11 Domain Facets

- The **prerequisite** for studying this chapter is that you, as a domain engineer, need to know: *which are the constituents of a proper model of a domain?*
- The **aims** are to introduce the concept that a proper domain description is made up from most of the following constituent descriptions, i.e., facets: domain-facilitating business processes, domain intrinsics, domain support technologies, domain management and organisation, domain rules and regulations, domain scripts, human behaviour, etc., and to present principles, techniques and tools for the description of these facets.
- The **objective** is to ensure that you will become a thoroughly professional domain engineer.
- The **treatment** is from systematic to formal.

11.1 Introduction

Let us remind ourselves of what it is all about. Software development is all about getting software to the market, software that can and will be sold. Hence it must be software whose use pleases people, software which solves problems, that is, software which fits, hand in glove, the application domain in which it is to serve.

Therefore describing the domain is important. If we cannot describe the domain, then we are not trustworthy. We simply cannot be trusted to develop software for that domain. Describing the domain is thus of utmost importance. And hence it is of primary importance to know and to practice what a description consists of.

This chapter is all about that: to identify the various facets of a domain that are describable, and, hence, most likely, are parts of a proper domain description. So, in this chapter we will identify those facets, and we will present principles, techniques and tools for their proper description.

The present chapter constitutes a first high point of the present volume, because in this chapter we present principles and techniques of software development that are not otherwise available in any other textbook on software engineering. So take your time to become thoroughly familiar with the contents of the present chapter.

Characterisation. By a *domain facet* we 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. ▪

In this section we identify a number of *domain facets* and we survey principles and techniques for modelling, relative to identified domain stakeholder classes, each of the identified facets. So far we have been able to identify the following facets:

- (i) *intrinsics*,
- (ii) *support technology*,
- (iii) *management and organisation*,
- (iv) *rules and regulations* including
- (v) *script* facets, and
- (vi) *human behaviour*.

We enlarge upon the above enumeration using the following brief characterisations:

- (i) **Domain intrinsics:** *That which is common to all facets* (Sect. 11.3).
- (ii) **Domain support technologies:** *That in terms of which several other facets (intrinsics, business processes, management and organisation, and rules and regulations) are implemented* (Sect. 11.4).
- (iii) **Domain management and organisation:** *That which primarily determines and constrains communication between enterprise stakeholders* (Sect. 11.5).
- (iv–v) **Domain rules, regulations and scripts:** *That which guides the work of enterprise stakeholders, their interaction, and the interaction with non-enterprise stakeholders* (Sects. 11.6–11.7).
- (vi) **Domain human behaviour:** *The way in which domain stakeholders dispatch their actions and interactions wrt. the enterprise: dutifully, forgetfully, sloppily, yes, even criminally* (Sect. 11.8).

To help us identify parts of the above facets we suggest that rough sketch descriptions first be made of what we shall call the domain business process facilitators:

- **Domain business process facilitators:** *Those processes — carried out primarily by people — in terms of which the intrinsics (and so on) are implemented* (Sect. 11.2).

11.1.1 Separation of Concerns

We shall now treat each of these facets in some detail. For each we venture to express some specification pattern that most closely captures the essence of the facet. Separating the treatment of each of these (and possibly other) facets reflects the following principle:

Principle. *Separation of Facets:* When possible, one should identify distinguishable facets and, when appropriate, i.e., if feasible and pleasing, treat them separately. ▪

We believe that the facets we shall present can be treated separately in most developments — but not necessarily always. *Separation or not* is a matter of development as well as of presentation style.

11.1.2 Discussion of the Separation Principle

The separation, in more generality, of computing systems development into the triptych of domain engineering, requirements engineering and machine (hardware + software) design is also a result of separation of concerns. So are the separations of domain requirements, interface requirements and machine requirements (within requirements engineering), as well as the separation of software architecture and component and module design.

11.1.3 Structure of Chapter

Before we cover each of the facets individually (Sects. 11.3–11.8) we cover the concept of business process facilitators (Sect. 11.2). The material of Sect. 11.2, in addition to helping the domain describer to identify the various facets of a domain, also covers the important notion of business processes. Describing business processes is not only the responsibility of a software developer, but also of managers in any business enterprise. Before having, even superficially, understood current business processes how could a business manager mandate the reengineering of these processes? Section 11.2 therefore also serves as a prerequisite for the section on business process reengineering (a domain requirements facet, Sect. 19.3).

11.2 Domain Facilitators: Business Processes

A domain is often known to its stakeholders by the various actions they play in that domain. That is, the domain is known by the various sequences of entities, functions and events the stakeholders are exposed to, are performing and are influenced by. Such sequences are what we shall here understand as business processes.

In our ongoing example, that of railway systems, informal examples of business processes are: for a potential passenger to plan, buy tickets for, and undergo a journey. For the driver of the locomotive the sequence of undergoing a briefing of the train journey plan, taking possession of the train, checking some basic properties of that train, negotiating its start, driving it down the line, obeying signals and the plan, and, finally entering the next station, stopping at a platform, and concluding a trip of the train journey — all that constitutes a business process. For a train dispatcher, the monitoring and control of trains and signals during a work shift constitutes a business process.

Describing domain intrinsics focuses on the very essentials of a domain. It can sometimes be a bit hard for a domain engineer, in collaboration with stakeholders, to decide which are the domain intrinsics. It can often help (the process of identifying the domain intrinsics) if one alternatively, or hand in hand analyses and describes what is known as the business processes. From a description of business processes one can then analyse which parts of such a description designate, i.e., are about or relate to, which facets.

Principle. *Describing Domain Business Process Facets:* As part of understanding any (at least human-made) domain it is important to delineate and describe its business processes. Initially that should preferably be done in the form of rough sketches. These rough sketches should — again initially — focus on identifiable entities, functions, events and behaviours. Naturally, being business processes, identification of behaviours comes first. Then be prepared to rework these descriptions as other facets are being described in depth. ▪

11.2.1 Business Processes

Characterisation. By a *business process* we understand the procedurally describable aspects, of one or more of the ways in which a business, an enterprise, a factory, etc., conducts its yearly, quarterly, monthly, weekly and daily processes, that is, regularly occurring chores. The processes may include strategic, tactical and operational management and workflow planning and decision activities; and the administrative, and where applicable, the marketing, the research and development, the production planning and execution, the sales and the service (workflow) activities — to name some. ▪

Example 11.1 *Some Business Processes:*

(i) A Business Plan Business Process: The board of any company instructs its chief executive officer (CEO) to formulate revised business plans.[1] Briefly, a business plan is a plan for how the company strategically, tactically and, to some extent, operationally wishes to conduct its business: what it strives for,

[1] A business plan is not the same as a description of the business's processes.

productwise, imagewise, market-share-wise, financially, etc. The CEO develops a business plan in consultation with executive layers of (i.e., with strategic) management. Strategic management (in-between) discusses the plan (which the CEO wishes to submit to the Board) with tactical management, etc. Once generally agreed upon, the CEO submits the plan to the Board.

(ii) A Purchase Regulation Business Process: In our "example company", purchase of equipment must adhere to the following — roughly sketched — process: Once the need for acquisition of one or more units of a certain equipment, or a related set of equipment, has been identified, the staff most relevant to take responsibility for the use of this equipment issues a purchase inquiry request. The purchase inquiry request is sent to the purchasing department. The purchasing department investigates the market and reports back to the person who issued the request with a purchase inquiry report containing facts about zero, one or more possible equipment choices, their prices, and their purchase (i.e., payment), delivery, service and guarantee conditions. The person who issued the purchase inquiry request may now proceed to issue a purchase request order, attach the purchase inquiry report and send this to the relevant budget controlling manager for acceptance. If purchase is approved then the purchasing department is instructed to issue, to the chosen supplier, a purchase request order. Once the supplier delivers the ordered equipment, the purchasing department inspects the delivery and issues an equipment inspection report. An invoice from the supplier for the above-mentioned equipment is only paid if the equipment inspection report recommends to do so. Otherwise the delivered equipment is returned to the supplier. The above is but a rough sketch. Much more precision is needed, as are descriptions of exceptions, etc. ■

Example 11.2 *Some More Business Processes:* The University of California at Irvine (UCI), had their Administrative and Business Services department suggest, as a learning example, the description of a number of business processes. The "learning" had to do, actually, with business process reengineering (BPR). So we really should bring the below example into Sect. 19.3 instead of here! We quote from their home page [357]:

- **Human Resources:** "Examine the hiring business process of the University, including the applicant process. Special emphasis should be given to simplifying the process, identifying those parts where there is no value added — i.e., where those parts of the process which one considers *simplifying "away"* add no value. Increase speed of response to applicant and units, and reduce process costs while achieving high quality."
- **Renovation:** "Review the campus' remodelling and alterations business process, and develop recommendations to improve Facilities Management services to UCI departments for small projects (under $50,000) and minor capital projects (up to $250,000). Special emphasis should be given to simplifying the process, identifying those parts where there is no value added

to the customer's product; to increase speed and flexibility of response; and to reduce process costs while achieving high quality."

- **Procurement:** "Review the campus procurement business process and develop recommendations/solutions for process improvement. The redesigned process should provide "hassle-free" purchasing, give a quick response time to the purchaser, be economical in terms of all costs, be reasonably error-free and be compliant with (US) Federal procurement standards."
- **Travel:** "Study the travel business process from the beginning stage when a faculty/staff member identifies the need to travel to the time when reimbursement is received. Analyze and redesign the process through a six step program based on the following business process improvement (BPI) principles: (i) simplify the process, (ii) identify those parts where there is no value added to the customer, increase (iii) speed and (iv) flexibility of response, (v) improve clarity for responsibilities and (vi) reduce process costs while meeting customer expectations from travel services. The redesign should reflect customer needs, service, economy of operation and be in compliance with applicable regulations."
- **Accounts payable:** "Redesign the accounts payable business process to meet the following functional objectives (in addition to BPI measures): Payment for goods and services must assure that vendors receive remittance in a timely manner for all goods and services provided to the University. Significantly improve the operation's ability to serve campus customers while maintaining financial solvency and adequate internal controls."
- **Parking:** "Review how parking permits[2] are sold to students, faculty and staff with the intent of omitting unnecessary steps and redundant data collection. The redesigned process should achieve a dramatic reduction in time spent by people standing in line to purchase a permit, and reduce administrative time (and cost) in recording and tracking permit sales."

Please observe that the above examples illustrate requests for possible business process reengineering — but that they also give rough-sketch glimpses of underlying business processes. ■

Characterisation. By *business process engineering* we understand the identification of which business processes should be subject to precise description, describing these and securing their general adoption (acceptance) in the business, and enacting these business process descriptions. ■

[2] We here assume that the company is a very large company with extensive, but still limited, parking facilities.

Example 11.3 *Example Business Process Engineering:*

(i) *Business plans:* We assume, about our example company, that — up to a certain time — there was no set procedure wrt. the creation, etc., of business plans. As the company grows, a need is felt for "stricter" procedures wrt. business plans. Therefore the CEO and/or the board drafts the business plan very implicitly hinted at in Example 11.1 (i). The last two sentences, above, portray an example business process engineering.

