Development of Transportation Systems*

Dines Bjørner, Professor Emeritus

Faculté des Sciences, Bureau 266,
INRIA, LORIA and Université Henri Poincaré Nancy 1,
BP 239, F-54506 Vandœuvre lès Nancy, France.†

E–Mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

December 5, 2007, compiled June 11, 2008: 16:33

Abstract

Road systems[1], railway systems[2], air traffic systems[3], and, for example, container vessel shipping[4], all share underlying abstractions such as transportation nets with hubs (road intersections, train stations, airports and harbours) and links (road segments, train tracks, air and sea lanes) and their states of being open or closed for certain flows of traffic across hubs and along links, etc.

In this paper we shall first hint at an abstract formal model for such transportation and then show how it can be refined into models for road traffic, train traffic and air traffic. Then we likewise hint at how such, so-called domain models — which reflect only what there is "out there", in reality, before computing and communication — can be rigorously transformed into requirements for respective traffic monitoring and control systems.

The paper concludes with a discussion of issues of development of the right systems, that is, the systems that customers (that is, transportation and traffic authorities) expect to receive, and of systems which are right, that is, are correct.

Contents

*Invited paper for ISOLA 2007, Workshop On Leveraging Applications of Formal Methods, Verification and Validation. Special Workshop Theme: Formal Methods in Avionics, Space and Transport. Poitiers, France, December 12-14 2007

†This paper was written with the financial support of Université Henri Poincaré and INRIA during the author's two month visit. The author is Professor Emeritus at DTU Informatics, The Technical University of Denmark, DK-2800 Kgs. Lyngby, Denmark. Home address is Fredsvej 11, DK-2840 Holte, Danmark.

[1]Road systems: including ordinary traffic monitoring and street signal control systems and toll way (peage) systems with toll gate tickets and payment

[2]Railway systems: incl. mixed train traffic on general nets with, for example switch [aiguillage] interlocking and other signaling subsystems

[3]Air traffic systems: across ground control, terminal or tower control, area control, etc.

[4]Container line industry: across sea lane between harbours

1 Introduction

1.1 The Software Engineering Triptych

1.1.1 The Triptych Dogma

Before software (in general: the machines, i.e., systems of computers and communication and of sensors and actuators etcetera connected to them) can be designed we must understand "the" requirements. Before requirements, that is, prescriptions for the machine, what it should do, not how, can be prescribed we must understand the domain.

1.1.2 The Triptych Doctrine Consequences

In consequence we prefer to develop software professionally, that is: First we study an available — or develop ourselves an as "complete" as possible — description of the domain; then we develop, from such a domain description, the requirements prescription; and from the requirements prescription we then design the software. In this paper we shall mainly cover the issues of domain descriptions and we do so around the presentation of an extensive model of a conceptual transportation domain.

1.2 Narrative versus Formal Specifications

1.2.1 Three Forms of Specification

By a specification we shall here (a bit narrowly) mean a narrated and a formal description of a domain, a narrated and a formal prescription of a (set of) requirements, or a narrated and a formal design (document[ation]) of some software. So the term has three instantiations: description, prescription and design (document[ation]).

1.2.2 Narration and Formalisation

By a narrative (specification) we shall here mean an informal, English, French, Danish, or the like, specification; usually it is extensively "peppered" with technical terms. By a formalisation (a formal specification) we shall here mean a specification which is formulated in a formal specification language, that is, a language with a formal syntax, a formal semantics, and a proof system, that is, a set of proof rules. It is not possible to prove that a narrative specification an a supposedly related formal specification expresses the same !

1.2.3 Motivation for Both Informal and Formal Specifications

Domain as well as requirements stakeholders can not usually read formal specifications. And software correctness cannot be meaningfully claimed and ascertained unless the whole development, from domain via requirements to software is formally expressed.

Domain and requirements stakeholders shall validate the specifications in order to help guarantee that the software is the right software: does what is expected. $\mathcal{D}$omain engineers, $\mathcal{R}$equirements engineers and $\mathcal{S}$oftware designers, that is, software engineers shall (ultimately) verify the specifications and the phases of development from domain descriptions via requirements prescriptions to software designs in order to help guarantee that the software is correct, i.e., is that the software is right: has no "bugs".

$$\text{Correctness:} \qquad \mathcal{D}, \mathcal{S} \models \mathcal{R}$$

1.3 Background Work

To be a professional software engineer therefore requires ability to analyse domains and requirements in order to find suitable and pleasing abstractions, a very good command of the informal

language of the narrative in order to concisely, but informally express abstract models, and a very good command of a number for formal specification languages including (ultimately) their proof systems in order to succinctly and formally express abstract models and to (eventually) prove properties of formal specifications and phases, stages and steps of development (refinements).

There are, by now, quite a respectable set of publications covering the field of formal development techniques (aka: "formal methods"). To develop formal domain descriptions and requirements prescriptions requires not just one, but a set of formal specification languages together with their abstraction and modelling and verification, model checking and formal testing techniques. We shall mention a few.

1.3.1 VDM and RAISE

The author's work, since 1973 has focused on formal techniques for software development: The design, study, use and propagation of VDM: The Vienna (software) Development Method [22, 23, 33, 32] — in the period 1973 till late 1980s. The design, study, use and propagation of RAISE: Rigorous Approach to Industrial Software Engineering [35, 37, 12, 13, 14, 34] — in the period late 1980s till at present.

1.3.2 The Software Engineering Book

My most recent book is a three volume book: Software Engineering [12, 13, 14]. Volume 1 of that book covers Abstraction and Modelling using RSL, the RAISE Specification Language. Volume 2 covers Specification of Systems and Languages additionally using such formal languages as Petri Nets, Message and Live Sequence Charts, Statecharts and the Duration Calculus. Volume 3 covers Domain and Requirements Engineering — and brings it all together in Software Design.

1.3.3 Other Specification Approaches

Other specification languages, techniques and tools, that, in addition to VDM and RAISE, cover the spectrum of domain and requirements specification, refinement and verification, are dealt with in Alloy [53], ASM [25, 73, 74], B, Event-B [1, 28], CSP [48, 76, 77, 49], DC [82, 83, 39] (Duration Calculus), Live Sequence Charts [29, 44, 56], Message Sequence Charts [50, 51, 52], Petri nets [55, 66, 71, 70, 72], Statecharts [40, 41, 43, 45, 42], Temporal Logic of Reactive Systems [59, 60, 65, 67], TLA+ [57, 58, 61, 62] (Temporal Logic of Actions), Z [78, 79, 81, 47, 46]. Techniques for integrating "different" formal techniques are covered in [2, 38, 26, 24, 75]. The recent book on Logics of Specification Languages [21] covers ASM, B/event B, CafeObj, CASL, DC, RAISE, TLA+, VDM and Z.

1.4 Structure of Paper

Section 2 contains an extensive example of a model of a generic domain claimed to represent an abstraction of the transport domain which includes road, rail, air and sea transport. Section 3 contains a development of this generic transport domain model to show that it indeed does capture some essence of road transport. The development is formulated as a "derivation" of domain and interface requirements. Thus we also manage to illustrate how we can develop requirements prescriptions from domain descriptions. But we could, as well claim that the "derivation of requirements" is a refinement of a generic domain model into an instantiated domain model. That is, the (domain to requirements) operations of projection, instantiation, determination, extension and fitting can, as well, be thought of as domain refinement operations. Finally Sect. 4 contains an analysis of the generic transport domain model (of Sect. 2) to show that it indeed does capture some essence also of rail, air and sea transport. In this more discursive section we rely on the more formal parts of Sect. 3 to hint that the claims for rail and air transport can indeed be rigorously developed. For sea transport we refer to [17].

2 A Generic Model of Transportation

2.1 First Example of a Generic Domain Description

We exemplify a transportation domain. By transportation we shall mean *the movement of vehicles from hubs to hubs along the links of a net.*

2.1.1 Rough Sketching — Business Processes

The basic *entities* of the transportation "business" are the (i) *net*s with their (ii) *hub*s and (iii) *link*s, the (iv) *vehicle*s, and the (v) *traffic* (of vehicles on the net). The basic *functions* are those of (vi) vehicles *entering* and *leaving* the net (here simplified to entering and leaving at hubs), (vii) for vehicles to *make movement* transitions along the net, and (viii) for *inserting* and *removing links* (and associated hubs) into and from the net. The basic *events* are those of (ix) the *appearance* and *disappearance* of vehicles, and (x) the *breakdown* of links. And, finally, the basic *behaviours* of the transportation business are those of (xi) *vehicle journey* through the net and (xii) *net development & maintenance* including insertion into and removal from the net of links (and hubs).

2.1.2 Narrative — Entities

By an *entity* we mean *something we can point to, i.e., something manifest, or a concept abstracted from, such a phenomenon or concept thereof.*
Among the many entities of transportation we start with nets, hubs, and links.

A transportation net consists of hubs and links. Hubs and links are different kinds of entities. Conceptually hubs (links) can be uniquely identified. From a link one can observe the identities of the two distinct hubs it links. From a hub one can observe the identities of the one or more distinct links it connects.

Other entities such as vehicles and traffic could as well be described. Please think of these descriptions of entities as descriptions of the real phenomena and (at least postulated) concepts of an actual domain.

2.1.3 Formalisation — Entities

type
  H, HI, L, LI
  N = H-**set** × L-**set**
value
  obs_HI: H→HI, obs_LI: L→LI,
  obs_HIs: L→HI-**set**,obs_LIs: H→LI-**set**
axiom
  ∀ (hs,ls):N •
    card hs≥2 ∧ **card** ls≥1 ∧
    ∀ h:H • h ∈ hs ⇒
      ∀ li:LI • li ∈ obs_LIs(h) ⇒
        ∃ l′:L • l′ ∈ ls ∧ li=obs_LI() ∧ obs_HI(h) ∈ obs_HIs(l′) ∧
    ∀ l:L • l ∈ ls ⇒
      ∃ h′,h″:H • h′≠h″ ∧ {h′,h″}⊆hs ∧ obs_HIs(l)={obs_HI(h′),obs_HI(h″)}
value
  xtr_HIs: N → HI-**set**,xtr_LIs: N → LI-**set**

2.1.4 Narrative — Operations

By an *operation* (of a domain) we mean a *function that applies to entities of the domain and yield entities of that domain — whether these entities are actual phenomena or concepts of these or of other phenomena.*

Actions (by domain stakeholders) amount to the execution of operations.

Among the many operations performed in connection with transportation we illustrate some on nets. To a net one can join new link in either of three ways: The new link connects two new hubs — so these must also be joined , or The new link connects a new hub with an existing hub — so it must also be joined, or The new link connects two existing hubs. In any case we must either provide the new hubs or identify the existing hubs.

From a net one can remove a link. Three possibilities now exists: The removed link would leave its two connected hubs isolated unless they are also removed — so they are; The removed link would leave one of its connected hubs isolated unless it is also removed — so it is; or The removed link connects two hubs into both of which other links are connected — so all is OK. (Note our concern for net invariance.) Please think of these descriptions of operations as descriptions of the real phenomena and (at least postulated) concepts of an actual domain. (Thus they are not prescriptions of requirements to software let alone specifications of software operations.)

