7 June 2016, 08:16 am: Submitted to **Formal Aspects of Computing**

Domain Facets

Analysis & Description — 7 June 2016, 08:16 am

Dines Bjørner

Fredsvej 11, DK-2840 Holte, Denmark.
DTU Compute, Technical University of Denmark, DK-2800 Kgs. Lyngby, Denmark.
E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~dibj

Abstract. This paper is a continuation of [Bjø16c, Bjø16b]. Where [Bjø16c] covered a method for analysing and describing the intrinsics of manifest domains, the present paper covers principles and techniques for describing domain facets — not covered in [Bjø16c]. Where [Bjø16b] covered some basic principles and techniques for structuring requirements analysis and prescription, the present paper hints at requirements that can be derived from domain facets. By a domain facet we shall understand 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. We shall outline the following domain facets: *intrinsics, support technologies, rules & regulations, scripts, license languages, management & organisation,* and *human behaviour*. The present paper is a substantial reformulation and extension of [Bjø10] on the background of [Bjø16c, Bjø16b].

1. Introduction

In [Bjø16c] we outline a method for analysing &[1] describing domains. By a **method** we shall understand a set of principles, techniques and tools for analysing and constructing (synthesizing) an artifact, as here a description ⊙[2] By a **domain** we shall understand a potentially infinite set of endurants and a usually finite set of perdurants (actions, events and behaviours) [the latter map endurants into endurants] such that these entities are observable in the world and can be described ⊙ In this paper we cover domain analysis & description principles and techniques not covered in [Bjø16c]. That paper focused on manifest domains. Here we, on one side, go "outside" the realm of manifest domains, and, on the other side, cover, what we shall refer to as, facets, not covered in [Bjø16c].

1.1. Facets of Domains

By a **domain facet** we shall understand *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* ⊙ Now, the definition of what a domain facet is can seem vague. It cannot

Correspondence and offprint requests to: Dines Bjørner, Fredsvej 11, DK-2840 Holte, Denmark. E-mail: bjorner@gmail.com
[1] We use the ampersand (logogram), &, in the following sense: Let A and B be two concepts. By *A and B* we mean to refer to these two concepts. With $A\&B$ we mean to refer to a composite concept "containing" elements of both A and B.
[2] The ⊙ symbol delimits a definition.

be otherwise. The definition is sharpened by the definitions of the specific facets. You can say, that the definition of domain facet is the "sum" of the definitions of these specific facets. The specific facets – so far[3] – are: intrinsics (Sect. 2), support technology (Sect. 3), rules & regulations (Sect. 4), scripts (Sect. 5), license languages (Sect. 6), management & organisation (Sect. 7) and human behaviour (Sect. 8). Of these, the rules & regulations, scripts and license languages are closely related. Vagueness may "pop up", here and there, in the delineation of facets. It is necessarily so. We are not in a domain of computer science, let alone mathematics, where we can just define ourselves precisely out of any vagueness problems. We are in the domain of (usually) really world facts. And these are often hard to encircle.

1.2. Relation to Previous Work

The present paper is a rather complete rewrite of [Bjø10]. The reason for the rewriting is the expected[4] publication of [Bjø16c]. The [Bjø10] was finalised already in 2006, 10 years ago, before the analysis & description calculus of [Bjø16c] had emerged. It was time to revise [Bjø10] rather substantially.

1.3. Structure of Paper

The structure of the paper follows the seven specific facets, as listed above. Each section, 2.–8., starts by a definition of the specific facet, Then follows an analysis of the abstract concepts involved usually with one or more examples – with these examples making up most of the section. We then "speculate" on derivable requirements thus relating the present paper to [Bjø16b]. We close each of the sections, 2.–8., with some comments on how to model the specific facet of that section.

• • •

Examples 1–22 of sections 2.–8. present quite a variety. In that, they reflect the wide spectrum of facets.

• • •

More generally, domains can be characterised by intrinsically being endurant, or function, or event, or behaviour intensive. Software support for activities in such domains then typically amount to database systems, computation-bound systems, real-time embedded systems, respectively distributed process monitoring and control systems. Other than this brief discourse we shall not cover the "intensity"-aspect of domains in this paper.

2. Intrinsics

- *By domain* **intrinsics** *we shall understand 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* ⊙

2.1. Conceptual Analysis

The principles and techniques of domain analysis & description, as unfolded in [Bjø16c], focused on and resulted in descriptions of the intrinsics of domains. They did so in focusing the analysis (and hence the description) on the basic endurants and their related perdurants, that is, on those parts that most readily present themselves for observation, analysis & description.

Example: 1. Railway Net Intrinsics: We narrate and formalise three railway net intrinsics.

From the view of *potential train passengers* a railway net consists of lines, l:L, with names, ln:Ln, stations, s:S, with names sn:Sn, and trains, tn:TN, with names tnm:Tnm. A line connects exactly two distinct stations.

[3] We write: 'so far' in order to "announce", or hint that there may be other specific facets. The one listed are the ones we have been able to "isolate", to identify, in the most recent 10-12 years.

[4] Edit: to be edited, either, hopefully, into 'recent', or sentence removed.

```
scheme N0 =
  class
    type
      N, L, S, Sn, Ln, TN, Tnm
    value
      obs_Ls: N → L-set, obs_Ss: N → S-set
      obs_Ln: L → Ln, obs_Sn: S → Sn
      obs_Sns: L → Sn-set, obs_Lns: S → Ln-set
    axiom
      ...
  end
```

N, L, S, Sn and Ln designate nets, lines, stations, station names and line names. One can observe lines and stations from nets, line and station names from lines and stations, pair sets of station names from lines, and lines names (of lines) into and out from a station from stations. Axioms ensure proper graph properties of these concepts.

From the view of *actual train passengers* a railway net — in addition to the above — allows for several lines between any pair of stations and, within stations, provides for one or more platform tracks, tr:Tr, with names, trn:Trn, from which to embark on or alight from a train.

```
scheme N1 = extend N0 with
  class
    type
      Tr, Trn
    value
      obs_Trs: S → Tr-set, obs_Trn: Tr → Trn
    axiom
      ...
  end
```

The only additions are that of track and track name types, related observer functions and axioms.

From the view of *train operating staff* a railway net — in addition to the above — has lines and stations consisting of suitably connected rail units. A rail unit is either a simple (i.e., linear, straight) unit, or is a switch unit, or is a simple crossover unit, or is a switchable crossover unit, etc. Simple units have two connectors. Switch units have three connectors. Simple and switchable crossover units have four connectors. A path, p:P, (through a unit) is a pair of connectors of that unit. A state, $\sigma : \Sigma$, of a unit is the set of paths, in the direction of which a train may travel. A (current) state may be empty: The unit is closed for traffic. A unit can be in any one of a number of states of its state space, $\omega : \Omega$.

```
scheme N2 = extend N1 with
  class
    type
      U, C
      P′ = U × (C×C)
      P = {| p:P′ • let (u,(c,c′))=p in (c,c′)∈ ∪ obs_Ω(u) end |}
      Σ = P-set
      Ω = Σ-set
    value
      obs_Us: (N|L|S) → U-set
      obs_Cs: U → C-set
      obs_Σ: U → Σ
      obs_Ω: U → Ω
    axiom
      ...
  end
```

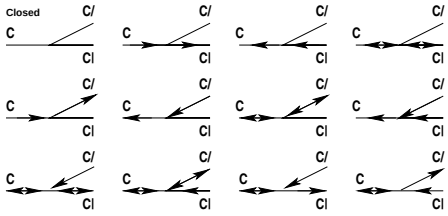

Fig. 1. Possible states of a rail switch

Unit and connector types have been added as have concrete types for paths, unit states, unit state spaces and related observer functions, including unit state and unit state space observers. •

Different stakeholder perspectives, not only of intrinsics, as here, but of any facet, lead to a number of different models. The name of a phenomenon of one perspective, that is, of one model, may coincide with the name of a "similar" phenomenon of another perspective, that is, of another model, and so on. If the intention is that the "same" names cover comparable phenomena, then the developer must state the comparison relation.

Example: 2. Intrinsics of Switches: The intrinsic attribute of a rail switch is that it can take on a number of states. A simple switch $({}^{c_|}Y_{c}^{c_/})$ has three connectors: $\{c, c_|, c_/\}$. c is the connector of the common rail from which one can either "go straight" $c_|$, or "fork" $c_/$ (Fig. 1). So we have that a possible state space of such a switch could be ω_{g_s} :

$\{\{\},$
$\{(c, c_|)\}, \{(c_|, c)\}, \{(c, c_|), (c_|, c)\},$
$\{(c, c_/)\}, \{(c_/, c)\}, \{(c, c_/), (c_/, c)\}, \{(c_/, c), (c_|, c)\},$
$\{(c, c_|), (c_|, c), (c_/, c)\}, \{(c, c_/), (c_/, c), (c_|, c)\}, \{(c_/, c), (c, c_|)\}, \{(c, c_/), (c_|, c)\}\}$

The above models a general switch ideally. Any particular switch ω_{p_s} may have $\omega_{p_s} \subset \omega_{g_s}$. Nothing is said about how a state is determined: who sets and resets it, whether determined solely by the physical position of the switch gear, or also by visible or virtual (i.e., invisible, intangible) signals up or down the rail, away from the switch. •

Example: 3. An Intrinsics of Documents: Think of documents, written, by hand, or typed "onto" a computer text processing system. One way of considering such documents is as follows. First we abstract from the syntax that such a document, or set of more-or-less related documents, or just documents, may have: whether they are letters, with sender and receive addressees, dates written, sent and/or received, opening and closing paragraphs, etc., etc.; or they are books, technical, scientific, novels, or otherwise, or they are application forms, tax returns, patient medical records, or otherwise. Then we focus on the operations that one may perform on documents: their creation, editing, reading, copying, authorisation, "transfer"[5], "freezing"[6], and shredding. Finally we consider documents as manifest parts, cf. [Bjø16c]. Parts, so documents have unique identifications, in this case, changeable mereology, and a number of attributes. The mereology of a document, d, reflects those other documents upon which a document is based, i.e., refers to, and/or refers to d. Among the attributes of a document we can think of (i) a trace of what has happened to a document, i.e., a trace of all the operations performed on "that" document, since and including creation — with that trace, for example, consisting of time-stamped triples of the essence of the operations, the "actor" of the operation (i.e., the operator), and possibly some abstraction of the locale of the document when operated upon; (ii) a synopsis of what the document text "is all about", (iii) and some "rendition" of the document text. •

This view of documents, whether "implementable" or "implemented" or not, is at the basis of our view of license languages (for *digital media*, *health-care* (patient medical record), *documents*, and *transport* (contracts) as that facet is covered in Sect. 6.

[5] to other editors, readers, etc.
[6] i.e., prevention of future operations

2.2. Requirements

[Bjø16b] illustrated requirements "derived" from the intrinsics of a road transport system – as outlined in [Bjø16c]. So this paper has little to add to the subject of requirements "derived" from intrinsics.

2.3. On Modeling Intrinsics

[Bjø16c] outlined basic principles, techniques and tools for modeling the intrinsics of manifest domains. Modeling the domain intrinsics can often be expressed in property-oriented specification languages (like CafeOBJ [FNT00, FD98, DFO03]), model-oriented specification languages (like Alloy [Jac06], B [Abr09], VDM-SL [BJ78, BJ82, FL97], RSL [GHH+92], or Z [Spi88, Spi92, WD96, HRB03]), event-based languages (like Petri nets or [Jen97, Pet62, Rei85, Rei92, Rei98] or CSP [Hoa85, Ros97, Sch00, Hoa04]), respectively in process-based specification languages (like MSCs [IT92, IT96, IT99], LSCs [DH01, HM03, KW01], Statecharts [Har87, Har88, HLN+90, HN96, HG97], or CSP [Hoa85, Ros97, Sch00, Hoa04]). An area not well-developed is that of modeling continuous domain phenomena like the dynamics of automobile, train and aircraft movements, flow in pipelines, etc. We refer to [ORW16].

3. Support Technologies

- *By a domain* **support technology** *we shall understand ways and means of implementing certain observed phenomena or certain conceived concepts* ⊙

The "ways and means" may be in the form of "soft technologies": human manpower, see, however, Sect. 8, or in the form of "hard" technologies: electro-mechanics, etc. The term 'implementing' is crucial. It is here used in the sense that, $\psi\tau$, which is an 'implementation' of a endurant or perdurant, ϕ, is an extension of ϕ, with ϕ being an abstraction of $\psi\tau$. We strive for the extensions to be proof theoretic conservative extensions [Mai97].