(ii) *Purchase regulations:* We assume, about our example company, that — up to a certain time — there was no set procedure wrt. purchase of equipment. As the company grows, a need is felt for "stricter" procedures wrt. procurement. Therefore some (say, operations) manager drafts the purchase process roughly sketched in Example 11.1 item (ii). The previous two sentences portray an example business process engineering. ■

11.2.2 Overall Principles

We summarise:

Principles. Human-made universes of discourse[3] entail the concept of business processes. The principle of *business processes* states that the description of business processes is indispensable in any description of a human-made universe of discourse. The principle of *business processes* also states that describing these is not sufficient: all facets must be described. ■

Techniques. *Business Processes:* The basic technique of describing a human-made universe of discourse involves: (i) identification and description of a suitably comprehensive set of *behaviours:* the behaviours of interest and the environment; (ii) identification and description, for each behaviour, of the *entities* characteristic of this behaviour; (iii) identification and description, for each entity, of the *functions* that apply to entities, or from which entities are yielded; (iv) identification and description, for each behaviour, of the *events* that it shares — either with other specifically identified behaviours of interest, or with a further, abstract, environment. ■

Tools. *Business Processes:* Further techniques and the basic tools for describing business processes include: (1) `RSL/CSP` definition of processes, where one suitably defines their *in*put/*out*put signatures, associated *channel* names and *type*s, and their process definition bodies;[4] (2) Petri nets;[5] (3) message

[3] Examples of human-made universes of discourse are: public administration, manufacturing industries (mechanical, chemical, medical, woodworking, etc.), transportation, the financial service industry (banks, insurance companies, securities instrument brokers, traders and exchanges, portfolio management, etc.), agriculture, fisheries, mining, etc.

[4] `RSL/CSP` [168, 301, 311] was covered in detail in Vol. 1, Chap. 21.

[5] Petri Nets [196, 273, 293–295] were covered in detail in Vol. 2, Chap. 12.

and live sequence charts for the definition of interaction between behaviours;[6] (4) statecharts for the definition of highly complex, typically interwoven behaviours;[7] and (5) the usual, full complement of RSL's *type*, function *value*, and *axiom* constructs and their abstract techniques for modelling entities and functions. ∎

11.2.3 Informal and Formal Examples

We rough-sketch a number of examples. In each example we start, according to the principles and techniques enunciated above, with identifying behaviours, events, and hence channels and the type of entities communicated over channels, i.e. participating in events. Hence we shall emphasise, in these examples, the behaviour, or process diagrams. We leave it to other examples to present other aspects, so that their totality yields the principles, the techniques and the tools of domain description.

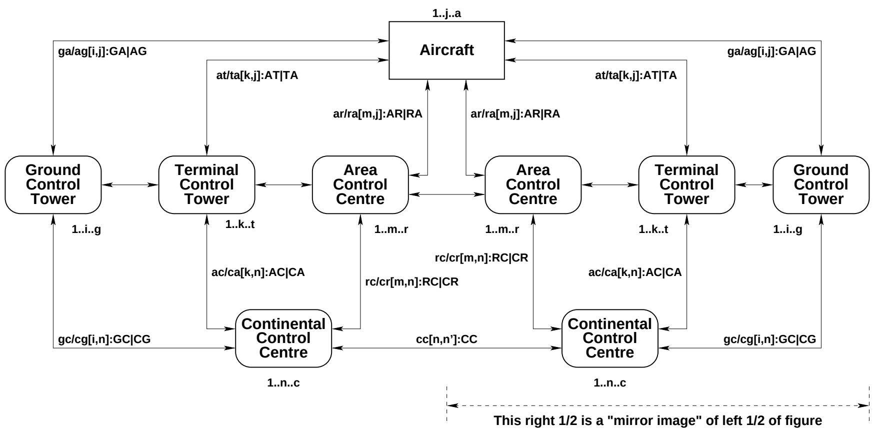

Fig. 11.1. An air traffic behavioural system abstraction

Example 11.4 *Air Traffic Business Processes:* The main business process behaviours of an air traffic system are the following: (i) the aircraft, (ii) the ground control towers, (iii) the terminal control towers, (iv) the area control centres and (v) the continental control centres (Fig. 11.1).

We describe each of these behaviours separately:

[6] Message [182–184] and live sequence charts [73,149,203] were covered in detail in Vol. 2, Chap. 13.

[7] Statecharts [144,145,147,148,150] were covered in detail in Vol. 2, Chap. 14.

(i) *Aircraft* get permission from ground control towers to depart; proceed to fly according to a flight plan (an entity); keep in contact with area control centres along the route, (upon approach) contacting terminal control towers from which they, simplifying, get permission to land; and upon touchdown, changing over from terminal control tower to ground control tower guidance.

(ii) The ground control towers, on one hand, take over monitoring and control of landing aircraft from terminal control towers; and, on the other hand, hand over monitoring and control of departing aircraft to area control centres. Ground control towers, on behalf of a requesting aircraft, negotiate with destination ground control tower and (simplifying) with continental control centres when a departing aircraft can actually start in order to satisfy certain “slot” rules and regulations (as one business process). Ground control towers, on behalf of the associated airport, assign gates to landing aircraft, and guide them from the spot of touchdown to that gate, etc. (as another business process).

(iii) The terminal control towers play their major role in handling aircraft approaching airports with intention to land. They may direct these to temporarily wait in a holding area. They — eventually — guide the aircraft down, usually “stringing” them into an ordered landing queue. In doing this the terminal control towers take over the monitoring and control of landing aircraft from regional control centres, and pass their monitoring and control on to the ground control towers.

(iv) The area control centres handle aircraft flying over their territory: taking over their monitoring and control either from ground control towers, or from neighbouring area control centres. Area control centres shall help ensure smooth flight, that aircraft are allotted to appropriate air corridors, if and when needed (as one business process), and are otherwise kept informed of “neighbouring” aircraft and weather conditions en route (other business processes). Area control centres hand over aircraft either to terminal control towers (as yet another business process), or to neighbouring area control centres (as yet another business process).

(v) The continental control centres monitor and control, in collaboration with regional and ground control centres, overall traffic in an area comprising several regional control centres (as a major business process), and can thus monitor and control whether contracted (landing) slot allocations and schedules can be honoured, and, if not, reschedule these (landing) slots (as another major business process).

From the above rough sketches of behaviours the domain engineer then goes on to describe types of messages (i.e., entities) between behaviours, types of entities specific to the behaviours, and the functions that apply to or yield those entities. ▪

Example 11.5 *Freight Logistics Business Processes:* The main business process behaviours of a freight logistics system are the following: (i) the senders of freight, (ii) the logistics firms which plan and coordinate freight transport, (iii) the transport companies on whose conveyors freight is being transported, (iv) the hubs between which freight conveyors "ply their trade", (v) the conveyors themselves and (vi) the receivers of freight (Fig. 11.2). A detailed description for each of the freight logistics business process behaviours listed above should now follow. We leave this as an exercise to the reader to complete. ∎

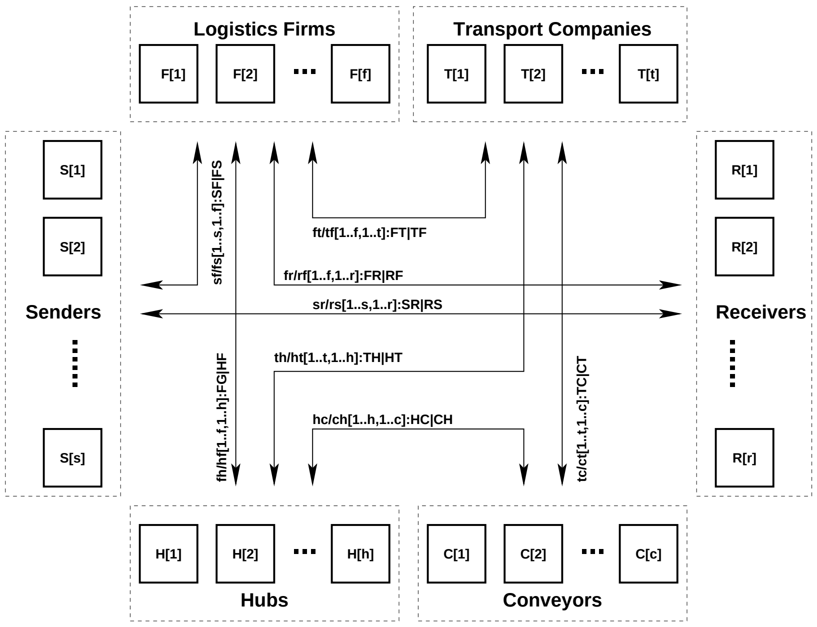

Fig. 11.2. A freight logistics behavioural system abstraction

Example 11.6 *Harbour Business Processes:* The main business process behaviours of a harbour system are the following: (i) the ships who seek harbour to unload and load cargo at a harbour quay, (ii) the harbourmaster who allocates and schedules ships to quays, (iii) the quays at which ships berth and unload and load cargo (to and from a container area) and (iv) the container area which temporarily stores ("houses") containers (Fig. 11.3). There may be other parts of a harbour: a holding area for ships to wait before being allowed to properly enter the harbour and be berthed at a buoy or a quay, or for ships to rest before proceeding; as well as buoys at which ships may be anchored while unloading and loading. We shall assume that the reader can properly complete an appropriate, realistic harbour domain.

A detailed description for each of the harbour business process behaviours listed above should now follow. We leave this as an exercise to the reader to complete. ∎

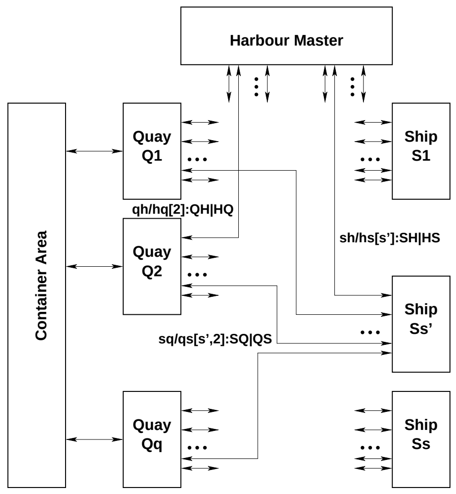

Fig. 11.3. A harbour behavioural system abstraction

Example 11.7 *Financial Service Industry Business Processes:* The main business process behaviours of a financial service system are the following: (i) clients, (ii) banks, (iii) securities instrument brokers and traders, (iv) portfolio managers, (v) (the, or a, or several) stock exchange(s), (vi) stock incorporated enterprises and (vii) the financial service industry "watchdog". We rough-sketch the behaviour of a number of business processes of the financial service industry.

(i) Clients engage in a number of business processes: (i.1) they open, deposit into, withdraw from, obtain statements about, transfer sums between and close demand/deposit, mortgage and other accounts; (i.2) they request brokers to buy or sell, or to withdraw buy/sell orders for securities instruments (bonds, stocks, futures, etc.); and (i.3) they arrange with portfolio managers to look after their bank and securities instrument assets, and occasionally they reinstruct portfolio managers in those respects.

(ii) Banks engage with clients, portfolio managers, and brokers and traders in exchanges related to client transactions with banks, portfolio managers, and brokers and traders, as well as with these on their own behalf, as clients.

(iii) Securities instrument brokers and traders engage with clients, portfolio managers and the stock exchange(s) in exchanges related to client transactions with brokers and traders, and, for traders, as well as with the stock exchange(s) on their own behalf, as clients.

(iv) Portfolio managers engage with clients, banks, and brokers and traders in exchanges related to client portfolios.

(v) Stock exchanges engage with the financial service industry watchdog, with brokers and traders, and with the stock listed enterprises, reinforcing trading practices, possibly suspending trading of stocks of enterprises, etc.

(vi) Stock incorporated enterprises engage with the stock exchange: They send reports, according to law, of possible major acquisitions, business developments, and quarterly and annual stockholder and other reports.