2.1.5 Formalisation — Operations

```
type
  NetOp = InsLnk | RemLnk
  InsLnk == 2Hs(h1:H,l:L,h2:H)|1H(hi:HI,l:L,h:H)|0H(hi1:HI,l:L,hi2:HI)
  RemLnk == RmvL(li:LI)
value
  int_NetOp: NetOp → N ~→ N
  pre int_NetOp(op)(hs,ls) ≡
    case op of
      2Hs(h1,l,h2) → {h1,h2}∩ hs={} ∧ l∉ ls ∧ obs_HIs(l)={obs_HI(h1),obs_HI(h2)} ∧
          {obs_HI(h1),obs_HI(h2)}∩ xtr_HIs(hs)={} ∧ obs_LIs(h1)={li} ∧ obs_LIs(h2)={li},
      1H(hi,l,h) → h∉ hs ∧ obs_HI(h)∉ xtr_HIs(hs,ls) ∧
          l∉ ls ∧ obs_LI(l)∉ xtr_LIs(hs,ls) ∧ ∃ h′:H•h′ ∈ hs∧obs_HI(h′)=hi,
      0H(hi1,l,hi2) → l∉ ls ∧ hi1≠hi2 ∧ {hi1,hi2}⊆∈ xtr_HIs(hs,ls) ∧
          ∃ h1,h2:H•{h1,h2}∈ hs∧{hi1,hi2}={obs_HI(h1),obs_HI(h2)},
      RmvL(li) → ∃ l:L • l ∈ ls ∧ obs_LI(l)=li
    end


  int_NetOp(op)(hs,ls) ≡
    case op of
      2Hs(h1,l,h2) → (hs ∪ {h1,h2},ls ∪ {l}),
      1H(hi,l,h) →
        (hs\{xtr_H(hi,hs)}∪{h,aLI(xtr_H(hi,hs),obs_LI(l))},ls ∪ {l}),
      0H(hi1,l,hi2) →
        let hsδ = {aLI(xtr_H(hi1,hs),obs_LI(l)),aLI(xtr_H(hi2,hs),obs_LI(l))} in
        (hs\{xtr_H(hi1,hs),xtr_H(hi2,hs)}∪ hsδ,ls ∪ {l}) end,
      RmvL(li) → ...
    end

  xtr_H: HI × H-set ~→ H
  xtr_H(hi,hs) ≡ let h:H • h ∈ hs ∧ obs_HI(h)=hi in h end
  pre ∃ h:H • h ∈ hs ∧ obs_HI(h)=hi


  aLI: H × LI → H, sLI: H × LI → H
  aLI(h,li) as h′
    pre li ∉ obs_LIs(h)
    post obs_LIs(h′) = {li} ∪ obs_LIs(h) ∧ ...
```

```
  sLI(h′,li) as h
    pre li ∈ obs_LIs(h′)
    post obs_LIs(h) = obs_LIs(h′)\{li} ∧ ...
```

The ellipses, ..., shall indicate that previous properties of h holds for h′.

2.1.6 Narrative — Events

By an *event* of a domain we shall here mean *an instantaneous change of domain state (here, for example, "the" net state) not directly brought about by some willed action of the domain but either by "external" forces or implicitly, as an unintended result of a willed action.*

Among the "zillions" of events that may occur in transportation we single out just one. A link of a net ceases to exist as a link.[5]

In order to model transportation events we — ad hoc — introduce a transportation state notion of a net paired with some — ad hoc — "conglomerate" of remaining state concepts referred to as $\omega : \Omega$.

2.1.7 Formalisation — Events

```
type
  Link_Disruption == LiDi(li:LI)
channel
  x:(Link_Disruption|...)
value
  transportation_transition: (N × Ω) → in x  (N × Ω)
  transportation_transition(n,ω) ≡
    ...
    ⌈⌉ let xv = x? in
         case xv of
           LiDi(li) → (int_NetOp(RmvL(li))(hs,ls),line_dis(ω))
           ...
         end end
    ⌈⌉ ...

  line_dis: Ω → Ω
```

2.1.8 Narrative — Behaviours

By a *behaviour* we mean *a possibly infinite sequence of zero, one or more actions and events.* We illustrate just one of very many possible transportation behaviours.

A net behaviour is a sequence of zero, one or more executed net operations: the openings (insertions) of new links (and implied hubs) and the closing (removals) of existing links (and implied hubs), and occurrences of external events (limited here to link disruptions).

2.1.9 Formalisation — Behaviours

```
channel
  x:...
value
  transportation_transition: (N × Ω) → in x  (N × Ω)
  transportation_transition(n,ω) ≡
    ...
```

[5] There may be many different causes of such an event: a road link segment, say a bridge or a building next to the link, collapses; a traffic accident; or other.

```
    ⌈⌉ let xv = x? in case xv of ... end end
    ⌈⌉ let op:NetOp • pre IntNetOp(op)(n) in IntNetOp(op)(n) end
    ...

transportation: (N × Ω) → in x Unit
transportation(n,ω) ≡
    let (n′,ω′) = transportation_transition(n,ω) in
    transportation (n′,ω′) end
```

2.2 Domain Modelling: Describing Facets

In this, a major, methodology section of the current paper we shall focus on principles and techniques domain modelling, that is, developing abstractions and verifying properties. We shall only cover 'developing abstractions'.

Domain modelling, as we shall see, entails modelling a number of domain facets.

By a *domain facet* we mean *one amongst a finite set of generic ways of analysing a domain: a view of the domain, such that the different facets cover conceptually different views, and such that these views together cover the domain.*

These are the facets that we find "span" a domain in a pragmatically sound way: intrinsics, support technology, management & organisation, rules & regulations, scripts and human behaviour: We shall now survey these facets.

2.2.1 Domain Intrinsics

By *domain intrinsics* we mean *those phenomena and concepts of a domain which are basic to any of the other facets (listed earlier and treated, in some detail, below), with such domain intrinsics initially covering at least one specific, hence named, stakeholder view.*

In the large example of Sect. 2.1, we claim that the net, hubs and links were intrinsic phenomena of the transportation domain; and that the operations of joining and removing links were not: one can explain transportation without these operations. We will now augment the domain description of Sect. 2.1 with an intrinsic concept, namely that of the states of hubs and links: where these states indicate desirable directions of flow of movement.

A Transportation Intrinsics — Narrative. With a hub we can associate a concept of hub state. The pragmatics of a hub state is that it indicates desirable directions of flow of vehicle movement from (incoming) links to (outgoing) links. The syntax of indicating a hub state is (therefore) that of a possibly empty set of triples of two link identifiers and one hub identifier where the link identifiers are those observable from the identified hub.

With a link we can associate a concept of link state. The pragmatics of a link state is that it indicates desirable directions of flow of vehicle movement from (incoming, identified) hubs to (outgoing, identified) hubs along an identified link. The syntax of indicating a link state is (therefore) that of a possibly empty set of triples of pairs of identifiers of link connected hub and a link identifier where the hub identifiers are those observable from the identified link.

A Transportation Intrinsics — Formalisation.

```
type
   X = LI×HI×LI [ crossings of a hub ]
   P = HI×LI×HI [ paths of a link ]
   HΣ = X-set [ hub states ]
   LΣ = P-set [ link states ]
value
   obs_HΣ: H → HΣ
```

obs_LΣ: L → LΣ
xtr_Xs: H → X-**set**, xtr_Ps: L → P-**set**
xtr_Xs(h) ≡ {(li,hi,li′)|li,li′:LI,hi:HI•{li,li′}⊆obs_LIs(h)∧hi=obs_HI(h)}
xtr_Ps(l) ≡ {(hi,li,hi′)|hi,hi′:HI,li:LI•{hi,hi′}=obs_HIs(l)∧li=obs_LI(l)}
axiom
∀ n:N,h:H;l:L • h ∈ obs_Hs(n)∧l ∈ obs_Ls(n) ⇒
obs_HΣ(h)⊆xtr_Xs(h) ∧ obs_LΣ(l)⊆xtr_Ps(l)

2.2.2 Domain Support Technologies

By *domain support technologies* we mean *ways and means of implementing certain observed phenomena or certain conceived concepts.*

A Transportation Support Technology Facet — Narrative, 1. Earlier we claimed that the concept of hub and link states was an intrinsics facet of transport nets. But we did not describe how hubs or links might change state, yet hub and link state changes should also be considered intrinsic facets. We there introduce the notions of hub and link state spaces and hub and link state changing operations. A hub (link) state space is the set of all states that the hub (link) may be in. A hub (link) state changing operation can be designated by the hub and a possibly new hub state (the link and a possibly new link state).

A Transportation Support Technology Facet — Formalisation, 1.

type
HΩ = HΣ-**set**, LΩ = LΣ-**set**
value
obs_HΩ: H → HΩ, obs_LΩ: L → LΩ
axiom
∀ h:H • obs_HΣ(h) ∈ obs_HΩ(h) ∧ ∀ l:L • obs_LΣ(l) ∈ obs_LΩ(l)
value
chg_HΣ: H × HΣ → H, chg_LΣ: L × LΣ → L
chg_HΣ(h,hσ) **as** h′
pre hσ ∈ obs_HΩ(h) **post** obs_HΣ(h′)=hσ
chg_LΣ(l,lσ) **as** l′
pre lσ ∈ obs_LΩ(h) **post** obs_HΣ(l′)=lσ

A Transportation Support Technology Facet — Narrative, 2. Well, so far we have indicated that there is an operation that can change hub and link states. But one may debate whether those operations shown are really examples of a support technology. (That is, one could equally well claim that they remain examples of intrinsic facets.) We may accept that and then ask the question: How to effect the described state changing functions ? In a simple street crossing a semaphore does not instantaneously change from red to green in one direction while changing from green to red in the cross direction. Rather there is are intermediate sequences of green/yellow/red and red/yellow/green states to help avoid vehicle crashes and to prepare vehicle drivers. Our "solution" is to modify the hub state notion.

A Transportation Support Technology Facet — Formalisation, 2.

type
Colour == red | yellow | green
X = LI×HI×LI×Colour [crossings **of** a hub]
HΣ = X-**set** [hub states]

value
obs_HΣ: H → HΣ, xtr_Xs: H → X-**set**
xtr_Xs(h) ≡ {(li,hi,li′,c)|li,li′:LI,hi:HI,c:Colour•{li,li′}⊆obs_LIs(h)∧hi=obs_HI(h)}
axiom
∀ n:N,h:H • h ∈ obs_Hs(n) ⇒ obs_HΣ(h)⊆xtr_Xs(h) ∧
∀ (li1,hi2,li3,c),(li4,hi5,li6,c′):X • {(li1,hi2,li3,c),(li4,hi5,li6,c′)}⊆obs_HΣ(h) ∧
li1=li4 ∧ hi2=hi5 ∧ li3=li6 ⇒ c=c′

A Transportation Support Technology Facet — Narrative, 3. We consider the colouring, or any such scheme, an aspect of a support technology facet. There remains, however, a description of how the technology that supports the intermediate sequences of colour changing hub states.

We can think of each hub being provided with a mapping from pairs of "stable" (that is non-yellow coloured) hub states (hσ_i,hσ_f) to well-ordered sequences of intermediate "un-stable' (that is yellow coloured) hub states paired with some time interval information $\langle(h\sigma', t\delta'), (h\sigma'', t\delta''), \ldots, (h\sigma'^{\cdots\prime}, t\delta'^{\cdots\prime})\rangle$ and so that each of these intermediate states can be set, according to the time interval information,[6] before the final hub state (hσ_f) is set.