3.1. Conceptual Analysis

There are [always] basically two approaches the task of analysing & describing the support technology facets of a domain. One either stumbles over it, or one tries to tackle the issue systematically. The "stumbling" approach occurs when one, in the midst of analysing & describing a domain realises that one is tackling something that satisfies the definition of a support technology facet. In the systematic approach to the analysis & description of the support technology facets of a domain one usually starts with a basically intrinsics facet-oriented domain description. We then suggest that the domain engineer "inquires" of ever endurant and perdurant whether it is an intrinsic entity or, perhaps a support technology.

Example: 4. Railway Support Technology: We give a rough sketch description of possible rail unit switch technologies.

(i) In "ye olde" days, rail switches were "thrown" by manual labour, i.e., by railway staff assigned to and positioned at switches.

(ii) With the advent of reasonably reliable mechanics, pulleys and levers[7] and steel wires, switches were made to change state by means of "throwing" levers in a cabin tower located centrally at the station (with the lever then connected through wires etc., to the actual switch).

(iii) This partial mechanical technology then emerged into electro-mechanics, and cabin tower staff was "reduced" to pushing buttons.

(iv) Today, groups of switches, either from a station arrival point to a station track, or from a station track to a station departure point, are set and reset by means also of electronics, by what is known as interlocking (for example, so that two different routes cannot be open in a station if they cross one another). •

[7] https://en.wikipedia.org/wiki/Pulley and http://en.wikipedia.org/wiki/Lever

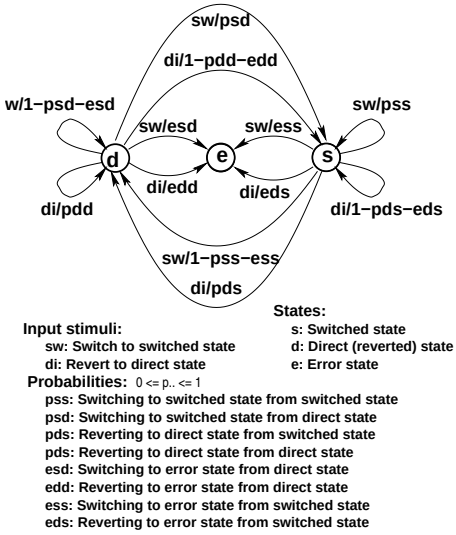

Fig. 2. Probabilistic state switching

It must be stressed that Example 4 is just a rough sketch. In a proper narrative description the software (cum domain) engineer must describe, in detail, the subsystem of electronics, electro-mechanics and the human operator interface (buttons, lights, sounds, etc.). An aspect of supporting technology includes recording the state-behaviour in response to external stimuli. We give an example.

Example: 5. Probabilistic Rail Switch Unit State Transitions: Figure 2 indicates a way of formalising this aspect of a supporting technology. Figure 2 intends to model the probabilistic (erroneous and correct) behaviour of a switch when subjected to settings (to switched (s) state) and re-settings (to direct (d) state). A switch may go to the switched state from the direct state when subjected to a switch setting s with probability psd. •

Example: 6. Traffic Signals: We continue Examples 17, 18, 25 and 33 of [Bjø16c]. This example should, however, be understandable without reference to [Bjø16c]. A traffic signal represents a technology in support of visualising hub states (transport net road intersection signaling states) and in effecting state changes.

1 A traffic signal, *ts:TS*, is considered a part with observable hub states and hub state spaces. Hub states and hub state spaces are programmable, respectively static attributes of traffic signals.
2 A hub state space, $h\omega$, is a set of hub states such that each current hub state is in that hubs' hub state space.
3 A hub state, $h\sigma$, is now modeled as a set of hub triples.
4 Each hub triple has a link identifier l_i ("coming from"), a colour (**red**, yellow or **green**), and another link identifier l_j ("going to").
5 Signaling is now a sequence of one or more pairs of next hub states and time intervals, *ti:TI*, for example: $<(h\sigma_1,ti_1),(h\sigma_2,ti_2),...,(h\sigma_{n-1},ti_{n-1}),(h\sigma_n,ti_n)>$, n>0. The idea of a signaling is to first change the designated hub to state $h\sigma_1$, then wait ti_1 time units, then set the designated hub to state $h\sigma_2$, then wait ti_2 time units, etcetera, ending with final state σ_n and a (supposedly) long time interval ti_n before any decisions are to be made as to another signaling. The set of hub states $\{h\sigma_1,h\sigma_2,...,h\sigma_{n-1}\}$ of $<(h\sigma_1,ti_1),(h\sigma_2,ti_2),...,(h\sigma_{n-1},ti_{n-1}),(h\sigma_n,ti_n)>$, n>0, is called the set of intermediate states. Their purpose is to secure an orderly phase out of green via yellow to red and phase in of red via yellow to green in some order for the various directions. We leave it to the reader to devise proper well-formedness conditions for signaling sequences as they depend on the hub topology.
6 A street signal (a semaphore) is now abstracted as a map from pairs of hub states to signaling sequences. The idea is that given a hub one can observe its semaphore, and given the state, $h\sigma$ (not in the above set), of the hub "to be signaled" and the state $h\sigma_n$ into which that hub is to be signal-led "one looks up" under that pair in the semaphore and obtains the desired signaling.

type
1 TS ≡ H, HΣ, HΩ
value
2 obs_HΣ: H,TS → HΣ
2 obs_HΩ: H,TS → HΩ
type
3 HΣ = Htriple-**set**
3 HΩ = HΣ-**set**
4 Htriple = LI×Colour×LI
axiom
2 ∀ ts:TS • obs_HΣ(ts) ∈ obs_HΩ(ts)
type
4 Colour == **red** | **yellow** | **green**
5 Signaling = (HΣ×TI)*
5 TI
6 Sempahore = (HΣ×HΣ) $\overrightarrow{m}$ Signalling
value
6 obs_Semaphore:TS → Sempahore

7 Based on [Bjø16c] we treat hubs as processes with hub state spaces and semaphores as static attributes and hub states as programmable attributes. We ignore other attributes and input/outputs.
8 We can think of the change of hub states as taking place based the result of some internal, non-deterministic choice.

value
7. hub: HI × LI-**set** × (HΩ×Semaphore) → HΣ **in** ... **out** ... **Unit**
7. hub(hi,lis,(hω,sema))(hσ) ≡
7. ...
8. ⌈⌉ **let** hσ′:HI • ... **in** hub(hi,lis,(hω,sema))(signaling(hσ,hσ′)) **end**
7. ...
7. **pre**: {hσ,hσ′} ⊆ hω

where we do not bother about the selection of $h\sigma'$.

9 Given two traffic signal, i.e., hub states, $h\sigma_{\text{init}}$ and $h\sigma_{\text{end}}$, where $h\sigma_{\text{init}}$ designates a present hub state and $h\sigma_{\text{end}}$ designates a desired next hub state after signaling.
10 Now *signaling* is a sequence of one or more successful hub state changes.

value
9 signaling: (HΣ×HΣ) × Semaphore → HΣ → HΣ
10 signaling(hσ$_{\text{init}}$,hσ$_{\text{end}}$,sema)(hσ) ≡ **let** sg = sema(hσ$_{\text{init}}$,hσ$_{\text{end}}$) **in** signal_sequence(sg)(hσ) **end**
10 **pre** hσ$_{\text{init}}$ = hσ ∧ (hσ$_{\text{init}}$,hσ$_{\text{end}}$) ∈ **dom** sema

If a desired hub state change fails (i.e., does not meet the **pre**-condition, or for other reasons (e.g., failure of technology)), then we do not define the outcome of signaling.

10 signal_sequence(⟨⟩)(hσ) ≡ hσ
10 signal_sequence(⟨(hσ‘,ti)⟩⌢sg)(hσ) ≡ **wait**(ti); signal_sequence(sg)(hσ‘)

We omit expression of a number of well-formedness conditions, e.g., that the *htriple* link identifiers are those of the corresponding mereology (*lis*), etcetera. The design of the semaphore, for a single hub or for a net of connected hubs has many similarities with the design of interlocking tables for railway tracks [HPK11]. •

Another example shows another aspect of support technology: Namely that the technology must guarantee certain of its own behaviours, so that software designed to interface with this technology, together with the technology, meets dependability requirements.

Example: 7. Railway Optical Gates: Train traffic (itf:iTF), intrinsically, is a total function over some time interval, from time (t:T) to continuously positioned (p:P) trains (tn:TN). Conventional optical gates sample, at

regular intervals, the intrinsic train traffic. The result is a sampled traffic (stf:sTF). Hence the collection of all optical gates, for any given railway, is a partial function from intrinsic to sampled train traffics (stf). We need to express quality criteria that any optical gate technology should satisfy — relative to a necessary and sufficient description of a closeness predicate. The following axiom does that:

- *For all intrinsic traffics, itf, and for all optical gate technologies, og, the following must hold: Let stf be the traffic sampled by the optical gates. For all time points, t, in the sampled traffic, those time points must also be in the intrinsic traffic, and, for all trains, tn, in the intrinsic traffic at that time, the train must be observed by the optical gates, and the actual position of the train and the sampled position must somehow be check-able to be close, or identical to one another.*

Since units change state with time, n:N, the railway net, needs to be part of any model of traffic.

```
type
  T, TN
  P = U*
  NetTraffic == net:N  trf:(TN ~m~> P)
  iTF = T → NetTraffic
  sTF = T ~m~> NetTraffic
  oG = iTF ~~> sTF
value
  close: NetTraffic × TN × NetTraffic ~~> Bool
axiom
  ∀ itt:iTF, og:OG • let stt = og(itt) in
    ∀ t:T • t ∈ dom stt  ⇒
      ∀ Tn:TN • tn ∈ dom trf(itt(t))
        ⇒ tn ∈ dom trf(stt(t)) ∧ close(itt(t),tn,stt(t)) end
```

Check-ability is an issue of testing the optical gates when delivered for conformance to the closeness predicate, i.e., to the axiom. •

3.2. Requirements

Section 4.4 [Extension] of [Bjø16b] illustrates a possible toll-gate, whose behaviour exemplifies a support technology. So do pumps of a pipe-line system such as illustrated in Examples 24, 29 and 42–44 in [Bjø16c]. A pump of a pipe-line system gives rise to several forms of support technologies: from the Egyptian Shadoof [irrigation] pumps, and the Hellenic Archimedian screw pumps, via the 11th century Su Song pumps of China[8], and the hydraulic "technologies" of Moorish Spain[9] to the centrifugal and gear pumps of the early industrial age, etcetera, The techniques – to mention those that have influenced this author – of [ZH04, JHJ07, OD08, HPK11] appears to apply well to the modeling of support technology requirements.

3.3. On Modeling Support Technologies

Support technologies in their relation to the domain in which they reside typically reflect real-time embeddedness. As such the techniques and languages for modeling support technologies resemble those for modeling event and process intensity, while temporal notions are brought into focus. Hence typical modeling notations include event-based languages (like Petri nets [Jen97, Pet62, Rei85, Rei92, Rei98] or CSP) [Hoa85, Ros97, Sch00, Hoa04]), respectively process-based specification languages (like MSCs, [IT92, IT96, IT99], LSCs [DH01, HM03, KW01], Statecharts [Har87, Har88, HLN+90, HN96, HG97], or CSP) [Hoa85, Ros97, Sch00, Hoa04]), as well as temporal languages (like the Duration Calculus and [ZH04, ZHR92] and Temporal Logic of Actions, TLA+) [Lam95, Lam02, Mer03]).

[8] https://en.wikipedia.org/wiki/Su_Song
[9] http://www.islamicspain.tv/Arts-and-Science/The-Culture-of-Al-Andalus/Hydraulic-Technology.htm

4. Rules &[10] Regulations

- *By a* **domain rule** *we shall understand some text (in the domain) which prescribes how people or equipment are expected to behave when dispatching their duties, respectively when performing their functions* ⊙
- *By a* **domain regulation** *we shall understand 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* ⊙

The domain rules & regulations need or may not be explicitly present, i.e., written down. They may be part of the "folklore", i.e., tacitly assumed and understood.

4.1. Conceptual Analysis

Example: 8. Trains at Stations:

- Rule: In China the arrival and departure of trains at, respectively from, railway stations is subject to the following rule:

  In any three-minute interval at most one train may either arrive to or depart from a railway station.