(vii) The financial industry watchdog engages with banks, portfolio managers, brokers and traders and with the stock exchanges. ∎

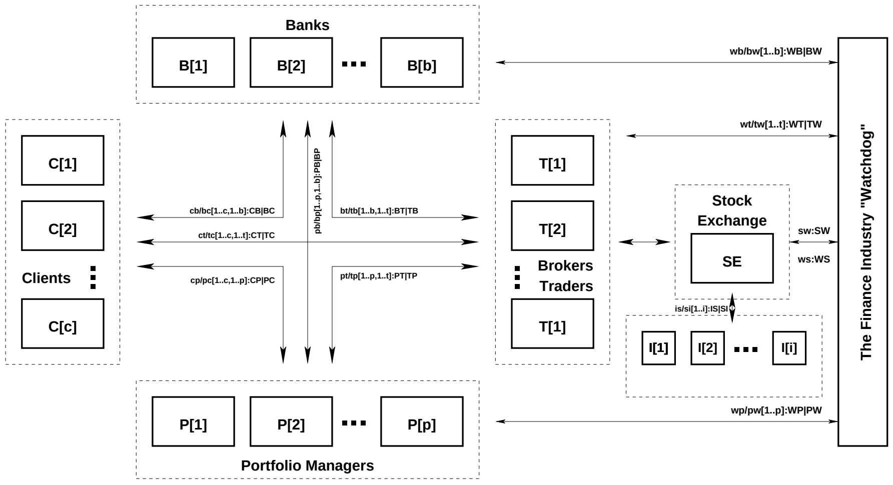

Fig. 11.4. A financial behavioural system abstraction

11.2.4 Discussion

The reader is to be properly warned.

An essence of the examples is not the specific diagrams shown, but that one can indeed draw such behavioural rough sketches. These can include square or rounded boxes designating behaviours; single- or, as here shown, double-ended arrows, designating the possibility of typed communication of messages (say over channels); the (entity) typing of these messages; and so on.

Another essence of the examples is hence that there is a diagrammatic language of behaviours, and that this language has textual counterparts — say in the form of `CSP` or `RSL/CSP`. Other diagrammatic forms might be chosen, depending on properties not revealed in the above examples. These other forms might be Petri nets, message or live sequence charts, or, for example, statecharts.

Furthermore, the examples are sketchy, but they provide an immediate, constructive start to the arduous task of carefully and painstakingly describing a domain.

In all examples we have sketched the suggested arrays of channels and their types (as sorts). These are just suggestions. Interactions between behaviours are then modelled in terms of messages communicated over these channels. But such models are just that: there is no obligation on the part of any, subsequent software design to implement channels as something anywhere similar to channels!

The reader should understand that to describe domains fully satisfactorily requires at least the full complement of principles, techniques and tools covered in all chapters of Vols. 1 and 2, as well as in all the chapters up to and including all of the present chapter in this volume!

11.2.5 Summary

The purpose of first rough-sketching a number, not necessarily all, identifiable business processes is to use these descriptions to identify

- entities,
- functions,
- events and
- behaviours,

as well as to classify these into their "facethood":

- intrinsics,
- support technologies,
- management and organisation,
- rules and regulations,
- scripts and
- human behaviour.

11.2.6 Reminder

We remind the reader of the principle stated at the outset of this section on domain business process facets.

Principle. *Describing Domain Business Process Facets:* As part of understanding any (at least human-made) domain it is important to delineate and describe its business processes. Initially that should preferably be done in the form of rough sketches. These rough sketches should — again initially — focus on identifiable entities, functions, events and behaviours. Naturally, being business processes, identification of behaviours comes first. Then be prepared to rework these descriptions as other facets are being described in depth. ■

A main reason for initially describing the business processes of a domain is to discover, identify and capture entities, functions, events and behaviours of that domain. Another good reason is to get the process of description started — somewhere!

11.3 Domain Intrinsics

Railways, although they have many "players and actors" revolve around some core notions: the rail net and trains on these. Overlapping groups of players and actors (i.e., stakeholders), hence different perspectives, in general, have a core of common entities and phenomena. We refer to this core as *the intrinsics of the domain*.

Principles. *Domain Intrinsics:* From the outset of describing a domain: Analyse it with respect to its intrinsic phenomena and concepts. Focus on describing these first. Make sure that the descriptions of subsequently described domain facets are subordinated descriptions of the domain facets. ∎

Principle. *Describing the Domain Intrinsics Facets:* So from the outset of describing a domain analyse it with respect to its intrinsic phenomena and concepts. Focus on describing these first, and make sure that the descriptions of all other (subsequently described) domain facets are subordinated descriptions of the domain intrinsics. ∎

11.3.1 Overall Principles

Each stakeholder group typically has its view of a domain. Different stakeholder groups may thus have different views of their — otherwise shared — domain. In developing a description of the domain intrinsics we must first develop one description per stakeholder group. Then, in some step of development, reconcile possible domain description inconsistencies and conflicts. To do so systematically we first need to form a basis, the intrinsics, which is common to all subsequent facets.

Characterisation. 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. ∎

In the next many examples we show typical fragments of rough-sketch or narrative descriptions — such as the software developer has to construct when creating a domain description.

Example 11.8 *Railway Net Intrinsics:* We narrate and formalise three railway net intrinsics.

- From the view of *potential train passengers* a railway net consists of lines, stations and trains. A line connects exactly two distinct stations.
- 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 from which to embark or alight a train.
- 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 (through a unit) is a pair of connectors of that unit. A state 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 either one of a number of states of its state space.

Formal Presentation: Railway Net Intrinsics

A summary formalisation of the three narrated railway net intrinsics could be:

- *Potential train passengers:*

```
scheme N0 =
    class
        type
            N, L, S, Sn, Ln
        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.

- *Actual train passengers:*

```
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 sorts, related observer functions and axioms.

- *Train operating staff:*

```
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
```

Unit and connector sorts 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.

The reader is invited to compare the three narrative descriptions with the three formal descriptions, line by line. ■

Different stakeholder perspectives, not only of intrinsics, as here, but of any facet, leads 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 11.9 *Comparable Intrinsics:* We refer to Example 11.8. We claim that the concept of nets, lines and stations in the three models of Example 11.8 must relate. The simplest possible relationships are to let the third model be the common "unifier" and to mandate

- that the model of nets, lines and stations of the *potential train passengers* formalisation is that of nets, lines and stations of the *train operating staff* model; and
- that the model of nets, lines, stations and tracks of the *actual train passengers* formalisation is that of nets, lines, stations of the *train operating staff* model.

Thus the third model is seen as the definitive model for the stakeholder views initially expressed. ■

In general the relationships to be expressed between different stakeholder models require more elaborate expressions. To express these formally, in `RSL`, we make use of `RSL`'s *scheme* facility. We refer to Vol. 2, Chap. 10 (Modularisation) in which we cover the *scheme* concept of `RSL` (Sect. 10.2 (`RSL` Classes, Objects and Schemes) of that volume). More elaborate stakeholder schemes can be expressed by *extending* basic (i.e., intrinsic) schemes *with* additional types, values and axioms. The *hiding* facility of schemes can likewise be used to express different, but commensurate models.

The comparison relations are in this case quite simple, namely those of *conservative algebra inclusions*. One algebra is conservatively included in another algebra if all the entities and operations (etcetera) of the former are included in the latter, and hence if all theorems true of the former algebra hold in the latter.

In the above description such things as lines, stations and units, including their particular kind (linear, switch, etc.) are phenomena, that is, they can be pointed to. Such things as connectors and paths could be considered either phenomena or concepts. Unit states and unit state spaces, including the idea of open and closed units, will here be considered concepts. The above example is only indicative. Much care must be taken to ensure that a description is consistent and complete. Care must also be taken to not describe phenomena or concepts that more properly belong to some other facets, as covered next. Identifying and describing intrinsics is also an art!

Example 11.10 *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. 11.5). 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. ∎

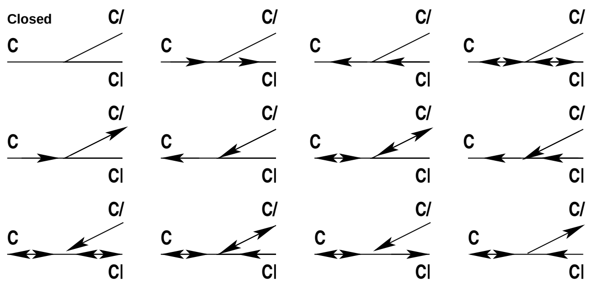

Fig. 11.5. Possible states of a rail switch

11.3.2 Conceptual Versus Actual Intrinsics

In order to bring an otherwise seemingly complicated domain across to the reader, one may decide to present it piecemeal:[8] First, one presents the very basics, the fewest number of inescapable entities, functions and behaviours. Then, in a step of enrichment, one adds a few more (intrinsic) entities, functions and behaviours. And so forth. In a final step one adds the last (intrinsic) entities, functions and behaviours. In order to develop what initially may seem to be a complicated domain, one may decide to develop it piecemeal: We basically do as for the presentation steps: Steps of enrichments — from a big lie, via increasingly smaller lies, till one reaches a truth!

[8] That seemingly complicated domain may seem very complicated, containing hundreds of entities, functions and behaviours. Instead of presenting all the entities, functions, events and behaviours in one "fell swoop", one presents them in stages: first, around seven such (entities, functions, events and behaviours), then seven more, etc.

Example 11.11 *Conceptual Intrinsics: Freight Transport:* The very essence of freight transport is: Entities: Senders, freight, "the system of transport", and receivers. Functions: submitting an item of freight for transport, and receiving an item of freight having been transported. Behaviour: Being transported.

Formal Presentation: Freight Transport

```
type
    Sndr, Frei, Rcvr
value
    submit: Sndr × Frei → System → System
    receiv: Rcvr → System → System × Frei
    transport: System → System
```

Observe that we have said nothing, really, about "the system of transport. ∎

Example 11.12 *Actual Intrinsics: Freight Logistics:* We now elaborate on "the system of transport" alluded to in Example 11.11. The system entities are: harbours, bills of lading, ships and ship routes (from harbours to harbours). We assume that there is no need to detail what are harbours, ships and ship routes. A bill of lading is a document, say attached to a piece of freight, which stipulates properties of the freight (sender, receiver, origin of transport, destination of transport and route of transport: sequence of harbours and ships, sailing times, etc.). The system functions are: submit a piece of freight to a harbour (of origin) indicating a receiver and a harbour of destination, and obtaining a bill of lading; load a piece of freight from a harbour to a ship, as prescribed by that freight's bill of lading; unload a piece of freight from a ship to a harbour, as prescribed by that freight's bill of lading; fetching, by a receiver, a piece of freight from a destination harbour, as prescribed by that freight's bill of lading. A system behaviour could be the sequence of one submission, one or more pairs of loadings and unloadings, ended by one fetch. The above behaviour has abstracted "away" any notion of sailings, i.e., of actual movement!

Formal Presentation: Freight Logistics

```
type
    Sndr, Sndr_Na, Frei, Rcvr, Rcvr_Na,
    Harb, H_Na, Ship, S_Na, System, BoL
    Dest = H_Na
value
    obs_Harbs: System → Harb-set
    obs_HNa: Harb → H_Na
```

```
obs_Route : BoL → (H_Na × S_Na)*
obs_Dest : BoL → HNa
obs_RcvrNa : BoL → Rcvr_Na
obs_RcvrNa : Rcvr → Rcvr_Na

submit: Sndr × Frei × Dest → System → BoL
load: Frei × BoL × Ship × Harb → Ship × Harb
unload: BoL × Ship × Harb → Ship × Harb × Frei
receiv: Rcvr → Harb → System → Frei × BoL
transp: System → System
```

The formalisation, as does the narrative, only rough-sketches some intrinsics of freight logistics. ∎

We leave the two versions, the virtual and the "more realistic", further undefined. Both descriptions were kept in the form of rough sketches. The latter can take being further refined, i.e., made more precise.

