1

From Domain to Requirements*

Dines Bjørner, Professor Emeritus

[1] Faculté des Sciences, Bureau 266, LORIA & Université Henri Poincaré Nancy 1, BP 239, F-54506 Vandœuvre lès Nancy, France.**

[2] Department of Informatics and Mathematical Modelling, The Technical University of Denmark, DK-2800 Kgs. Lyngby, Denmark.

[3] Fredsvej 11, DK-2840 Holte, Danmark.
E–Mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

Abstract This is a discursive paper. That is, it shows some formulas (but only as examples so that the reader may be convinced that there is, perhaps, some substance to our claims), no theorems, no proofs. Instead it postulates. The postulates are, however, firmly rooted, we think, in Vol.3 ('Domains, Requirements and Software Design') of the three volume book 'Software Engineering' (Springer March 2006) [6, 7, 8].

First we present a summary of essentials of domain engineering, its motivation, and its modelling of abstractions of domains through the modelling of the intrinsics, support technologies, management and organisation, rules and regulations, scripts, and human behaviour of whichever domain is being described.

Then we present the essence of two (of three) aspects of requirements: the domain requirements and the interface requirements prescriptions as they relate to domain descriptions and we survey the basic operations that "turn" a domain description into a domain requirements prescription: projection, instantiation, determination, extension and fitting. An essence of interface requirements is also presented: the "merging" of shared entities, operations, events and behaviours of the domain with those of the machine (i.e., the hardware and software to be designed).

An objective of the paper is to summarise my work in recent years. Another objective is make a plea for what I consider a more proper approach to software development.

1.1 Introduction

This paper is not a computer science paper — where by computer science, sometimes strangely referred to even as theoretical computer science, we mean the study and knowledge of the things that may exist inside computers.

The paper is more of a computing science paper — where by computing science we mean the study and knowledge of how to construct the things that can exist inside computers.

The borderline between these two disciplines is sharp, but most interesting papers which purports to be computing science papers also, oftentimes strongly, contains text of computer science nature.

Some computer science papers present new models of computation and/or analyses and theorems about such models.

This paper presents a model of early stages of software development that is not conventional. The model is presented in two alternating ways: (i) we present some of the principles and techniques

* Invited paper for the Festschrift of Ugo Montanari on the occasion of his 65th anniversary May 2008. Edited by Rocco di Nicola et al. To be published by Springer.

** This paper was written with the financial support of Université Henri Poincaré and INRIA during the author's two month visit.

of that unconventional software development method, and (ii) we present — what in the end, that is, taken across the paper, amounts to a relatively large example.

One aspect of the non-conventionality of the present paper is its total lack of 'references to related work' by others. Instead we shall solely refer, now, to our own work related to the topic of the current paper. In those referenced works you should find 'references to related work'.

The topic pair of domain and of requirements engineering — on which the present paper relies — is treated in depth in [8, Chaps. 8–24, Pages 193–524 (!)]. Recent papers elaborate on related (possible) research topics [9], or on software management [10, to appear], or gives more extensive summaries on domain engineering, one without a leading, extensive example but with a more proper discussion of domain modelling issues and 'related work' [11, to appear], and one with a considerably larger example [12, to appear (the example appendix, pages 32–97, illustrates a Container Line Industry domain)].

In summary: the objective of the present paper is to relate domain engineering to requirements engineering and to show that one can obtain an altogether different basis for requirements engineering.

1.2 The Triptych Principle of Software Engineering

We start, unconventionally, by enunciating a principle. The principle expresses how we see software development as centrally consisting of three "programming-like" phases based on the following observation: before software can be designed we must understand its requirements, and before requirements can be prescribed we must understand the application domain. We therefore see software development proceeding, ideally, in three phases: a first phase of domain engineering, a second phase of requirements engineering, and a third phase of software design.

The first paragraphs of Sects. 1.3 and 1.4 explain what the objectives of domain engineering and requirements engineering are. The sections otherwise outline major development stages and steps of these two phases.

1.3 Domain Engineering

The objective of domain engineering is to create a domain description. A domain description specifies entities, functions, events and behaviours of the domain such as the domain stakeholders think they are. A domain description thus (indicatively [46]) expresses what there is. A domain description expresses no requirements let alone anything about the possibly desired (required) software.