- Regulation: *If it is discovered that the above rule is not obeyed*, then there is some regulation which prescribes administrative or legal management and/or staff action, as well as some correction to the railway traffic.

•

Example: 9. Trains Along Lines:

- Rule: In many countries railway lines (between stations) are segmented into blocks or sectors. The purpose is to stipulate that if two or more trains are moving along the line, then:

  There must be at least one free sector (i.e., without a train) between any two trains along a line.

- Regulation: *If it is discovered that the above rule is not obeyed*, then there is some regulation which prescribes administrative or legal management and/or staff action, as well as some correction to the railway traffic.

•

At a meta-level, i.e., explaining the general framework for describing the syntax and semantics of the human-oriented domain languages for expressing rules and regulations, we can say the following: There are, abstractly speaking, usually three kinds of languages involved wrt. (i.e., when expressing) rules and regulations (respectively when invoking actions that are subject to rules and regulations). Two languages, Rules and Reg, exist for describing rules, respectively regulations; and one, Stimulus, exists for describing the form of the [always current] domain action stimuli. A syntactic stimulus, sy_sti, denotes a function, se_sti:STI: $\Theta \to \Theta$, from any configuration to a next configuration, where configurations are those of the system being subjected to stimulations. A syntactic rule, sy_rul:Rule, stands for, i.e., has as its semantics, its meaning, rul:RUL, a predicate over current and next configurations, $(\Theta \times \Theta) \to \mathbf{Bool}$, where these next configurations have been brought about, i.e., caused, by the stimuli. These stimuli express: If the predicate holds then the stimulus will result in a valid next configuration.

type
  Stimulus, Rule, Θ
  $\mathrm{STI} = \Theta \to \Theta$
  $\mathrm{RUL} = (\Theta \times \Theta) \to \mathbf{Bool}$
value
  meaning: Stimulus → STI
  meaning: Rule → RUL
  valid: Stimulus × Rule → Θ → **Bool**
  valid(sy_sti,sy_rul)(θ) ≡ meaning(sy_rul)(θ,(meaning(sy_sti))(θ))

[10] See footnote 16.

A syntactic regulation, sy_reg:Reg (related to a specific rule), stands for, i.e., has as its semantics, its meaning, a semantic regulation, se_reg:REG, which is a pair. This pair consists of a predicate, pre_reg:Pre_REG, where Pre_REG = (Θ × Θ) → **Bool**, and a domain configuration-changing function, act_reg:Act_REG, where Act_REG = Θ → Θ, that is, both involving current and next domain configurations. The two kinds of functions express: If the predicate holds, then the action can be applied. The predicate is almost the inverse of the rules functions. The action function serves to undo the stimulus function.

```
type
  Reg
  Rul_and_Reg = Rule × Reg
  REG = Pre_REG × Act_REG
  Pre_REG = Θ × Θ → Bool
  Act_REG = Θ → Θ
value
  interpret: Reg → REG
```

The idea is now the following: Any action (i.e., event) of the system, i.e., the application of any stimulus, may be an action (i.e., event) in accordance with the rules, or it may not. Rules therefore express whether stimuli are valid or not in the current configuration. And regulations therefore express whether they should be applied, and, if so, with what effort. More specifically, there is usually, in any current system configuration, given a set of pairs of rules and regulations. Let (sy_rul,sy_reg) be any such pair. Let sy_sti be any possible stimulus. And let θ be the current configuration. Let the stimulus, sy_sti, applied in that configuration result in a next configuration, θ', where $\theta' = (\mathsf{meaning}(\mathsf{sy\_sti}))(\theta)$. Let θ' violate the rule, $\sim\mathsf{valid}(\mathsf{sy\_sti},\mathsf{sy\_rul})(\theta)$, then if predicate part, pre_reg, of the meaning of the regulation, sy_reg, holds in that violating next configuration, $\mathsf{pre\_reg}(\theta,(\mathsf{meaning}(\mathsf{sy\_sti}))(\theta))$, then the action part, act_reg, of the meaning of the regulation, sy_reg, must be applied, $\mathsf{act\_reg}(\theta)$, to remedy the situation.

```
axiom
  ∀ (sy_rul,sy_reg):Rul_and_Reg •
    let se_rul = meaning(sy_rul),
        (pre_reg,act_reg) = meaning(sy_reg) in
    ∀ sy_sti:Stimulus, θ:Θ •
        ~valid(sy_sti,se_rul)(θ)
          ⇒ pre_reg(θ,(meaning(sy_sti))(θ))
              ⇒ ∃ nθ:Θ • act_reg(θ)=nθ ∧ se_rul(θ,nθ)
    end
```

It may be that the regulation predicate fails to detect applicability of regulations actions. That is, the interpretation of a rule differs, in that respect, from the interpretation of a regulation. Such is life in the domain, i.e., in actual reality.

4.2. Requirements

Implementation of rules & regulations implies monitoring and partially controlling the states symbolised by Θ in Sect. 4.1. Thus some partial implementation of Θ must be required; as must some monitoring of states θ:Θ and implementation of the predicates *meaning*, *valid*, *interpret*, *pre_reg* and action(s) *act_reg*. The emerging requirements follow very much in the line of support technology requirements.

4.3. On Modeling Rules and Regulations

Usually rules (as well as regulations) are expressed in terms of domain entities, including those grouped into "the state", functions, events, and behaviours. Thus the full spectrum of model-ling techniques and notations may be needed. Since rules usually express properties one often uses some combination of axioms and wellformedness predicates. Properties sometimes include temporality and hence temporal notations (like Duration Calculus or Temporal Logic of Actions) are used. And since regulations usually express state

(restoration) changes one often uses state changing notations (such as found in Allard [Jac06], B or event-B [Abr09], RSL [GHH+92], VDM-SL [BJ78, BJ82, FL97], and Z [Spi88, Spi92, WD96, HRB03]). In some cases it may be relevant to model using some constraint satisfaction notation [Apt03] or some Fuzzy Logic notations [VVD90].

5. Scripts

- *By a* **domain script** *we shall understand the structured, almost, if not outright, formally expressed, wording of a procedure on how to proceed, one that has legally binding power, that is, which may be contested in a court of law* ⊙

5.1. Conceptual Analysis

Rules & regulations are usually expressed, even when informally so, as predicates. Scripts, in their procedural form, are like instructions, as for an algorithm.

Example: 10. A Casually Described Bank Script: Our formulation amounts to just a (casual) rough sketch. It is followed by a series of four large examples. Each of these elaborate on the theme of (bank) scripts. The problem area is that of how repayments of mortgage loans are to be calculated. At any one time a mortgage loan has a balance, a most recent previous date of repayment, an interest rate and a handling fee. When a repayment occurs, then the following calculations shall take place: (i) the interest on the balance of the loan since the most recent repayment, (ii) the handling fee, normally considered fixed, (iii) the effective repayment — being the difference between the repayment and the sum of the interest and the handling fee — and the new balance, being the difference between the old balance and the effective repayment. We assume repayments to occur from a designated account, say a demand/deposit account. We assume that bank to have designated fee and interest income accounts. (i) The interest is subtracted from the mortgage holder's demand/deposit account and added to the bank's interest (income) account. (ii) The handling fee is subtracted from the mortgage holder's demand/deposit account and added to the bank's fee (income) account. (iii) The effective repayment is subtracted from the mortgage holder's demand/deposit account and also from the mortgage balance. Finally, one must also describe deviations such as overdue repayments, too large, or too small repayments, and so on. •

Example: 11. A Formally Described Bank Script: First we must informally and formally define the bank state: There are clients (c:C), account numbers (a:A), mortgage numbers (m:M), account yields (ay:AY) and mortgage interest rates (mi:MI). The bank registers, by client, all accounts (ρ:A_Register) and all mortgages (μ:M_Register). To each account number there is a balance (α:Accounts). To each mortgage number there is a loan (ℓ:Loans). To each loan is attached the last date that interest was paid on the loan.

value
  r, r′:**Real axiom** ...
type
  C, A, M, Date
  AY′ = **Real**, AY = {| ay:AY′ • 0<ay≤r |}
  MI′ = **Real**, MI = {| mi:MI′ • 0<mi≤r′ |}
  Bank′ = A_Register × Accounts × M_Register × Loans
  Bank = {| β:Bank′ • wf_Bank(β)|}
  A_Register = C $\xrightarrow{m}$ A-**set**
  Accounts = A $\xrightarrow{m}$ Balance
  M_Register = C $\xrightarrow{m}$ M-**set**
  Loans = M $\xrightarrow{m}$ (Loan × Date)
  Loan,Balance = P
  P = **Nat**

Then we must define well-formedness of the bank state:

value
  ay:AY, mi:MI

```
  wf_Bank: Bank → Bool
  wf_Bank(ρ,α,μ,ℓ) ≡ ∪ rng ρ = dom α ∧ ∪ rng μ = dom ℓ
axiom
  ay<mi [ ∧ ... ]
```

We — perhaps too rigidly — assume that mortgage interest rates are higher than demand/deposit account interest rates: ay<mi. Operations on banks are denoted by the commands of the bank script language. First the syntax:

```
type
  Cmd = OpA | CloA | Dep | Wdr | OpM | CloM | Pay
  OpA == mkOA(c:C)
  CloA == mkCA(c:C,a:A)
  Dep == mkD(c:C,a:A,p:P)
  Wdr == mkW(c:C,a:A,p:P)
  OpM == mkOM(c:C,p:P)
  Pay == mkPM(c:C,a:A,m:M,p:P,d:Date)
  CloM == mkCM(c:C,m:M,p:P)
  Reply = A | M | P | OkNok
  OkNok == ok | notok
value
  period: Date × Date → Days [for calculating interest]
  before: Date × Date → Bool [first date is earlier than last date]
```

And then the semantics:

```
  int_Cmd(mkPM(c,a,m,p,d))(ρ,α,μ,ℓ) ≡
    let (b,d′) = ℓ(m) in
    if α(a)≥p
      then
        let i = interest(mi,b,period(d,d′)),
            ℓ′ = ℓ † [m↦ℓ(m)−(p−i)]
            α′ = α † [a↦α(a)−p,a_i↦α(a_i)+i] in
        ((ρ,α′,μ,ℓ′),ok) end
      else
        ((ρ,α′,μ,ℓ),nok)
    end end
    pre c ∈ dom μ ∧ a ∈ dom α ∧ m ∈ μ(c)
    post before(d,d′)

    interest: MI × Loan × Days → P
```

•

The idea about scripts is that they can somehow be objectively enforced: that they can be precisely understood and consistently carried out by all stakeholders, eventually leading to computerisation. But they are, at all times, part of the domain.

5.2. Requirements

Script requirements call for the possibly interactive computerisation of algorithms, that is, for rather classical computing problems. But sometimes these scripts can be expressed, computably, in the form of programs in a domain specific language. As an example we refer to [CGN+]. [CGN+] illustrates how the design of pension and life insurance products, and their administration, reserve calculations, and audit, can be based on a common formal notation. The notation is human-readable and machine-processable,and specialised to the actuarial domain, achieving great expressive power combined with ease of use and safety. More specifically (a) product definitions based on standard actuarial models, including arbitrary continuous-time Markov and

semi-Markov models, with cyclic transitions permitted; (b) calculation descriptions for reserves and other quantities of interest, based on differential equations; and (c) administration rules.

5.3. On Modeling Scripts

Scripts (as are licenses) are like programs (respectively like prescriptions program executions). Hence the full variety of techniques and notations for modeling programming (or specification) languages apply [Bak95, Gun92, Rey99, Sch86, Ten97, Win93]. [Bjø06b, Chaps. 6–9] cover pragmatics, semantics and syntax techniques for defining functional, imperative and concurrent programming languages.

6. License Languages

License: a right or permission
granted in accordance with law
by a competent authority
to engage in some business or occupation,
to do some act, or to engage in some transaction
which but for such license would be unlawful ⊙

Merriam Webster Online [Mer04]

6.1. Conceptual Analysis

6.1.1. The Settings

A special form of scripts are increasingly appearing in some domains, notably the domain of electronic, or digital media. Here licenses express that a licensor, o, permits a licensee, u, to *render* (i.e., play) works of proprietary nature CD ROM-like music, DVD-like movies, etc. while obligating the licensee to pay the licensor on behalf of the owners of these, usually artistic works. Classical digital rights license languages, [Ben02, AH05, CCD+06, CEH03, CCE03, Inc00, C.E02, GWW01, HW04, Ltd01, MB02, MVJD05, Lyo02, KLMM04, Sam03, PW04, PW02, All05, MHB03] applied to the electronic "downloading", payment and rendering (playing) of artistic works (for example music, literature readings and movies). In this paper we generalise such applications languages and we extend the concept of licensing to also cover work authorisation (work commitment and promises) in health care, public government and schedule transport. The digital works for these new application domains are patient medical records, public government documents and bus/train/aircraft transport contracts. Digital rights licensing for artistic works seeks to safeguard against piracy and to ensure proper payments for the rights to render these works. Health care and public government license languages seek to ensure transparent and professional (accurate and timely) health care, respectively 'good governance'. Transport contract languages seeks to ensure timely and reliable transport services by an evolving set of transport companies. Proper mathematical definition of licensing languages seeks to ensure smooth and correct computerised management of licenses and contracts.