11.3.3 Methodological Consequences

Principles. In any modelling one first forms and describes *intrinsic* facets. ∎

Techniques. The *intrinsics* model of a domain is a partial specification. As such, it involves the use of well-nigh all description principles. Typically we resort to property-oriented models, i.e., sorts and axioms. ∎

11.3.4 Discussion

Thus the intrinsics become part of every one of the next facets. From an algebraic semantics point of view these latter are extensions of the above. We have presented a story of intrinsics as truthfully as we could. To decide on what is intrinsics and what is not is an art — it is a matter of choice, hence of style. There is no clear-cut criterion according to which a line of separation between intrinsics and nonintrinsics can be drawn.

11.3.5 Utter Barebones Intrinsics

It was implied above that an absolute barebones intrinsics of railways was the atomic trains and the rail net abstracted to atomic lines and atomic stations. Similarly one could claim that an absolute barebones intrinsics of a hospital system was the atomic patients, atomic medical staff and atomic beds. Without the beds the first two kinds of entities would pass only for a physician's office. And similarly one could claim that an absolute barebones intrinsics for air traffic would be the aircraft, the airports and the air space. And so on.

The reason we bring this concept of *utter barebones intrinsics* up is threefold. First, the domain engineer must "think very hard" in trying to isolate, identify and capture the, or an utter barebones intrinsics of a domain. Secondly, the "more frugal" the domain engineer has been in selecting the utter barebones entities, functions, events and behaviours, the more time that domain engineer has to care about properly extending that utter barebones intrinsics with the remaining domain facets covered next. Thirdly, by "forcibly" trying to isolate an utter barebone intrinsics the domain engineer is actually trying to establish a scientific basis for the domain. The domain describer is more of a researcher than an engineer. This is basically untrodden land: few have tried to formulate domain descriptions let alone intrinsics, and very few, if any may have attempted to identify the utter barebones of a domain. We claim that it is a prerequisite for good domain descriptions to have tried to discover utter barebones intrinsics.

11.3.6 Reminder

We remind the reader of the principle stated at the outset of this section on domain intrinsics:

Principle. *Describing the Domain Intrinsics Facets:* So from the outset of describing a domain analyse it with respect to its intrinsic phenomena and concepts. Focus on describing these first, and make sure that the descriptions of all other (subsequently described) domain facets are subordinated descriptions of the domain intrinsics. ∎

11.4 Domain Support Technologies

Technology is meant to support human activities. Usually technology replaces human actions one to one, i.e., rather directly. (That is, for one human action kind there is usually a substitute technology.) In other instances technology radically transforms the ways in which things are done. Hence:

Principle. *Describing the Domain Support Technologies Facets:* When describing a domain analyse it with respect to its support technology phenomena and concepts, focus on possibly describing these separately, and make sure that descriptions of other described domain facets are commensurate with possibly multiple, alternative descriptions of domain support technologies. ∎

11.4.1 Overall Principles

In Example 11.8, we implied that a switch may take on a number of states: linking, into paths, suitable pairs of connectors, or none. But how such states came about was abstracted (away).

Characterisation. By domain *support technology* we shall understand ways and means of implementing certain observed phenomena. ■

The above characterisation is deliberately loose. It is so, so that we are not, later, constrained by a too tight characterisation. Therefore it is important to illustrate the idea, so as to aid the reader's intuition, and thus enable proper identification and description of support technologies.

Example 11.13 *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[9] (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 electromechanics, 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).[10] ■

It must be stressed that Example 11.13 is just a rough sketch. In a proper narrative description the software (cum domain) engineer must describe, in detail, the subsystem of electronics, electromechanics 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 11.14 *Probabilistic Rail Switch Unit State Transitions:* Figure 11.6 indicates a way of formalising this aspect of a supporting technology.

[9] For pulley see: http://www.walter-fendt.de/ph11e/pulleysystem.htm. For lever see: http://www.edhelper.com/ReadingComprehension_24_90.html.

[10] In Vol. 2, Chap. 12, Petri nets, in Sect. 12.3.4 we exemplified this concept of interlocking by specifying a software design based on place transition nets. See also: http://irfca.org/faq/faq-signal4.html.

Figure 11.6 intends to model the probabilistic (erroneous and correct) behaviour of a switch when subjected to settings (to switched (s) state) and resettings (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. ■

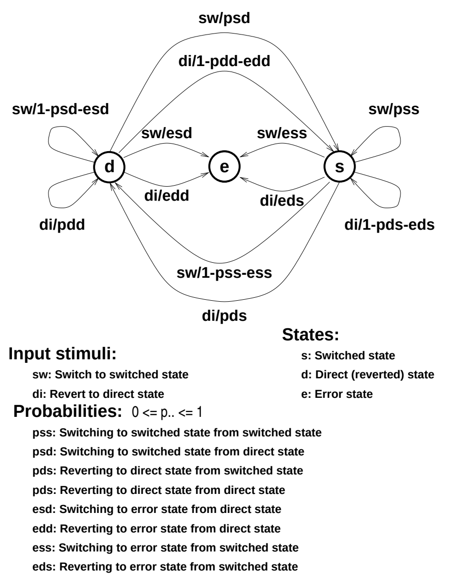
sw/psd
di/1-pdd-edd
sw/1-psd-esd
sw/pss
sw/esd
sw/ess
d
e
s
di/edd
di/eds
di/pdd
di/1-pds-eds
sw/1-pss-ess
di/pds
States:
s: Switched state
d: Direct (reverted) state
e: Error state
Input stimuli:
sw: Switch to switched state
di: Revert to direct state
Probabilities: 0 <= p.. <= 1
pss: Switching to switched state from switched state
psd: Switching to switched state from direct state
pds: Reverting to direct state from switched state
pds: Reverting to direct state from direct state
esd: Switching to error state from direct state
edd: Reverting to error state from direct state
ess: Switching to error state from switched state
eds: Reverting to error state from switched state

Fig. 11.6. Probabilistic state switching

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 11.15 *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 checkable 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. We have defined railway nets in Example 11.8 (Sect. 11.3.1).

Formal Presentation: Railway Optical Gate Technology Requirements

```
type
    T, TN
    P = U*
    NetTraffic == net:N  trf:(TN $\overrightarrow{m}$ P)
    iTF = T → NetTraffic
    sTF = T $\overrightarrow{m}$ NetTraffic
    oG = iTF $\xrightarrow{\sim}$ sTF
value
    [close] c: NetTraffic × TN × NetTraffic $\xrightarrow{\sim}$ Bool
axiom
    ∀ itt:iTF, og:OG • let stt = og(itt) in
        ∀ t:T • t ∈ dom stt •
            t ∈ $\mathcal{D}$ itt ∧ ∀ Tn:TN • tn ∈ dom trf(itt(t))
                ⇒ tn ∈ dom trf(stt(t)) ∧ c(itt(t),tn,stt(t)) end
```

$\mathcal{D}$ is not an RSL operator. It is a mathematical way of expressing the definition set of a general function. Hence it is not a computable function.

Checkability is an issue of testing the optical gates when delivered for conformance to the closeness predicate, i.e., to the axiom. ■

The next example shows another aspect of the technology support facet. Example 11.15 relates any support technology to an intrinsics whose entity values that support technology was supposed to monitor. The next example, i.e., Example 11.16, again shows the relativeness of support technologies, as did Example 11.13.

Example 11.16 *Air Traffic Control:* We first refer to Example 11.4. Then we make the following remarks: The particular decomposition of air traffic control into the domain described, the ground, terminal, area and continental (monitoring and) control centres, represents but one composition of technologies. The pragmatics, i.e., the assumptions underlying that combined ground, terminal, area and continental control centre support technology is that all monitoring and control was to take place from the ground. Future technologies, easily implementable today, facilitate the following alternative "sum total" technologies: Most, if not all, of the human guidance that today takes place at these control centres can be automated and physically moved either to fixed space-positioned satellites, or to each aircraft itself. Intermediate support technologies shall then feature solutions that are intermediary to the present and the future support technologies. ■

11.4.2 Methodological Consequences

Techniques. The *support technologies* model of a domain is a partial specification, hence all the usual abstraction and modelling principles, techniques and tools apply. More specifically, support technologies (st:ST) "implement" intrinsic contexts and states: $\theta_i : \Theta_i$ in terms of "actual" contexts and states: $\theta_a : \Theta_a$:

type
  Θ_i, Θ_a
  $\mathrm{ST} = \Theta_i \to \Theta_a$
axiom
  $\forall$ sts:ST-**set**, st:ST $\bullet$ st $\in$ sts $\Rightarrow \forall\ \theta_i{:}\Theta_i,\ \exists\ \theta_a{:}\Theta_a \bullet \mathrm{st}(\theta_i) = \theta_a$

The formal requirements can be narrated: Let Θ_i and Θ_a designate the spaces of intrinsic and actual-world configurations (contexts and states).[11] For each intrinsic configuration model — that we know is support technology assisted — there exists a support technology solution, that is, a total function from all intrinsic configurations to corresponding actual configurations. If we are not convinced that there is such a function then there is little hope that we can trust this technology. ■

Support technology is not a refinement, but an extension. Support technology typically introduces considerations of technology accuracy, reliability, fault tolerance, availability, accessability, safety, and so on. Axioms characterise members of the set of support technologies (sts). An example axiom was given in the optical gate example (Example 11.15). We shall have much (more) to say about support technologies, and the above dependability (etc.) issues, as we — much later in these volumes — move into machine requirements.

Principles. The *support technology* principle is relative to all other domain facets. It expresses that one must first describe essential intrinsics. Then it expresses that support technology is any means of implementing concrete instantiations of some intrinsics, of some management and organisation, and/or of some rules and regulations, and so on. ■

[11] The concept of configurations, in terms of contexts and states, was treated in detail in Vol. 2, Chap. 4.

Generally the principle states that one must always be on the look out for and inspire new support technologies. The most abstract form of the principle is: *What is a support technology one day becomes part of the domain intrinsics a future day.*

11.4.3 Discussion

Skakkebæk et al. [337] exemplify the use of the duration calculus [381, 382] in describing supporting technologies that help achieve safe operation of a road-rail level crossing. This was exemplified very extensively in Vol. 2, Chap. 15, Sect. 15.3.6 (the road-rail level crossing example).

Chapters 12–15 of Vol. 2 cover a somewhat extensive variety of principles, techniques and tools for formally modelling support technologies. The support technology descriptions reappear in the requirements definitions: as projected, instantiated, extended and initialised (see Chap. 19). In the domain description we only record our understanding of aspects of support technology failures. In the requirements definition we then follow up and make decisions as to which kinds of breakdowns the computing system, the machine, is to handle, and what is to be achieved by such handling.

11.4.4 Reminder

We remind the reader of the principle stated at the outset of this section on domain support technologies:

Principle. *Describing the Domain Support Technologies Facets:* When describing a domain analyse it with respect to its support technology phenomena and concepts, focus on possibly describing these separately, and make sure that descriptions of other described domain facets are commensurate with possibly multiple, alternative descriptions of domain support technologies. ■

11.5 Domain Management and Organisation

It is a basic characteristic of human-made systems that they are managed by humans and that their management and the managed are structured in organisational structures. This section is about how we model this facet.

Principle. *Describing the Domain Management and Organisation Facets:* When describing a domain analyse it with respect to its management and organisation phenomena and concepts. Focus on possibly describing these separately, and make sure that descriptions of other described domain facets are commensurate with possibly multiple, alternative descriptions of domain management and organisation. ■

11.5.1 Overall Principles

Activities of some (application) domains are made up by the actions of many people. It is therefore common to organise these into levels of management and many groups of "floor", i.e., nonmanagement staff.

Railway systems are usually characterised by highly structured management organisations, and rules and regulations set up by upper echelons of management to be followed by lower levels and by ground staff and users.