A Transportation Support Technology Facet — Formalisation, 3.

type
TI [time interval]
Signalling = (HΣ × TI)*
Sema = (HΣ × HΣ) $\xrightarrow[m]{}$ Signalling
value
obs_Sema: H → Sema, chg_HΣ: H × HΣ → H, chg_HΣ_Seq: H × HΣ → H
chg_HΣ(h,hσ) **as** h′ **pre** hσ ∈ obs_HΩ(h) **post** obs_HΣ(h′)=hσ
chg_HΣ_Seq(h,hσ) ≡
let sigseq = (obs_Sema(h))(obs_Σ(h),hσ) **in** sig_seq(h)(sigseq) **end**

sig_seq: H → Signalling → H
sig_seq(h)(sigseq) ≡
if sigseq=⟨⟩ **then** h **else**
let (hσ,tδ) = **hd** sigseq **in**
let h′ = chg_HΣ(h,hσ); **wait** tδ;
sig_seq(h′)(**tl** sigseq) **end end end**

2.2.3 Domain Management & Organisation

By *domain management* we mean *people (such decisions) (i) who (which) determine, formulate and thus set standards (cf. rules and regulations, a later lecture topic) concerning strategic, tactical and operational decisions; (ii) who ensure that these decisions are passed on to (lower) levels of management, and to "floor" staff; (iii) who make sure that such orders, as they were, are indeed carried out; (iv) who handle undesirable deviations in the carrying out of these orders cum decisions; and (v) who "backstop" complaints from lower management levels and from floor staff.*

We use the connective '&' (ampersand) in lieu of the connective 'and' in order to emphasise that the joined concepts (A & B) hang so tightly together that it does not make sense to discuss one without discussing the other.

By *domain organisation* we mean *the structuring of management and non-management staff levels; the allocation of strategic, tactical and operational concerns to within management and non-management staff levels; and hence the "lines of command": who does what and who reports to whom — administratively and functionally.*

[6] Hub state $h\sigma''$ is set $t\delta'$ time unites after hub state $h\sigma'$ was set.

A Transportation Management & Organisation Facet — Narrative. In the previous section on support technology we did not describe who or which "ordered" the change of hub states. We could claim that this might very well be a task for management.

(We here look aside from such possibilities that the domain being modelled has some further support technology which advices individual hub controllers as when to change signals and then into which states. We are interested in finding an example of a management & organisation facet — and the upcoming one might do!)

So we think of a 'net hub state management' for a given net. That management is divided into a number of 'sub-net hub state managements' where the sub-nets form a partitioning of the whole net. For each sub-net management there are two kinds of management interfaces: one to the overall hub state management, and one for each of interfacing sub-nets. What these managements do, what traffic state information they monitor, etcetera, you can yourself "dream" up. Our point is this: We have identified a management organisation.

A Transportation Management & Organisation Facet — Formalisation.

type
  HIsLIs = HI-**set** × LI-**set**
  MgtNet′ = HIsLIs × N
  MgtNet = {| mgtnet:MgtNet′ • wf_MgtNet(mgtnet)|}
  Partitioning′ = HIsLIs-**set** × N
  Partitioning = {| partitioning:Partitioning′ • wf_Partitioning(partitioning)|}
value
  wf_MgtNet: MgtNet′ → **Bool**
  wf_MgtNet((his,lis),n) ≡
    [The his component contains all the hub ids.\ of links identified in lis]
  wf_Partitioning: Partitioning′ → **Bool**
  wf_Partitioning(hisliss,n) ≡
    ∀ (his,lis):HIsLIs • (his,lis) ∈ hisliss ⇒ wf_MgtNet((his,lis),n)∧
    [no sub−net overlap and together they ″span″ n]

Etcetera.

2.2.4 Domain Rules & Regulations

Domain Rules. By a *domain rule* we mean *some text (in the domain) which prescribes how people or equipment are expected to behave when dispatching their duty, respectively when performing their function.*

Domain Regulations. By a *domain regulation* we mean *some text (in the domain) which prescribes what remedial actions are to be taken when it is decided that a rule has not been followed according to its intention.*

A Transportation Rules & Regulations Facet — Narrative. The purpose of maintaining an appropriate set of hub (and link) states may very well be to guide traffic into "smooth sailing" — avoiding traffic accidents etc. But this requires that vehicle drivers obey the hub states, that is, the signals. So there is undoubtedly a rule that says: *Obey traffic signals.*And, in consequence of human nature, overlooking or outright violating signals there is undoubtedly a regulation that says: *Violation of traffic signals is subject to fines and . . . .*

A Transportation Rules & Regulations Facet — Formalisation. We shall, regretfully, not show any formalisation of the above mentioned rule and regulation. To do a proper job at such a formalisation would require that we formalise traffics, say as (a type of) continuous functions from time to pairs of net and vehicle positions, that we define a number of auxiliary

(traffic monitoring) functions, including such which test whether from one instance of traffic, say at time t to a "next" instance of time, t', some one or more vehicles have violated the rule[7], etc. The "etcetera" is ominous: It implies modelling traffic wardens (police trying to apprehend the "sinner"), 'etc.' ! We rough-sketch an incomplete formalisation.

```
type
   T [ time ]
   V [ vehicle ]
   Rel_Distance = {| f:Rel • 0<f<1 |}
   VPos == VatH(h:H) | VonL(hif:HI,l:L,hit:HI,rel_distance:Rel_Distance)
   Traffic = T → (N × (V m→ VPos))
value
   violations: Traffic → (T×T) → V-set
```

Vehicle positions are either at hubs or some fraction f down a link (l) from some hub (hit) towards the connected hub (hit). Traffic maps time into vehicle positions. We omit a lengthy description of traffic well-formedness.

2.2.5 Domain Scripts

By a *domain script* we mean *the structured, almost, if not outright, formally expressed, wording of a rule or a regulation that has legally binding power, that is, which may be contested in a court of law.*

A Transportation Script Facet — Narrative. Regular buses ply the network according to some time table. We consider a train time table to be a script. Let us take the following to be a sufficiency narrative description of a train time table. For every train line, identified by a line number unique to within, say a year of operation, there is a list of train hub visits. A train hub visit informs of the intended arrival and departure times at identified hubs (i.e., train stations) such that "neighbouring" hub visits, (t_{a_i}, h_i, t_{d_i}) and (t_{a_j}, h_j, t_{d_j}), satisfy the obvious that a train cannot depart before it has arrived, and cannot arrive at the next, the "neighbouring" station before it has departed from the previous station, in fact, $t_{a_j} - t_{d_i}$ must be commensurate with the distance between the two stations.

A Transportation Script Facet — Formalisation.

```
type
   TLin
   HVis = T × HI × T
   Journey′ = HVis*, Journey = {|j:Journey′•len j≥2|}
   TimTbl′ = (TLin m→ Journey) × N
   TimTbl = {| timtbl:TimTbl′ • wf_TimTbl(timtbl) |}
value
   wf_TimTbl: TimTbl′ → Bool
   wf_TimTbl(tt,n) ≡
      [ all hubs designated in tt must be hubs of n ]
      [ and all journeys must be along feasible links of n ]
      [ and with commensurate timing net n constraints ]
```

[7] Here the time interval $t' - t$ could be thought of as long enough to cross a hub, h from link l to link l' where the signal at time t (or during the time $[t, t']$) does not allow traffic down l' from l.

2.2.6 Domain Human Behaviour

By *human behaviour* we mean *any of a quality spectrum of carrying out assigned work: from (i)* ***careful, diligent*** *and* ***accurate,*** *via (ii)* ***sloppy*** *dispatch, and (iii)* ***delinquent*** *work, to (iv) outright* ***criminal*** *pursuit.*

Transportation Human Behaviour Facets — Narrative. We have already exemplified aspects of human behaviour in the context of the transportation domain, namely vehicle drivers not obeying hub states. Other example can be given: drivers moving their vehicle along a link in a non-open direction, drivers waving their vehicle off and on the link, etcetera. Whether rules exists that may prohibit this is, perhaps, irrelevant. In any case we can "speak" of such driver behaviours — and then we ought formalise them !

Transportation Human Behaviour Facets — Formalisation. But we decide not to. For the same reason that we skimped proper formalisation of the violation of the *"obey traffic signals"* rule. But, by now, you've seen enough formulas and you ought trust that it can be done.

off_on_link: Traffic → (T×T) $\xrightarrow{\sim}$ (V $\xrightarrow[m]{}$ VPos×VPos)
wrong_direction: Traffic → T $\xrightarrow{\sim}$ (V $\xrightarrow[m]{}$ VPos)

2.3 Discussion

We have given a mere glimpse of a domain description. A full description of a reasonably "convincing" domain description will take years to develop and will fill many pages (hundreds, . . . (!)).

3 From Domains to Requirements

The objective of requirements engineering is to create a requirements prescription: A requirements prescription specifies externally observable properties of entities, functions, events and behaviours of *the machine* such as the requirements stakeholders wish them to be. The *machine* is what is required: that is, the *hardware* and *software* that is to be designed and which are to satisfy the requirements. A *requirements prescription* thus (*putatively* [54]) expresses what there should be. A requirements prescription expresses nothing about the design of the possibly desired (required) software. We shall show how a major part of a requirements prescription can be "derived" from "its" prerequisite domain description.

3.1 The Example Requirements

The domain was that of transportation. The requirements is now basically related to the issuance of tickets upon vehicle entry to a toll road net[8]and payment of tickets upon the vehicle leaving the toll road net both issuance and collection/payment of tickets occurring at toll booths[9]which are hubs somehow linked to the toll road net proper. Add to this that vehicle tickets are sensed and updated whenever the vehicle crosses an intermediate toll road intersection.

3.2 Stages of Requirements Engineering

The following are the stages of requirements engineering: stakeholder identification, *business process re-engineering*, *domain requirements development*, *interface development*, machine requirements development, requirements verification and validation, and requirements satisfiability and feasibility.

[9]Toll road: in other forms of English; tollway, turnpike, pike or toll-pike, in French péage.
[9]Toll plazas, toll stations, or toll gates

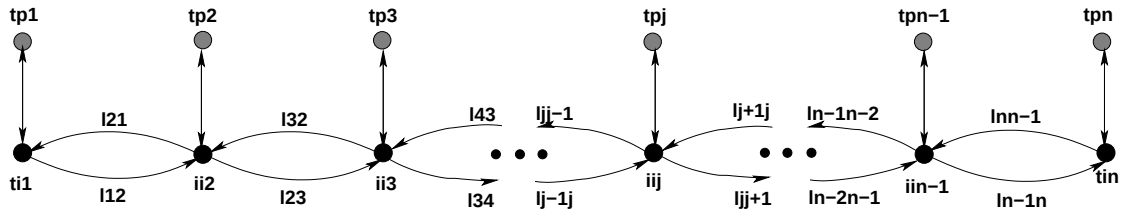

Figure 1: A simple, linear toll road net:
tp_i: *t*oll *p*laza i,
ti_1, ti_n: *t*erminal *i*ntersection k,
ii_k: *i*ntermediate *i*ntersection k, $1<k<n$
l_{xy}: tollway *l*ink from i_x to i_y, $y=x+1$ or $y=x-1$ and $1\leq x<n$.

The domain requirements development stage consists of a number of steps: projection, instantiation, determination, extension, and fitting

We shall basically only cover business process re-engineering, domain requirements development, and interface development

3.3 Business Process Re-engineering

Business process re-engineering (BPR) re-evaluates the intrinsics, support technologies, management & organisation, rules & regulations, scripts, and human behaviour facets while possibly changing some or all of these, that is, possibly rewriting the corresponding parts of the domain description.

3.3.1 Re-engineering Domain Entities

The net is arranged as a linear sequence of two or more (what we shall call) intermediate intersection hubs. Each intersection hub has a single two-way link to (what we shall call) an entry/exit hub (a toll plaza); and each intersection hub has either two or four one-way (what we shall call) tollway links: the first and the last intersection hub (in the sequence) has two tollway links and all intermediate intersections has four tollway links. We introduce a pragmatic notion of net direction: "up" and "down" the net, "from one end to the other". This is enough to give a hint at the re-engineered domain.