1.3.1 Stages of Domain Engineering

To develop a proper domain description necessitates a number of development stages: (i) identification of stakeholders, (ii) domain knowledge acquisition, (iii) business process rough-sketching, (iv) domain analysis, (v) domain modelling: developing abstractions and verifying properties, (vi) domain validation and (vii) domain theory building.

Business process (BP) rough-sketching amount to rough, narrative outlines of the set of business processes as experienced by each of the stakeholder groups. BP engineering is in contrast to BR re-engineering (BPR) which we shall cover later, but briefly in Sect. 1.4.2.

We shall only cover domain modelling.

1.3.2 First Example of a 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.*

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).

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.

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″}⊆hs ∧ obs_HIs(l)={obs_HI(h′),obs_HI(h″)}
value
   xtr_HIs: N → HI-set,xtr_LIs: N → LI-set
```

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.)

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 $\overset{\sim}{\to}$ 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 $\overset{\sim}{\to}$ 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′.

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.[4]

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 $\omega : \Omega$.

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
```

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).

Formalisation — Behaviours

```
channel
   x:...
value
   transportation_transition: (N × Ω) → in x  (N × Ω)
```

```
transportation_transition(n,$\omega$) $\equiv$
    ...
    $\lceil\rceil$ let xv = x? in case xv of ... end end
    $\lceil\rceil$ let op:NetOp • pre IntNetOp(op)(n) in IntNetOp(op)(n) end
    ...

transportation: (N $\times$ $\Omega$) $\to$ in x  Unit
transportation(n,$\omega$) $\equiv$
    let (n$'$,$\omega'$) = transportation_transition(n,$\omega$) in
    transportation (n$'$,$\omega'$) end
```

1.3.3 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.

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. 1.3.2, 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. 1.3.2 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)
```

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]
    H$\Sigma$ = X-set [hub states]
value
    obs_H$\Sigma$: H → H$\Sigma$, 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_H$\Sigma$(h)⊆xtr_Xs(h) ∧
        ∀ (li1,hi2,li3,c),(li4,hi5,li6,c$'$):X • {(li1,hi2,li3,c),(li4,hi5,li6,c$'$)}⊆obs_H$\Sigma$(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 $\langle(h\sigma', t\delta'), (h\sigma'', t\delta''), \ldots, (h\sigma'^{\cdots'}, t\delta'^{\cdots'})\rangle$ and so that each of these intermediate states can be set, according to the time interval information,[5] before the final hub state (hσ_f) is set.

A Transportation Support Technology Facet — Formalisation, 3.

```
type
    TI [time interval]
    Signalling = (H$\Sigma$ × TI)$^*$
    Sema = (H$\Sigma$ × H$\Sigma$) $\overrightarrow{m}$ Signalling
value
    obs_Sema: H → Sema, chg_H$\Sigma$: H × H$\Sigma$ → H, chg_H$\Sigma$_Seq: H × H$\Sigma$ → H
    chg_H$\Sigma$(h,h$\sigma$) as h$'$ pre h$\sigma$ ∈ obs_H$\Omega$(h) post obs_H$\Sigma$(h)=h$\sigma$
    chg_H$\Sigma$_Seq(h,h$\sigma$) ≡ let h$'$ = sig_seq(h)(obs_Sema(h,h$\sigma$)) in chg_H$\Sigma$(h$'$,obs_$\Sigma$(h)) end

    sig_seq: H → Signalling → H
    sig_seq(h)(sigseq) ≡
        if sigseq=⟨⟩ then h else
        let (h$\sigma$,t$\delta$) = hd sigseq in
        let h$'$ = chg_H$\Sigma$(h,h$\sigma$); wait t$\delta$;
        sig_seq(h$'$)(tl sigseq) end end end
```

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.*

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 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.

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[6], 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.

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]
```

