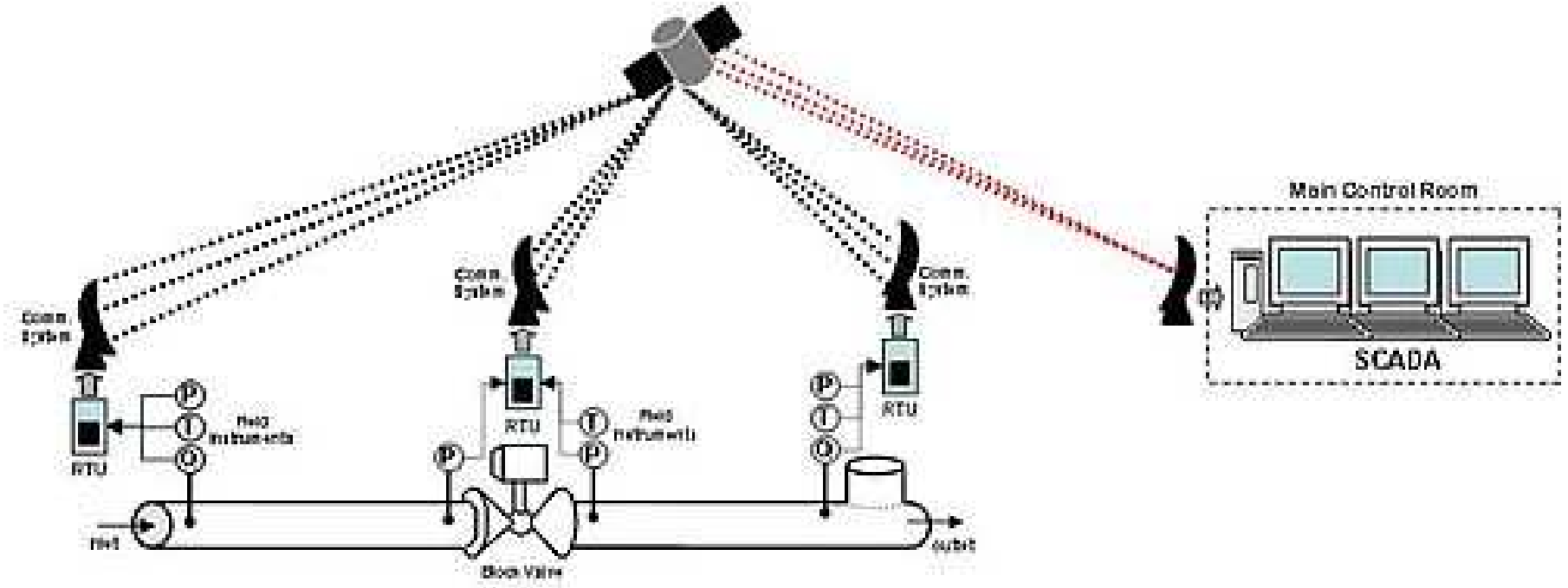

Figure 1: A supervisory control and data acquisition system

- SCADA is then part of the pipeline_system behaviour.

104.

104. pipeline_system: PLS → **in**,**out** { pls_u_ch[ui]:ui:UI·i ∈ UIs(pls) } **Uni**
104. pipeline_system(pls) ≡ scada(props(pls)) ‖ ‖{ unit(u)|u:U·u ∈ obs_Us(p

- props was defined on Slide 131.

105. scada non-deterministically (internal choice, ⌈⌉), alternates between continually

    a doing own work,

    b acquiring data from pipeline units, and

    c controlling selected such units.

type

105. Props

value

105. scada: Props → **in**,**out** { pls_ui_ch[ui] | ui:UI·ui ∈ ∈ uis } **Unit**

105. scada(props) ≡

105a. scada(scada_own_work(props))

105b. ⌈⌉ scada(scada_data_acqui_work(props))

105c. ⌈⌉ scada(scada_control_work(props))

- We leave it to the listeners imagination to describe scada_own_work.

