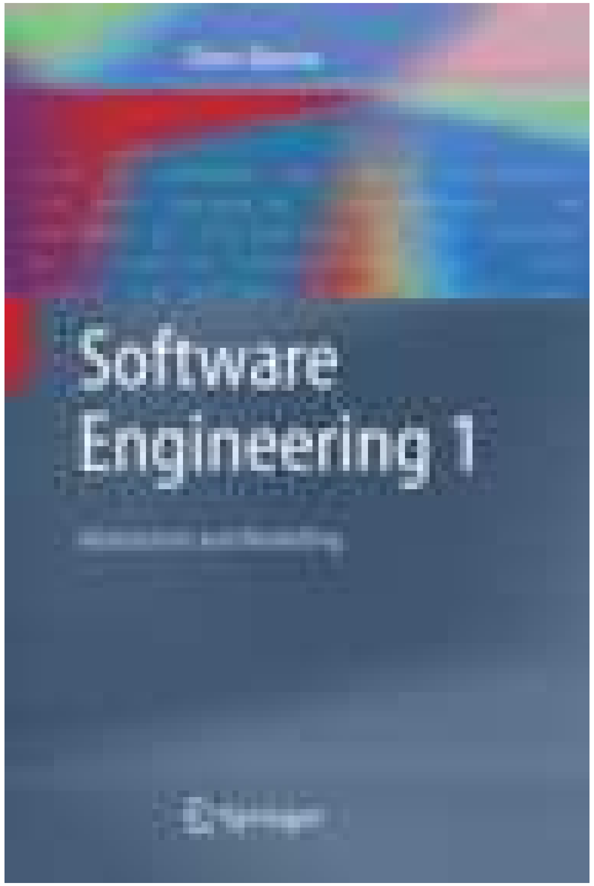
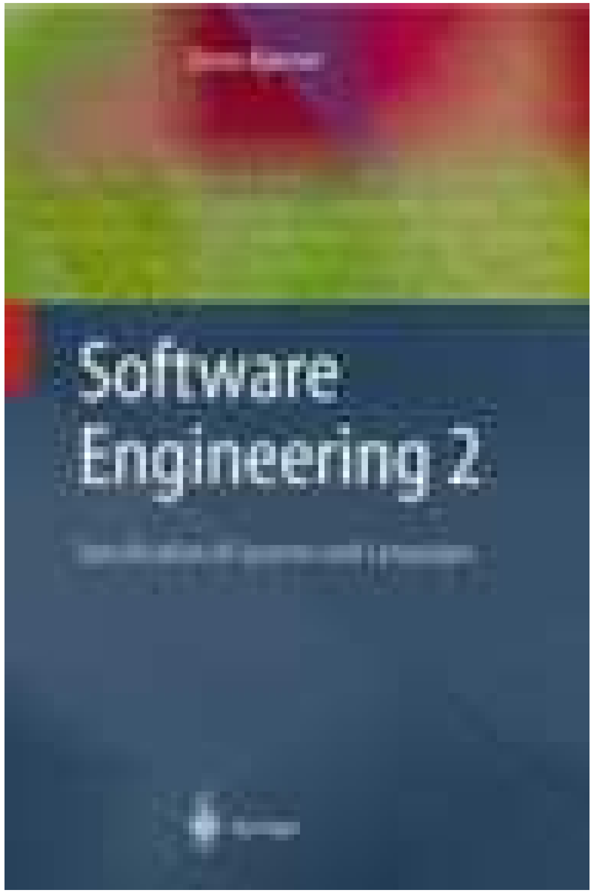
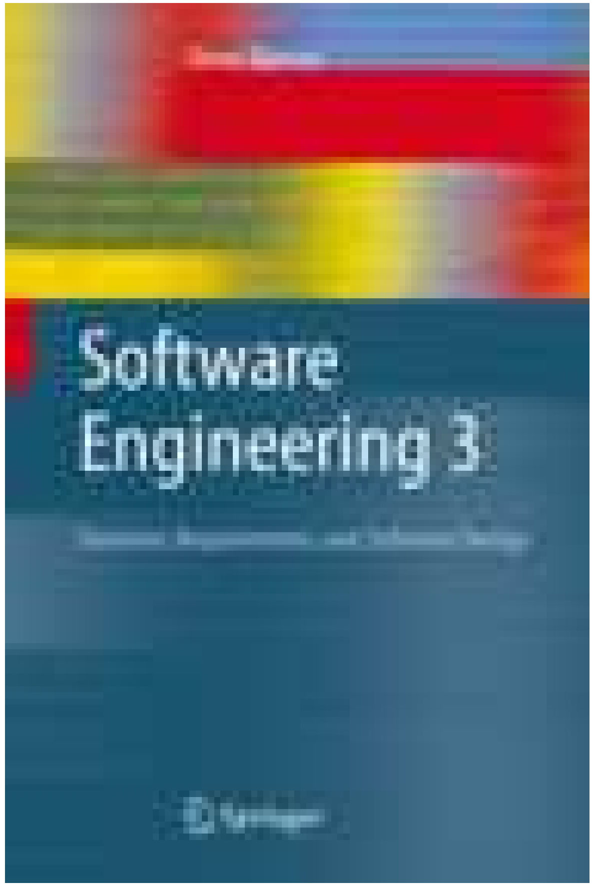

Software Engineering 1


Software Engineering 2


Software Engineering 3

WELCOME

Domain Science & Engineering

A Precursor for Requirements Engineering

Dines Bjørner

DTU Informatics

August 25, 2012: 17:17

Lecture Schedule

• Lecture 1:	09:00–09:40 + 09:50–10:30
• Lecture 2:	11:00–11:40 + 11:50–12:30
• Lecture 3:	14:00–14:40 + 14:50–15:30
• Lecture 4:	16:00–16:40 + 16:50–17:30

Lecture 1: 9:00–9:40 + 9:50–10:30
Introduction and Main Example

Summary

- This seminar covers
  - a **new science & engineering of domains** as well as
  - a **new foundation for software development**.

We treat the latter first.