Example 11.17 *Train Monitoring, I:* In China, as an example, 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. ▪

Characterisation. By domain *management* we shall understand such people (such decisions) (i) who (which) determine, formulate and thus set standards (cf. rules and regulations, Sect. 11.6) concerning strategic, tactical and operational decisions; (ii) who ensure that these decisions are passed on to (lower) levels of management, and to floor staff; (iii) who make sure that such orders, as they were, are indeed carried out; (iv) who handle undesirable deviations in the carrying out of these orders cum decisions; and (v) who "backstop" complaints from lower management levels and from floor staff. ▪

In Example 9.2 (Chap. 9, Sect. 9.3.1) we illustrated the distinctions indicated in the above characterisation of management (item (i)) between strategies, tactics and operations.

Characterisation. By domain *organisation* we shall understand the structuring of management and nonmanagement staff levels; the allocation of strategic, tactical and operational concerns to within management and nonmanagement staff levels; and hence the "lines of command": who does what, and who reports to whom, administratively and functionally. ▪

Example 11.18 *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 nonmanagement 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.[12] A (middle-level operations) manager for regional traffic, i.e., train traffic involving several stations and lines, resolves possible disputes and conflicts. ■

The above, albeit rough-sketch description, illustrated the following management and organisation issues: There is a set of lowest-level (as here: train traffic scheduling and rescheduling) supervisors and their staff. They are organised into one such group (as here: per station). 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. The guidelines issued jointly by local and regional (...) supervisors and managers imply an organisational structuring of lines of information provision and command.

11.5.2 A Conceptual Analysis, I

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 (cf. Example 9.2) — 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 nonmanagement 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 nonmanagement 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.

[12] 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.

11.5.3 Methodological Consequences, I+II

Techniques. The *management and organisation* model of a domain is a partial specification; hence all the usual abstraction and modelling principles, techniques and tools apply. More specifically, management is a set of predicates, observer and generator functions which either parameterise 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 modelled as sets (of recursively invoked sets) of equations. This was illustrated in Example 9.2.

11.5.4 Conceptual Analysis, II

To relate classical organigrams to formal descriptions we first show such an organigram (Fig. 11.7), and then we show schematic processes which — for a rather simple scenario — model managers and the managed!

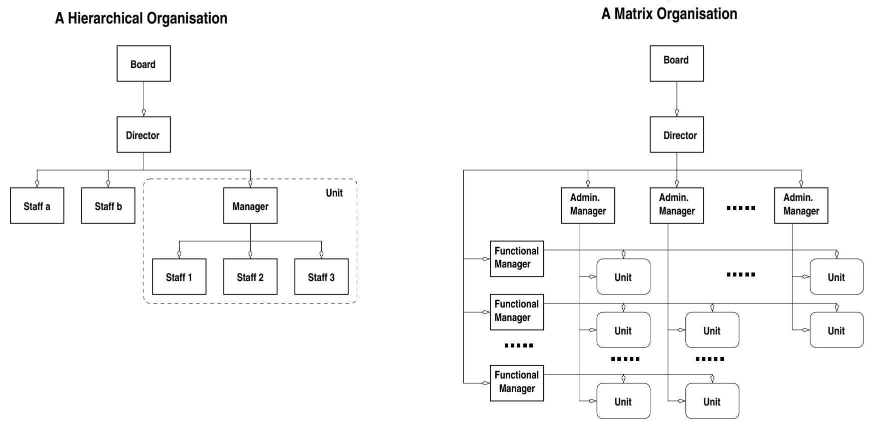

Fig. 11.7. Organisational structures

Based on such a diagram, and modelling 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 modelled by ψ. Each staff, stf(i) operates in a context and a state modelled by sσ(i).

Formal Presentation: Conceptual Model of a Manager-Staff Relation, I

type

```
    Msg, $\Psi$, $\Sigma$, Sx
    S$\Sigma$ = Sx $\overrightarrow{m}$ $\Sigma$
channel
    { ms[i]:Msg | i:Sx }
value
    s$\sigma$:S$\Sigma$, $\psi$:$\Psi$

    sys: Unit $\rightarrow$ Unit
    sys() $\equiv$ ‖ { stf(i)(s$\sigma$(i)) | i:Sx } ‖ mgr($\psi$)
```

In this system the manager, mgr, (1) either broadcasts messages, msg, to all staff via message channel ms[i]. The manager's concoction, mgr_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, mgr_in(i,msg)(ψ), the manager state. In both cases the manager resumes work as from the new state. The manager chooses — in this model — which of the two things (1 or 2) to do by a so-called nondeterministic internal choice ($\lceil\rceil$).

Formal Presentation: Conceptual Model of a Manager-Staff Relation, II

```
    mgr: $\Psi$ $\rightarrow$ in,out { ms[i] | i:Sx } Unit
    mgr($\psi$) $\equiv$
(1)  (let ($\psi'$,msg) = mgr_out($\psi$) in
       ‖ { ms[i]!msg | i:Sx } ; mgr($\psi'$) end)
     $\lceil\rceil$
(2)  (let  $\psi'$ = [] {let msg = ms[i]? in
       mgr_in(i,msg)($\psi$) end | i:Sx } in mgr($\psi'$) end)

    mgr_out: $\Psi$ $\rightarrow$ $\Psi$ $\times$ MSG,
    mgr_in: Sx $\times$ MSG $\rightarrow$ $\Psi$ $\rightarrow$ $\Psi$
```

And in this system, staff i, stf(i), (1) either is willing to receive a message, msg, from the manager, and then to change, stf_in(msg)(σ), state accordingly, or (2) to concoct, stf_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 the two "things" (1 or 2) to do by a nondeterministic internal choice ($\lceil\rceil$).

Formal Presentation: Conceptual Model of a Manager-Staff Relation, III

```
    stf: i:Sx $\rightarrow$ $\Sigma$ $\rightarrow$ in,out ms[i] Unit
    stf(i)($\sigma$) $\equiv$
(1)  (let msg = ms[i]? in stf(i)(stf_in(msg)($\sigma$)) end)
```

```
    $\lceil\rceil$
(2) (**let** ($\sigma'$,msg) = stf_out($\sigma$) **in** ms[i]!msg; stf(i)($\sigma'$) **end**)

    stf_in: MSG $\rightarrow \Sigma \rightarrow \Sigma$,
    stf_out: $\Sigma \rightarrow \Sigma \times$ MSG
```

Both manager and staff processes recurse (i.e., iterate) over possibly changing states. The management process nondeterministically, external choice, "alternates" between "broadcast"-issuing orders to staff and receiving individual messages from staff. Staff processes likewise nondeterministically, external choice, alternate between receiving orders from management and issuing individual messages to management.

The conceptual example also illustrates modelling stakeholder behaviours as interacting (here CSP-like [168, 301, 311]) processes.[13]

11.5.5 Methodological Consequences, III

The *strategic, tactical and operations resource management* example of Example 9.2 (Sect. 9.3.1) illustrated another management and organisation description pattern. It is based on a set of, in this case, recursive equations. Any way of solving these equations, finding a suitable fix point, or an approximation thereof, including just choosing and imposing an arbitrary "solution", reflects some management communication. The syntactic ordering of the equations — in this case a linear passing of enterprise results from upper equations onto lower equations — reflects some organisation.

Principles. The *management and organisation* principle expresses that relations between resources, and decisions to acquire and dispose resources, to deschedule, reschedule and schedule resources, to deallocate, reallocate and allocate resources and to deactivate, reactivate and activate resources, are the prerogatives of well-functioning management, reflect a functioning organisation and imply invocation of procedures that are modelled as actions that "set up" and "take down" contexts and change states. As such, these principles tell us which subproblems of development to tackle. ■

Techniques. *Management and Organisation:* We have already, under techniques for modelling stakeholder and stakeholder perspectives, mentioned some of the techniques (cf. Sect. 9.3.1). Two extremes were shown: Earlier we modelled individual management groups by their respective functions (strm, trm, orm), and their interaction (i.e., organisation) by solutions to a set of recursive equations! Presently we modelled management and organisation, especially the latter, by communicating sequential behaviours. ■

[13] We covered the use of CSP in Vol. 1, Chap. 21.

11.5.6 Discussion

The domain models of management and organisation eventually find their way into requirements and, hence, the software design — for those cases in which the requirements are about computing support of management and its organisation. Support in the solution of the recursive equations of the earlier stakeholder example (Example 9.2 Resource Management) may be offered in the form of constraint-satisfaction solvers [15]. These may partially handle logic characterisations of the strategic and tactical management functions. They might then do so in the form of computerised support of message passing between the various management groups (of, for example, that stakeholder example), as well as of the generic example of the present part.

11.5.7 Reminder

We remind the reader of the principle stated at the outset of this section on domain management and organisation:

Principle. *Describing the Domain Management and Organisation Facets:* When describing a domain analyse it with respect to its management and organisation phenomena and concepts. Focus on possibly describing these separately, and make sure that descriptions of other described domain facets are commensurate with possibly multiple, alternative descriptions of domain management and organisation. ▪

11.6 Domain Rules and Regulations

Railway systems, for example, are characterised by large varieties of rules for appropriate behaviour of: trains, train dispatch, monitoring and control, supporting technology, and hence of humans at all levels. This is also true for most other systems that we might care to consider.

When rules are broken regulations take effect: Humans may be disciplined, and activities of the domain may be adjusted.

Principle. *Describing the Domain Rules and Regulations Facets:* When describing a domain analyse it with respect to its rules and regulations phenomena and concepts. Focus on possibly describing these separately, and make sure that the descriptions of other domain facets are commensurate with possibly multiple, alternative descriptions of domain rules and regulations. ▪

11.6.1 Overall Principles

Earlier, when we dealt with management and organisation, it was hinted that management may issue certain guidelines. We now look at a special class of these.

Characterisation. 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 duty, respectively when performing their function. ■

Characterisation. 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. ■

Rules are like one part of a law: *Thou shalt!* Regulations are like another part of a law: *If you break this law "thou" can expect the following punishment!*

Rules and regulations are set by enterprises, by equipment manufacturers, by enterprise associations, by [government] regulatory agencies, and by society (the latter in the form of laws). Adherence to rules is likewise monitored by these or similar institutions. Enforcement of (i.e., the imposition of what is specified in) regulations is similarly ensured by these or similar institutions.

Example 11.19 *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 11.20 *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.[14]
- 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.

■

It is, as for all other domain facets, crucially important that rules and regulations are captured and precisely described — as we often shall find that requirements of software either assume these rules to hold, or expect such rules to be enforced.[15]

11.6.2 Methodological Consequences

Techniques. *Rules and Regulations:* At a metalevel, 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.

Formal Explication: Conceptual Model of Rules, 1

```
type
    Stimulus, Rule, Θ
    STI = Θ → Θ
    RUL = (Θ × Θ) → 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))

    valid: Stimulus × RUL → Θ → Bool
    valid(sy_sti,se_rul)(θ) ≡ se_rul(θ,(meaning(sy_sti))(θ))
```

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,

[14] In Vol. 2, Chap. 14, Sect. 14.4.1, an example, Automatic Line Blocking, illustrates how one might implement this rule.

[15] As, for example, in Sect. 14.4.1 of Vol. 2, Chap. 14.

which is a pair. This pair consists of a predicate, pre_reg:Pre_REG, where Pre_REG = $(\Theta \times \Theta) \rightarrow$ **Bool**, and a domain configuration-changing function, act_reg:Act_REG, where Act_REG = $\Theta \rightarrow \Theta$, 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.

Formal Explication: Conceptual Model of Regulations, 2

```
type
    Reg
    Rul_and_Reg = Rule × Reg
    REG = Pre_REG × Act_REG
    Pre_REG = $\Theta \times \Theta \rightarrow$ Bool
    Act_REG = $\Theta \rightarrow \Theta$
value
    interpret: Reg → REG
```

The idea is now the following: Any action of the system, i.e., the application of any stimulus, may be an action 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 θ' = (meaning(sy_sti))(θ). Let θ' violate the rule, ~valid(sy_sti,sy_rul)(θ), then if predicate part, pre_reg, of the meaning of the regulation, sy_reg, holds in that violating next configuration, pre_reg(θ,(meaning(sy_sti))(θ)), then the action part, act_reg, of the meaning of the regulation, sy_reg, must be applied, act_reg(θ), to remedy the situation.