3.3.2 Re-engineering Domain Operations

We first briefly sketch the tollgate Operations. Vehicles enter and leave the tollway net only at entry/exit hubs (toll plazas). Vehicles collect and return their tickets from and to tollgate ticket issuing, respectively payment machines. Tollgate ticket issuing machines respond by issuing ticket as the result of sensor signals from "passing" vehicles or vehicle drivers pressing a button of the ticket issuing machine. Tollgate payment machines accept credit cards, bank notes or coins in designated currencies as payment and returns appropriate change.

We then briefly introduce and sketch an operation performed when vehicles cross intersections: The vehicle is assumed to possess the ticket issued upon entry (in)to the net (at a tollgate). At the crossing of each intersection, by a vehicle, its ticket is sensed and is updated with the fact that the vehicle crossed the intersection: time and date.

The updated domain description section on support technology will detail the exact workings of these tollgate and internal intersection machines; and the updated domain description section on human behaviour will likewise explore the man/machine facet. Erroneous operations as well as statistically determined undesirable behaviours of machines and humans need be carefully described.

3.3.3 Re-engineering Domain Events

The intersections are highway-engineered in such a way as to deter vehicle entry into opposite direction tollway links, yet, one never knows, there might still be (what we shall call ghost) vehicles, that is vehicles which have somehow defied the best intentions, and are observed moving along a tollway link in the wrong direction.

3.3.4 Re-engineering Domain Behaviours

The intended behaviour of a vehicle of the tollway is to enter at an entry hub (collecting a ticket at the toll gate), to move to the associated intersection, to move into, where relevant, either an upward or a downward tollway link, to proceed (i.e., move) along a sequence of one or more tollway links via connecting intersections, until turning into an exit link and leaving the net at an exit hub (toll plaza) while paying the toll.

• • •

This should be enough of a BPR rough sketch for us to meaningfully proceed to requirements prescription proper.

3.4 Domain Requirements Prescription

A domain requirements prescription is that part of the overall requirements prescription which can be expressed solely using terms from the domain description. Thus to construct the domain requirements prescription all we need is collaboration with the requirements stakeholders (who, with the requirements engineers, developed the BPR) and the possibly rewritten (resulting) domain description.

3.4.1 Domain Projection

By *domain projection* we mean a *subset of the domain description, one which leaves out all those entities, functions, events, and (thus) behaviours that the stakeholders do not wish represented by the machine.*

The resulting document is a *partial domain requirements prescription.*

Domain Projection — Narrative. We copy the domain description and call the copy a 0th version domain requirements prescription. From that document we remove all mention of link insertion and removal functions, to obtain a 1st version domain requirements prescription.[10]

Domain Projection — Formalisation. We do not show the resulting formalisation.

3.4.2 Domain Instantiation

By *domain instantiation* we mean a *refinement of the partial domain requirements prescription, resulting from the projection step, in which the refinements aim at rendering the entities, functions, events, and (thus) behaviours of the partial domain requirements prescription more concrete, more specific. Instantiations usually render these concepts less general.*

Domain Instantiation — Narrative. The 1st version domain requirements prescription is now updated with respect to the properties of the toll way net: We refer to Fig. 1 and the preliminary description given in Sect. 3.3.1. There are three kinds of hubs: tollgate hubs and intersection hubs: terminal intersection hubs and proper, intermediate intersection hubs. Tollgate hubs have one connecting two-way link. linking the tollgate hub to its associated intersection hub. Terminal intersection hubs have three connecting links: one, a two way link, to a tollgate hub, one

[10] Restrictions of the net to the toll road nets hinted at earlier will follow in the next domain requirements steps.

one-way link emanating to a next up (or down) intersection hub, and one one-way link incident upon this hub from a next up (or down) intersection hub. Proper intersection hubs have five connecting links: one, a two-way link, to a tollgate hub, two one-way links emanating to next up and down intersection hubs, and two one-way links incident upon this hub from next up and down intersection hub. (Much more need be narrated.) As a result we obtain a 2nd version domain requirements prescription.

Domain Instantiation — Formalisation, Toll Way Net.

type
  TN = ((H × L) × (H × L × L))* × H × (L × H) → N′
value
  abs_N: TN → N′
  abs_N(tn) ≡ (tn_hubs(tn),tn_hubs(tn))
  pre wf_TN(tn)

  tn_hubs: TN → H-**set**,
  tn_hubs(hll,h,(_,hn)) ≡
    {h,hn} ∪ {thj,hj|((thj,tlj),(hj,lj,lj′)):((H×L)×(H×L×L))•((thj,tlj),(hj,lj,lj′))∈ **elems** hlll}
  tn_links: TN → L-**set**
  tn_links(hll,_,(ln,_)) ≡
    {ln} ∪ {tlj,lj,lj′|((thj,tlj),(hj,lj,lj′)):((H×L)×(H×L×L))•((thj,tlj),(hj,lj,lj′))∈ **elems** hlll}

theorem ∀ tn:TN • wf_TN(tn) ⇒ wf_N(abs_N(tn))

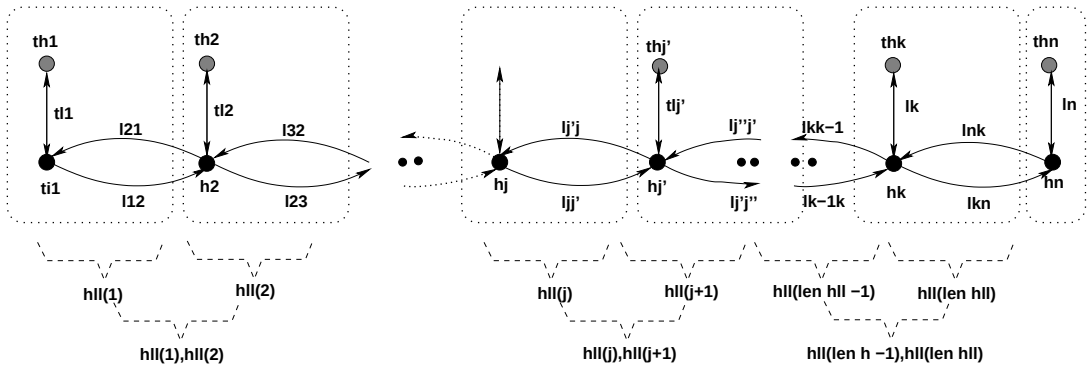

Figure 2: A simple, linear toll road net:
  th_i: *t*oll *p*laza *i*,
  h_1, h_n: *t*erminal *i*ntersections,
  h_2, h_j, h'_j, h_k: *i*ntermediate *i*ntersections, $1<j\leq k$, $k=n$-1
  l_{xy}, l_{yx}: tollway *l*ink from h_x to h_y and from h_y to h_x, $1\leq x<n$.
  l_{x-1x}, l_{xx-1}: tollway *l*ink from h_{x-1} to h_x and h_x to h_{x-1}, $1\leq x<n$,
  dashed links are not in formulas.

Domain Instantiation — Formalisation, Wellformedness.

type
  LnkM == plaza | way
value

```
wf_TN: TN → Bool
wf_TN(tn:(hll,h,(ln,hn))) ≡
    wf_Toll_Lnk(h,ln,hn)(plaza) ∧ wf_Toll_Ways(hll,h) ∧
    wf_State_Spaces(tn) [ to be defined under Determination ]

value
  wf_Toll_Ways: ((H×L)×(H×L×L))* × H → Bool
  wf_Toll_Ways(hll,h) ≡
    ∀ j:Nat • {j,j+1}⊆inds hll ⇒
      let ((thj,tlj),(hj,ljj′,lj′j)) = hll(j),
          (_,(hj′,_,_)) = hll(j+1) in
      wf_Toll_Lnk(thj,tlj,hj)(plaza) ∧
      wf_Toll_Lnk(hj,ljj′,hj′)(way) ∧ wf_Toll_Lnk(hj′,lj′j,hj)(way) end ∧
    let ((thk,tlk),(hk,lk,lk′)) = hll(len hll) in
      wf_Toll_Lnk(thk,tlk,hk)(plaza) ∧
      wf_Toll_Lnk(hk,lk,hk′)(way) ∧ wf_Toll_Lnk(hk′,lk′,hk)(way)  end

value
  wf_Toll_Lnk: (H×L×H) → LnkM → Bool
  wf_Toll_Lnk(h,l,h′)(m) ≡
    obs_Ps(l)={(obs_HI(h),obs_LI(l),obs_HI(h′)),
               (obs_HI(h′),obs_LI(l),obs_HI(h))} ∧
    obs_Σ(l) = case m of
                 plaza → obs_Ps(l),
                 way → {(obs_HI(h),obs_LI(l),obs_HI(h′))} end
```

3.4.3 Domain Determination

By *domain determination* we mean *a refinement of the partial domain requirements prescription, resulting from the instantiation step, in which the refinements aim at rendering the entities, functions, events, and (thus) behaviours of the partial domain requirements prescription less non-determinate, more determinate. Instantiations usually render these concepts less general.*

Domain Determination — Narrative. We single out only two 'determinations': There is only one link state: the set of all paths through the link; thus any link state space is the singleton set of its only link state. Similarly the hub state spaces are the singleton sets of the "current" hub states which allow these crossings: from terminal link back to terminal link, from terminal link to emanating tollway link, from incident tollway link to terminal link, and from incident tollway link to emanating tollway link This, then, is what free- and tollways are all about: no restriction on link and hub movement. Special provision must be made for expressing the entering from the outside and leaving toll plazas to the outside.

Domain Determination — Formalisation.

```
wf_State_Spaces: TN → Bool
wf_State_Spaces(hll,hn,(thn,tln)) ≡
  let ((th1,tl1),(h1,l12,l21)) = hll(1),
      ((thk,ljk),(hk,lkn,lnk)) = hll(len hll) in
  wf_Plaza(th1,tl1,h1) ∧ wf_Plaza(thn,tln,hn) ∧
  wf_End(h1,tl1,l12,l21,h2) ∧ wf_End(hk,tln,lkn,lnk,hn) ∧
  ∀ j:Nat • {j,j+1,j+2}⊆inds hll ⇒
    let (,(hj,ljj,lj′j)) = hll(j),((thj′,tlj′),(hj′,ljj′,lj′j′)) = hll(j+1) in
    wf_Plaza(thj′,tlj′,hj′) ∧ wf_Interm(ljj,lj′j,hj′,tlj′,ljj′,lj′j′) end end
```

wf_Plaza(th,tl,h) ≡
  obs_HΣ(th) = { [crossings at toll plazas]
    (″external″,obs_HI(th),obs_LI(tl)),(obs_LI(tl),obs_HI(th),″external″),
    (obs_LI(tl),obs_HI(th),obs_LI(tl))} ∧
  obs_HΩ(th) = {obs_HΣ(th)} ∧ obs_LΩ(tl) = {obs_LΣ(tl)}

wf_End(h,tl,l,l′) ≡
  obs_HΣ(h) = {[crossings at 3−link end hubs]
    (obs_LI(tl),obs_HI(h),obs_LI(tl)),(obs_LI(tl),obs_HI(h),obs_LI(l)),
    (obs_LI(l′),obs_HI(h),obs_LI(tl)),(obs_LI(l′),obs_HI(h),obs_LI(l))} ∧
  obs_HΩ(h) = {obs_HΣ(h)} ∧
  obs_LΩ(l) = {obs_LΣ(l)} ∧ obs_LΩ(l′) = {obs_LΣ(l′)}

wf_Interm(ul_1,dl_1,h,tl,ul,dl) ≡
  obs_HΣ(h) = {[crossings at properly intermediate, 5−link hubs]
    (obs_LI(tl),obs_HI(h),obs_LI(tl)),(obs_LI(tl),obs_HI(h),obs_LI(dl_1)),
    (obs_LI(tl),obs_HI(h),obs_LI(ul)),(obs_LI(ul_1),obs_HI(h),obs_LI(tl)),
    (obs_LI(ul_1),obs_HI(h),obs_LI(ul)),(obs_LI(ul_1),obs_HI(h),obs_LI(dl_1)),
    (obs_LI(dl),obs_HI(h),obs_LI(tl)),(obs_LI(dl),obs_HI(h),obs_LI(dl_1)),
    (obs_LI(dl),obs_HI(h),obs_LI(ul))} ∧
    obs_HΩ(h) = {obs_HΣ(h)} ∧ obs_LΩ(tl) = {obs_LΣ(tl)} ∧
    obs_LΩ(ul) = {obs_LΣ(ul)} ∧ obs_LΩ(dl) = {obs_LΣ(dl)}

Not all determinism issues above have been fully explained. But for now we should — in principle — be satisfied.