- Instead of commencing with requirements engineering,
  - whose pursuit may involve repeated,
  - but unstructured forms of domain analysis,
  - we propose a predecessor phase of domain engineering.
- That is, we single out domain analysis as an activity to be pursued prior to requirements engineering.

- In emphasising **domain engineering** as a predecessor phase
  - we, at the same time, introduce a number of facets
  - that are **not present,** we think,
  - in current software engineering studies and practices.
- **One facet** is **the construction of separate domain descriptions.**
  - Domain descriptions are void of any reference to requirements
  - and encompass the **modelling** of **domain phenomena**
  - without regard to their being computable.

- **Another facet** is **the pursuit of domain descriptions as a free-standing activity.**
  - In this seminar we emphasize domain description development need not lead to software development.
  - This gives a new meaning to business process engineering, and should lead to
    - a deeper understanding of a domain
    - and to possible non-IT related business process re-engineering of areas of that domain.

- In this seminar we shall investigate
  - a method for analysing domains,
  - for constructing domain descriptions
  - and some emerging scientific bases.

- **Our contribution** to **domain analysis** is
  - that we view domain analysis
  - as a variant of formal concept analysis `[Wille:ConceptualAnalysis1999]`,
    - a contribution which can be formulated by the "catch phrase"
    - *domain entity types and signatures form a Galois connection*,
  - and further contribute with a methodology of
  - necessary corresponding principles and techniques of domain analysis.

- Those corresponding principles and techniques hinge on our view of domains as having the following **ontology**.
  - There are the **entities** that we can describe and then there is "the rest" which we leave un-described.
  - We **analyse** entities into
    - **endurant entities** and
    - **perdurant entities** ,

    that is,
    - **parts** and **materials** as **endurant entities** and
    - **discrete actions**, **discrete events** and **behaviours** as **perdurant entities** , respectively.
- Another way of looking at **entities** is as
  - **discrete entities** , or as
  - **continuous entities**.

- We also contribute to the **analysis** of **discrete endurants** in terms of the following notions:
  - ⟐ **part types** and **material types**,
  - ⟐ **part unique identifiers**,
  - ⟐ **part mereology** and
  - ⟐ **part attributes** and **material attributes** and
  - ⟐ **material laws**.
- Of the above we point to the introduction, into **computing science** and **software engineering** of the notions of
  - ⟐ **materials** and
  - ⟐ **continuous behaviours**

as novel.

- The example formalisations are expressed in
  - `RAISE [RaiseMethod]`
- but could as well have been expressed in for example
  - `Alloy [alloy]`,
  - `Event B [JRAbrial:TheBBooks]` ,
  - `VDM [e:db:Bj78bwo,e:db:Bj82b,JohnFitzgerald+PeterGormLarsen]` or
  - `Z [m:z:jd+jcppw96]`.

1. Introduction

- This is primarily a **methodology** paper.
- By a **method**$_\delta$ we shall understand
  - a set of **principles**
  - for **selecting** and **applying**
  - a number of **techniques** and **tools**
  - in order to **analyse** a **problem**
  - and **construct** an **artefact**.
- By **methodology**$_\delta$ we shall understand
  - the study and knowledge about methods.

- This seminar contributes to
  - the study and knowledge
  - of software engineering development methods.
- Its contributions are those of suggesting and exploring
  - domain engineering and
  - domain engineering as a basis for requirements engineering.
- We are not saying
  - *"thou must develop software this way"*,
- but we do suggest
  - that since it is possible
  - and makes sense to do so
  - it may also be wise to do so.

1.1. Domains: Some Definitions

- By a **domain**$_\delta$ we shall here understand
  - an area of human activity
  - characterised by observable phenomena:
    - **entities**
      - whether **endurants** (manifest **parts** and **materials**)
      - or **perdurants** (**actions**, **events** or **behaviours**),
    - whether
      - **discrete** or
      - **continuous**;
    - and of their **properties**.

Example: 1 Some Domains Some examples are:

air traffic,
airport,
banking,
consumer market,
container lines,
fish industry,
health care,
logistics,
manufacturing,
pipelines,
securities trading,
transportation
etcetera. ■

1.1.1. Domain Analysis

- By domain analysis$_\delta$ we shall understand
  - an inquiry into the domain,
  - its entities
  - and their properties.

Example: 2 A Container Line Analysis.

We omit enumerating entity properties.

- *parts:*
  - container,
  - vessel,
  - terminal port, etc.;
- *actions:*
  - container loading,
  - container unloading,
  - vessel arrival in port, etc.;
- *events:*
  - container falling overboard;
  - container afire;
  - etc.;
- *behaviour:*
  - vessel voyage,
  - across the seas,
  - visiting ports, etc.

Length of a **container** is a container *property*.
Name of a **vessel** is a vessel *property*.
Location of a **container terminal port** is a port *property*.

1.1.2. Domain Descriptions

- By a domain description$_\delta$ we shall understand
  - a narrative description
  - tightly coupled (say line-number-by-line-number)
  - to a formal description.
- To develop a domain description
  requires a thorough amount of domain analysis.

Example: 3 A Transport Domain Description.

- *Narrative:*
  - a transport net, n:N, consists of an aggregation of hubs, hs:HS, which we "concretise" as a set of hubs, **H-set**, and an aggregation of links, ls:LS, that is, a set **L-set**,
- *Formalisation:*
  - **type** N, HS, LS, Hs = H-**set**, Ls = L-**set**, H, L
    value
    obs_HS: N→HS,
    obs_LS: N→LS.
    obs_Hs: HS→H-**set**,
    obs_Ls: LS→L-**set**.

1.1.3. Domain Engineering

- By **domain engineering**$_\delta$ we shall understand
  - the **engineering** of a domain description,
  - that is,
    - the rigorous construction of domain descriptions, and
    - the further analysis of these, creating **theories of domains**.

- The size, structure and complexity of interesting domain descriptions is usually such as to put a special emphasis on engineering:
  - the management and organisation of several, typically 5–6 collaborating domain describers,
  - the ongoing check of description quality, completeness and consistency, etcetera.

1.1.4. Domain Science

- By **domain science**$_\delta$ we shall understand
  - two things:
    - the general study and knowledge of
      - how to create and handle domain descriptions
      - (a general theory of domain descriptions)

      and
    - the specific study and knowledge of a particular domain.
  - The two studies intertwine.