Formal Explication: Conceptual Model of Rules and Regulations, 3

```
axiom
    ∀ (sy_rul,sy_reg):Rul_and_Regs •
        let se_rul = meaning(sy_rul),
                (pre_reg,act_reg) = meaning(sy_reg) in
        ∀ sy_sti:Stimulus, $\theta$:$\Theta$ •
            ~valid(sy_sti,se_rul)($\theta$)
                ⇒ pre_reg($\theta$,(meaning(sy_sti))($\theta$))
            ⇒ $\exists$ n$\theta$:$\Theta$ • act_reg($\theta$)=n$\theta$ $\wedge$ se_rul($\theta$,n$\theta$)
    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. ■

We have given an outline of the basic conditions under which a set of rules and regulations must be designed. Whether they are, in actual life, designed, by people, and to be interpreted and followed by people, as described here is not for us to decide. Such concerns are the prerogatives of business process reengineering and domain requirements (Sects. 19.3 and 19.4).

11.6.3 Rules and Regulation Languages

We have outlined the basic properties any set of rules and regulations must imply in a properly functioning organisation. The axioms prescribed above are abstract. They also apply, inter alia, to natural language expressions of rules and regulations.

It would be nice if rules and regulations could be formalised. Then, given an appropriate model of the domain, one might be able to analyse the consistency and completeness of rules and regulations with respect to the domain model.

It is inside the scope, but outside the span of this book to bring in — as of 2006 — research material on this subject. In other words: Expect it to come, one day, probably couched in terms of some modal logics of knowledge and belief, and/promise and commitment, etc. We refer to the nice book by Fagin, Halpern, Moses and Vardi: *Reasoning About Knowledge* [98].

Essentially, the issues are: first, to design and use languages (one or more, Rul, Reg), with proper, possibly modal constructs, for expressing rules and regulations. Second, we need to compile such expressions of rules and regulations. Finally, we need to let a computer check "all the time" whether stimuli (whether human or otherwise generated) might cause transitions that may result in violations of the rules.

11.6.4 Principles and Techniques

Principle. *Rules and Regulations:* Domains are governed by rules and regulations: by laws of nature or edicts by humans. Laws of nature can be part of intrinsics, or can be modelled as rules and regulations constraining the intrinsics. Edicts by humans usually change, but are normally considered part of an irregularly changing context, not a recurrently changing state. Modelling techniques follow these principles. ■

Techniques. *Rules and regulations,* in the domain, are therefore domain-modelled by abstract or concrete syntaxes of syntactic rules, by abstract types of denotations and by semantics definitions, usually in the form of axioms or denotation-ascribing functions. Such rules and regulations modelling must allow for conflicts between rule and regulation interpretations: that rules are interpreted to state that a next configuration is not valid, while a regulation (applicability) predicate does not hold. Stimuli, without here going into details, may be modelled by nondeterministic external events, i.e., `CSP`-like inputs. ∎

11.6.5 Reminder

We remind the reader of the principle stated at the outset of this section on domain rules and regulations:

Principle. *Describing the Domain Rules and Regulations Facets:* When describing a domain analyse it with respect to its rules and regulations phenomena and concepts. Focus on possibly describing these separately, and make sure that the descriptions of other domain facets are commensurate with possibly multiple, alternative descriptions of domain rules and regulations. ∎

11.7 Domain Scripts

Usually rules and regulations form a contract between levels of staff in an enterprise. We may call these intrainstitutional rules and regulations. Rules that pertain to contracts between, say, a private enterprise and its customers, or a government and its citizens, often need be far more stringently phrased than intrainstitutional rules and regulations. We may call such rules legal rules and regulations. Legal rules and regulations often need be scripted.

Principle. *Describing the Domain Script Facets:* When describing a domain analyse it with respect to its script phenomena and concepts. Focus on possibly describing these separately, and make sure that descriptions of other described domain facets are commensurate with possibly multiple, alternative descriptions of domain scripts. ∎

11.7.1 The Description of Scripts

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

Scripts are like programs. They are expected to prescribe step-by-step actions to be applied in order to determine whether a rule should be applied, and, if so, exactly how it should be applied.

Example 11.21 *A Casually Described Bank Script, I:* We deviate, momentarily, from our line of railway examples, to exemplify one from banking. 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. ■

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.

In the next example we systematically describe a bank, informally and formally. The formal description is in the classical style of semantics. Each formal description is followed by an informal, almost rough-sketch description. You may consider the latter to be in some casual script language. Example 11.23 then attempts a formalisation of the rough-sketch scripts into a "bank-friendly" script.

Example 11.22 *Bank Scripts, II:* Without much informal explanation, i.e., narrative, we define a small bank, small in the sense of offering but a few services. One can open and close demand/deposit accounts. One can obtain and close mortgage loans, i.e., obtain loans. One can deposit into and withdraw from demand/deposit accounts. And one can make payments on the loan. In this example we illustrate informal rough-sketch scripts while also formalising these scripts.

In the following we first give the formal specification, then a rough-sketch script. You may prefer to read the pairs, formal specification and rough-sketch script, in the reverse order.

Bank State

Formal Presentation: Bank State

type
  C, A, M
  $\text{AY}' = \textbf{Real}$, $\text{AY} = \{|\ \text{ay}:\text{AY}' \bullet 0<\text{ay}\leq 10\ |\}$
  $\text{MI}' = \textbf{Real}$, $\text{MI} = \{|\ \text{mi}:\text{MI}' \bullet 0<\text{mi}\leq 10\ |\}$
  $\text{Bank}' = \text{A\_Register} \times \text{Accounts} \times \text{M\_Register} \times \text{Loans}$
  $\text{Bank} = \{|\ \beta:\text{Bank}' \bullet \text{wf\_Bank}(\beta)|\}$
  $\text{A\_Register} = \text{C} \xrightarrow[m]{} \text{A-}\textbf{set}$
  $\text{Accounts} = \text{A} \xrightarrow[m]{} \text{Balance}$
  $\text{M\_Register} = \text{C} \xrightarrow[m]{} \text{M-}\textbf{set}$
  $\text{Loans} = \text{M} \xrightarrow[m]{} (\text{Loan} \times \text{Date})$
  Loan,Balance = P
  P = **Nat**

There are clients (c:C), account numbers (a:A), mortgage number (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.

State Well-formedness

Formal Presentation: State Well-formedness

value
  ay:AY, mi:MI

  $\text{wf\_Bank}: \text{Bank} \to \textbf{Bool}$
  $\text{wf\_Bank}(\rho,\alpha,\mu,\ell) \equiv \cup\ \textbf{rng}\ \rho = \textbf{dom}\ \alpha \wedge \cup\ \textbf{rng}\ \mu = \textbf{dom}\ \ell$
axiom
  ai<mi

We assume a fixed yield, ai, on demand/deposit accounts, and a fixed interest, mi, on loans. A bank is well-formed if all accounts named in the accounts register are indeed accounts, and all loans named in the mortgage register are indeed mortgages. No accounts and no loans exist unless they are registered.

Client Transactions

Formal Presentation: Syntax of Client Transactions

```
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)
    CloM == mkCM(c:C,m:M,p:P)
    Reply = A | M | P | OkNok
    OkNok == ok | notok
```

The client can issue the following commands: Open Account, Close Account, Deposit monies (p:P), Withdraw monies (p:P), Obtain loans (of size p:P) and Pay installations on loans (by transferring monies from an account). Loans can be Closed when paid down.

Open Account Transaction

Formal Presentation: Semantics of Open Account Transaction

```
value
    int_Cmd: Cmd → Bank → Bank × Reply

    int_Cmd(mkOA(c))($\rho$,$\alpha$,$\mu$,$\ell$) ≡
        let a:A • a ∉ dom $\alpha$ in
        let as = if c ∈ dom $\rho$ then $\rho$(c) else {} end ∪ {a} in
        let $\rho'$ = $\rho$ † [c↦as],
              $\alpha'$ = $\alpha$ ∪ [a↦0] in
        (($\rho'$,$\alpha'$,$\mu$,$\ell$),a) end end end
```

When opening an account the new account number is registered and the new account set to 0. The client obtains the account number.

Close Account Transaction

Formal Presentation: Semantics of Close Account Transaction

```
int_Cmd(mkCA(c,a))($\rho$,$\alpha$,$\mu$,$\ell$) $\equiv$
    let $\rho' = \rho \dagger [c \mapsto \rho(c) \setminus \{a\}]$,
        $\alpha' = \alpha \setminus \{a\}$ in
    (($\rho'$,$\alpha'$,$\mu$,$\ell$),$\alpha$(a)) end
    pre c $\in$ dom $\rho \wedge$ a $\in \rho$(c)
```

When closing an account the account number is deregistered, the account is deleted, and its balance is paid to the client. It is checked that the client is a bona fide client and presents a bona fide account number. The well-formedness condition on banks secures that if an account number is registered then there is also an account of that number.

Deposit Transaction

Formal Presentation: Semantics of Deposit Transaction

```
int_Cmd(mkD(c,a,p))($\rho$,$\alpha$,$\mu$,$\ell$) $\equiv$
    let $\alpha' = \alpha \dagger [a \mapsto \alpha(a)+p]$ in
    (($\rho$,$\alpha'$,$\mu$,$\ell$),ok) end
    pre c $\in$ dom $\rho \wedge$ a $\in \rho$(c)
```

When depositing into an account that account is increased by the amount deposited. It is checked that the client is a bona fide client and presents a bona fide account number.

Withdraw Transaction

Withdrawing monies can only occur if the amount is not larger than that deposited in the named account. Otherwise the amount, p:P, is subtracted from the named account. It is checked that the client is a bona fide client and presents a bona fide account number.

Formal Presentation: Semantics of Withdraw Transaction

```
int_Cmd(mkW(c,a,p))($\rho$,$\alpha$,$\mu$,$\ell$) $\equiv$
    if $\alpha$(a)$\geq$p
        then
            let $\alpha' = \alpha \dagger [a \mapsto \alpha(a)-p]$ in
            (($\rho$,$\alpha'$,$\mu$,$\ell$),p) end
        else
            (($\rho$,$\alpha$,$\mu$,$\ell$),nok)
    end
    pre c ∈ dom ρ ∧ a ∈ dom α
```

Open Mortgage Account Transaction

Formal Presentation: Semantics of Open Mortgage Account Transaction

```
int_Cmd(mkOM(c,p))($\rho$,$\alpha$,$\mu$,$\ell$) $\equiv$
    let m:M • m $\notin$ dom $\ell$in
    let ms = if c $\in$ dom $\mu$ then $\mu$(c) else {} end $\cup$ {m} in
    let mu$'$ = $\mu$ $\dagger$ [c$\mapsto$ms],
        $\alpha'$ = $\alpha$ $\dagger$ [a$_\ell$$\mapsto$$\alpha$(a$_\ell$)$-$p],
        $\ell'$ = $\ell$ $\cup$ [m$\mapsto$p] in
    (($\rho$,$\alpha'$,$\mu'$,$\ell'$),m) end end end
```

To obtain a loan, p:P, is to open a new mortgage account with that loan (p:P) as its initial balance. The mortgage number is registered and given to the client. The loan amount, p, is taken from a specially designated bank capital acount, a_ℓ. The bank well-formedness condition should be made to reflect the existence of this account.

Close Mortgage Account Transaction

Formal Presentation: Semantics of Close Mortgage Account Transaction

```
int_Cmd(mkCM(c,m))($\rho$,$\alpha$,$\mu$,$\ell$) $\equiv$
    if $\ell$(m) = 0
        then
            let $\mu'$ = $\rho$ $\dagger$ [c$\mapsto$$\mu$(c) \ {m}],
                $\ell'$ = $\ell$ \ {m} in
            (($\rho$,$\alpha$,$\mu'$,$\ell'$),ok) end
        else
            (($\rho$,$\alpha$,$\mu$,$\ell$),nok)
    end
    pre c $\in$ dom $\mu$ $\wedge$ m $\in$ $\mu$(c)
```

One can only close a mortgage account if it has been paid down (to 0 balance). If so, the loan is deregistered, the account removed and the client given an OK. If not paid down the bank state does not change, but the client is given a NOT OK. It is checked that the client is a bona fide loan client and presents a bona fide mortgage account number.

Loan Payment Transaction

Formal Presentation: Semantics of Loan Payment Transaction

To pay off a loan is to pay the interest on the loan since the last time interest was paid. That is, interest, i, is calculated on the balance, b, of the loan for the period $d' - d$, at the rate of mi. (We omit defining the interest computation.) The payment, p, is taken from the client's demand/deposit account, a; i is paid into a bank (interest earning account) a_i and the loan is diminished with the difference $p-i$. It is checked that the client is a bona fide loan client and presents a bona fide mortgage account number. The bank well-formedness condition should be made to reflect the existence of account a_i.