6.1.2. On Licenses

The concepts of licenses and licensing express relations between (i) *actors* (licensors (the authority) and licensees), (ii) *entities* (artistic works, hospital patients, public administration, citizen documents) and bus transport contracts and (iii) *functions* (on entities), and as performed by actors. By issuing a license to a licensee, a licensor wishes to express and enforce certain permissions and obligations: which functions on which entities the licensee is allowed (is licensed, is permitted) to perform. In this paper we shall consider four kinds of entities: (i) digital recordings of artistic and intellectual nature: music, movies, readings ("audio books"), and the like, (ii) patients in a hospital as represented also by their patient medical records, (iii) documents related to public government, and (iv) transport vehicles, time tables and transport nets (of a buses, trains and aircraft).

6.1.3. Permissions and Obligations

The *permissions* and *obligations* issues are, (1) for the owner (agent) of some intellectual property to be paid (an *obligation*) by users when they perform *permitted* operations (rendering, copying, editing, sub-licensing) on their works; (2) for the patient to be professionally treated — by medical staff who are basically *obliged* to try to cure the patient; (3) for public administrators and citizens to enjoy good governance: transparency in law making (national parliaments and local prefectures and city councils), in law enforcement (i.e., the daily administration of laws), and law interpretation (the judiciary) — by agents who are basically *obliged* to produce certain documents while being *permitted* to consult (i.e., read, perhaps copy) other documents; and (4) for bus passengers to enjoy reliable bus schedules — offered by bus transport companies on contract to, say public transport authorities and on sub-contract to other such bus transport companies where these transport companies are *obliged* to honour a contracted schedule.

6.2. The Pragmatics

By **pragmatics** *we understand the study and practice of the factors that govern our choice of language in social interaction and the effects of our choice on others.*

In this section we shall rough-sketch-describe pragmatic aspects of the four domains of (1) production, distribution and consumption of artistic works, (2) the hospitalisation of patient, i.e., hospital health care, (3) the handling of law-based document in public government and (4) the operational management of schedule transport vehicles. The emphasis is on the pragmatics of the terms, i.e., the language used in these four domains.

6.2.1. Digital Media

Example: 12. Digital Media: The intrinsic entities of the performing arts are the artistic works: drama or opera performances, music performances, readings of poems, short stories, novels, or jokes, movies, documentaries, newsreels, etc. We shall limit our span to the scope of electronic renditions of these artistic works: videos, CDs or other. In this paper we shall not touch upon the technical issues of "downloading"(whether "streaming" or copying, or other). That and other issues should be analysed in [XB06].

Operations on Digital Works For a consumer to be able to enjoy these works that consumer must (normally first) usually "buy a ticket" to their performances. The consumer, i.e., the theatre, opera, concert, etc., "goer" (usually) cannot copy the performance (e.g., "tape it"), let alone edit such copies of performances. In the context of electronic, i.e., digital renditions of these performances the above "cannots" take on a new meaning. The consumer may copy digital recordings, may edit these, and may further pass on such copies or editions to others. To do so, while protecting the rights of the producers (owners, performers), the consumer requests permission to have the digital works transferred ("downloaded") from the owner/producer to the consumer, so that the consumer can render ("play") these works on own rendering devices (CD, DVD, etc., players), possibly can copy all or parts of them, then possibly can edit all or parts of the copies, and, finally, possibly can further license these "edited" versions to other consumers subject to payments to "original" licensor.

License Agreement and Obligation To be able to obtain these permissions the user agrees with the wording of some license and pays for the rights to operate on the digital works.

Two Assumptions Two, related assumptions underlie the pragmatics of the electronics of the artistic works. The first assumption is that the format, the electronic representation of the artistic works is proprietary, that is, that the producer still owns that format. Either the format is publicly known or it is not, that is, it is somehow "secret". In either case we "derive" the second assumption (from the fulfillment of the first). The second assumption is that the consumer is not allowed to, or cannot operate[11] on the works by own means (software, machines). The second assumption implies that acceptance of a license results in the consumer receiving software that supports the consumer in performing all operations on licensed works, their copies and edited versions: rendering, copying, editing and sub-licensing.

[11] render, copy and edit

Protection of the Artistic Electronic Works The issue now is: how to protect the intellectual property (i.e., artistic) and financial (exploitation) rights of the owners of the possibly rendered, copied and edited works, both when, and when not further distributed. •

6.2.2. Health-care

Example: 13. Health-care: Citizens go to hospitals in order to be treated for some calamity (disease or other), and by doing so these citizens become patients. At hospitals patients, in a sense, issue a request to be treated with the aim of full or partial restitution. This request is directed at medical staff, that is, the patient authorises medical staff to perform a set of actions upon the patient. One could claim, as we shall, that the patient issues a license.

Patients and Patient Medical Records So patients and their attendant patient medical records (PMRs) are the main entities, the "works" of this domain. We shall treat them synonymously: PMRs as surrogates for patients. Typical actions on patients — and hence on PMRs — involve admitting patients, interviewing patients, analysing patients, diagnosing patients, planning treatment for patients, actually treating patients, and, under normal circumstance, to finally release patients.

Medical Staff Medical staff may request ('refer' to) other medical staff to perform some of these actions. One can conceive of describing action sequences (and 'referrals') in the form of hospitalisation (not treatment) plans. We shall call such scripts for licenses.

Professional Health Care The issue is now, given that we record these licenses, their being issued and being honoured, whether the handling of patients at hospitals follow, or does not follow properly issued licenses. •

6.2.3. Government Documents

Example: 14. Documents: By public government we shall, following Charles de Secondat, baron de Montesquieu (1689–1755)[12], understand a composition of three powers: the law-making (legislative), the law-enforcing and the law-interpreting parts of public government. Typically national parliament and local (province and city) councils are part of law-making government. Law-enforcing government is called the executive (the administration). And law-interpreting government is called the judiciary [system] (including lawyers etc.).

Documents A crucial means of expressing public administration is through *documents*.[13] We shall therefore provide a brief domain analysis of a concept of documents. (This document domain description also applies to patient medical records and, by some "light" interpretation, also to artistic works — insofar as they also are documents.) Documents are *created*, *edited* and *read*; and documents can be *copied*, *distributed*, the subject of *calculations* (interpretations) and be *shared* and *shredded*.

Document Attributes With documents one can associate, as attributes of documents, the *actors* who created, edited, read, copied, distributed (and to whom distributed),shared, performed calculations and shredded documents. With these operations on documents, and hence as attributes of documents one can, again conceptually, associate the *location* and *time* of these operations.

Actor Attributes and Licenses With actors (whether agents of public government or citizens) one can associate the *authority* (i.e., the *rights*) these actors have with respect to performing actions on documents. We now intend to express these *authorisations as licenses*.

Document Tracing An issue of public government is whether citizens and agents of public government act in accordance with the laws — with actions and laws reflected in documents such that the action documents enables a trace from the actions to the laws "governing" these actions. We shall therefore assume that every document can be traced back to its law-origin as well as to all the documents any one document-creation or -editing was based on. •

6.2.4. Transportation

Example: 15. Passenger and Goods Transport:

[12] *De l'esprit des lois* (*The Spirit of the Laws*), published 1748
[13] Documents are, for the case of public government to be the "equivalent" of artistic works.

A Synopsis Contracts obligate transport companies to deliver bus traffic according to a timetable. The timetable is part of the contract. A contractor may sub-contract (other) transport companies to deliver bus traffic according to timetables that are sub-parts of their own timetable. Contractors are either public transport authorities or contracted transport companies. Contracted transport companies may cancel a subset of bus rides provided the total amount of cancellations per 24 hours for each bus line does not exceed a contracted upper limit The cancellation rights are spelled out in the contract. A sub-contractor cannot increase a contracted upper limit for cancellations above what the sub-contractor was told (in its contract) by its contractor. Etcetera.

A Pragmatics and Semantics Analysis The "works" of the bus transport contracts are two: the timetables and, implicitly, the designated (and obligated) bus traffic. A bus timetable appears to define one or more bus lines, with each bus line giving rise to one or more bus rides. Nothing is (otherwise) said about regularity of bus rides. It appears that bus ride cancellations must be reported back to the contractor. And we assume that cancellations by a sub-contractor is further reported back also to the sub-contractor's contractor. Hence eventually that the public transport authority is notified. Nothing is said, in the contracts, such as we shall model them, about passenger fees for bus rides nor of percentages of profits (i.e., royalties) to be paid back from a sub-contractor to the contractor. So we shall not bother, in this example, about transport costs nor transport subsidies. But will leave that necessary aspect as an exercise. The opposite of cancellations appears to be 'insertion' of extra bus rides, that is, bus rides not listed in the time table, but, perhaps, mandated by special events[14] We assume that such insertions must also be reported back to the contractor. We assume concepts of acceptable and unacceptable bus ride delays. Details of delay acceptability may be given in contracts, but we ignore further descriptions of delay acceptability. but assume that unacceptable bus ride delays are also to be (iteratively) reported back to contractors. We finally assume that sub-contractors cannot (otherwise) change timetables. (A timetable change can only occur after, or at, the expiration of a license.) Thus we find that contracts have definite period of validity. (Expired contracts may be replaced by new contracts, possibly with new timetables.)

Contracted Operations, An Overview The actions that may be granted by a contractor according to a contract are: (i) *start:* to commence, i.e., to start, a bus ride (obligated); (ii) *end:* to conclude a bus ride (obligated); (iii) *cancel:* to cancel a bus ride (allowed, with restrictions); (iv) *insert:* to insert a bus ride; and (v) *subcontract:* to sub-contract part or all of a contract. •

6.3. Schematic Rendition of License Language Constructs

There are basically two aspects to licensing languages: (i) the [actual] licensing [and sub-licensing], in the form of licenses, ℓ, by licensors, o, of permissions and thereby implied obligations, and (ii) the carrying-out of these obligations in the form of licensee, u, actions. We shall in this paper treat licensors and licensees on par, that is, some os are also us and vice versa. And we shall think of licenses as not necessarily material entities (e.g., paper documents), but allow licenses to be tacitly established (understood).

6.3.1. Licensing

The granting of a license ℓ by a licensor o, to a set of licensees $u_{u_1}, u_{u_2}, ..., u_{u_u}$ in which ℓ expresses that these may perform actions $a_{a_1}, a_{a_2}, ..., a_{a_a}$ on work items $e_{e_1}, e_{e_2}, ..., e_{e_e}$ can be schematised:

ℓ : **licensor** o **contracts licensees** $\{u_{u_1}, u_{u_2}, ..., u_{u_u}\}$
to perform actions $\{a_{a_1}, a_{a_2}, ..., a_{a_a}\}$ **on work items** $\{e_{e_1}, e_{e_2}, ..., e_{e_e}\}$
allowing sub-licensing of actions $\{a_{a_i}, a_{a_j}, ..., a_{a_k}\}$ **to** $\{u_{u_x}, u_{u_y}, ..., u_{u_z}\}$

The two sets of action designators, das :$\{a_{a_1}, a_{a_2}, ..., a_{a_a}\}$ and sas :$\{a_{a_x}, a_{a_y}, ..., a_{a_z}\}$ need not relate. **Sub-licensing:** Line 3 of the above schema, ℓ, expresses that licensees $u_{u_1}, u_{u_2}, ..., u_{u_u}$, may act as licensors and (thereby sub-)license ℓ to licensees us : $\{u_{u_x}, u_{u_y}, ..., u_{u_z}\}$, distinct from sus : $\{u_{u_1}, u_{u_2}, ..., u_{u_u}\}$, that is, $us \cap sus = \{\}$. **Variants:** One can easily "cook up" any number of variations of the above license schema. **Revoke Licenses:** We do not show expressions for revoking part or all of a previously granted license.