1.2. The Triptych of Software Development

- We suggest a "dogma":
  - before **software** can be **designed** one must understand[1] the **requirements**; and
  - before **requirements** can be expressed one must understand[2] the **domain**.
- We can therefore view software development as ideally proceeding in three (i.e., TripTych) phases:
  - an initial phase of **domain engineering**, followed by
  - a phase of **requirements engineering**, ended by
  - a phase of **software design**.

[1] Or maybe just: have a reasonably firm grasp of
[2] See previous footnote!

- In the **domain engineering phase** ($\mathcal{D}$)
  - a domain is analysed, described and "theorised",
  - that is, the beginnings of a **specific domain theory** is established.
- In the **requirements engineering phase** ($\mathcal{R}$)
  - a **requirements prescription** is constructed —
  - significant fragments of which are "derived",
  - systematically, from the **domain description**.
- In the **software design phase** ($\mathcal{S}$)
  - a **software design**
  - is derived, systematically, rigorously or formally,
  - from the **requirements prescription**.
- Finally the $\mathcal{S}$oftware is proven correct with respect to the $\mathcal{R}$equirements under assumption of the $\mathcal{D}$omain: $\mathcal{D}, \mathcal{S} \models \mathcal{R}$.

- By a **machine**$_\delta$ we shall understand the **hardware** and **software** of a target, i.e., a required **IT system**.
- In [dines:ugo65:2008,psi2009,Kiev:2010ptI] we indicate how one can “derive” significant parts of requirements from a suitably comprehensive domain description – basically as follows.
  - **Domain projection**: from a domain description one **project**s those areas that are to be somehow manifested in the software.
  - **Domain initialisation**: for that resulting projected requirements prescription one **initialise**s a number of part types as well as action and behaviour definitions, from less **abstract** to more **concrete**, specific types, respectively definitions.

- **Domain determination:** hand-in-hand with domain initialisation a[n interleaved] stage of making values of types less **non-deterministic**, i.e., more **deterministic**, can take place.
- **Domain extension:** Requirements often arise in the context of new business processes or technologies either placing old or replacing human processes in the domain. Domain extension is now the 'enrichment' of the domain requirements, so far developed, with the description of these new business processes or technologies.
- Etcetera.

- The result of this part of "**requirements derivation**" is the **domain requirements**.

- A set of domain-to-requirements operators similarly exists for constructing **interface requirements**
  - from the domain description and,
  - independently, also from knowledge of the **machine**
  - for which the required IT system is to be developed.
- We illustrate the **techniques** of **domain requirements** and **interface requirements** in Sect. 8.
- Finally **machine requirements** are "derived"
  - from just the knowledge of the **machine**,
  - that is,
    - the target hardware and
    - the software system tools for that hardware.

- When you review this section ('A Triptych of Software Development')
  - then you will observe how 'the domain'
  - predicates both the requirements
  - and the software design.
- For a specific domain one may develop
  - many (thus related) requirements
  - and from each such (set of) requirements
  - one may develop many software designs.
- We may characterise this multitude of domain-predicated requirements and designs as a **product line** [dines-maurer].
- You may also characterise domain-specific developments as representing another 'definition' of **domain engineering**.

1.3. Issues of Domain Science & Engineering

- We specifically focus on the following issues of domain science &[3] engineering:
  - (i) which are the "things" to be described[4],
  - (ii) how to analyse these "things" into description structures[5],
  - (iii) how to describe these "things" informally and formally,
  - (iv) how to further structure descriptions[6], and a further study of
  - (v) mereology[7].

[3] When we put '&' between two terms that the compound term forms a whole concept.

[4] endurants [manifest entities henceforth called parts and materials] and perdurants [actions, events, behaviours]

[5] atomic and composite, unique identifiers, mereology, attributes

[6] *intrinsics, support technology, rules & regulations, organisation & management, human behaviour* etc.

[7] the study and knowledge of parts and relations of parts to other parts and a "whole".

1.4. Structure of Paper

- First (Sect. 1) we introduce the problem. And that was done above.
- Then, in (Sects. 4–6)
  - we bring a rather careful analysis of
  - the concept of the **observable, manifest phenomena**
  - that we shall refer to as **entities**.
- We strongly think that these sections of this seminar
  - brings, to our taste, a simple and elegant
  - reformulation of what is usually called *"data modelling"*,
  - in this case for domains —
  - but with major aspects applicable as well to
  - **requirements development** and **software design**.

- That analysis focuses on
  - **endurant entities**, also called **parts** and **materials**,
    - those that can be observed at no matter what time,
    - i.e., entities of substance or continuant, and
  - **perdurant entities**: **action**, **event** and **behaviour** entities, those
    - that occur,
    - that happen,
    - that, in a sense, are accidents.

- **We think** that this “decomposition” of the “data analysis” problem into
  - **discrete parts** and **continuous materials**,
  - **atomic** and **composite parts**,
  - their **unique identifiers** and **mereology**, and
  - their **attributes**
  - **is novel**,
  - and differs from past practices in **domain analysis**.

- In Sect. 7 we suggest
  - for each of the entity categories
    - parts,
    - materials,
    - actions,
    - events and
    - behaviours,
  - a calculus of meta-functions:
    - analytic functions,
      - that guide the domain description developer
      - in the process of selection,

      and
    - so-called discovery functions,
      - that guide that person
      - in "generating" appropriate domain description texts, informal and formal.

- The **domain description calculus** is to be thought of
  - as directives to the **domain engineer**,
  - mental aids that help a team of **domain engineers**
  - to steer it simply through the otherwise daunting task
  - of constructing a usually large **domain description**.
- Think of the **calculus**
  - as directing
  - a **human calculation**
  - of **domain descriptions**.
- Finally the **domain description calculus** section
  - suggests a number of **laws** that the
  - **domain description process** ought satisfy.

- In Sect. 8 we bring a brief survey of the kind of **requirements engineering**
  - that one can now pursue based on a reasonably comprehensive **domain description**.
  - We show how one can systematically, but not automatically
  - "derive" significant fragments
    - of **requirements prescriptions**
    - from **domain description**s.