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) ~→ (V →ₘ VPos×VPos)
    wrong_direction: Traffic → T ~→ (V →ₘ VPos)
```

1.3.4 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, . . . (!)).

1.4 Requirements Engineering

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* [46]) 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.

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[7] and payment of tickets upon the vehicle leaving the toll road net both issuance and collection/payment of tickets occurring at toll booths[8] 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.

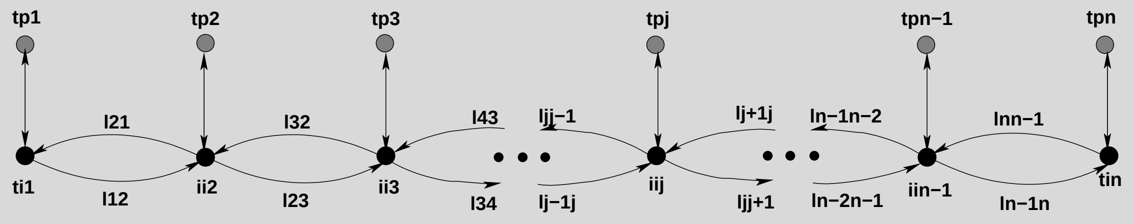

Fig. 1.1. A simple, linear toll road net:
tp_i: *t*oll *p*laza i,
ti_1, ti_n: *t*erminal *i*ntersection k,
ii_k: *i*ntermediate *i*ntersection k, $1<k<n$
l_{xy}: tollway *l*ink from i_x to i_y, y=x+1 or y=x-1 and $1\leq x<n$.

1.4.1 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.

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

1.4.2 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.

Re-engineering Domain Entities

The net is arranged as a linear sequence of two or more (what we shall call) intersection hubs. Each intersection hub has a single two-way link to (what we shall call) an entry/exit hub (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 (what we shall call) 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.

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 to sensor pressure from "passing" vehicles or by vehicle drivers pressing ticket issuing machine button by issuing ticket. Tollgate payment machines accept credit cards, bank notes or coins in designated currencies as payment and returns any change.

We then briefly introduce and sketch an operation performed when vehicles cross intersections: The vehicle is assume 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.

The updated domain description section on support technology will detail the exact workings of these tollgate and internal intersectrion machines and the domain description section on human behaviour will likewise explore the man/machine facet.

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.

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.

1.4.3 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.

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.[9]

Domain Projection — Formalisation.

We do not show the resulting formalisation.

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. 1.1 and the preliminary description given in Sect. 1.4.2. 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 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))
```

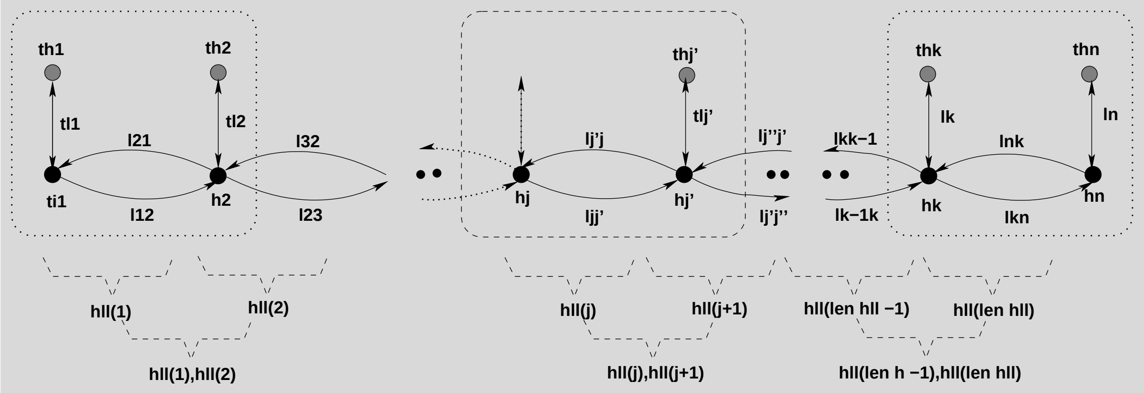

Fig. 1.2. A simple, linear toll road net:
th_i: *t*oll plaza *i*,
h_1, h_n: *t*erminal *i*ntersections,
h_2, h_j, h'_j, h_k: *i*ntermediate *i*ntersections, $1<j\leq k$, $k=n$-1
l_{xy}, l_{yx}: tollway *l*ink from h_x to h_y and from h_y to h_x, $1\leq x<n$.
l_{x-1x}, l_{xx-1}: tollway *l*ink from h_{x-1} 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