6.3.2. Licensors and Licensees

Example: 16. Licensors and Licensees:

[14] Special events: breakdown (that is, cancellations) of other bus rides, sports event (soccer matches), etc.

Digital Media : For digital media the original licensors are the original producers of music, film, etc. The "original" licensees are you and me! Thereafter some of us may become licensors, etc.

Heath-care : For health-care the original licensors are, say in Denmark, the Danish governments' National Board of Health[15]; and the "original" licensees are the national hospitals. These then sub-license their medical clinics (rheumatology, cancer, urology, gynecology, orthopedics, neurology, etc.) which again sub-licenses their medical staff (doctors, nurses, etc.). A medical doctor may, as is the case in Denmark for certain actions, not [necessarily] perform these but may sub-license their execution to nurses, etc.

Documents : For government documents the original licensor are the (i) heads of parliament, regional and local governments, (ii) government (prime minister) and the heads of respective ministries, respectively the regional and local agencies and administrations. The "original" licensees are (i′) the members of parliament, regional and local councils charged with drafting laws, rules and regulations, (ii′) the ministry, respectively the regional and local agency department heads. These (the ′s) then become licensors when licensing their staff to handle specific documents.

Transport : For scheduled passenger (etc.) transportation the original licensors are the state, regional and/or local transport authorities. The "original" licensees are the public and private transport firms. These latter then become licensors licensors licensing drivers to handle specific transport lines and/or vehicles. •

6.3.3. Actors and Actions

Example: 17. Actors and Actions:

Digital Media : w refers to a digital "work" with w' designating a newly created one; s_i refers to a sector of some work. **render** $w(s_i, s_j, ..., s_k)$: sectors $s_i, s_j, ..., s_k$ of work w are rendered (played, visualised) in that order. w' **:= copy** $w(s_i, s_j, ..., s_k)$: sectors $s_i, s_j, ..., s_k$ of work w are copied and becomes work w'. w' **:= edit** w **with** $\mathcal{E}(w_\alpha(s_a, s_b, ..., s_c), ..., w_\gamma(s_p, s_q, ..., s_r))$: work w is edited while [also] incorporating references to or excerpts from [other] works $w_\alpha(s_a, s_b, ..., s_c), ..., w_\gamma(s_p, s_q, ..., s_r)$. **read** w: work w is read, i.e., information about work w is somehow displayed. ℓ **: licensor m contracts licensees** $\{\mathbf{u}_{u_1}, \mathbf{u}_{u_2}, ..., \mathbf{u}_{u_u}\}$ **to perform actions** {RENDER, COPY, EDIT, READ} **on work items** $\{w_{i_1}, w_{i_2}, ..., w_{i_w}\}$. Etcetera: other forms of actions can be thought of.

Heath-care : Actors are here limited to the patients and the medical staff. We refer to Fig. 3 on the following page. It shows an archetypal hospitalisation plan and identifies a number of actions; π designates patients, t designates treatment (medication, surgery, ...). Actions are performed by medical staff, say h, with h being an implicit argument of the actions. **interview** π: a PMR with name, age, family relations, addresses, etc., is established for patient π. **admit** π: the PMR records the anamnese (medical history) for patient π. **establish analysis plan** π: the PMR records which analyses (blood tests, ECG, blood pressure, etc.) are to be carried out. **analyse** π: the PMR records the results of the analyses referred to previously. **diagnose** π: medical staff h diagnoses, based on the analyses most recently performed. **plan treatment for** π: medical staff h sets up a treatment plan for patient π based on the diagnosis most recently performed. **treat** π **wrt.** t: medical staff h performs treatment t on patient π, observes "reaction" and records this in the PMR. Predicate "actions": **more analysis** π**?**, **more treatment** π**?** and **more diagnosis** π**?**. **release** π: either the patient dies or is declared ready to be sent 'home'. ℓ **: licensor o contracts medical staff** $\{m_{m_1}, m_{m_2}, ..., m_{m_m}\}$ **to perform actions** {INTERVIEW, ADMIT, PLAN ANALYSIS, ANALYSE, DIAGNOSE, PLAN TREATMENT, TREAT, RELEASE} **on patients** $\{\pi_{p_1}, \pi_{p_2}, ..., \pi_{p_p}\}$. Etcetera: other forms of actions can be thought of.

Documents : d refer to documents with d' designating new documents. d' **:= create based on** $d_x, d_y, ..., d_z$: A new document, named d', is created, with no information "contents", but referring to existing documents $d_x, d_y, ..., d_z$. **edit** d **with** $\mathcal{E}$ **based on** $d_{n_\alpha}, d_\beta, ..., d_\gamma$: document d is edited with $\mathcal{E}$ being the editing function and $\mathcal{E}^{-1}$ being its "undo" inverse. **read** d: document d is being read. d' **:= copy** d: document d is copied into a new document named d'. **freeze** d: document d can, from now on, only be read. **shred** d: document d is shredded. That is, no more actions can be performed on d. ℓ **: licensor o contracts civil service staff** $\{c_{c_1}, c_{c_2}, ..., c_{c_c}\}$ **to perform actions** {CREATE, EDIT, READ, COPY, FREEZE, SHRED} **on documents** $\{d_{d_1}, d_{d_2}, ..., d_{d_d}\}$. Etcetera: other forms of actions can be thought of.

Transport : We restrict, without loss of generality, to bus transport. There is a timetable, tt. It records bus lines, l, and specific instances of bus rides, b. **start bus ride** l, b **at time** t: Bus line l is recorded in tt and its departure in tt is recorded as τ. Starting that bus ride at t means that the start is either on time, i.e., $t{=}\tau$, or the

[15] In the UK: the NHS, etc.

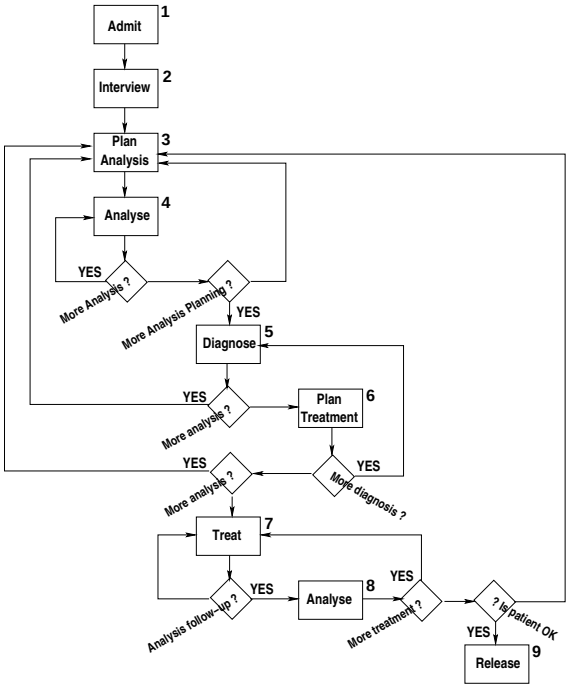

Fig. 3. An example single-illness non-fatal hospitalisation plan. States: {1,2,3,4,5,6,7,8,9}

start is delayed $\delta_d : \tau\text{-}t$ or advanced $\delta_a : t\text{-}\tau$ where δ_d and δ_a are expected to be small intervals. All this is to be reported, in due time, to the contractor. **end bus ride** l, b **at time** t: Ending bus ride l, b at time t means that it is either ended on time, or earlier, or delayed. This is to be reported, in due time, to the contractor. **cancel bus ride** l, b **at time** t: t must be earlier than the scheduled departure of bus ride l, b. **insert an extra bus** l, b' **at time** t: t must be the same time as the scheduled departure of bus ride l, b with b' being a "marked" version of b. ℓ : **licensor o contracts transport staff** $\{b_{b_1}, b_{b_2}, ..., b_{b_b}\}$ **to perform actions** {START, END, CANCEL, INSERT} **on work items** $\{\mathbf{e}_{e_1}, \mathbf{e}_{e_2}, ..., \mathbf{e}_{e_e}\}$. Etcetera: other forms of actions can be thought of. •

6.4. Requirements

Requirements for license language implementation basically amounts to requirements for three aspects. (i) The design of the license language, its abstract and concrete syntax, its interpreter, and its interfaces to distributed licensor and licensee behaviours; (ii) the requirements for a distributed system of licensor and licensee behaviours; and (iii) the monitoring and partial control of the states of licensor and licensee behaviours. The structuring of these distributed licensor and licensee behaviours differ from slightly to somewhat, but not that significant in the four license languages examples. Basically the licensor and licensee behaviours form a set of behaviours. Basically everyone can communicate with everyone. For the case of digital media licensee behaviours communicate back to licensor behaviours whenever a properly licensed action is performed – resulting in the transfer of funds from licensees to licensors. For the case of health care some central authority is expected to validate the granting of licenses and appear to be bound by medical training. For the case of documents such checks appear to be bound by predetermined authorisation rules. For the case of transport one can perhaps speak of more rigid management & organisation dependencies as licenses are traditionally transferred between independent authorities and companies.

6.5. On Modeling License Languages

Licensors are expected to maintain a state which records all the licenses it has issued. Whenever at licensee "reports back" (the begin and/or the end) of the performance of a granted action, this is recorded in its state. Sometimes these granted actions are subject to fees. The licensor therefore calculates outstanding fees — etc. Licensees are expected to maintain a state which records all the licenses it has accepted. Whenever an action is to be performed the licensee records this and checks that it is permitted to perform this action. In many cases the licensee is expected to "report back", both the beginning and the end of performance of that action, to the licensor. A typical technique of modeling licensors, licensees and patients, i.e., their PMRs, is to model them as (never ending) processes, a la CSP [Hoa85, Ros97, Sch00, Hoa04]with input/output, ch ?/ch ! m, communications between licensors, licensees and PMRs. Their states are modeled as programmable attributes.

7. Management &[16] Organisation

- *By* **domain management** *we shall understand such people (such decisions) (i) who (which) determine, formulate and thus set standards (cf. rules and regulations, Sect. 4) 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 "backstops" complaints from lower management levels and from "floor" staff* ⊙
- *By* **domain organisation** *we shall understand (vi) the structuring of management and non-management staff "overseeable" into clusters with "tight" and "meaningful" relations; (vii) the allocation of strategic, tactical and operational concerns to within management and non-management staff clusters; and hence (viii) the "lines of command": who does what, and who reports to whom, administratively and functionally* ⊙

The '&' is justified from the interrelations of items *(i–viii)*.

7.1. Conceptual Analysis

We first bring some examples.

Example: 18. Train Monitoring, I: In China, as an example, till the early 1990s, rescheduling of trains occurs at stations and involves telephone negotiations with neighbouring stations ("up and down the lines"). Such rescheduling negotiations, by phone, imply reasonably strict management and organisation (M&O). This kind of M&O reflects the geographical layout of the rail net. •

Example: 19. Railway Management and Organisation: Train Monitoring, II: We single out a rather special case of railway management and organisation. Certain (lowest-level operational and station-located) supervisors are responsible for the day-to-day timely progress of trains within a station and along its incoming and outgoing lines, and according to given timetables. These supervisors and their immediate (middle-level) managers (see below for regional managers) set guidelines (for local station and incoming and outgoing lines) for the monitoring of train traffic, and for controlling trains that are either ahead of or behind their schedules. By an incoming and an outgoing line we mean part of a line between two stations, the remaining part being handled by neighbouring station management. Once it has been decided, by such a manager, that a train is not following its schedule, based on information monitored by non-management staff, then that manager directs that staff: (i) to suggest a new schedule for the train in question, as well as for possibly affected other trains, (ii) to negotiate the new schedule with appropriate neighbouring stations, until a proper reschedule can be decided upon, by the managers at respective stations, (iii) and to enact that new schedule.[17] A (middle-level operations) manager for regional traffic, i.e., train traffic involving several stations and lines, resolves possible disputes and conflicts. •

[16] The concept unifier '&' expresses that $A\&B$ designates one concept, not two: A and B.
[17] That enactment may possibly imply the movement of several trains incident upon several stations: the one at which the manager is located, as well as possibly at neighbouring stations.