- The formal descriptions will here be expressed in the RAISE [RaiseMethod] Specification Language, RSL.
- We otherwise refer to [TheSEBook1wo].
- Appendix C of the tutorial notes brings a short primer, mostly on the syntactic aspects of RSL.
- But other model-oriented formal specification languages can be used with equal success; for example:
  - Alloy [alloy],
  - Event B [JRAbrial:TheBBooks] ,
  - VDM [e:db:Bj78bwo,e:db:Bj82b,JohnFitzgerald+PeterGormLarsen] and
  - Z [m:z:jd+jcppw96].

2. The Main Example – Example 3: Road Traffic System

- 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.

2.1. Parts

2.1.1. 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
1(a). N
1(b). F
1(c). M

value
1(a). **obs**_N: $\Delta \rightarrow$ N
1(b). **obs**_F: $\Delta \rightarrow$ F
1(c). **obs**_M: $\Delta \rightarrow$ M

2.1.2. 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
2(a). **obs_**HS: N → HS
2(b). **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
4(a). Vs = V-**set**
4(b). V
value
4(a). **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.

2.1.3. 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

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

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

5(b). H

6(b). 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.

2.2. 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.

2.2.1. 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
7(a). HI
7(b). LI
7(c). VI
value
7(a). **uid**_H: H → HI
7(b). **uid**_L: L → LI
7(c). **uid**_V: V → VI

2.2.2. Mereology

2.2.2.1 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

8(a). $\underline{\textbf{mereo}}$_L: L → HI-**set**, **axiom** ∀ l:L·**card** $\underline{\textbf{mereo}}$_L(l)=2

axiom

8(b). ∀ n:N,l:L,hi:HI · l ∈ $\underline{\textbf{obs}}$_Ls($\underline{\textbf{obs}}$_LS(n)) ∧ hi ∈ $\underline{\textbf{mereo}}$_L(l)

8(b). ⇒ ∃ h:H·h ∈ $\underline{\textbf{obs}}$_Hs($\underline{\textbf{obs}}$_HS(n))∧$\underline{\textbf{uid}}$_H(h)=hi

value

9(a). $\underline{\textbf{mereo}}$_H: H → LI-**set**

axiom

9(b). ∀ n:N,h:H,li:LI · h ∈ $\underline{\textbf{obs}}$_Hs($\underline{\textbf{obs}}$_HS(n)) ∧ li ∈ $\underline{\textbf{mereo}}$_H(h)

9(b). ⇒ ∃ l:L·l ∈ $\underline{\textbf{obs}}$_Ls($\underline{\textbf{obs}}$_LS(n))∧$\underline{\textbf{uid}}$_L(l)=li

2.2.2.2 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.

2.2.3. 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.

2.2.3.1 Attributes of Links

10. The following are attributes of links.
    (a) Link states, $\mathsf{l}\sigma$:$\mathsf{L}\Sigma$, 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, $\mathsf{l}\omega$:$\mathsf{L}\Omega$ 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
10(a). $L\Sigma = (HI \times HI)$**-set**
axiom
10(a). $\forall\ l\sigma{:}L\Sigma \cdot 0 \leq$ **card** $l\sigma \leq 2$
value
10(a). **attr_**$L\Sigma$: $L \rightarrow L\Sigma$
axiom
10(a). $\forall$ l:L · **let** {hi,hi′}=**mereo_**L(l) **in** **attr_**LΣ(l)⊆{(hi,hi′),(hi′,hi)} **end**
type
10(b). $L\Omega = L\Sigma$**-set**
value
10(b). **attr_**$L\Omega$: $L \rightarrow L\Omega$
axiom
10(b). $\forall$ l:L · **let** {hi,hi′}=**mereo_**L(l) **in** **attr_**LΣ(l)∈ **attr_**LΩ(l) **end**
type
10(c). LOC, LEN, ...
value
10(c). **attr_**LOC: L → LOC, **attr_**LEN: L → LEN, ...

2.2.3.2 Attributes of Hubs

11. The following are attributes of hubs:
   (a) Hub states, hσ:HΣ, 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, hω:HΩ 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
11(a). $\mathrm{H}\Sigma = (\mathrm{LI} \times \mathrm{LI})$**-set**
value
11(a). **attr**_HΣ: $\mathrm{H} \rightarrow \mathrm{H}\Sigma$
axiom
11(a). $\forall$ h:H • **attr**_HΣ(h)$\subseteq\{(\mathrm{li},\mathrm{li}')|\mathrm{li},\mathrm{li}':\mathrm{LI}\cdot\{\mathrm{li},\mathrm{li}'\}\subseteq$**mereo**_H(h)$\}$
type
11(b). $\mathrm{H}\Omega = \mathrm{H}\Sigma$**-set**
value
11(b). **attr**_HΩ: $\mathrm{H} \rightarrow \mathrm{H}\Omega$
axiom
11(b). $\forall$ h:H • **attr**_HΣ(h) $\in$ **attr**_HΩ(h)
type
11(c). LOC, ...
value
11(c). **attr**_LOC: $\mathrm{L} \rightarrow \mathrm{LOC}$, ...

2.2.3.3 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

12(a). VP = atH | onL

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

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

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

12(b). 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

2.2.3.4 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[8] 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.