3.4.4 Domain Extension

By domain extension we understand the *introduction of domain entities, functions, events and behaviours that were not feasible in the original domain, but for which, with computing and communication, there is the possibility of feasible implementations, and such that what is introduced become part of the emerging domain requirements prescription.*

Domain Extension — Narrative. The domain extension is that of the controlled access of vehicles to and departure from the toll road net: the entry to (and departure from) tollgates from (respectively to) an `"an external"` net — which we do not describe; the new entities of tollgates with all their machinery; the user/machine functions: upon entry: driver pressing entry button, tollgate delivering ticket; upon exit: driver presenting ticket, tollgate requesting payment, driver providing payment, etc.

One added (extended) domain requirements: as vehicles are allowed to cruise the entire net payment is a function of the totality of links traversed, possibly multiple times. This requires, in our case, that tickets be made such as to be sensed somewhat remotely, and that intersections be equipped with sensors which can record and transmit information about vehicle intersection crossings. (When exiting the tollgate machine can then access the exiting vehicles' sequence of intersection crossings — based on which a payment fee calculation can be done.)

All this to be described in detail — including all the thinks that can go wrong (in the domain) and how drivers and tollgates are expected to react.

Domain Extension — Formalisation. We suggest only some signatures:

type
  Mch, Tik, Csh, Pmt, Map_TN
  Gat == open | closed

value
obs_Csh: Mch → Csh, obs_Tiks: Mach → Tik-**set**, obs_Gat: Mch → Gat
obs_Entry, obs_Exit: Tik → HI, obs_Tik: V → (Ticket|nil)
obs_Journey: Ticket → HI*

calc_Pmtt: HI* → Map_TN → Pmt

press_Entry: Mch → Mch × Ticket [gate up]
press_Exit: Mch × Ticket → Mch × Payment
payment: Mch × Payment → Mch × Cash [gate up]

Domain Extension — Formalisation of Support Technology. This example provides a classical requirements engineering setting for embedded, safety critical, real-time systems, requiring, ultimately, the techniques and tools of such things as Petri nets, statecharts, message sequence charts (MSCs) or live sequence charts (LSCs), and temporal logics (ITL, DC, TLA+).

3.4.5 Requirements Fitting

The issue of requirements fitting arises when two or more software development projects are based on what appears to be the same domain. The problem then is to harmonise the two or more software development projects by harmonising, if not too late, their requirements developments.

We thus assume that there are n domain requirements developments, d_{r_1}, d_{r_2}, ..., d_{r_n}, being considered, and that these pertain to the same domain — and can hence be assumed covered by a same domain description.

By requirements fitting we mean *a harmonisation of $n > 1$ domain requirements that have overlapping (common) not always consistent parts and which results in n 'modified and partial domain requirements', and m 'common domain requirements' that "fit into" to two or more of the 'modified and partial domain requirements'.*

By a *modified and partial domain requirements* we mean a domain requirements which is short of (that is, is missing) some description parts: text and formula. By a *common domain requirements* we mean a domain requirements. By the m common domain requirements parts, *cdrs*, *fitting into* the n modified and partial domain requirements we mean that there is for each modified and partial domain requirements, $mapdr_i$, an identified subset of *cdrs* (could be all of *cdrs*), *scdrs*, such that textually conjoining *scdrs* to *mapdr* can be claimed to yield the "original" d_{r_i}.

Requirements Fitting Procedure — A Sketch. Requirements fitting consists primarily of a pragmatically determined sequence of analytic and synthetic ('fitting') steps. It is first decided which n domain requirements documents to fit. Then a 'manual' analysis is made of the selected, n domain requirements. During this analysis tentative common domain requirements are identified. It is then decided which m common domain requirements to single out. This decision results in a tentative construction of n modified and partial domain requirements. An analysis is made of the tentative modified and partial and also common domain requirements. A decision is then made whether to accept the resulting documents or to iterate the steps above.

Requirements Fitting — Narrative. We postulate two domain requirements: We have outlined a domain requirements development for software support for a toll road system. We have earlier hinted at domain operations related to insertion of new and removal of existing links and hubs. We can therefore postulate that there are two domain requirements developments, both based on the transport domain: one, $d_{r_{\text{toll}}}$, for a toll road computing system monitoring and controlling vehicle flow in and out of toll plazas, and another, $d_{r_{\text{maint.}}}$, for a toll link and intersection

(i.e., hub) building and maintenance system monitoring and controlling link and hub quality and for development.

The fitting procedure now identifies the shared net entities by both $d_{r_{\text{toll}}}$ and $d_{r_{\text{maint.}}}$. That is: nets (N), hubs (H) and links (L) appear in both $d_{r_{\text{toll}}}$ and $d_{r_{\text{maint.}}}$. We conclude from this that we can single out a common requirements for software that manages net, hubs and links. Such software requirements basically amounts to requirements for a database system. A suitable such system, say a relational database management system, DB_{rel}, may already be available with the customer.

In any case, where there before were two requirements $(d_{r_{\text{toll}}}, d_{r_{\text{maint.}}})$ there are now four: (i) $d'_{r_{\text{toll}}}$, a modification of $d_{r_{\text{toll}}}$ which omits the description parts pertaining to the net; (ii) $d'_{r_{\text{maint.}}}$, a modification of $d_{r_{\text{maint.}}}$ which likewise omits the description parts pertaining to the net; (iii) $d_{r_{\text{net}}}$, which contains what was basically omitted in $d'_{r_{\text{toll}}}$ and $d'_{r_{\text{maint.}}}$; and (iv) $d_{r_{\text{db:i/f}}}$ (for database interface) which prescribes a mapping between type names of $d_{r_{\text{net}}}$ and relation and attribute names of DB_{rel}.

Much more can and should be said, but this suffices as an example in a software engineering methodology paper.

Requirements Fitting — Formalisation. We omit lengthy formalisation.

3.4.6 Domain Requirements Consolidation

After projection, instantiation, determination, extension and fitting, it is time to review, consolidate and possibly restructure (including re-specify) the domain requirements prescription before the next stage of requirements development.

3.5 Interface Requirements Prescription

By an *interface requirements* we mean *a requirements prescription which refines and extends the domain requirements by considering those requirements of the domain requirements whose entities, operations, events and behaviours are "shared" between the domain and the machine (being requirements prescribed).*

'Sharing' means *(a) that an entity is represented both in the domain and "inside" the machine, and that its machine representation must at suitable times reflect its state in the domain; (b) that an operation which is to be supported by the machine requires a sequence of several "online" interactions between the machine (being requirements prescribed) and the domain, usually a person or another machine; (c) that an event arises either in the domain, that is, in the environment of the machine, or in the machine, and need be communicated to the machine, respectively to the environment; and (d) that a behaviour is manifested both by actions and events of the domain and by actions and events of the machine.*

3.6 Interface Requirements Prescription

So a systematic reading of the domain requirements shall result in an identification of all shared entities, operations, events and behaviours.

Each such shared phenomenon shall then be individually dealt with: *entity sharing* shall lead to interface requirements for data initialisation and refreshment; *operation sharing* shall lead to interface requirements for interactive dialogues between the machine and its environment; *event sharing* shall lead to interface requirements for how such event are communicated between the environment of the machine and the machine, both directions; *behaviour sharing* shall lead to interface requirements for action and event dialogues between the machine and its environment.

• • •

We shall now illustrate these domain interface requirements development steps with respect to our ongoing example.

3.6.1 Shared Entities

The main shared entities are the net, hence the hubs and the links. As domain entities they continuously undergo changes with respect to the values of a great number of attributes and otherwise possess attributes — most of which have not been mentioned so far: length, cadestral information, namings, wear and tear (where-ever applicable), last/next scheduled maintenance (where-ever applicable), state and state space, and many others.

We "split" our interface requirements development into two separate steps: the development of $d_{r_{\text{net}}}$ (the common domain requirements for the shared hubs and links), and the co-development of $d_{r_{\text{db:i/f}}}$ (the common domain requirements for the interface between $d_{r_{\text{net}}}$ and DB_{rel} — under the assumption of an available relational database system DB_{rel}

When planning the common domain requirements for the net, i.e., the hubs and links, we enlarge our scope of requirements concerns beyond the two so far treated ($d_{r_{\text{toll}}}$, $d_{r_{\text{maint.}}}$) in order to make sure that the shared relational database of nets, their hubs and links, may be useful beyond those requirements. We then come up with something like hubs and links are to be represented as tuples of relations; each net will be represented by a pair of relations a hubs relation and a links relation; each hub and each link may or will be represented by several tuples; etcetera. In this database modelling effort it must be secured that "standard" operations on nets, hubs and links can be supported by the chosen relational database system DB_{rel}.

Data Initialisation. As part of $d_{r_{\text{net}}}$ one must prescribe data initialisation, that is provision for an interactive user interface dialogue with a set of proper display screens, one for establishing net, hub or link attributes (names) and their types and, for example, two for the input of hub and link attribute values. Interaction prompts may be prescribed: next input, on-line vetting and display of evolving net, etc. These and many other aspects may therefore need prescriptions.

Essentially these prescriptions concretise the insert link operation.

Data Refreshment. As part of $d_{r_{\text{net}}}$ one must also prescribe data refreshment: an interactive user interface dialogue with a set of proper display screens one for updating net, hub or link attributes (names) and their types and, for example, two for the update of hub and link attribute values. Interaction prompts may be prescribed: next update, on-line vetting and display of revised net, etc. These and many other aspects may therefore need prescriptions.

These prescriptions concretise remove and insert link operations.

3.6.2 Shared Operations

The main shared operations are related to the entry of a vehicle into the toll road system and the exit of a vehicle from the toll road system.

Interactive Operation Execution. As part of $d_{r_{\text{toll}}}$ we must therefore prescribe the varieties of successful and less successful sequences of interactions between vehicles (or their drivers) and the toll gate machines.

The prescription of the above necessitates determination of a number of external events, see below.

(Again, this is an area of embedded, real-time safety-critical system prescription.)

3.6.3 Shared Events

The main shared external events are related to the entry of a vehicle into the toll road system, the crossing of a vehicle through a toll way hub and the exit of a vehicle from the toll road system.

As part of $d_{r_{\text{toll}}}$ we must therefore prescribe the varieties of these events, the failure of all appropriate sensors and the failure of related controllers: gate opener and closer (with sensors and actuators), ticket "emitter" and "reader" (with sensors and actuators), etcetera.

The prescription of the above necessitates extensive fault analysis.

3.6.4 Shared Behaviours

The main shared behaviours are therefore related to the journey of a vehicle through the toll road system and the functioning of a toll gate machine during "its lifetime". Others can be thought of, but are omitted here.