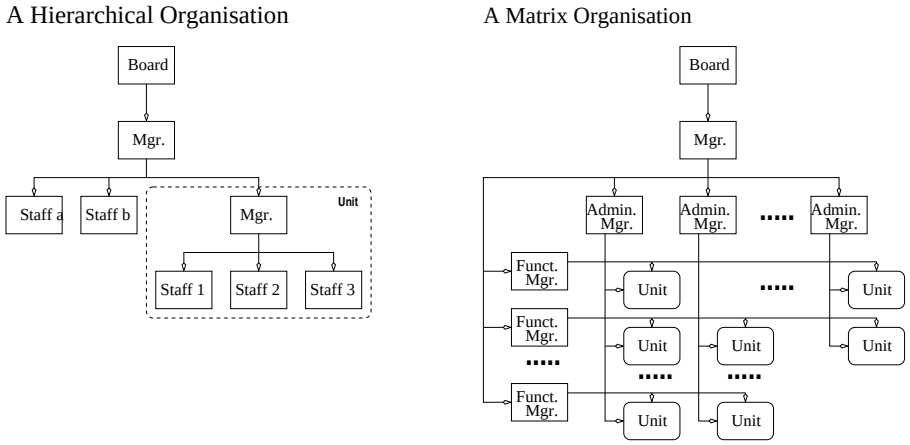

Fig. 4. Organisational structures

The above, albeit rough-sketch description, illustrated the following management and organisation issues: (i) There is a set of lowest-level (as here: train traffic scheduling and rescheduling) supervisors and their staff; (ii) they are organised into one such group (as here: per station); (iii) there is a middle-level (as here: regional train traffic scheduling and rescheduling) manager (possibly with some small staff), organised with one such per suitable (as here: railway) region; and (iv) the guidelines issued jointly by local and regional (...) supervisors and managers imply an organisational structuring of lines of information provision and command.

People staff enterprises, the components of infrastructures with which we are concerned, i.e., for which we develop software. The larger these enterprises — these infrastructure components — the more need there is for management and organisation. The role of management is roughly, for our purposes, twofold: first, to perform strategic, tactical and operational work, to set strategic, tactical and operational policies — and to see to it that they are followed. The role of management is, second, to react to adverse conditions, that is, to unforeseen situations, and to decide how they should be handled, i.e., conflict resolution. Policy setting should help non-management staff operate normal situations — those for which no management interference is thus needed. And management "backstops" problems: management takes these problems off the shoulders of non-management staff. To help management and staff know who's in charge wrt. policy setting and problem handling, a clear conception of the overall organisation is needed. Organisation defines lines of communication within management and staff, and between these. Whenever management and staff has to turn to others for assistance they usually, in a reasonably well-functioning enterprise, follow the command line: the paths of organigrams — the usually hierarchical box and arrow/line diagrams.

The *management and organisation* model of a domain is a partial specification; hence all the usual abstraction and modeling principles, techniques and tools apply. More specifically, management is a set of predicate functions, or of observer and generator functions These either parametrise other, the operations functions, that is, determine their behaviour, or yield results that become arguments to these other functions. Organisation is thus a set of constraints on communication behaviours. Hierarchical, rather than linear, and matrix structured organisations can also be modeled as sets (of recursively invoked sets) of equations.

To relate classical organigrams to formal descriptions we first show such an organigram (Fig. 4), and then we show schematic processes which — for a rather simple scenario — model managers and the managed! Based on such a diagram, and modeling only one neighbouring group of a manager and the staff working for that manager we get a system in which one manager, mgr, and many staff, stf, coexist or work concurrently, i.e., in parallel. The mgr operates in a context and a state modeled by ψ. Each staff, stf(i) operates in a context and a state modeled by $s\sigma$(i).

type
  Msg, Ψ, Σ, Sx
  $S\Sigma = Sx \xrightarrow[m]{} \Sigma$
channel

```
  { ms[i]:Msg | i:Sx }
value
  sσ:SΣ, ψ:Ψ

  sys: Unit → Unit
  sys() ≡ ‖ { stf(i)(sσ(i)) | i:Sx } ‖ mgr(ψ)
```

In this system the manager, mgr, (1) either broadcasts messages, m, to all staff via message channel ms[i]. The manager's concoction, m_out(ψ), of the message, msg, has changed the manager state. Or (2) is willing to receive messages, msg, from whichever staff i the manager sends a message. Receipt of the message changes, m_in(i,m)(ψ), the manager state. In both cases the manager resumes work as from the new state. The manager chooses — in this model — which of thetwo things (1 or 2) to do by a so-called non-deterministic internal choice (⌈⌉).

```
  mg: Ψ → in,out {ms[i]|i:Sx} Unit
  mgr(ψ) ≡
(1)   let (ψ′,m)=m_out(ψ) in ‖ {ms[i]!m|i:Sx};mgr(ψ′) end
      ⌈⌉
(2)   let ψ′ = [] {let m=ms[i]? in m_in(i,m)(ψ) end|i:Sx} in mgr(ψ′) end

  m_out: Ψ → Ψ × MSG,
  m_in: Sx × MSG → Ψ → Ψ
```

And in this system, staff i, stf(i), (1) either is willing to receive a message, msg, from the manager, and then to change, st_in(msg)(σ), state accordingly, or (2) to concoct, st_out(σ), a message, msg (thus changing state) for the manager, and send it ms[i]!msg. In both cases the staff resumes work as from the new state. The staff member chooses — in this model — which of thetwo "things" (1 or 2) to do by a non-deterministic internal choice (⌈⌉).

```
  stf: i:Sx → Σ → in,out ms[i] Unit
  stf(i)(σ) ≡
(1)   let m = ms[i]? in stf(i)(stf_in(m)(σ)) end
      ⌈⌉
(2)   let (σ′,m) = st_out(σ) in ms[i]!m; stf(i)(σ′) end

  st_in: MSG → Σ → Σ,
  st_out: Σ → Σ × MSG
```

Both manager and staff processes recurse (i.e., iterate) over possibly changing states. The management process non-deterministically, internal choice, "alternates" between "broadcast"-issuing orders to staff and receiving individual messages from staff. Staff processes likewise non-deterministically, internal choice, alternate between receiving orders from management and issuing individual messages to management. The conceptual example also illustrates modeling stakeholder behaviours as interacting (here CSP-like) processes.

Example: 20. Strategic, Tactical and Operations Management: We think of (i) strategic, (ii) tactic, and (iii) operational managers as well as (iv) supervisors, (v) team leaders and the rest of the (vi) staff (i.e., workers) of a domain enterprise as functions. Each category of staff, i.e., each function, works in state and updates that state according to schedules and resource allocations — which are considered part of the state. To make the description simple we do not detail the state other than saying that each category works on an "instantaneous copy" of "the" state. Now think of six staff category activities, strategic managers, tactical managers, operational managers, supervisors, team leaders and workers as six simultaneous sets of actions. Each function defines a step of collective (i.e., group) (strategic, tactical, operational) management, supervisor, team leader and worker work. Each step is considered "atomic". Now think of an enterprise as the "repeated" step-wise simultaneous performance of these category activities. Six "next" states arise. These are, in the reality of the domain, ameliorated, that is reconciled into one state. however with the next iteration, i.e., step, of work having each category apply its work to a reconciled version of the state resulting from that category's previously yielded state and the mediated "global" state. Caveat: The below is not a mathematically proper definition. It suggests one!

```
type
0. Σ
value
1. str, tac, opr, sup, tea, wrk: Σ → Σ
2. stra, tact, oper, supr, team, work: Σ → (Σ×Σ×Σ×Σ×Σ) → Σ
3. objective: (Σ×Σ×Σ×Σ×Σ) → Bool
4. enterprise, ameliorate: (Σ×Σ×Σ×Σ×Σ×Σ) → Σ
5. enterprise(⟨σs,σt,σo,σu,σe,σw⟩) ≡
6.   let σ's = stra(str(σs))(σ't,σ'o,σ'u,σ'e,σ'w),
7.       σ't = tact(tac(σt))(σ's,σ'o,σ'u,σ'e,σ'w),
8.       σ'o = oper(opr(σo))(σ's,σ't,σ'u,σ'e,σ'w),
9.       σ'u = supr(sup(σu))(σ's,σ't,σ'o,σ'e,σ'w),
10.      σ'e = team(tea(σe))(σ's,σ't,σ'o,σ'u,σ'w),
11.      σ'w = work(wrk(σw))(σ's,σ't,σ'o,σ'u,σ'e) in
12.  if objective(σ's,σ't,σ'o,σ'u,σ'e,σ'w)
13.    then ameliorate(σ's,σ't,σ'o,σ'u,σ'e,σ'w)
14.    else enterprise(σ's,σ't,σ'o,σ'u,σ'e,σ'w) end end
```

0. Σ is a further undefined and unexplained enterprise state space. The various enterprise players view this state in their own way.
1. Six staff group operations, str, tac, opr, sup, tea and wrk, each act in the enterprise state such as conceived by respective groups to effect a resulting enterprise state such as achieved by respective groups.
2. Six staff group state amelioration functions, ame_s,ame _t, ame_o, ame_u, ame_e and ame_w, each apply to the resulting enterprise states such as achieved by respective groups to yield a result state such as achieved by that group.
3. An overall objective function tests whether a state summary reflects that the objectives of the enterprise has been achieved or not.
4. The enterprise function applies to the tuple of six group-biased (i.e., ameliorated) states. Initially these may all be the same state. The result is an ameliorated state.
5. An iteration, that is, a step of enterprise activities, lines 5.–13. proceeds as follows:
6. strategic management operates
   - in its state space, $\sigma_s : \Sigma$;
   - effects a next (un-ameliorated strategic management) state σ'_s;
   - and ameliorates this latter state in the context of all the other player's ameliorated result states.

7.–11. The same actions take place, simultaneously for the other players: tac, opr, sup, tea and wrk.

12. A test, *has* objectives *been met*, is made on the six ameliorated states.
13. If test is successful, then the enterprise terminates in an ameliorated state.
14. Otherwise the enterprise recurses, that is, "repeats" itself in new states.

The above "function" definition is suggestive. It suggests that a solution to the fix-point 6-tuple of equations over "intermediate" states, σ'_x, where x is any of s, t, o, u, e, w, is achieveable by iteration over just these 6 equations. •

7.2. Requirements

Top-level, including strategic management tends to not be amenable to "automation". Increasingly tactical management tends to "divide" time between "bush-fire, stop-gap" actions – hardly automatable and formulating, initiating and monitoring main operations. The initiation and monitoring of tactical actions appear amenable to partial automation. Operational management – with its reliance on rules & regulations, scripts and licenses – is where computer monitoring and partial control has reaped the richest harvests.

7.3. On Modeling Management and Organisation

Management and organisation basically spans entity, function, event and behaviour intensities and thus typically require the full spectrum of modeling techniques and notations — summarised in Sect. 2.3.

8. Human Behaviour

- *By* **domain human behaviour** *we shall understand 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* ⊙

8.1. Conceptual Analysis

To model human behaviour "smacks" like modeling human actors, the psychology of humans, etc.! We shall not attempt to model the psychological side of humans — for the simple reason that we neither know how to do that nor whether it can at all be done. Instead we shall be focusing on the effects on non-human manifest entities of human behaviour.

Example: 21. Banking — or Programming — Staff Behaviour: Let us assume a bank clerk, "in ye olde" days, when calculating, say mortgage repayments (cf. Example 10). We would characterise such a clerk as being *diligent,* etc., if that person carefully follows the mortgage calculation rules, and checks and double-checks that calculations "tally up", or lets others do so. We would characterise a clerk as being *sloppy* if that person occasionally forgets the checks alluded to above. We would characterise a clerk as being *delinquent* if that person systematically forgets these checks. And we would call such a person a *criminal* if that person intentionally miscalculates in such a way that the bank (and/or the mortgage client) is cheated out of funds which, instead, may be diverted to the cheater. Let us, instead of a bank clerk, assume a software programmer charged with implementing an automatic routine for effecting mortgage repayments (cf. Example 11). We would characterise the programmer as being *diligent* if that person carefully follows the mortgage calculation rules, and throughout the development verifies and tests that the calculations are correct with respect to the rules. We would characterise the programmer as being *sloppy* if that person forgets certain checks and tests when otherwise correcting the computing program under development. We would characterise the programmer as being *delinquent* if that person systematically forgets these checks and tests. And we would characterise the programmer as being a *criminal* if that person intentionally provides a program which miscalculates the mortgage interest, etc., in such a way that the bank (and/or the mortgage client) is cheated out of funds. •

Example: 22. A Human Behaviour Mortgage Calculation: Example 11 gave a semantics to the mortgage calculation request (i.e., command) as would a diligent bank clerk be expected to perform it. To express, that is, to model, how sloppy, delinquent, or outright criminal persons (staff?) could behave we must modify the int_Cmd(mkPM(c,a,m,p,d′))(ρ,α,μ,ℓ) definition.

```
int_Cmd(mkPM(c,a,m,p,d))(ρ,α,μ,ℓ) ≡
    let (b,d′) = ℓ(m) in
    if q(α(a),p) [ α(a)≤p∨α(a)=p∨α(a)≤p∨... ]
        then
            let i = f1(interest(mi,b,period(d,d′))),
                ℓ′ = ℓ † [ m↦f2(ℓ(m)−(p−i)) ],
                α′ = α † [ a↦f3(α(a)−p),ai↦f4(α(ai)+i),a"staff"↦f"staff"(α(a"staff")+i) ] in
            ((ρ,α′,μ,ℓ′),ok) end
        else
            ((ρ,α′,μ,ℓ),nok)
    end end
    pre c ∈ dom μ ∧ m ∈ μ(c)

q: P × P ~→ Bool
f1,f2,f3,f4,f"staff": P ~→ P  [ typically: f"staff" = λp.p ]
```

•

The predicate q and the functions f_1, f_2, f_3, f_4 and f"staff" of Example 22 are deliberately left undefined. They are being defined by the "staffer" when performing (incl., programming) the mortgage calculation routine. The point of Example 22 is that one must first define the mortgage calculation script precisely as one would like to see the diligent staff (programmer) to perform (incl., correctly program) it before one can "pinpoint" all the places where lack of diligence may "set in". The invocations of q, f_1, f_2, f_3, f_4 and f"staff" designate those places. The point of Example 22 is also that we must first domain-define, "to the best of our ability" all the places where human behaviour may play other than a desirable role. If we cannot, then we cannot claim that some requirements aim at countering undesirable human behaviour.