[8] 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**

14(b). vps(n) ≡

14(a). **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) }

14(c). ∪

14((c))i. { atH(fli,**uid**_H(h),tli) | fli,tli:LI,h:H ·

14((c))ii. h ∈ hs ∧ {fli,tli}⊆**mereo**_H(h) }

14(a). **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. $\mathrm{VPM}' = \mathrm{VI} \underset{m}{\overrightarrow{\ }} \mathrm{VP}$

15. $\mathrm{VPM} = \{|\ \mathrm{vpm}:\mathrm{VPM}' \cdot \textbf{card dom}\ \mathrm{vpm} = \textbf{card rng}\ \mathrm{vpm}\ |\}$

value

16. $\mathrm{vpr}\colon \mathrm{V}\textbf{-set} \times \mathrm{N} \rightarrow \mathrm{VMP}$

17. $\mathrm{vpr(vs)(n)} \equiv$

18. $\quad \textbf{let}\ \mathrm{vpa}:\mathrm{VP}\textbf{-set} \cdot \mathrm{vpa} \subseteq \mathrm{vps(vs)(n)} \wedge \textbf{card}\ \mathrm{vpa} = \mathrm{vard\ vs}\ \textbf{in}$

19. $\quad \textbf{let}\ \mathrm{vpm}:\mathrm{VPM} \cdot \textbf{dom}\ \mathrm{vpm} = \mathrm{vps} \wedge \textbf{rng}\ \mathrm{vpm} = \mathrm{vpa}\ \textbf{in}$

20. $\quad \mathrm{vpm}\ \textbf{end end}$

2.3. 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

26. get_H: HI → N $\xrightarrow{\sim}$ H
26. get_H(hi)(n) ≡ ι h:H·h ∈ **obs**_Hs(**obs**_HS(n))∧**uid**_H(h)=hi
26. **pre**: hi ∈ xtr_HIs(n)

26(a). get_L: LI → N $\xrightarrow{\sim}$ L
26(a). get_L(li)(n) ≡ ι l:L·l ∈ **obs**_Ls(**obs**_LS(n))∧**uid**_L(l)=li
26(a). **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.

2.4. Some Derived Traffic System Concepts

2.4.1. Maps

25. A road map is an abstraction of a road net. We define one model of maps below.
    (a) A road map, RM, is a finite definition set function, M, (a specification language map) from
    - hub identifiers (the source hub)
    - to (such finite definition set) functions
    - from link identifiers
    - to hub identifiers (the target hub).

type

25(a). $\mathrm{RM}' = \mathrm{HI} \xrightarrow[m]{} (\mathrm{LI} \xrightarrow[m]{} \mathrm{HI})$

- If a hub identifier in the source or an rm:RM maps into the empty map then the "corresponding" hub is "isolated": has no links emanating from it.

26. These road maps are subject to a well-formedness criterion.
    (a) The target hubs must be defined also as source hubs.
    (b) If a link is defined from source hub (referred to by its identifier) shi via link li to a target hub thi, then, vice versa, link li is also defined from source thi to target shi.

type

26. $\text{RM} = \{|\ \text{rm:RM}' \cdot \text{wf\_RM(rm)}\ |\}$

value

26. $\text{wf\_RM: RM}' \to \textbf{Bool}$

26. $\text{wf\_RM(rm)} \equiv$

26(a). $\quad \cup\ \{\ \textbf{rng}(\text{rm(hi)})|\text{hi:HI}\cdot\text{hi} \in \textbf{dom}\ \text{rm}\ \} \subseteq \textbf{dom}\ \text{rm}$

26(b). $\quad \wedge\ \forall\ \text{shi:HI}\cdot\text{shi} \in \textbf{dom}\ \text{rm} \Rightarrow$

26(b). $\qquad \forall\ \text{li:LI} \cdot \text{li} \in \textbf{dom}\ \text{rm(shi)} \Rightarrow$

26(b). $\qquad\quad \text{li} \in \textbf{dom}\ \text{rm((rm(shi))(li))} \wedge \text{(rm((rm(shi))(li)))(li)=shi}$

27. Given a road net, n, one can derive "its" road map.
    (a) Let hs and ls be the hubs and links, respectively of the net n.
    (b) Every hub with no links emanating from it is mapped into the empty map.
    (c) For every link identifier uid_L(l) of links, l, of ls and every hub identifier, hi, in the mereology of l
    (d) hi is mapped into a map from uid_L(l) into hi'
    (e) where hi' is the other hub identifier of the mereology of l.

value

27. derive_RM: N → RM
27. derive_RM(n) ≡
27(a). **let** hs = **obs_**Hs(**obs_**HS(n)), ls = **obs_**Ls(**obs_**LS(n)) **in**
27(b). [hi ↦ [ ] | hi:HI · ∃ h:H · h ∈ hs ∧ **mereo_**H(h) = {}] ∪
27(d). [hi ↦ [**uid_**L(l) ↦ hi′
27(e). | hi′:HI · hi′ = **mereo_**L(l)\{hi}]
27(c). | l:L,hi:HI · l ∈ ls ∧ hi ∈ **mereo_**L(l)] **end**

- **Theorem:** If the road net, n, is well-formed then wf_RM(derive_RM(n)).

2.4.2. Traffic Routes

28. A traffic route, tr, is an alternating sequence of hub and link identifiers such that
    (a) li:LI is in the mereology of the hub, h:H, identified by hi:HI, the predecessor of li:LI in route r, and
    (b) hi':HI, which follows li:LI in route r, is different from hi, and is in the mereology of the link identified by li.

type
28. $R' = (HI|LI)^*$
28. $R = \{|\ r{:}R' \cdot \exists\ n{:}N \cdot wf\_R(r)(n)\ |\}$
value
28. $wf\_R: R' \to N \to \mathbf{Bool}$
28. $wf\_R(r)(n) \equiv$
28. $\quad \forall\ i{:}\mathbf{Nat} \cdot \{i,i+1\} \subseteq \mathbf{inds}\ r \Rightarrow$
28(a). $\qquad \underline{\mathbf{is}}\_HI(r(i)) \Rightarrow \underline{\mathbf{is}}\_LI(r(i+1)) \wedge r(i+1) \in \underline{\mathbf{mereo}}\_H(get\_H(r(i))(n)),$
28(b). $\qquad \underline{\mathbf{is}}\_LI(r(i)) \Rightarrow \underline{\mathbf{is}}\_HI(r(i+1)) \wedge\ r(i+1) \in \underline{\mathbf{mereo}}\_L(get\_L(r(i))(n))$

29. From a well-formed road map (i.e., a road net) we can generate the possibly infinite set of all routes through the net.
    (a) **Basis Clauses:**
        i. The empty sequence of identifiers is a route.
        ii. The one element sequences of link and hub identifiers of links and hubs of a road map (i.e., a road net) are routes.
        iii. If hi maps into some li in rm then $\langle$hi,li$\rangle$ and $\langle$li,hi$\rangle$ are routes of the road map (i.e., of the road net).
    (b) **Induction Clause:**
        i. Let r$^\frown\langle$i$\rangle$ and $\langle$i$'\rangle^\frown$r$'$ be two routes of the road map.
        ii. If the identifiers i and i$'$ are identical, then r$^\frown\langle$i$\rangle^\frown$r$'$ is a route.
    (c) **Extremal Clause:**
        i. Only such routes that can be formed from a finite number of applications of the above clauses are routes.