In consequence of considering, for example, the journey of a vehicle behaviour, we may "add" some further, extended requirements: (a) requirements for a vehicle statistics "package"; (b) requirements for tracing supposedly "lost" vehicles; (c) requirements limiting toll road system access in case of traffic congestion; etcetera.

3.7 Discussion

We have indicated how to "derive" domain and interface requirements from a domain model. Development of machine requirements, such as performance, dependability, maintenance, platform, documentation and other machine requirements, follow more general guidelines as these requirements are usually expressible without any specific reference to the domain.

4 Instantiations of Generic Model of Transportation

4.1 Road Transport

In Sect. 3 we have shown how to refine the model of Sect. 2 into a model of a toll road system. Thus with no changes to the model of Sect. 2 we simply refined that model. The net became a toll road net between toll plazas with the toll gate ticket issuing and ticket and payment collection machines. In Sect. 3 furthermore illustrated how to develop requirement prescriptions from domain descriptions.

4.2 Rail Transport

4.2.1 Rail Nets

The concepts of rail nets, train stations, trains, etc., has to be related to generic nets, sub-nets of these, vehicles (or convoys of these), etc. We shall basically "equate" the generic nets with rail nets, provided we can provide suitable, believable, relations between hubs and the special rail units of switches, crossovers, switchable crossovers, etcetera, that is between hub states and the states of rail units, and between link segment states and states of linear rail units, between vehicles and convoys and train cars and trains, etc. We shall do so presently.

For rail transport we thus need to introduce a notion of sub-nets. Out of a net one can "carve islands" of sub-nets which we shall use to represent train stations. What has not been carved out is then the net of rail tracks between stations. The hubs of the generic nets become such rail units as switches, crossovers, and switchable crossovers. The links of the generic nets become linear sequences of one or more linear rail units. There are notably many hubs within station sub-nets and thus fewer outside these — as there are notably many links (link segments) outside station sub-nets and somewhat fewer inside. Figure 3[A] shows four such rail units and Fig. 3[B] shows a "toy" rail net.

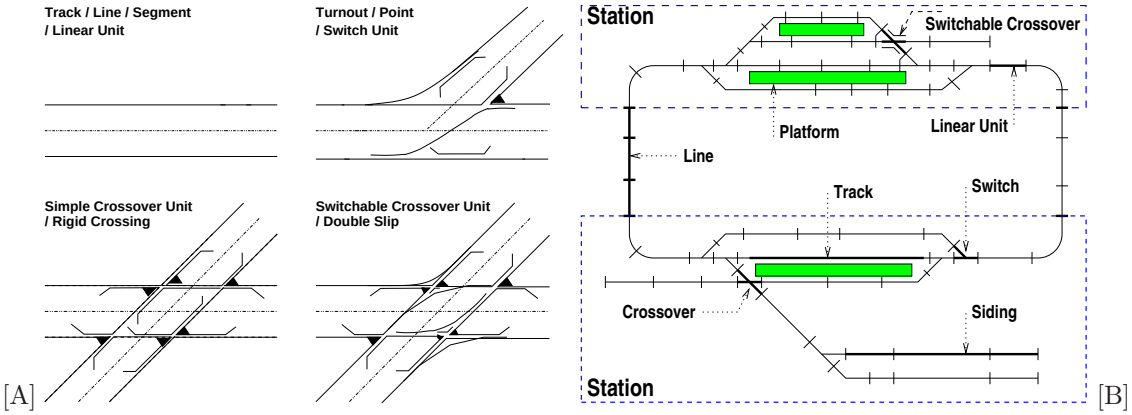

Figure 3: [A] Rail units. [B] A "toy" rail net

4.2.2 Rail Units

And we need to introduce, based on hubs and links — or rather link segments — a notion of a rail unit. We shall consider four kinds of rail units: *simple linear* units with up to four states: closed, open (only) in one direction, open (only) in the other direction and open in both directions; *simple switches*, see Fig. 4, with up to 12 states;

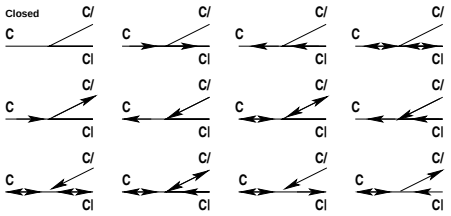

Figure 4:

simple crossovers, with 16 states:[11], (i) (i) cD1 + cD2, (ii) cD1 + uD2, (iii) cD1 + dD2, (iv) cD1 + bD2, (v) cD2 + uD1, (vi) cD2 + dD1, (vii) cD2 + bD1, (viii) uD1 + uD2, (ix) uD1 + dD2, (x) uD1 + bD2, (xi) dD1 + uD2, (xii) dD1 + dD2, (xiii) dD1 + bD2, (xiv) bD1 + uD2, (xv) bD1 + dD2 and (xvi) bD1 + bD2. — some of these states are pretty dangerous; and *switchable crossovers* — we leave it to you to figure out how many potential states they may have !

Hubs are switch, crossover and switchable crossover units. Links are sets of linear units, that is, link segments are linear units, and these can be ordered into appropriate sequences.

4.2.3 Stations

Sub-nets of the net are stations such that any two distinct sub-nets of a net do not overlap but if the two stations are to be directly connected then it is via a route with at least one (linear ?) unit. From a sub-net one can observe a sub-net name. No two distinct sub-nets have the same (station) name. Etcetera.

4.2.4 Discussion

We refrain from formalising the above simple refinements of generic nets to rail nets. More refinements need to be shown: refinements of vehicles and convoys into railway rolling stock

[11] D1: diagonal 1, D2: diagonal 2, c: closed, u: open upwards, d: open downwards, b: open both ways.

(locomotives, passenger wagons and freight cars); refinements of certain states of units into states of signals; etc., before we can fully claim that the generic model of transportation given earlier "maps" into railways. But we trust that the reader can be convinced.

4.3 Air Transport

4.3.1 Airports and Air Lanes

We must refine generic transportation nets into air traffic nets. So we need to establish refinements between air lanes, taxiways, runways, etc. and links, and between airport ramps (or aprons) and hubs, and between airports and sub-nets of generic nets. Then we can consider air transport (air traffic) nets as consisting of air lanes and airport nets, with these latter including runway links, taxiway links, the ramp (apron) hub and the gates (terminal ramp and hub) as well as the hubs that connect runways and taxiways, taxiways and the apron hub, and the apron hub to gates (the terminal ramps).

Add to the above that air-based holding areas can be considered refinements of hubs, and the reader should be convinced that generic transportation nets refine into air traffic nets.

The ground-based air traffic links and hubs are fixed, i.e., changes only as a consequence of removing and inserting links. The air-based holding area hubs are usually pretty stable. The set of air-based links, i.e., the air lanes, however, vary dynamically. You may consider for the sake of argument, that there is, from any airport an air lane to any other airport, no matter how long the distance is. But one must consider two things in this respect: that these air lanes overlap in their occupancy of air space, and that an air flight may change from one air lane to another that is, from one link to another — during any flight. This possibility of vehicles "jumping" across links was not considered for generic transportation so we must "go back" and "repair" our generic model. The simplest way to "repair" the generic model is to not repair it, but to consider any air lane from any airport to — as we mandated above — any other airport not as a single link but as an indefinite, modelled formally as an infinite, set of air lanes: a "stem" (master) air lane "decorated" (augmented, interspersed) with hubs which additionally connect to links to all "nearest" airports should the aircraft pilot decide to divert a flight originally planned for — and "presently" cruising the stem air lane.

We have retained our desire to refine the air traffic net from the generic transportation net at the no-cost of a great number of maintenance and operations cost-free air-based links and hubs.

4.3.2 Link and Hub States and Their Control

The states and state spaces of links and hubs need be refined into the monitoring and control of air traffic on the ground as well as in the air. Not into requirements for ATC but as descriptions of how it is now, today.

Today the air traffic monitoring and control domain is handled that is, the setting of link and hub states by something along the following (schematised) line: ground control (GCs) and terminal control towers (TCTs), and area (regional) (ACCs)and continental control centers (CCCs). See Fig. 5.

Ground control (GC) handle airport net traffic (surface movement) handed over to them from TCTs, clears scheduled flights for departure with (remote) TCTs at respective destinations, often via CCCs, and hands over such flights once ready at runways to local TCTs.

Terminal control towers (TCTs), that is, local and air controls handle incoming flights handed over to them from ACCs, possibly assigning to or releasing them from holding areas, guiding them into landing, while handing them over to GCs. The TCTs also guide departing aircraft (handed over to them from GCs) onto runways and into the air while handing them over to ACCs. There might be an occasional need for consulting CCCs.

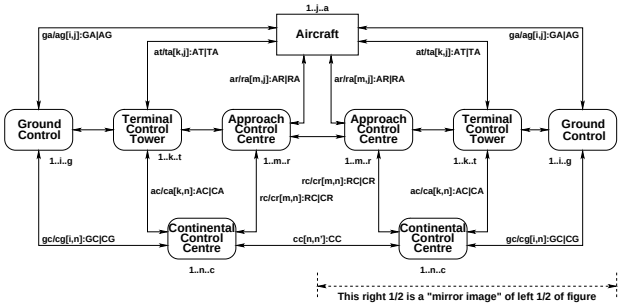

Figure 5: An idealised diagramming of air traffic control tower and centre relations to aircraft.

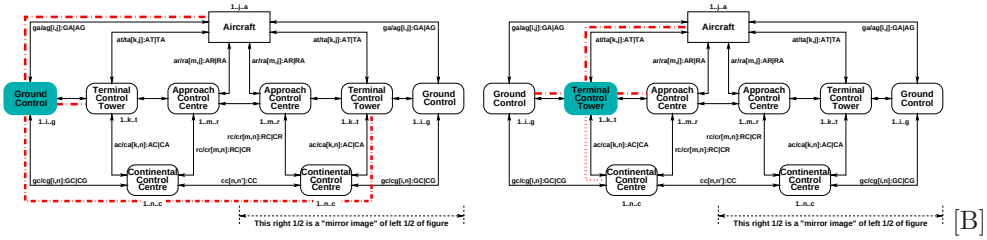

Figure 6: [A] Ground control and [B] terminal control tower ATC.

Approach control centers (ARCs), also referred to as terminal control centers, handle incoming and transit flights handed to them from other ACCs (with handling means: monitoring and possibly "controlling", i.e., guiding them); possibly handing them (transit flights) over to other ACCs, or handing them (incoming flights) over to TCTs.

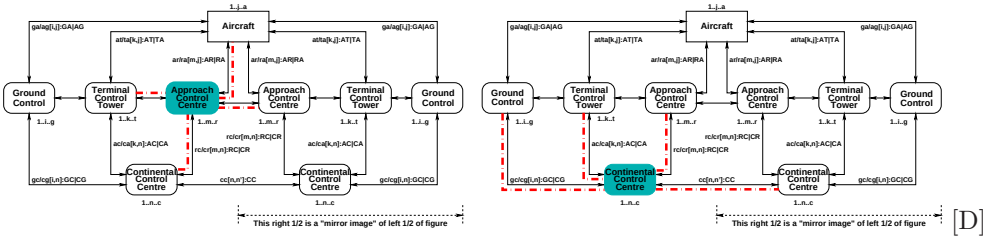

Figure 7: [C] Approach and [D] continental control center ATC.

Continental control centers (CCCs) usually handle requests specific intercontinental flights by coordinating GC, TCT and ACC actions. CCCs usually allocates and schedules continental flights, not on a per flight basis, but on a by volume of traffic basis.

It is thus we see that the monitoring and control of traffic implicit in the changing of link and hub states in the generic transportation model is being significantly detailed in the air traffic model.