vqlue
  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
```

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': The link state spaces There is only one link state: the set of all paths through the link, thus any link state space is the singleton set its only link state. 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 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.

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
   Mach, Ticket, Cash, Payment, Map_TN
value
   obs_Cash: Mach → Cash, obs_Tickets: M → Ticket-set
   obs_Entry, obs_Exit: Ticket → HI, obs_Ticket: V → (Ticket|nil)

   calculate_Payment: (HI×HI) → Map_TN → Payment

   press_Entry: M → M × Ticket            [gate up]
   press_Exit: M × Ticket → M × Payment
   payment: M × Payment → M × 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 or live sequence charts and temporal logics (DC, TLA+).

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 of awareness of the net by both $d_{r_{\text{toll}}}$ and $d_{r_{\text{maint.}}}$ of nets (N), hubs (H) and links (L). 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.

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.

1.4.4 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 requires a sequence of several "on-line" 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.*

1.4.5 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. *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.

Shared Entities

The main shared entites 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 attibutes — most of which have not been mentioned so far: length, cadestral information, namings, wear and tear (whereever applicable), last/next scheduled maintenance (whereever applicable), state and state space, and many others.

We "split" our interface requirements development into two separate steps: the development of $d_{r_{\mathrm{net}}}$ (the common domain requirements for the shared hubs and links), and the co-development of $d_{r_{\mathrm{db:i/f}}}$ (the common domain requirements for the interface between $d_{r_{\mathrm{net}}}$ 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 ($d_{r_{\mathrm{toll}}}$, $d_{r_{\mathrm{maint.}}}$) 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 $d_{r_{\mathrm{net}}}$ 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 $d_{r_{\mathrm{net}}}$ 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.

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 $d_{r_{\mathrm{toll}}}$ we must therefore prescribe the varieties of successfull 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.)

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 $d_{r_{\mathrm{toll}}}$ 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.

Shared Behaviours

The main shared behaviours are therefore related to the journey of a vehicle throuh 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.

1.5 Discussion

1.5.1 An ‘Odysey’

Our ‘Odysey’ has ended. A long example has been given.

We have shown that requirements engineering can have an abstraction basis in domain engineering; and we have shown that we do not have to start software development with requirements engineering, but that we can start software development with domain engineering and then proceed to a more orderly requirements engineering phase than witnessed today.

1.5.2 Claims of Contribution

What is essentially new here is the claim and its partial validation that one can and probably should put far more emphasis on domain modelling, the domain modelling concepts, principles and techniques of business process domain intrinsics, domain support technologies, domain management and organisation, domain rules and regulations, domain scripts and domain human behaviour; the identification of, and the decomposition of the requirements development process into, domain requirements, interface requirements and machine requirements; the domain requirements “derivation” concepts, principles and techniques of projection, instantiation, determination, extension and fitting and the identification of structuring of the interfce qround requirements shared entities, shared operations, shared events and shared behaviours.

1.5.3 Comparison to Other Work

Jackson’s Problem Frame approach [47] cleverly alternates between domain analysis, requirements development and software design. For more satisfactory comparisons between our domain engineering approach and past practices and writings on domain analysis we refer to [11].

1.5.4 A Critique

A major presentation of domain and of requirements engineering is given in [8, Chaps. 8–16 and 17–24]. [11] provides a summary, more complete presentation of domain engineering than the present paper allows, while [9] discusses a set of research issues for domain engineering. Papers, like [11, 9], but for requirements engineering, with more a complete presentation, respectively a discussion of research issues for this new kind of reauirements engineering might be desirable. The current paper's Sect. 1.4 provided a slightly revised structuring of the interface requirements engineering.