value

29. gen_routes: M → Routes-**infset**

29. gen_routes(m) ≡

29((a))i. **let** rs = {⟨⟩}

29((a))ii. ∪ {⟨li,hi⟩,⟨hi,li⟩|li:LI,hi:HI·...}

29((b))i. ∪ {**let** r⌢⟨li⟩,⟨li′⟩⌢r′:R · {r⌢⟨li⟩,⟨li′⟩⌢r′}⊆rs,

29((b))i. r″⌢⟨hi⟩,⟨hi′⟩⌢r‴:R · {r″⌢⟨hi⟩,⟨hi′⟩⌢r‴}⊆rs **in**

29((b))ii. r⌢⟨li⟩⌢r′,r″⌢⟨hi⟩⌢r‴ **end**} **in**

29((c))i. rs **end**

2.4.2.1 Circular Routes

30. A route is circular if the same identifier occurs more than once.

value

30. is_circular_route: R → **Bool**

30. is_circular_route(r) ≡ ∃ i,j:**Nat** · {i,j}⊆**inds** r ∧ i≠j ⇒ r(i)=r(j)

2.4.2.2 Connected Road Nets

31. A road net is connected if there is a route from any hub (or any link) to any other hub or link in the net.

```
31.  is_conn_N: N → Bool
31.  is_conn_N(n) ≡
31.      let m = derive_RM(n) in
31.      let rs = gen_routes(m) in
31.      ∀ i,i′:(LI|HI) • {i,i′}⊆xtr_LIs(n)∪ xtr_HIs(n)
31.         ∃ r:R • r ∈ rs ∧ r(1)=i ∧ r(len r)=i′ end end
```

2.4.2.3 Set of Connected Nets of a Net

32. The set, cns, of connected nets of a net, n, is
    (a) the smallest set of connected nets, cns,
    (b) whose hubs and links together "span" those of the net n.

value

32. conn_Ns: N → N-**set**
32. conn_Ns(n) **as** cns
32(a). **pre**: **true**
32(b). **post**: conn_spans_HsLs(n)(cns)
32(a). ∧ ∼∃ kns:N-**set** · **card** kns < **card** cns
32(a). ∧ conn_spans_HsLs(n)(kns)

32(b). conn_spans_HsLs: N → N → **Bool**
32(b). conn_spans_HsLs(n)(cns) ≡
32(b). ∀ cn:N•cn ∈ cns ⇒ is_connected_N(n)(cn)
32(b). ∧ **let** (hs,ls) = (**obs_**Hs(**obs_**HS(n)),**obs_**Ls(**obs_**LS(n))),
32(b). chs = ∪{**obs_**Hs(**obs_**HS(cn))|cn ∈ cns},
32(b). cls = ∪{**obs_**Ls(**obs_**LS(cn))|cn ∈ cns} **in**
32(b). hs = chs ∧ ls = cls **end**

2.4.2.4 Route Length

33. The length attributes of links can be
    (a) added and subtracted,
    (b) multiplied by reals to obtain lengths,
    (c) divided to obtain fractions,
    (d) compared as to whether one is shorter than another, etc., and
    (e) there is a "zero length" designator.

value

33(a). $+,- : \text{LEN} \times \text{LEN} \rightarrow \text{LEN}$

33(b). $* : \text{LEN} \times \textbf{Real} \rightarrow \text{LEN}$

33(c). $/ : \text{LEN} \times \text{LEN} \rightarrow \textbf{Real}$

33(d). $<,\leq,=,\neq,\geq,> : \text{LEN} \times \text{LEN} \rightarrow \textbf{Bool}$

33(e). $\ell_0 : \text{LEN}$

34. One can calculate the length of a route.

value
34. length: R → N → LEN
34. length(r)(n) ≡
34. **case** r **of**:
34. ⟨⟩ → ℓ_0,
34. ⟨si⟩⌢r′ →
34. is_LI(si)→**attr_**LEN(get_L(si)(n))+length(r′)(n)
34. is_HI(si)→length(r′)(n)
34. **end**

2.4.2.5 Shortest Routes

35. There is a predicate, is_R, which,
    (a) given a net and two distinct hub identifiers of the net,
    (b) tests whether there is a route between these.

value

35. is_R: N → (HI×HI) → **Bool**
35. is_R(n)(fhi,thi) ≡
35(a). fhi ≠ thi ∧ {fht,thi}⊆xtr_HIs(n)
35(b). ∧ ∃ r:R • r ∈ routes(n) ∧ **hd** r = fhi ∧ r(**len** r) = thi

36. The shortest between two given hub identifiers
    (a) is an acyclic route, r,
    (b) whose first and last elements are the two given hub identifiers
    (c) and such that there is no route, r′ which is shorter.

value

36. shortest_route: N → (HI×HI) → R
36(a). shortest_route(n)(fhi,thi) **as** r
36(b). **pre**: pre_shortest_route(n)(fhi,thi)
36(c). **post**: pos__shortest_route(n)(r)(fhi,thi)

36(b). pre_shortest_route: $N \rightarrow (HI \times HI) \rightarrow$ **Bool**
36(b). pre_shortest_route(n)(fhi,thi) $\equiv$
36(b). is_R(n)(fhi,thi) $\wedge$ fhi$\neq$thi $\wedge$ $\{$fhi,thi$\}\subset$xtr_HIs(n)

36(c). pos_shortest_route: $N \rightarrow R \rightarrow (HI \times HI) \rightarrow$ **Bool**
36(c). pos_shortest_route(n)(r)(fhi,thi) $\equiv$
36(c). r $\in$ routes(n)
36(c). $\wedge \sim\exists$ r':R $\cdot$ $r' \in$ routes(n) $\wedge$ length(r') $<$ length(r)

2.5. 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.