```
int_Cmd(mkPM(c,a,m,p,d$'$))($\rho$,$\alpha$,$\mu$,$\ell$) $\equiv$
    **let** (b,d) = $\ell$(m) **in**
    **if** $\alpha$(a)$\geq$p
        **then**
            **let** i = interest(mi,b,d$'$$-$d),
                $\ell'$ = $\ell$ $\dagger$ [ m$\mapsto$$\ell$(m)$-$(p$-$i) ]
                $\alpha'$ = $\alpha$ $\dagger$ [ a$\mapsto$$\alpha$(a)$-$p,$a_i$$\mapsto$$\alpha$($a_i$)+i ] **in**
            (($\rho$,$\alpha'$,$\mu$,$\ell'$),ok) **end**
        **else**
            (($\rho$,$\alpha'$,$\mu$,$\ell$),nok)
    **end end**
    **pre** c $\in$ **dom** $\mu$ $\wedge$ m $\in$ $\mu$(c)
```

This ends the first stage of the development of a script language. ■

Example 11.22 gave the formal description of banking transactions and their informal, rough-sketch script counterparts. We now "derive", without much further ado, pseudo-formal "bank-friendly" scripts.

Example 11.23 *Bank Scripts, III:* From each of the informal/formal bank script descriptions we systematically "derive" a script in a possible bank script language. The derivation, for example, for how we get from the formal descriptions of the individual transactions to the scripts in the "formal" bank script language is not formalised. In this example we simply propose possible scripts in the formal bank script language.

Open Account Transaction

Formal Presentation: Open Account Transaction

```
value
    int_Cmd(mkOA(c))($\rho$,$\alpha$,$\mu$,$\ell$) $\equiv$
        let a:A • a $\notin$ dom $\alpha$ in
        let as = if c $\in$ dom $\rho$ then $\rho$(c) else {} end $\cup$ {a} in
        let $\rho'$ = $\rho$ $\dagger$ [c$\mapsto$as],
             $\alpha'$ = $\alpha$ $\cup$ [a$\mapsto$0] in
        (($\rho'$,$\alpha'$,$\mu$,$\ell$),a) end end end
```

Derived Bank Script: Open Account Transaction

```
routine open_account(c in "client",a out "account") $\equiv$
        do
            register c with new account a ;
            return account number a to client c
        end
```

Close Account Transaction

Formal Presentation: Close Account Transaction

```
    int_Cmd(mkCA(c,a))($\rho$,$\alpha$,$\mu$,$\ell$) $\equiv$
        let $\rho'$ = $\rho$ $\dagger$ [c$\mapsto\rho$(c)$\setminus${a}],
             $\alpha'$ = $\alpha \setminus$ {a} in
        (($\rho'$,$\alpha'$,$\mu$,$\ell$),$\alpha$(a)) end
        pre c $\in$ dom $\rho$ $\wedge$ a $\in$ $\rho$(c)
```

Derived Bank Script: Close Account Transaction

```
routine close_account(c in "client",a in "account" out "monies") $\equiv$
        do
            check that account client c is registered ;
            check that account a is registered with client c ;
            if
                checks fail
                    then
                        return NOT OK to client c
                    else
                        do
                            return account balance a to client c ;
                    delete account a
                  end
            fi
          end
```

Deposit Transaction

Formal Presentation: Deposit Transaction

```
  int_Cmd(mkD(c,a,p))($\rho$,$\alpha$,$\mu$,$\ell$) $\equiv$
    let $\alpha'$ = $\alpha$ † [a$\mapsto\alpha$(a)+p] in
    (($\rho$,$\alpha'$,$\mu$,$\ell$),ok) end
    pre c $\in$ dom $\rho$ $\wedge$ a $\in$ $\rho$(c)
```

Derived Bank Script: Deposit Transaction

```
routine deposit(c in "client",a in "account",ma in "monies") ≡
    do
      check that account client c is registered ;
      check that account a is registered with client c ;
      if
        checks fail
          then
            return NOT OK to client c
          else
            do
              add ma to account a ;
              return OK to client c
            end
      fi
    end
```

Withdraw Transaction

Formal Presentation: Withdraw Transaction

```
  int_Cmd(mkW(c,a,p))($\rho$,$\alpha$,$\mu$,$\ell$) $\equiv$
    if $\alpha$(a)$\geq$p
      then
        let $\alpha'$ = $\alpha$ † [a$\mapsto\alpha$(a)$-$p] in
        (($\rho$,$\alpha'$,$\mu$,$\ell$),p) end
      else
```

```
        $((\rho,\alpha,\mu,\ell)$,nok)
    end
    pre c $\in$ dom $\rho \wedge$ a $\in$ dom $\alpha$
```

Derived Bank Script: Withdraw Transaction

```
routine withdraw(c in ″client″,a in ″account″,
                        ma in ″amount″ out ″monies″) $\equiv$
    do
      check that account client c is registered ;
      check that account a is registered with client c ;
      check that account a has ma or more balance;
      if
        checks fail
          then
            return NOT OK to client c
          else
            do
              subtract ma from account a ;
              return ma to client c
            end
      fi
    end
```

Obtain Loan Transaction

Formal Presentation: Obtain Loan Transaction

```
  int_Cmd(mkOM(c,p))$(\rho,\alpha,\mu,\ell) \equiv$
    let m:M • m $\notin$ dom $\ell$in
    let ms = if c $\in$ dom $\mu$ then $\mu$(c) else {} end $\cup$ {m} in
    let mu′ = $\mu$ † [c$\mapsto$ms],
        $\alpha'$ = $\alpha$ † [$a_\ell \mapsto \alpha(a_\ell)$−p],
        $\ell'$ = $\ell$ $\cup$ [m$\mapsto$p] in
    $((\rho,\alpha',\mu',\ell')$,m) end end end
```

Derived Bank Script: Obtain Loan Transaction

```
routine get_loan(c in ″client″,p in ″amount″,m out ″loan number″) $\equiv$
    do
      register c with loan m amount p;
      subtract p from account bank’s loan capital
```

```
        return loan number m to client c
    end
```

Close Loan Transaction

Formal Presentation: Close Loan Transaction

```
int_Cmd(mkCM(c,m))($\rho$,$\alpha$,$\mu$,$\ell$) $\equiv$
    if $\ell$(m) = 0
        then
            let $\mu' = \rho \dagger [\mathrm{c} \mapsto \mu(\mathrm{c}) \setminus \{\mathrm{m}\}]$,
                $\ell' = \ell \setminus \{\mathrm{m}\}$ in
            (($\rho$,$\alpha$,$\mu'$,$\ell'$),ok) end
        else
            (($\rho$,$\alpha$,$\mu$,$\ell$),nok)
    end
    pre c $\in$ dom $\mu$ $\wedge$ m $\in$ $\mu$(c)
```

Derived Bank Script: Close Loan Transaction

```
routine close_loan(c in "client",m in "loan number") ≡
    do
        check that loan client c is registered ;
        check that loan m is registered with client c ;
        check that loan m has 0 balance;
        if
            checks fail
                then
                    return NOT OK to client c
                else
                    do
                        close loan m
                        return OK to client c
                    end
        fi
    end
```

Loan Payment Transaction

Formal Presentation: Loan Payment Transaction

```
int_Cmd(mkPM(c,a,m,p,d$'$))($\rho$,$\alpha$,$\mu$,$\ell$) $\equiv$
    **let** (b,d) = $\ell$(m) **in**
    **if** $\alpha$(a)$\geq$p
        **then**
            **let** i = interest(mi,b,d$'$$-$d),
                $\ell'$ = $\ell$ $\dagger$ [m$\mapsto$$\ell$(m)$-$(p$-$i)]
                $\alpha'$ = $\alpha$ $\dagger$ [a$\mapsto$$\alpha$(a)$-$p,$a_i$$\mapsto$$\alpha$($a_i$)+i] **in**
            (($\rho$,$\alpha'$,$\mu$,$\ell'$),ok) **end**
        **else**
            (($\rho$,$\alpha'$,$\mu$,$\ell$),nok)
    **end end**
    **pre** c $\in$ **dom** $\mu$ $\wedge$ m $\in$ $\mu$(c)
```

Derived Bank Script: Loan Payment Transaction

```
routine pay_loan(c in "client",m in "loan number",p in "amount") ≡
    do
        check that loan client c is registered ;
        check that loan m is registered with client c ;
        check that account a is registered with client c ;
        check that account a has p or more balance ;
        if
            checks fail
                then
                    return NOT OK to client c
                else
                    do
                        compute interest i for loan m on date d ;
                        subtract p−i from loan m ;
                        subtract p from account a ;
                        add i to account bank's interest
                        return OK to client c ;
                    end
        fi
    end
```

This ends the second stage of the development of a script language. ■

From the sketch attempts of bank-friendly scripts we establish, in the next example, a syntax for the bank-friendly script language.

Example 11.24 *Bank Scripts, IV:* We now examine the proposed scripts. Our objective is to design a syntax for the language of bank scripts. First, we list the statements as they appear in Example 11.23, except for the first two statements.

Routine Headers

We first list all routine “headers”:

open_account(c **in** ″`client`″,a **out** ″`account`″)
close_account(c **in** ″`client`″,a **in** ″`account`″ **out** ″`monies`″)
deposit(c **in** ″`client`″,a **in** ″`account`″,ma **in** ″`monies`″)
withdraw(c **in** ″`client`″,a **in** ″`account`″,ma **in** ″`amount`″ **out** ″`monies`″)
get_loan(c **in** ″`client`″,p **in** ″`amount`″,m **out** ″`loan number`″)
close_loan(c **in** ″`client`″,m **in** ″`loan number`″)
pay_loan(c **in** ″`client`″,m **in** ″`loan number`″,p **in** ″`amount`″)

We then schematise a routine “header”:

routine name(v1 io ″`t`″,v2 io ″`t2`″,...,vn io ″`tn`″) ≡

where:

io = **in** | **out**

and:

ti is any text

Example Statements

do stmt_list ***end***
if test_expr ***then*** stmt ***else*** stmt ***fi***

register c ***with new account*** a
register c ***with loan*** m ***amount*** p

add p ***to account*** a
subtract p ***from account*** a
subtract p−i ***from loan*** m
add i ***to account bank’s interest***
subtract p ***from account bank’s loan capital***

add p ***to account bank's loan capital***
compute interest i ***for loan*** m ***on date*** d

delete account a
close loan m

return ret_expr ***to client*** c
check that check_expr

The interest variable i is a ***local*** variable. The date variable d is an "oracle" (see below), but will be treated as a ***local*** variable.

Example Expressions

test_expr:

checks fail

ret_expr:

account number a
account balance a
NOT OK
OK
p
loan number m

check_expr:

account client c ***is registered***
account a ***is registered with client*** c
account a ***has*** p ***or more balance***
loan client c ***is registered***
loan m ***is registered with client*** c
loan m ***has*** 0 ***balance***

Abstract Syntax for Syntactic Types

We analyse the above concrete schemas (i.e., examples). Our aim is to find a reasonably simple syntax that allows the generation of the scripts of Example 11.23. After some experimentation we settle on the syntax shown next.

Formal Presentation: Bank Script Language Syntax

type