Commensurate with the above, humans interpret rules and regulations differently, and, for some humans, not always consistently — in the sense of repeatedly applying the same interpretations. Our final specification pattern is therefore:

```
type
    Action = Θ ~→ Θ-infset
value
    hum_int: Rule → Θ → RUL-infset
    action: Stimulus → Θ → Θ
```

hum_beha: Stimulus × Rules → Action → Θ $\xrightarrow{\sim}$ **Θ-infset**
hum_beha(sy_sti,sy_rul)(α)(θ) **as** θset
post
θset = α(θ) ∧ action(sy_sti)(θ) ∈ θset
∧ ∀ θ′:Θ•θ′ ∈ θset ⇒
∃ se_rul:RUL•se_rul ∈ hum_int(sy_rul)(θ)⇒se_rul(θ,θ′)

The above is, necessarily, sketchy: There is a possibly infinite variety of ways of interpreting some rules. A human, in carrying out an action, interprets applicable rules and chooses one which that person believes suits some (professional, sloppy, delinquent or criminal) intent. "Suits" means that it satisfies the intent, i.e., yields **true** on the pre/post-configuration pair, when the action is performed — whether as intended by the ones who issued the rules and regulations or not. We do not cover the case of whether an appropriate regulation is applied or not. The above-stated axioms express how it is in the domain, not how we would like it to be. For that we have to establish requirements.

8.2. Requirements

Requirements in relation to the human behaviour facet is not requirements about software that "replaces" human behaviour. Such requirements were hinted at in Sects. 5.2–7.2. Human behaviour facet requirements are about software that checks human behaviour; that its remains diligent; that it does not transgress into sloppy, delinquent, let alone criminal behaviour. When transgressions are discovered, appropriate remedial actions may be prescribed.

8.3. On Modeling Human Behaviour

To model human behaviour is, "initially", much like modeling management and organisation. But only 'initially'. The most significant human behaviour modeling aspect is then that of modeling non-determinism and looseness, even ambiguity. So a specification language which allows specifying non-determinism and looseness (like `CafeOBJ` [FNT00, FD98, DFO03] and `RSL` [GHH+92]) is to be preferred. To prescribe requirements is to prescribe the monitoring of the human input at the computer interface.

9. Conclusion

We have introduced the scientific and engineering concept of domain theories and domain engineering; and we have brought but a mere sample of the principles, techniques and tools that can be used in creating domain descriptions.

9.1. Completion

Domain acquisition results in typically up to thousands of units of domain descriptions. Domain analysis subsequently also serves to classify which facet any one of these description units primarily characterises. But some such "compartmentalisations" may be difficult, and may be deferred till the step of "completion". It may then be, "at the end of the day", that is, after all of the above facets have been modeled that some description units are left as not having been described, not deliberately, but "circumstantially". It then behooves the domain engineer to fit these "dangling" description units into suitable parts of the domain description. This "slotting in" may be simple, and all is fine. Or it may be difficult. Such difficulty may be a sign that the chosen model, the chosen description, in its selection of entities, functions, events and behaviours to model — in choosing these over other possible selections of phenomena and concepts is not appropriate. Another attempt must be made. Another selection, another abstraction of entities, functions, etc., may need be chosen. Usually however, after having chosen the abstractions of the intrinsic phenomena and concepts, one can start checking whether "dangling" description units can be fitted in "with ease".

9.2. Integrating Formal Descriptions

We have seen that to model the full spectrum of domain facets one needs not one, but several specification languages. No single specification language suffices. It seems highly unlikely and it appears not to be desirable to obtain a single, "universal" specification language capable of "equally" elegantly, suitably abstractly modeling all aspects of a domain. Hence one must conclude that the full modeling of domains shall deploy several formal notations – includig plain, good old mathematics in all its forms. The issues are then the following which combinations of notations to select, and how to make sure that the combined specification denotes something meaningful. The ongoing series of "Integrating Formal Methods" conferences [AGT99, GSS00, BPS02, BDS04, RSvdP05] is a good source for techniques, compositions and meanings.

9.3. The Impossibility of Describing Any Domain Completely

Domain descriptions are, by necessity, abstractions. One can never hope for any notion of complete domain descriptions. The situation is no better for domains such as we define them than for physics. Physicists strive to understand the manifest world around us – the world that was there before humans started creating "their domains". The physicists describe the physical world "in bits and pieces" such that large collections of these pieces "fit together", that is, are based on some commonly accepted laws and in some commonly agreed mathematics. Similarly for such domains as will be the subject of domain science & engineering such as we cover that subject in [Bjø16c, Bjø16b] and in the present and an upcoming paper ([Bjø16a, Bjø16d]). Individual such domain descriptions will be emphasising some clusters of facets, others will be emphasising other aspects.

9.4. Rôles for Domain Descriptions

We can distinguish between a spectrum of rôles for domain descriptions. Some of the issues brought forward below may have been touched upon in [Bjø16c, Bjø16b].

Alternative Domain Descriptions: It may very well be meaningful to avail oneself of a variety of domain models (i.e., descriptions) for any one domain, that is, for what we may consider basically one and the same domain. In control theory (a science) and automation (an engineering) we develop specific descriptions, usually on the form of a set of differential equations, for any one control problem. The basis for the control problem is typically the science of mechanics. This science has many renditions (i.e., interpretations). For the control problem, say that of keeping a missile carried by a train wagon, erect during train movement and/or windy conditions, one may then develop a "self-contained" description of the problem based on some mechanics theory presentation. Similarly for domains. One may refer to an existing domain description. But one may re-develop a textually "smaller" domain description for any one given, i.e., specific problem.

Domain Science: A domain description designates a domain theory. That is, a bundle of propositions, lemmas and theorems that are either rather explicit or can be proven from the description. So a domain description is the basis for a theory as well as for the discovery of domain laws, that is, for a domain science. We have sciences of physics (incl. chemistry), biology, etc. Perhaps it is about time to have proper sciences, to the extent one can have such sciences for human-made domains.

Business Process Re-engineering: Some domains manifest serious amounts of human actions and interactions. These may be found to not be efficient to a degree that one might so desire. A given domain description may therefore be a basis for suggesting other *management & organisation* structures, and/or *rules & regulations* than present ones. Yes, even making explicit *scripts* or a *license language* which have hitherto been tacitly understood – without necessarily computerising any support for such a *script* or *license language*. The given and the resulting domain descriptions may then be the basis for operations research models that may show desired or acceptable efficiency improvements.

Software Development: [Bjø16b] shows one approach to requirements prescription. Domain analysis & description, i.e., domain engineering, is here seen as an initial phase, with requirements prescription engineering being a second phase, and software design being a third phase. We see domain engineering as indispensable, that is, an absolute must, for software development. [Bjø11, *Domains: Their Simulation, Monitoring and Control*] further illustrates how domain engineering is a base for the development of domain simulators, demos, monitors and controllers.

9.5. Grand Challenges of Informatics[19]

To establish a reasonably trustworthy and believable theory of a domain, say the transportation, or just the railway domain, may take years, possibly 10–15 ! Similarly for domains such as the financial service industry, the market (of consumers and producers, retailers, wholesaler, distribution cum supply chain), health care, and so forth. The current author urges younger scientists to get going! It is about time.

9.6. Acknowledgements

The author kindly and with thanks acknowledges the referees of the present paper.

10. Bibliograpy

10.1. Bibliographical Notes

To create domain descriptions, or requirements prescriptions, or software designs, properly, at least such as this author sees it, is a joy to behold. The beauty of carefully selected and balanced abstractions, their interplay with other such, the relations between phases, stages and steps, and many more conceptual constructions make software engineering possibly the most challenging intellectual pursuit today. For this and more consult [Bjø06a, Bjø06b, Bjø06c].

10.2. References

[Abr09] Jean-Raymond Abrial. The B Book: Assigning Programs to Meanings *and* Modeling in Event-B: System and Software Engineering. Cambridge University Press, Cambridge, England, 1996 and 2009.

[AGT99] Keijiro Araki, Andy Galloway, and Kenji Taguchi, editors. *IFM'99: Integrated Formal Methods*, London. England, June 1999. Springer–Verlag. Proceedings of 1st Intl. Conf. on IFM.

[AH05] Alapan Arnab and Andrew Hutchison. Fairer Usage Contracts for DRM. In *Proceedings of the Fifth ACM Workshop on Digital Rights Management (DRM'05)*, pages 65–74, Alexandria, Virginia, USA, Nov 2005.

[All05] Open Mobile Alliance. OMA DRM V2.0 Candidate Enabler. http://www.openmobilealliance.org/-release_program/drm_v2_0.html, Sep 2005.

[Apt03] Krzysztof R. Apt. *Principles of Constraint Programming*. Cambridge University Press, August 2003. ISBN 0521825830.

[Bak95] Jaco W. de Bakker. *Control Flow Semantics*. The MIT Press, Cambridge, Mass., USA, 1995.

[BDS04] 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.

[Ben02] Yochai Benkler. Coase's Penguin, or Linux and the Nature of the Firm. *The Yale Law Journal*, 112, 2002.

[BJ78] Dines Bjørner and Cliff B. Jones, editors. *The Vienna Development Method: The Meta-Language*, volume 61 of *LNCS*. Springer, 1978. This was the first monograph on *Meta-IV*.

[BJ82] Dines Bjørner and Cliff B. Jones, editors. *Formal Specification and Software Development*. Prentice-Hall, 1982.

[Bjø06a] Dines Bjørner. *Software Engineering, Vol. 1: Abstraction and Modelling*. Texts in Theoretical Computer Science, the EATCS Series. Springer, 2006. .

[Bjø06b] 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.

[Bjø06c] Dines Bjørner. *Software Engineering, Vol. 3: Domains, Requirements and Software Design*. Texts in Theoretical Computer Science, the EATCS Series. Springer, 2006.

[Bjø10] Dines Bjørner. Domain Engineering. In Paul Boca and Jonathan Bowen, editors, *Formal Methods: State of the Art and New Directions*, Eds. Paul Boca and Jonathan Bowen, pages 1–42, London, UK, 2010. Springer.

[Bjø11] Dines Bjørner. Domains: Their Simulation, Monitoring and Control – A Divertimento of Ideas and Suggestions. In *Rainbow of Computer Science, Festschrift for Hermann Maurer on the Occasion of His 70th Anniversary.*, Festschrift (eds. C. Calude, G. Rozenberg and A. Saloma), pages 167–183. Springer, Heidelberg, Germany, January 2011.

[Bjø16a] Dines Bjørner. Domain Facets: Analysis & Description. *Expected published by Formal Aspects of Computing*, 2016. http://www.imm.dtu.dk/~dibj/2016/facets/faoc-facets.pdf.

[19] In the early-to-mid 2000s there were a rush of research foundations and scientists enumerating *"Grand Challenges of Informatics"*

[Bjø16b] Dines Bjørner. From Domain Descriptions to Requirements Prescriptions – A Different Approach to Requirements Engineering. *Submitted for consideration by Formal Aspects of Computing*, 2016.