2.6. 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 (hσ),
  - setting of link state (lσ),
  - 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.

37. 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

37. $\text{ins\_H: N} \rightarrow \text{H} \xrightarrow{\sim} \text{N}$

37. ins_H(n)(h) **as** n', **pre**: pre_ins_H(n)(h), **post**: post_ins_H(n)(h)

37(a). pre_ins_H(n)(h) $\equiv$

37(a). $\sim\exists$ h':H $\cdot$ $h' \in$ **obs_**Hs(n) $\wedge$ **uid_**HI(h)=**uid_**HI(h')

37(b). $\wedge$ **mereo_**H(h) = {}

37(c). post_ins_H(n)(h)(n') $\equiv$

37(c). **obs_**Hs(n) $\cup$ {h} = **obs_**Hs(n')

37(d). $\wedge$ **obs_**Ls(n) = **obs_**Ls(n')

2.7. 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".

38. Link disappearance is expressed as a predicate on the "before" and "after" states of the net. The predicate identifies the "missing" ℓink (!).

39. 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**_$\Pi(\ell)$.

38. link_dis: N × N → **Bool**
38. link_dis(n,n′) ≡
38. ∃ ℓ:L · pre_link_dis(n,ℓ) ⇒ post_link_dis(n,ℓ,n′)
39. pre_link_dis: N × L → **Bool**
39. pre_link_dis(n,ℓ) ≡ ℓ ∈ **obs**_Ls(n)

40. 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. $\{l''_1, l''_2, \ldots, l''_q\} \setminus \{\mathsf{uid\_}\Pi(\ell)\} \cup \{\mathsf{uid\_}\Pi(\ell_j)\}$
41. All other hubs and links of n are unaffected.

42. 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

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

2.8. Behaviours

2.8.1. Traffic

2.8.1.1 Continuous Traffic

- For the road traffic system
  - perhaps the most significant example of a behaviour
  - is that of its traffic

43. the continuous time varying discrete positions of vehicles, vp:VP[9],
44. where time is taken as a dense set of points.

type

44. $c\mathbb{T}$

43. $cRTF = c\mathbb{T} \rightarrow (V \underset{m}{\overrightarrow{\quad}} VP)$

[9] For VP see Item 12(a) on Slide 56.

2.8.1.2 Discrete Traffic

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

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

46. $\mathrm{d}\mathbb{T}$
45. $\mathrm{dRTF} = \mathrm{d}\mathbb{T} \xrightarrow[m]{} (\mathrm{V} \xrightarrow[m]{} \mathrm{VP})$

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

type
47. $\mathrm{RTF} = \mathrm{d}\mathbb{T} \xrightarrow[m]{} (\mathrm{VI} \xrightarrow[m]{} \mathrm{VP})$

2.8.1.3 Time: An Aside

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

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

type

48. $\mathbb{T}$

49. $\mathbb{TI}$

value

50. $\delta:\mathbb{TI}$

51. $\mathbb{MIN}, \mathbb{MAX}: \ \mathbb{T} \rightarrow \mathbb{T}$

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

52. $-: \ \mathbb{T}\times\mathbb{T} \rightarrow \mathbb{TI}$

53. $+: \ \mathbb{T}\times\mathbb{TI},\mathbb{TI}\times\mathbb{T} \rightarrow \mathbb{T}$

53. $-,+: \ \mathbb{TI}\times\mathbb{TI} \rightarrow \mathbb{TI}$

53. $*: \ \mathbb{TI}\times$**Real** $\rightarrow \mathbb{TI}$

53. $/: \ \mathbb{TI}\times\mathbb{TI} \rightarrow$ **Real**

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

55. We declare a channel **clk_ch**.

value
54. clock: $\mathbb{T}$ → **out** clk_ch **Unit**
54. clock(t) ≡ ... clk_ch!t ... clock(t ⊓ t+δ)
channnel
55. clk_ch:$\mathbb{T}$

2.8.2. Globally Observable Parts

- There is given

56. a net, **n:N**,
57. a set of vehicles, **vs:V-set**, and
58. a monitor, **m:M**.

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

value

56. n:N = **obs**_N(Δ)
56. ls:L-**set** = **obs**_Ls(**obs**_LS(n)), hs:H-**set** = **obs**_Hs(**obs**_HS(n)),
56. lis:LI-**set** = {**uid**_L(l)|l:L•l ∈ ls}, his:HI-**set** = {**uid**_H(h)|h:H•h ∈ hs}
57. vs:V-**set** = **obs**_Vs(**obs**_VS(**obs**_F(Δ))), vis:V-**set** = {**uid**_V(v)|v:V•v
58. m:**obs**_M(Δ)

2.8.3. Road Traffic System Behaviours

59. 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

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

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

59(e). t_0:T = clk_ch?

59. rts() =

59(a). ∥ {veh(**uid_**V(v))(v)(vpm(**uid_**V(v)))|v:V•v ∈ vs}

59(b). ∥ mon(mi)(m)([$t_0 \mapsto$ 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.

2.8.4. 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.

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

channel

60. $\{vm\_ch[mi,vi]|vi:VI\bullet vi \in vis\}:VP$

2.8.5. Behaviour Signatures

61. The road traffic system behaviour, rts, takes no arguments; and "behaves", that is, continues forever.
62. 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".
63. The monitor behaviour takes monitor part, m:M, as argument and also the discrete road traffic, drtf:dRTF; the behaviour otherwise runs forever.

value

61. rts: **Unit** → **Unit**

62. veh: vi:VI → v:V → VP → **out** vm_ch[vi],mi:MI **Unit**

63. mon: mi:MI → m:M → dRTF → **in** {vm_ch[mi,vi]|vi:VI•vi ∈ vis},clk_c

2.8.6. The Vehicle Behaviour

64. 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**).

65. 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" !

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

66. We describe here an abstraction of the vehicle behaviour on a Link (ii). Either
    (a) the vehicle remains at that link position informing the monitor,
    (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.
67. or, internally non-deterministically,
68. the vehicle "disappears — off the radar" !