Some of the development steps within the domain modelling and likewise within the requirements modelling are refinments, and some are extensions. If we ensure that the extensions are what is known as conservatiove extensions then all theorems of the source of the extension go through and are also valid in the extension. Although such things are here rather clear much more should be said here about ensuring conservatiove extensions. We do not since the current paper is is not aimed at the finer issues of the development but at the domain to requirements "derivation" issues.

1.6 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 organsisers of Ugo Montanari's Festschrift, Pierpaolo Degano, Jose Meseguer and Rocco De Nicola, for inviting me — thus forcing me to willingly write this paper.

1.7 Bibliographical Notes

Specification languages, techniques and tools, that cover the spectrum of domain and requirements specification, refinement and verification, are dealt with in Alloy: [45], ASM: [17, 64, 65], B/event B: [1, 20], CSP [40, 67, 68, 41], DC [72, 73, 31] (Duration Calculus), Live Sequence Charts [21, 36, 49], Message Sequence Charts [42, 43, 44], RAISE [27, 29, 6, 7, 8, 26] (RSL), Petri nets [48, 59, 62, 61, 63], Statecharts [32, 33, 35, 37, 34], Temporal Logic of Reactive Systems [52, 53, 58, 60], TLA+ [50, 51, 54, 55] (Temporal Logic of Actions), VDM [14, 15, 25, 24], and Z [69, 70, 71, 39, 38]. Techniques for integrating "different" formal techniques are covered in [2, 30, 18, 16, 66]. The recent book on Logics of Specification Languages [13] covers ASM, B/event B, CafeObj, CASL, DC, RAISE, TLA+, VDM and Z.

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, [14]*, 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, [14]*, 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 Engineering, Vol. 1: Abstraction and Modelling*. Texts in Theoretical Computer Science, the EATCS Series. Springer, 2006.
7. 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.
8. Dines Bjørner. *Software Engineering, Vol. 3: Domains, Requirements and Software Design*. Texts in Theoretical Computer Science, the EATCS Series. Springer, 2006.
9. 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.
10. 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).
11. 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.
12. 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.
13. Dines Bjørner and Martin C. Henson, editors. *Logics of Specification Languages — see [65, 20, 23, 56, 31, 26, 55, 24, 38]*. EATCS Monograph in Theoretical Computer Science. Springer, Heidelberg, Germany, 2008.
14. 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].
15. Dines Bjørner and Cliff B. Jones, editors. *Formal Specification and Software Development*. Prentice-Hall, 1982.
16. 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.
17. E. Börger and R. Stärk. *Abstract State Machines. A Method for High-Level System Design and Analysis*. Springe, 2003. ISBN 3-540-00702-4.
18. 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.
19. 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: [64, 22, 57, 28, 54, 39] appearing in a double issue of the same journal: *Logics of Specification Languages* — edited by Dines Bjørner.
20. Dominique Cansell and Dominique Méry. *Logics of Specification Languages*, chapter The event-B Modelling Method: Concepts and Case Studies, pages in [13], 47–152. Springer, 2008.
21. 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.
22. 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: [64, 19, 57, 28, 54, 39] appearing in a double issue of the same journal: *Logics of Specification Languages* — edited by Dines Bjørner.
23. Răzvan Diaconescu. *Logics of Specification Languages*, chapter A Methodological Guide to the CafeOBJ Logic, pages in [13], 153–240. Springer, 2008.
24. John S. Fitzgerald. *Logics of Specification Languages*, chapter The Typed Logic of Partial Functions and the Vienna Development Method, pages in [13], 453–487. Springer, 2008.
25. John S. Fitzgerald and Peter Gorm Larsen. *Developing Software using VDM-SL*. Cambridge University Press, The Edinburgh Building, Cambridge CB2 1RU, England, 1997.
26. Chris George and Anne E. Haxthausen. *Logics of Specification Languages*, chapter The Logic of the RAISE Specification Language, pages in [13], 349–399. Springer, 2008.

27. 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.
28. 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: [64, 19, 22, 57, 54, 39] appearing in a double issue of the same journal: *Logics of Specification Languages* — edited by Dines Bjørner.
29. 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.
30. 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.
31. Michael R. Hansen. *Logics of Specification Languages*, chapter Duration Calculus, pages in [13], 299–347. Springer, 2008.
32. David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, 1987.
33. David Harel. On visual formalisms. *Communications of the ACM*, 33(5), 514–530 1988.