[Bjø16c] Dines Bjørner. Manifest Domains: Analysis & Description. *Expected published by Formal Aspects of Computing*, 2016.

[Bjø16d] Dines Bjørner. Manifest Domains: Analysis & Description – A Philosophical Basis. *Eventually to be submitted to Formal Aspects of Computing*, Incomplete draft 2016. http://www.imm.dtu.dk/~dibj/2016/apb/daad-apb.pdf.

[BPS02] 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.

[CCD+06] C. N. Chong, R. J. Corin, J. M. Doumen, S. Etalle, P. H. Hartel, Y. W. Law, and A. Tokmakoff. LicenseScript: a logical language for digital rights management. *Annals of telecommunications special issue on Information systems security*, 2006.

[CCE03] Cheun Ngen Chong, Ricardo Corin, and Sandro Etalle. LicenseScript: A novel digital rights languages and its semantics. In *Proc. of the Third International Conference WEB Delivering of Music (WEDELMUSIC'03)*, pages 122–129. IEEE Computer Society Press, 2003.

[C.E02] C.E.C. Digital Rights: Background, Systems, Assessment. Commission of The European Communities, Staff Working Paper, 2002. Brussels, 14.02.2002, SEC(2002) 197.

[CEH03] C. N. Chong, S. Etalle, and P. H. Hartel. Comparing Logic-based and XML-based Rights Expression Languages. In *Confederated Int. Workshops: On The Move to Meaningful Internet Systems (OTM)*, number 2889 in LNCS, pages 779–792, Catania, Sicily, Italy, 2003. Springer.

[CGN+] David R. Christiansen, Klaus Grue, Henning Niss, Peter Sestoft, and Kristján S. Sigtryggsson. Actulus Modeling Language - An actuarial programming language for life insurance and pensions. Technical Report, http://www.edlund.dk/sites/default/files/Downloads/paper_actulus-modeling-language.pdf, Edlund A/S, Denmark, Bjerregårds Sidevej 4, DK-2500 Valby. (+45) 36 15 06 30. edlund@edlund.dk, http://www.edlund.dk/en/insights/scientific-papers. This paper illustrates how the design of pension and life insurance products, and their administration, reserve calculations, and audit, can be based on a common formal notation. The notation is human-readable and machine-processable, and specialised to the actuarial domain, achieving great expressive power combined with ease of use and safety.

[DFO03] Ražvan Diaconescu, Kokichi Futatsugi, and Kazuhiro Ogata. CafeOBJ: Logical Foundations and Methodology. *Computing and Informatics*, 22(1–2), 2003.

[DH01] 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.

[FD98] Kokichi Futatsugi and Razvan Diaconescu. *CafeOBJ Report The Language, Proof Techniques, and Methodologies for Object-Oriented Algebraic Specification*. AMAST Series in Computing – Vol. 6. World Scientific Publishing Co. Pte. Ltd., 1998.

[FL97] John Fitzgerald and Peter Gorm Larsen. *Developing Software Using VDM-SL*. Cambridge University Press, Cambridge, UK, 1997.

[FNT00] K. Futatsugi, A.T. Nakagawa, and T. Tamai, editors. *CAFE: An Industrial–Strength Algebraic Formal Method*, Sara Burgerhartstraat 25, P.O. Box 211, NL–1000 AE Amsterdam, The Netherlands, 2000. Elsevier. Proceedings from an April 1998 Symposium, Numazu, Japan.

[GHH+92] 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.

[GSS00] 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.

[Gun92] C.A. Gunther. *Semantics of Programming Languages*. The MIT Press, Cambridge, Mass., USA, 1992.

[GWW01] Carl A. Gunter, Stephen T. Weeks, and Andrew K. Wright. Models and Languages for Digtial Rights. In *Proc. of the 34th Annual Hawaii International Conference on System Sciences (HICSS-34)*, pages 4034–4038, Maui, Hawaii, USA, January 2001. IEEE Computer Society Press.

[Har87] David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, 1987.

[Har88] David Harel. On visual formalisms. *Communications of the ACM*, 33(5), 514–530 1988.

[HG97] David Harel and Eran Gery. Executable object modeling with Statecharts. *IEEE Computer*, 30(7):31–42, 1997.

[HLN+90] 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.

[HM03] David Harel and Rami Marelly. *Come, Let's Play – Scenario-Based Programming Using LSCs and the Play-Engine*. Springer-Verlag, 2003.

[HN96] David Harel and Amnon Naamad. The STATEMATE semantics of Statecharts. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 5(4):293–333, 1996.

[Hoa85] Tony Hoare. *Communicating Sequential Processes*. C.A.R. Hoare Series in Computer Science. Prentice-Hall International, 1985.

[Hoa04] C.A.R. Hoare. Communicating Sequential Processes. Published electronically: `http://www.usingcsp.com/-cspbook.pdf`, 2004. Second edition of [Hoa85]. See also `http://www.usingcsp.com/`.

[HPK11] A.E. Haxthausen, J. Peleska, and S. Kinder. A formal approach for the construction and and verification of railway control systems. *Formal Aspects of Computing*, 23:191–219, 2011.
[HRB03] Martin C. Henson, Steve Reeves, and Jonathan P. Bowen. Z Logic and its Consequences. *Computing and Informatics*, 22(1–2), 2003.
[HW04] Joseph Y. Halpern and Vicky Weissman. A Formal Foundation for XrML. In *Proc. of the 17th IEEE Computer Security Foundations Workshop (CSFW'04)*, 2004.
[Inc00] ContentGuard Inc. XrML: Extensible rights Markup Language. http://www.xrml.org, 2000.
[IT92] ITU-T. CCITT Recommendation Z.120: Message Sequence Chart (MSC), 1992.
[IT96] ITU-T. ITU-T Recommendation Z.120: Message Sequence Chart (MSC), 1996.
[IT99] ITU-T. ITU-T Recommendation Z.120: Message Sequence Chart (MSC), 1999.
[Jac06] Daniel Jackson. *Software Abstractions: Logic, Language, and Analysis*. The MIT Press, Cambridge, Mass., USA, April 2006. ISBN 0-262-10114-9.
[Jen97] 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.
[JHJ07] Cliff B. Jones, Ian Hayes, and Michael A. Jackson. Deriving Specfications for Systems That Are Connected to the Physical World. In Cliff Jones, Zhiming Liu, and James Woodcock, editors, *Formal Methods and Hybrid Real-Time Systems: Essays in Honour of Dines Bjørner and Zhou Chaochen on the Occasion of Their 70th Birthdays*, volume 4700 of *Lecture Notes in Computer Science*, pages 364–390. Springer, 2007.
[KLMM04] R.H. Koenen, J. Lacy, M. Mackay, and S. Mitchell. The long march to interoperable digital rights management. *Proceedings of the IEEE*, 92(6):883–897, June 2004.
[KW01] 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.
[Lam95] Leslie Lamport. The Temporal Logic of Actions. *Transactions on Programming Languages and Systems*, 16(3):872–923, 1995.
[Lam02] Leslie Lamport. *Specifying Systems*. Addison–Wesley, Boston, Mass., USA, 2002.
[Ltd01] IPR Systems Pty Ltd. Open Digital Rights Language (ODRL). http://odrl.net, 2001.
[Lyo02] Gordon E. Lyon. Information Technology: A Quick-Reference List of Organizations and Standards for Digital Rights Management. NIST Special Publication 500-241, National Institute of Standards and Technology, Technology Administration, U.S. Department of Commerce, Oct 2002.
[Mai97] Tom Maibaum. Conservative Extensions, Interpretations Between Theories and All That. In Michel Bidoit and Max Dauchet, editors, *TAPSOFT'97: Theory and Practice of Software Development*, volume 1214 of *LNCS*, pages 40–66, 1997.
[MB02] D. Mulligan and A. Burstein. Implementing copyright limitations in rights expression languages. In *Proc. of 2002 ACM Workshop on Digital Rights Management*, volume 2696 of *Lecture Notes in Computer Science*, pages 137–154. Springer-Verlag, 2002.
[Mer03] Stephan Merz. On the Logic of TLA+. *Computing and Informatics*, 22(1–2), 2003.
[Mer04] Merriam Webster Staff. Online Dictionary: `http://www.m-w.com/home.htm`, 2004. Merriam–Webster, Inc., 47 Federal Street, P.O. Box 281, Springfield, MA 01102, USA.
[MHB03] Deirdre K. Mulligan, John Han, and Aaron J. Burstein. How DRM-Based Content Delivery Systems Disrupt Expectations of "Personal Use". In *Proc. of The 3rd International Workshop on Digital Rights Management*, pages 77–89, Washington DC, USA, Oct 2003. ACM.
[MVJD05] S. Michiels, K. Verslype, W. Joosen, and B. De Decker. Towards a Software Architecture for DRM. In *Proceedings of the Fifth ACM Workshop on Digital Rights Management (DRM'05)*, pages 65–74, Alexandria, Virginia, USA, Nov 2005.
[OD08] Ernst-Rüdiger Olderog and Henning Dierks. *Real-Time Systems: Formal Specification and Automatic Verification*. Cambridge University Press, UK, 2008.
[ORW16] Ernst Rüdiger Olderog, Anders Peter Ravn, and Rafael Wisniewski. Linking Discrete and Continuous Models, Applied to Traffic Maneuvers. In Jonathan Bowen, Michael Hinchey, and Ernst Rüdiger Olderog, editors, *BCS FACS – ProCoS Workshop on Provably Correct Systems*, Lecture Notes in Computer Science. Springer, 2016.
[Pet62] Carl Adam Petri. *Kommunikation mit Automaten*. Bonn: Institut für Instrumentelle Mathematik, Schriften des IIM Nr. 2, 1962.
[PW02] Riccardo Pucella and Vicky Weissman. A Logic for Reasoning about Digital Rights. In *Proc. of the 15th IEEE Computer Security Foundations Workshop (CSFW'02)*, pages 282–294. IEEE Computer Society Press, 2002.
[PW04] Riccardo Pucella and Vicky Weissman. A Formal Foundation for ODRL. In *Proc. of the Workshop on Issues in the Theory of Security (WIST'04)*, 2004.
[Rei85] Wolfgang Reisig. *Petri Nets: An Introduction*, volume 4 of *EATCS Monographs in Theoretical Computer Science*. Springer Verlag, May 1985.
[Rei92] Wolfang Reisig. *A Primer in Petri Net Design*. Springer Verlag, March 1992. 120 pages.
[Rei98] Wolfgang Reisig. *Elements of Distributed Algorithms: Modelling and Analysis with Petri Nets*. Springer Verlag, December 1998. xi + 302 pages.
[Rey99] John C. Reynolds. *The Semantics of Programming Languages*. Cambridge University Press, 1999.
[Ros97] 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.
[RSvdP05] 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.

[Sam03] Pamela Samuelson. Digital rights management {and, or, vs.} the law. *Communications of ACM*, 46(4):41–45, Apr 2003.

[Sch86] David A. Schmidt. *Denotational Semantics: a Methodology for Language Development*. Allyn & Bacon, 1986.

[Sch00] 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.

[Spi88] 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.

[Spi92] J. M. Spivey. *The Z Notation: A Reference Manual*. Prentice Hall International Series in Computer Science, 2nd edition, 1992.

[Ten97] Robert Tennent. *The Semantics of Programming Languages*. Prentice–Hall Intl., 1997.

[VVD90] F. Van der Rhee, H.R. Van Nauta Lemke, and J.G. Dukman. Knowledge based fuzzy control of systems. *IEEE Trans. Autom. Control*, 35(2):148–155, February 1990.

[WD96] J. C. P. Woodcock and J. Davies. *Using Z: Specification, Proof and Refinement*. Prentice Hall International Series in Computer Science, 1996.

[Win93] G. Winskel. *The Formal Semantics of Programming Languages*. The MIT Press, Cambridge, Mass., USA, 1993.

[XB06] JianWen Xiang and Dines Bjørner. The Electronic Media Industry: A Domain Analysis and a License Language. Technical note, JAIST, School of Information Science, 1-1, Asahidai, Tatsunokuchi, Nomi, Ishikawa, Japan 923-1292, Summer 2006.

[ZH04] 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.

[ZHR92] Chao Chen Zhou, C.A.R. Hoare, and Anders P. Ravn. A Calculus of Durations. *Information Proc. Letters*, 40(5), 1992.