64. veh(vi)(v)(vp:onL(fhi,li,f,thi)) $\equiv$
66(a). vm_ch[mi,vi]!vp ; veh(vi)(v)(vp)
66(b). $\prod$
66(c). **if** f + δ<1
66((c))i. **then** vm_ch[mi,vi]!onL(fhi,li,f+δ,thi) ;
66((c))i. veh(vi)(v)(onL(fhi,li,f+δ,thi))
66((c))ii. **else let** li′:LI•li′ $\in$ **mereo**_H(get_H(thi)(n)) **in**
66((c))iiA. vm_ch[mi,vi]!atH(li,thi,li′);
66((c))iiB. veh(vi)(v)(atH(li,thi,li′)) **end end**
67. $\prod$
68. **stop**

2.8.7. The Monitor Behaviour

69. The **mon**itor 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".

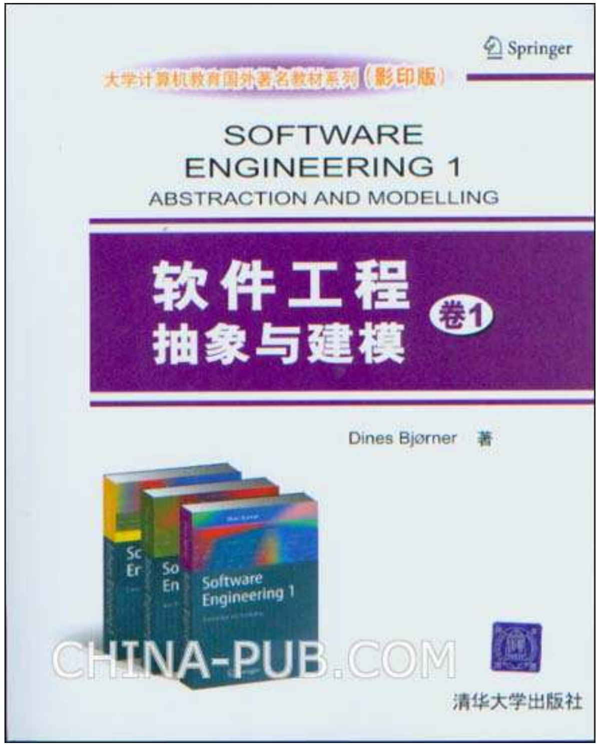
70. Either the monitor "does own work"

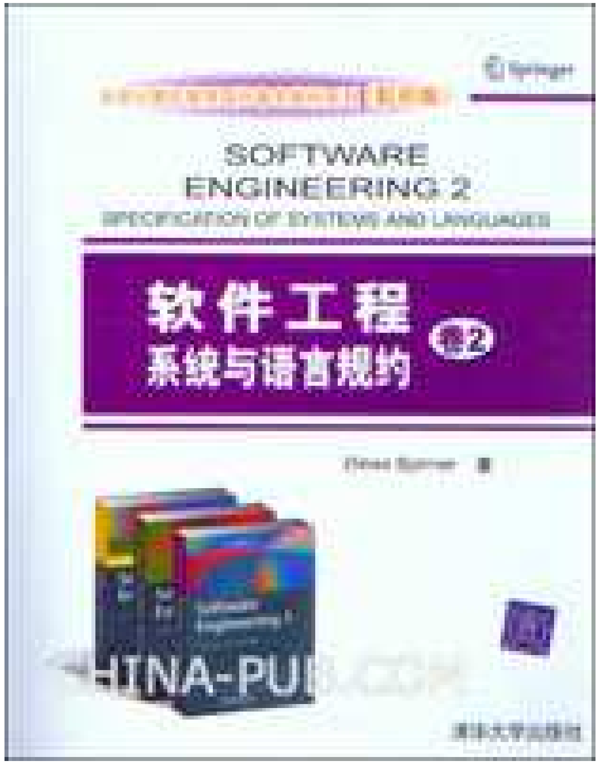
71. or, internally non-deterministically accepts messages from vehicles.
    (a) A vehicle position message, vp, may arrive from the vehicle identified by vi.
    (b) That message is appended to that vehicle's movement trace,
    (c) whereupon the monitor resumes its behaviour —
    (d) where the communicating vehicles range over all identified vehicles.

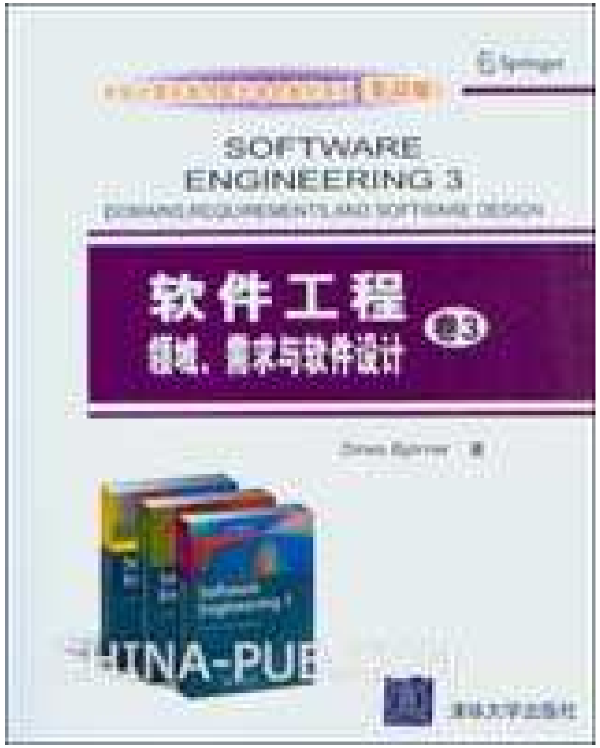
69. mon(mi)(m)(rtf) ≡
70. mon(mi)(own_mon_work(m))(rtf)
71. ⌈⌉
71(a). ⌈⌉ { **let** ((vi,vp),t) = (vm_ch[vi]?,clk_ch?) **in**
71(b). **let** rtf′ = rtf † [t ↦ rtf(max **dom** rtf) † [vi ↦ vp]] **in**
71(c). mon(mi)(m)(rtf′) **end**
71(d). **end** | vi:VI · vi ∈ vis }

70. 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**. ■







See You in 30 Minutes — Thanks !