There are no visible link and hub states as was not assumed in the generic transportation model, there are dialogues between ATC staff and with aircraft pilots. There is no a-priori set link and hub state spaces as was perhaps indicated in the in the generic transportation model. These state spaces evolve over time and with ATC staff and aircraft pilot experience.

4.3.3 Discussion — A Summary

We have sketched how to "re-interpret" generic transportation nets, that is, we have shown how to refine these into air traffic nets consisting of sub-nets of airport nets and an air lane net[12]. Airport nets have runway, taxiway and gate ramps connected (sometimes via "seamless" hubs) to air lanes and taxiways, to runways and the amorphous and large apron hub, respectively to the apron hub and the gate ramp. Air lane nets have an indefinite number of air lane link segments and holding area and an indefinite number of link segment-connecting hubs.

Air transportation has an elaborate "constellation" of vehicle, link and hub state and state space monitoring and control, organised and managed according to internationally set conventions.

But, despite all this, we think that the generic transportation model can be nicely refined into a model for current ATC which can serve as a basis for requirements for far more advanced ATC that we see today.

4.4 Sea Transport

We, perhaps somewhat surprisingly, leave it as an exercise, for the reader to sketch the refinement of the generic transportation model into one for — for example — container line shipping.

Such a rather complete model is given in [17].

4.5 Discussion

In Sect. 3 we have shown how to instantiate the generic transportation model of Sect. 2 to one for road transport. Sections 4.2, 4.3 and 4.4 indicated how to similarly instantiate the generic transportation model to models of rail, air and sea transport respectively. Where Sect. 3 strongly hinted at formalising the refinement, Sects. 4.2, 4.3 and 4.4 did not contain any formalisations. They could, of course, be provided, but the reader should be convinced, "proof by authority", that it can be done. We thus refer to a number of papers on railway specifications [20, 80, 69, 7, 8, 9, 10, 68, 11], to a paper and an MSc report on air traffic [6] and http://www.imm.dtu.dk/~db/airtraffic.pdf, as well as to an emerging paper containing an extensive example of a domain description for sea transport [17].

5 Conclusion

We set out to claim that, for certain domains, and the example here was the transportation domain, one can first develop a generic domain model, and then instantiate (refine) this generic model in several "directions", as here: road transport, rail transport, air transport and ship transport. We conclude now that we have achieved that claim.

5.1 What Have We Not Shown ?

Among, undoubtedly the most crucial aspects of software development — at least to the audience of this conference[13] — are the methodology and example of those facets of the domain, typically the supporting technologies which eventually lead to embedded systems depending on real-time properties and requiring safety. There are many other aspects that have not been shown, but which, when shown, do show the feasibility of this approach for professional, industry-scale software developments: the use of varieties of "co-operating, commensurate" formal techniques for full scale developments. Needless to say, most such available techniques can be and many have been integrated into the kind of formal techniques shown in this paper.

[12] The air lane net, as we have seen, is like a "soup" of links and hubs floating in a liquid of air space.

[13] ISOLA 2007: Workshop On Leveraging Applications of Formal Methods, Verification and Validation. Special Workshop Theme: Formal Methods in Avionics, Space and Transport; Poitiers, France, December 12-14 2007

5.2 Problems to Be Solved

Of course there is much more work to do: on the engineering side the work is to further develop such generic and instantiated models before they can properly serve as the basis for serious requirements engineering, and on the science side the work is to further develop the methodology: principles, techniques and tools for the systematic, or rigorous, or even formal development of provably correct transportation systems.

5.3 A Very-large Scale Systems Development Method

We have shown how we tackle the development of very large scale systems, systems that consists of hundreds of separably describable sub-systems, several scores of different job-profile staff, and of dozens of different kinds of interfaces to an external world. We have shown how to do it by domain engineering, requirements engineering, and, but not shown, software, etc., design. The simple fact is: to do systems development you must first describe their domain. There are no shortcuts !

6 Acknowledgments

I gratefully acknowledge support from Université Henri Poincaré (UHP), Nancy, and from INRIA (l'Institut National de Recherche en Informatique et en Automatique) both of France, for my two month stay at LORIA (Laboratoire Lorrain de Recherche en Informatique et ses Applications), Nancy, in the fall of 2008. I especially and warmly thank Dominique Méry for hosting me. And I thank the main organiser of this workshop at ISOLA 2007 Yamine Ait Ameur, LISI/ENSMA, Poitiers, France, for inviting me — thus forcing me to, all too willingly, write this paper.

7 Bibliographical Notes

In [18, to appear] I give a concise overview of domain engineering; in [19, to appear] one of domain and requirements engineering as they relate; and in [16, to appear] I relate domain enginering, requirements engineering and software design to software management. In [15] I present a number of domain engineering research challenges. In [17, to appear] — which also covers research challenges of domain engineering — I additionally present a rather large example of the container line industry domain.

References

[1] Jean-Raymond Abrial. *The B Book: Assigning Programs to Meanings*. Tracts in Theoretical Computer Science. Cambridge University Press, Cambridge, England, 1996.

[2] Keijiro Araki, Andy Galloway, and Kenji Taguchi, editors. *IFM 1999: Integrated Formal Methods*, volume 1945 of *Lecture Notes in Computer Science*, York, UK, June 1999. Springer. Proceedings of 1st Intl. Conf. on IFM.

[3] Dines Bjørner. Programming in the Meta-Language: A Tutorial. In Dines Bjørner and Cliff B. Jones, editors, *The Vienna Development Method: The Meta-Language, [22]*, LNCS, pages 24–217. Springer–Verlag, 1978.

[4] Dines Bjørner. Software Abstraction Principles: Tutorial Examples of an Operating System Command Language Specification and a PL/I-like On-Condition Language Definition. In Dines Bjørner and Cliff B. Jones, editors, *The Vienna Development Method: The Meta-Language, [22]*, LNCS, pages 337–374. Springer–Verlag, 1978.

[5] Dines Bjørner. The Vienna Development Method: Software Abstraction and Program Synthesis. In *Mathematical Studies of Information Processing*, volume 75 of *LNCS*. Springer–Verlag, 1979. Proceedings of Conference at Research Institute for Mathematical Sciences (RIMS), University of Kyoto, August 1978.

[6] Dines Bjørner. Software Systems Engineering — From Domain Analysis to Requirements Capture: An Air Traffic Control Example. In *2nd Asia-Pacific Software Engineering Conference (APSEC '95)*. IEEE Computer Society, 6–9 December 1995. Brisbane, Queensland, Australia.

[7] Dines Bjørner. Dynamics of Railway Nets: On an Interface between Automatic Control and Software Engineering. In *CTS2003: 10th IFAC Symposium on Control in Transportation Systems*, Oxford, UK, August 4-6 2003. Elsevier Science Ltd. Symposium held at Tokyo, Japan. Editors: S. Tsugawa and M. Aoki.

[8] Dines Bjørner. New Results and Trends in Formal Techniques for the Development of Software for Transportation Systems. In *FORMS2003: Symposium on Formal Methods for Railway Operation and Control Systems*. Institut für Verkehrssicherheit und Automatisierungstechnik, Techn.Univ. of Braunschweig, Germany, 15–16 May 2003. Conf. held at Techn.Univ. of Budapest, Hungary. Editors: G. Tarnai and E. Schnieder, Germany.

[9] Dines Bjørner. The Grand Challenge – FAQs of the R&D of a Railway Domain Theory. In *IFIP World Computer Congress, Topical Days: TRain: The Railway Domain*, IFIP, Amsterdam, The Netherlands, 2004. Kluwer Academic Press.

[10] Dines Bjørner. The TRain Topical Day. In *Building the Information Society, IFIP 18th World Computer Congress, Tpical Sessions, 22–27 August, 2004, Toulouse, France — Ed. Renéne Jacquart*, pages 607–611. Kluwer Academic Publishers, August 2004. A Foreword.

[11] Dines Bjørner. Towards a Formal Model of CyberRail. In *Building the Information Society, IFIP 18th World Computer Congress, Tpical Sessions, 22–27 August, 2004, Toulouse, France — Ed. Renéne Jacquart*, pages 657–664. Kluwer Academic Publishers, August 2004. Original report also listed some of DB's students as co–authors.

[12] Dines Bjørner. *Software Engineering, Vol. 1: Abstraction and Modelling*. Texts in Theoretical Computer Science, the EATCS Series. Springer, 2006.

[13] Dines Bjørner. *Software Engineering, Vol. 2: Specification of Systems and Languages*. Texts in Theoretical Computer Science, the EATCS Series. Springer, 2006. Chapters 12–14 are primarily authored by Christian Krog Madsen.

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

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

[16] Dines Bjørner. Believable Software Management. *Encyclopedia of Software Engineering*, 1(1):1–32, 2008. (This is a new journal, published by Taylor & Francis, New York and London, edited by Philip Laplante).

[17] Dines Bjørner. Domain Engineering. In *The 2007 Lipari PhD Summer School*, volume ???? of *Lecture Notes in Computer Science (eds. E. Börger and A. Ferro)*, pages 1–102, Heidelberg, Germany, 2008. Springer. To appear.

[18] Dines Bjørner. Domain Engineering. In *BCS FACS Seminars*, Lecture Notes in Computer Science, the BCS FAC Series (eds. Paul Boca and Jonathan Bowen), pages 1–42, London, UK, 2008. Springer. To appear.

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

[20] Dines Bjørner, Chris W. George, and Søren Prehn. Computing Systems for Railways — A Rôle for Domain Engineering. Relations to Requirements Engineering and Software for Control Applications. In *Integrated Design and Process Technology. Editors: Bernd Kraemer and John C. Petterson*, P.O.Box 1299, Grand View, Texas 76050-1299, USA, 24–28 June 2002. Society for Design and Process Science.

[21] Dines Bjørner and Martin C. Henson, editors. *Logics of Specification Languages — see [74, 28, 31, 63, 39, 34, 62, 32, 46].* EATCS Monograph in Theoretical Computer Science. Springer, Heidelberg, Germany, 2008.

[22] Dines Bjørner and Cliff B. Jones, editors. *The Vienna Development Method: The Meta-Language*, volume 61 of *LNCS*. Springer–Verlag, 1978. This was the first monograph on *Meta-IV*. [3, 4, 5].

[23] Dines Bjørner and Cliff B. Jones, editors. *Formal Specification and Software Development*. Prentice-Hall, 1982.

[24] Eerke A. Boiten, John Derrick, and Graeme Smith, editors. *IFM 2004: Integrated Formal Methods*, volume 2999 of *Lecture Notes in Computer Science*, London, England, April 4-7 2004. Springer. Proceedings of 4th Intl. Conf. on IFM. ISBN 3-540-21377-5.

[25] E. Börger and R. Stärk. *Abstract State Machines. A Method for High-Level System Design and Analysis*. Springe, 2003. ISBN 3-540-00702-4.

[26] Michael J. Butler, Luigia Petre, and Kaisa Sere, editors. *IFM 2002: Integrated Formal Methods*, volume 2335 of *Lecture Notes in Computer Science*, Turku, Finland, May 15-18 2002. Springer. Proceedings of 3rd Intl. Conf. on IFM. ISBN 3-540-43703-7.

[27] Dominique Cansell and Dominique Méry. Logical Foundations of the B Method. *Computing and Informatics*, 22(1–2), 2003. This paper is one of a series: [73, 30, 64, 36, 61, 47] appearing in a double issue of the same journal: *Logics of Specification Languages* — edited by Dines Bjørner.

[28] Dominique Cansell and Dominique Méry. *Logics of Specification Languages*, chapter The event-B Modelling Method: Concepts and Case Studies, pages 47–152 in [21]. Springer, 2008.