106. The scada_data_acqui_work

a non-deterministically, external choice, [], offers to accept data,
b and scada_input_updates the scada state —
c from any of the pipeline units.

```
value
106.  scada_data_acqui_work: Props → in,out { pls_ui_ch[ui] | ui:UI·ui ∈ ∈
106.  scada_data_acqui_work(props) ≡
106a.     [] { let (ui,data) = pls_ui_ch[ui] ? in
106b.          scada_input_update(ui,data)(props) end
106c.          | ui:UI · ui ∈ uis }

106b. scada_input_update: UI × Data → Props → Props
type
106a. Data
```

107. The scada_control_work

a analyses the scada state (props) thereby selecting a pipeline unit, ui, and the controls, ctrl, that it should be subjected to;

b informs the units of this control, and

c scada_output_updates the scada state.

```
107.  scada_control_work: Props → in,out { pls_ui_ch[ui] | ui:UI·ui ∈ ∈ uis
107.  scada_control_work(props) ≡
107a.     let (ui,ctrl) = analyse_scada(ui,props) in
107b.     pls_ui_ch[ui] ! ctrl ;
107c.     scada_output_update(ui,ctrl)(props) end

107c. scada_output_update UI × Ctrl → Props → Props
type
107a. Ctrl
```

Modelling Behaviours, I/II

- The domain describer has decided that an entity is a perdurant and is, or represents a behaviour.
  - The domain describer has further decided that the observed behaviour is of a class of behaviours — of the "same kind" — that need be described.
  - By behaviours of the 'same kind' is meant that these can be described by the same channel declarations, function signature and function definition.

Modelling Behaviours, II/II

- First the domain describer must decide on the underlying **function signature**.
  - It must be decided which synchronisation and communication
    - inputs and
    - outputs

    this behaviour requires, i.e., the **in,out** clause of the signature,
  - that also includes the "discovery" of necessary **channel declarations**.
- Finally the **function definition** must be decided upon.

5. Seminar Conclusion

5.1. Other Work on Domain Analysis

- Our comparison hinges on basically the following two facets:
  - domain analysis and
  - domain description.
- We shall see that the former term, seen across the surveyed literature,
  - covers techniques that are claimed used in many steps of software engineering,
  - but that they seldom, if ever, involve formal concept analysis as we understand it.

5.1.1. An Enumeration

- Formal Concept Analysis: Ganter & Will — **Mathematics**
- Miscellaneous Directions — **Software Engineering**
  - Business Process [Re-]engineering, BPE, BPRE
  - Ontological Engineering
  - Knowledge and Knowledge Engineering, KE
  - Prieto-Dĩaz's Domain Analysis
  - Software Product Line Engineering
  - M.A. Jackson's Problem Frames
  - Domain Specific Software Architectures, DSS
  - Domain Driven Design, DDD
  - Feature-oriented Domain Analysis, FODA
  - Unified Modelling Language, UML

5.1.2. Summary of Comparisons

- It should now be clear from the above that there are basically two notions from above that relate to our notion of **domain analysis**.
  - (i) Prieto-Dĩaz's notion of *'Domain Analysis'*, and
  - (ii) Jackson's notion of *Problem Frames*.
- But it should also be clear that none of the surveyed literature,
  - except, of course, Ganter & Wille's
    `[GanterWille:ConceptualAnalysis1999]`
    Formal Concept Analysis, Mathematical Foundations,
  - covers our notion of **domain analysis**
  - as it hinges crucially on Ganter & Wille's **formal concept analysis**.

5.2. From Domains to Requirements

- Requirements "reside" in the domain.
- That is, we base requirements development of domain analysis.
  - From a domain description we derive, systematically
    - the domain requirements
      which can all be expressed using terms only of the domain, and
    - the interface requirements
      which can be expressed using terms
      both of the domain and the machine (hardware + software) ro be designed.

Abstraction and 

Modelling

Languages and 

Systems

Domains, Requirements and 

System Design

5.3. What Have We Achieved

- We have sketched the concepts of
  - domains,
  - domain analysis and
  - domain descriptions.
- We have suggested that
  - domain descriptions be the
  - basis for requirements prescriptions.
- You can read more, much more, at
  - **http://www2.imm.dtu.dk/~dibj/dsae-a.pdf**

5.4. Acknowledgements

- The organisers of APSEC 2012 for accepting this tutorial.