34. David Harel and Eran Gery. Executable object modeling with Statecharts. *IEEE Computer*, 30(7):31–42, 1997.
35. 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.
36. David Harel and Rami Marelly. *Come, Let's Play – Scenario-Based Programming Using LSCs and the Play-Engine*. Springer-Verlag, 2003.
37. David Harel and Amnon Naamad. The STATEMATE semantics of Statecharts. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 5(4):293–333, 1996.
38. Martin C. Henson, Moshe Deutsch, and Steve Reeves. *Logics of Specification Languages*, chapter Z Logic and Its Applications, pages in [13], 489–596. Springer, 2008.
39. 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: [64, 19, 22, 57, 28, 54] appearing in a double issue of the same journal: *Logics of Specification Languages* — edited by Dines Bjørner.
40. Tony Hoare. *Communicating Sequential Processes*. C.A.R. Hoare Series in Computer Science. Prentice-Hall International, 1985.
41. Tony Hoare. Communicating Sequential Processes. Published electronically: `http://www.usingcsp.-com/cspbook.pdf`, 2004. Second edition of [40]. See also `http://www.usingcsp.com/`.
42. ITU-T. CCITT Recommendation Z.120: Message Sequence Chart (MSC), 1992.
43. ITU-T. ITU-T Recommendation Z.120: Message Sequence Chart (MSC), 1996.
44. ITU-T. ITU-T Recommendation Z.120: Message Sequence Chart (MSC), 1999.
45. Daniel Jackson. *Software Abstractions Logic, Language, and Analysis*. The MIT Press, Cambridge, Mass., USA, April 2006. ISBN 0-262-10114-9.
46. 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.
47. Michael A. Jackson. *Problem Frames — Analyzing and Structuring Software Development Problems*. ACM Press, Pearson Education. Addison–Wesley, Edinburgh Gate, Harlow CM20 2JE, England, 2001.
48. 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.
49. 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.
50. Leslie Lamport. The Temporal Logic of Actions. *Transactions on Programming Languages and Systems*, 16(3):872–923, 1995.
51. Leslie Lamport. *Specifying Systems*. Addison–Wesley, Boston, Mass., USA, 2002.
52. Zohar Manna and Amir Pnueli. *The Temporal Logic of Reactive Systems: Specifications*. Addison Wesley, 1991.
53. Zohar Manna and Amir Pnueli. *The Temporal Logic of Reactive Systems: Safety*. Addison Wesley, 1995.
54. Stephan Merz. On the Logic of TLA+. *Computing and Informatics*, 22(1–2), 2003. This paper is one of a series: [64, 19, 22, 57, 28, 39] appearing in a double issue of the same journal: *Logics of Specification Languages* — edited by Dines Bjørner.

55. Stephan Merz. *Logics of Specification Languages*, chapter The Specification Language TLA$^+$, pages in [13], 401–451. Springer, 2008.
56. T. Mossakowski, A. Haxthausen, D. Sannella, and A. Tarlecki. *Logics of Specification Languages*, chapter CASL – the Common Algebraic Specification Language, pages in [13], 241–298. Springer, 2008.
57. 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: [64, 19, 22, 28, 54, 39] appearing in a double issue of the same journal: *Logics of Specification Languages* — edited by Dines Bjørner.
58. Ben C. Moszkowski. *Executing Temporal Logic Programs*. Cambridge University Press, Cambridge, England, 1986.
59. Carl Adam Petri. *Kommunikation mit Automaten*. Bonn: Institut für Instrumentelle Mathematik, Schriften des IIM Nr. 2, 1962.
60. 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. .
61. Wolfang Reisig. *A Primer in Petri Net Design*. Springer Verlag, March 1992. 120 pages.
62. Wolfgang Reisig. *Petri Nets: An Introduction*, volume 4 of *EATCS Monographs in Theoretical Computer Science*. Springer Verlag, May 1985.
63. Wolfgang Reisig. *Elements of Distributed Algorithms: Modelling and Analysis with Petri Nets*. Springer Verlag, December 1998. xi + 302 pages.
64. Wolfgang Reisig. The Expressive Power of Abstract State Machines. *Computing and Informatics*, 22(1–2), 2003. This paper is one of a series: [19, 22, 57, 28, 54, 39] appearing in a double issue of the same journal: *Logics of Specification Languages* — edited by Dines Bjørner.
65. Wolfgang Reisig. *Logics of Specification Languages*, chapter Abstract State Machines for the Classroom, pages in [13], 15–46. Springer, 2008.
66. 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.
67. A. W. Roscoe. *Theory and Practice of Concurrency*. C.A.R. Hoare Series in Computer Science. Prentice-Hall, 1997.
68. 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.
69. 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.
70. J. M. Spivey. *The Z Notation: A Reference Manual*. Prentice Hall International Series in Computer Science, 2nd edition, 1992.
71. J. C. P. Woodcock and J. Davies. *Using Z: Specification, Proof and Refinement*. Prentice Hall International Series in Computer Science, 1996.
72. 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.
73. Chao Chen Zhou, Charles Anthony Richar Hoare, and Anders P. Ravn. A Calculus of Durations. *Information Proc. Letters*, 40(5), 1992.

A Laudatio: Ugo 65 Years

$\boxed{\text{To come}}$

Contents