[29] Werner Damm and David Harel. LSCs: Breathing life into Message Sequence Charts. *Formal Methods in System Design*, 19:45–80, 2001. Early version appeared as Weizmann Institute Tech. Report CS98-09, April 1998. An abridged version appeared in *Proc. 3rd IFIP Int. Conf. on Formal Methods for Open Object-based Distributed Systems* (FMOODS'99), Kluwer, 1999, pp. 293–312.

[30] Ražvan Diaconescu, Kokichi Futatsugi, and Kazuhiro Ogata. CafeOBJ: Logical Foundations and Methodology. *Computing and Informatics*, 22(1–2), 2003. This paper is one of a series: [73, 27, 64, 36, 61, 47] appearing in a double issue of the same journal: *Logics of Specification Languages* — edited by Dines Bjørner.

[31] Răzvan Diaconescu. *Logics of Specification Languages*, chapter A Methodological Guide to the CafeOBJ Logic, pages 153–240 in [21]. Springer, 2008.

[32] John S. Fitzgerald. *Logics of Specification Languages*, chapter The Typed Logic of Partial Functions and the Vienna Development Method, pages 453–487 in [21]. Springer, 2008.

[33] John S. Fitzgerald and Peter Gorm Larsen. *Developing Software using VDM-SL*. Cambridge University Press, The Edinburgh Building, Cambridge CB2 1RU, England, 1997.

[34] Chris George and Anne E. Haxthausen. *Logics of Specification Languages*, chapter The Logic of the RAISE Specification Language, pages 349–399 in [21]. Springer, 2008.

[35] Chris W. George, Peter Haff, Klaus Havelund, Anne Elisabeth Haxthausen, Robert Milne, Claus Bendix Nielsen, Søren Prehn, and Kim Ritter Wagner. *The RAISE Specification Language*. The BCS Practitioner Series. Prentice-Hall, Hemel Hampstead, England, 1992.

[36] Chris W. George and Anne E. Haxthausen. The Logic of the RAISE Specification Language. *Computing and Informatics*, 22(1–2), 2003. This paper is one of a series: [73, 27, 30, 64, 61, 47] appearing in a double issue of the same journal: *Logics of Specification Languages* — edited by Dines Bjørner.

[37] Chris W. George, Anne Elisabeth Haxthausen, Steven Hughes, Robert Milne, Søren Prehn, and Jan Storbank Pedersen. *The RAISE Method*. The BCS Practitioner Series. Prentice-Hall, Hemel Hampstead, England, 1995.

[38] Wolfgang Grieskamp, Thomas Santen, and Bill Stoddart, editors. *IFM 2000: Integrated Formal Methods*, volume of *Lecture Notes in Computer Science*, Schloss Dagstuhl, Germany, November 1-3 2000. Springer. Proceedings of 2nd Intl. Conf. on IFM.

[39] Michael R. Hansen. *Logics of Specification Languages*, chapter Duration Calculus, pages 299–347 in [21]. Springer, 2008.

[40] David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, 1987.

[41] David Harel. On visual formalisms. *Communications of the ACM*, 33(5), 514–530 1988.

[42] David Harel and Eran Gery. Executable object modeling with Statecharts. *IEEE Computer*, 30(7):31–42, 1997.

[43] David Harel, Hagi Lachover, Amnon Naamad, Amir Pnueli, Michal Politi, Rivi Sherman, Aharon Shtull-Trauring, and Mark B. Trakhtenbrot. STATEMATE: A working environment for the development of complex reactive systems. *Software Engineering*, 16(4):403–414, 1990.

[44] David Harel and Rami Marelly. *Come, Let's Play – Scenario-Based Programming Using LSCs and the Play-Engine*. Springer-Verlag, 2003.

[45] David Harel and Amnon Naamad. The STATEMATE semantics of Statecharts. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 5(4):293–333, 1996.

[46] Martin C. Henson, Moshe Deutsch, and Steve Reeves. *Logics of Specification Languages*, chapter Z Logic and Its Applications, pages 489–596 in [21]. Springer, 2008.

[47] Martin C. Henson, Steve Reeves, and Jonathan P. Bowen. Z Logic and its Consequences. *Computing and Informatics*, 22(1–2), 2003. This paper is one of a series: [73, 27, 30, 64, 36, 61] appearing in a double issue of the same journal: *Logics of Specification Languages* — edited by Dines Bjørner.

[48] Tony Hoare. *Communicating Sequential Processes*. C.A.R. Hoare Series in Computer Science. Prentice-Hall International, 1985.

[49] Tony Hoare. Communicating Sequential Processes. Published electronically: `http://www.-usingcsp.com/cspbook.pdf`, 2004. Second edition of [48]. See also `http://www.usingcsp.-com/`.

[50] ITU-T. CCITT Recommendation Z.120: Message Sequence Chart (MSC), 1992.

[51] ITU-T. ITU-T Recommendation Z.120: Message Sequence Chart (MSC), 1996.

[52] ITU-T. ITU-T Recommendation Z.120: Message Sequence Chart (MSC), 1999.

[53] Daniel Jackson. *Software Abstractions Logic, Language, and Analysis*. The MIT Press, Cambridge, Mass., USA, April 2006. ISBN 0-262-10114-9.

[54] Michael A. Jackson. *Software Requirements & Specifications: a lexicon of practice, principles and prejudices*. ACM Press. Addison-Wesley Publishing Company, Wokingham, nr. Reading, England; E-mail: ipc@awpub.add-wes.co.uk, 1995. ISBN 0-201-87712-0; xiv + 228 pages.

[55] Kurt Jensen. *Coloured Petri Nets*, volume 1: Basic Concepts (234 pages + xii), Vol. 2: Analysis Methods (174 pages + x), Vol. 3: Practical Use (265 pages + xi) of *EATCS Monographs in Theoretical Computer Science*. Springer–Verlag, Heidelberg, 1985, revised and corrected second version: 1997.

[56] Jochen Klose and Hartmut Wittke. An automata based interpretation of Live Sequence Charts. In T. Margaria and W. Yi, editors, *TACAS 2001*, LNCS 2031, pages 512–527. Springer-Verlag, 2001.

[57] Leslie Lamport. The Temporal Logic of Actions. *Transactions on Programming Languages and Systems*, 16(3):872–923, 1995.

[58] Leslie Lamport. *Specifying Systems*. Addison–Wesley, Boston, Mass., USA, 2002.

[59] Zohar Manna and Amir Pnueli. *The Temporal Logic of Reactive Systems: Specifications*. Addison Wesley, 1991.

[60] Zohar Manna and Amir Pnueli. *The Temporal Logic of Reactive Systems: Safety*. Addison Wesley, 1995.

[61] Stephan Merz. On the Logic of TLA+. *Computing and Informatics*, 22(1–2), 2003. This paper is one of a series: [73, 27, 30, 64, 36, 47] appearing in a double issue of the same journal: *Logics of Specification Languages* — edited by Dines Bjørner.

[62] Stephan Merz. *Logics of Specification Languages*, chapter The Specification Language TLA$^+$, pages 401–451 in [21]. Springer, 2008.

[63] T. Mossakowski, A. Haxthausen, D. Sannella, and A. Tarlecki. *Logics of Specification Languages*, chapter CASL – the Common Algebraic Specification Language, pages 241–298 in [21]. Springer, 2008.

[64] Till Mossakowski, Anne E. Haxthausen, Don Sanella, and Andzrej Tarlecki. CASL — The Common Algebraic Specification Language: Semantics and Proof Theory. *Computing and Informatics*, 22(1–2), 2003. This paper is one of a series: [73, 27, 30, 36, 61, 47] appearing in a double issue of the same journal: *Logics of Specification Languages* — edited by Dines Bjørner.

[65] Ben C. Moszkowski. *Executing Temporal Logic Programs*. Cambridge University Press, Cambridge, England, 1986.

[66] Carl Adam Petri. *Kommunikation mit Automaten*. Bonn: Institut für Instrumentelle Mathematik, Schriften des IIM Nr. 2, 1962.

[67] Amir Pnueli. The Temporal Logic of Programs. In *Proceedings of the 18th IEEE Symposium on Foundations of Computer Science*, IEEE CS FoCS, pages 46–57. Providence, Rhode Island, IEEE CS, 1977. .

[68] Martin Pěnička and Dines Bjørner. From Railway Resource Planning to Train Operation — a Brief Survey of Complementary Formalisations. In *Building the Information Society, IFIP 18th World Computer Congress, Topical Sessions, 22–27 August, 2004, Toulouse, France — Ed. Renéne Jacquart*, pages 629–636. Kluwer Academic Publishers, August 2004.

[69] Martin Pěnička, Albena Kirilova Strupchanska, and Dines Bjørner. Train Maintenance Routing. In *FORMS'2003: Symposium on Formal Methods for Railway Operation and Control Systems*. L'Harmattan Hongrie, 15–16 May 2003. Conf. held at Techn.Univ. of Budapest, Hungary. Editors: G. Tarnai and E. Schnieder, Germany.

[70] Wolfang Reisig. *A Primer in Petri Net Design*. Springer Verlag, March 1992. 120 pages.

[71] Wolfgang Reisig. *Petri Nets: An Introduction*, volume 4 of *EATCS Monographs in Theoretical Computer Science*. Springer Verlag, May 1985.

[72] Wolfgang Reisig. *Elements of Distributed Algorithms: Modelling and Analysis with Petri Nets*. Springer Verlag, December 1998. xi + 302 pages.

[73] Wolfgang Reisig. The Expressive Power of Abstract State Machines. *Computing and Informatics*, 22(1–2), 2003. This paper is one of a series: [27, 30, 64, 36, 61, 47] appearing in a double issue of the same journal: *Logics of Specification Languages* — edited by Dines Bjørner.

[74] Wolfgang Reisig. *Logics of Specification Languages*, chapter Abstract State Machines for the Classroom, pages 15–46 in [21]. Springer, 2008.

[75] Judi M.T. Romijn, Graeme P. Smith, and Jaco C. van de Pol, editors. *IFM 2005: Integrated Formal Methods*, volume 3771 of *Lecture Notes in Computer Science*, Eindhoven, The Netherlands, December 2005. Springer. Proceedings of 5th Intl. Conf. on IFM. ISBN 3-540-30492-4.

[76] A. W. Roscoe. *Theory and Practice of Concurrency*. C.A.R. Hoare Series in Computer Science. Prentice-Hall, 1997. Now available on the net: http://www.comlab.ox.ac.uk/people/bill.roscoe/publications/68b.pdf.

[77] Steve Schneider. *Concurrent and Real-time Systems — The CSP Approach*. Worldwide Series in Computer Science. John Wiley & Sons, Ltd., Baffins Lane, Chichester, West Sussex PO19 1UD, England, January 2000.

[78] J. M. Spivey. *Understanding Z: A Specification Language and its Formal Semantics*, volume 3 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, January 1988.

[79] J. M. Spivey. *The Z Notation: A Reference Manual*. Prentice Hall International Series in Computer Science, 2nd edition, 1992.

[80] Albena Kirilova Strupchanska, Martin Pěnička, and Dines Bjørner. Railway Staff Rostering. In *FORMS2003: Symposium on Formal Methods for Railway Operation and Control Systems*. L'Harmattan Hongrie, 15–16 May 2003. Conf. held at Techn.Univ. of Budapest, Hungary. Editors: G. Tarnai and E. Schnieder, Germany.

[81] J. C. P. Woodcock and J. Davies. *Using Z: Specification, Proof and Refinement*. Prentice Hall International Series in Computer Science, 1996.

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

[83] Chao Chen Zhou, Charles Anthony Richar Hoare, and Anders P. Ravn. A Calculus of Durations. *Information Proc. Letters*, 40(5), 1992.