```
RN, V, C, A, M, P, I, D

Routine = Header × Clause

   Header == mkH(rn:RN,vdm:(V →m (IOL × Text)))
   IOL == in | out | local

Clause = DoEnd | IfThEl | Return | RegA | RegL | Check
          | Add | Sub | 2Sub | DelA | DelM | ComI | RetE |

   DoEnd == mkDE(cl:Clause*)
   IfThEl == mkITE(tex:Test_Expr,cl:Clause,cl:Clause)

   Return == mkR(rex:Ret_Expr,c:V)
   RegA == mkRA(c:V,a:V)
   RegL == mkRL(c:V,m:V,p:V)
   Chk = mkC(cex:Chk_Expr)
   Add == mkA(p:V,t:(V|BA))
   Sub == mkS(p:V,t:(V|BA))
   2Sub == mk2S(p:V,i:V,t:(AN|MN|BA))
      AN == mkAN(a:V)
      MN == mkMN(m:V)
      BA == bank_i | bank_c
   DelA == mkDA(c:V,a:V)
   DelM == mkDM(c:V,m:V)
   Comp == mkCP(m:V,fn:Fn,argl:(V|D)*)

      Fn == interest | ...

Test_Expr = mkTE()

Chk_Expr == CisAReg(c:V) | AisReg(a:V,c:V) | AhasP(a:V,p:V)
           |   CisMReg(c:V) | MisReg(m:V,c:V) | Mhas0(m:V)

RetE == mkAN(a:V)|mkAB(a:V)|ok|nok|mkP(p:V)|mkMN(m:V)
```

■

Finally, in the next example, we establish a formal semantics of the bank-friendly script language. The reader is asked to compare the semantic types of the bank-friendly script language of Example 11.25 with the semantic types of Example 11.22.

Example 11.25 *Bank Scripts, V:*

Formal Presentation: Semantics of Bank Script Language

We now give semantics to the bank script language of Example 11.24.

Semantic Types Abstract Syntax

type
  V, C, A, M, P, I
type
  $\mathrm{AY}' = \mathbf{Real}$, $\mathrm{AY} = \{|\ \mathrm{ay}:\mathrm{AY}' \bullet 0<\mathrm{ay}\leq 10\ |\}$
  $\mathrm{MI}' = \mathbf{Real}$, $\mathrm{MI} = \{|\ \mathrm{mi}:\mathrm{MI}' \bullet 0<\mathrm{mi}\leq 10\ |\}$
  $\mathrm{Bank}' = \mathrm{A\_Register} \times \mathrm{Accounts} \times \mathrm{M\_Register} \times \mathrm{Loans}$
  $\mathrm{Bank} = \{|\ \beta:\mathrm{Bank}' \bullet \mathrm{wf\_Bank}(\beta)|\}$
  $\mathrm{A\_Register} = \mathrm{C} \ \overrightarrow{m}\ \mathrm{A}\textbf{-set}$
  $\mathrm{Accounts} = \mathrm{A} \ \overrightarrow{m}\ \mathrm{Balance}$
  $\mathrm{M\_Register} = \mathrm{C} \ \overrightarrow{m}\ \mathrm{M}\textbf{-set}$
  $\mathrm{Loans} = \mathrm{M} \ \overrightarrow{m}\ (\mathrm{Loan} \times \mathrm{Date})$
  $\mathrm{Loan},\mathrm{Balance} = \mathrm{P}$
  $\mathrm{P} = \mathbf{Nat}$
  $\Sigma = (\mathrm{V} \ \overrightarrow{m}\ (\mathrm{C}|\mathrm{A}|\mathrm{M}|\mathrm{P}|\mathrm{I})) \bigcup (\mathrm{Fn} \ \overrightarrow{m}\ \mathrm{FCT})$
  $\mathrm{FCT} = (...|\mathrm{Date})^* \rightarrow \mathrm{Bank} \rightarrow (\mathrm{P}|...)$
value
  $\mathrm{a}_\ell,\mathrm{a}_i:\mathrm{A}$
axiom
  $\forall\ (\rho,\alpha,\mu,\ell):\mathrm{B}\ \{\mathrm{a}_\ell,\mathrm{a}_i\} \subseteq \mathbf{dom}\ \alpha$

The only difference between the above semantics types and those of Example 11.23 is the Σ state. The purpose of this auxiliary bank state component is to provide (i) a binding between the (always fixed) formal parameters of the script routines and the actual arguments given by the bank client or bank clerk when invoking any one of the routines, and (ii) a binding of a variety of "primitive", fixed, banking functions, FCT, named Fn, like computing the interest on loans, etc.

Semantic Functions

channel
  k:(C|A|M|P|**Text**), d:Date

There is, in this simplifying example, one channel, k, between the bank and the client. It transfers text messages from the bank to the client, and client names (c:C), client account numbers (a:A), client mortgage numbers (m:M), and amount requests and monies (p:P) from the client to the bank. There is also a "magic", a demonic channel, d, which connects the bank to a date "oracle".

```
value
    date: Date → out d  Unit
    date(da) ≡ (d!da ; date(da+$\Delta$))
```

Each routine has a header and a clause. The purpose of the header is to initialise the auxiliary state component σ to appropriate bindings of formal routine parameters to actual, client-provided arguments. Once initialised, interpretation of the routine clause can take place.

```
    int_Routine: Routine → Bank → out k  Bank×$\Sigma$
    int_Routine(hdr,cla)($\beta$) ≡
        let $\sigma$ = initialise(hdr)([ ]) in
        Int_Clause(cla)($\sigma$)(true)($\beta$) end
```

For each formal parameter used in the body, i.e., in the clause, of the routine, there is a formal parameter definition in the header, and only for such. We have not expressed the syntactic well-formedness condition — but leave it as an exercise to the reader. And for each such formal parameter of the header a binding has now to be initially established. Some define input arguments, some define local variables and the rest define, i.e., name, output results. For each input argument the meaning of the header therefore specifies that an interaction is to take place, with the environment, as here designated by channel k, in order to obtain the actual value of that argument.

```
    initialise: Header → $\Sigma$ → out,in k  $\Sigma$
    initialise(hdr)($\sigma$) ≡
        if hdr = [ ]
            then $\sigma$
            else
                let v:V • v ∈ dom hdr in
                let (iol,txt) = hdr(v) in
                let $\sigma'$ =
                    case iol of
                        in → k!txt ; $\sigma$ ∪ [ v ↦ k? ],
                        _  → $\sigma$ ∪ [ v ↦ undefined ]
                    end in
                initialise(hdr\{v})($\sigma'$)
        end end end end
```

In general, a clause is interpreted in a configuration consisting of three parts: (i) the local, auxiliary state, $\sigma : \Sigma$, which binds routine formal parameters to their values; (ii) the current 'check' state, tf:Check, which records the "sum total", i.e., the conjunction status of the check commands so far interpreted, i.e., initially tf = **true**; and (iii) the proper bank state, β:Bank, exactly as also defined and used in Example 11.23. The result of interpreting a clause is a configuration: ($\Sigma\times$Check$\times$Bank).

```
type
    Check = Bool
value
    Int_Clause: Clause→Σ→Check→Bank→out k,in d (Σ×Check×Bank)
```

A **do ... end** clause is interpreted by interpreting each of the clauses within the clauses in the **do ... end** clause list, and in their order of appearance. The result of a check clause is "anded" (conjoined) to the current tf:Check status.

```
    Int_Clause(mkDE(cll))(σ)(tf)(β) ≡
        if cll = ⟨⟩
            then (σ,tf,β)
            else
                let (σ′,tf′,β′) = Int_Clause(hd cl)(σ)(tf)(β) in
                Int_Clause(mkDE(tl cll))(σ′)(tf∧tf′)(β′)
        end end
```

if ... then ... else fi clauses only test the current check status (and propagate this status).

```
    Int_Clause(mkITE(tex,ccl,acl))(σ)(tf)(β) ≡
        if tf
            then
                Int_Clause(ccl)(σ)(true)(β)
            else
                Int_Clause(acl)(σ)(false)(β)
        end
```

Interpretation of a **return** clause does not change the configuration "state". It only leads to an output, to the environment, via channel k, of a return value, and as otherwise directed by any of the six return expressions (rex).

```
    Int_Clause(mkRet(rex))(σ)(tf)(ρ,α,μ,ℓ) ≡
        k!(case rex of
                mkAN(a)
                    → "Your new account number:" σ(a),
                mkAB(a)
                    → "Your account balance paid out:" α(a),
                mkP(p)
```

```
            → "Monies withdrawn:" σ(p),
         mkMN(m)
            → "Your loan number:" σ(m),
         OK
            → "Transaction was successful",
         NOK
            → "Transaction was not successful"
      end);
   (σ,true,(ρ,α,μ,ℓ))
```

Interpretation of a **register account** clause is as you would expect from Example 11.23 — anything else would "destroy" the whole purpose of having a bank script. That purpose is, of course, to effect basically the same as the not yet "script-ised" semantics of Example 11.23.

```
Int_Clause(mkRA(c,a))(σ)(tf)(ρ,α,μ,ℓ) ≡
   let av:A • av ∉ dom α in
   let σ′ = σ † [a ↦ av],
       as = if c ∈ dom ρ then ρ(c) else {} end,
       ρ′ = ρ † [c ↦ as ∪ {av}],
       α′ = α ∪ [av ↦ 0] in
   (σ′,tf,(ρ′,α′,μ,ℓ))
   end end
```

The same holds for the **register loan** clause (as for the **register account** clause).

```
Int_Clause(mkRL(c,m,p))(σ)(tf)(ρ,α,μ,ℓ) ≡
   let mv:M • mv ∉ dom ℓin
   let σ′ = σ † [m ↦ mv],
       ms = if c ∈ dom μ then μ(c) else {} end,
       μ′ = μ † [c ↦ ms ∪ {mv}],
       ℓ′ = ℓ ∪ [mv ↦ p] in
   (σ′,tf,(ρ,α,μ′,ℓ′))
   end end
```

It can be a bit hard to remember the "meaning" of the mnemonics, so we repeat them here in another form:

- CisAReg: Client named in c is registered:
  $\sigma(\mathsf{c}) \in \textbf{dom}\ \rho$.
- AisReg: Client named in c has account named in a:
  $\sigma(\mathsf{c}) \in \textbf{dom}\rho \wedge \sigma(\sigma(\mathsf{a})) \in \rho(\sigma(\mathsf{c}))$.
- AhasP: Account named in a has at least the balance given in p:
  $\alpha(\sigma(\mathsf{a})) \geq \sigma(\mathsf{p})$.

- CisMReg: Client named in c has a mortgage:
  $\sigma(c) \in \mathbf{dom}\ \mu$.
- MisReg: Client named in c has mortgage named in m:
  $\sigma(c) \in \mathbf{dom}\mu \wedge \sigma(m) \in \mu(\sigma(c))$.
- Mhas0: Mortgage named in m is paid up fully:
  $\ell(\sigma(m))=0$.

Then it should be easier to “decipher” the logics:

```
Int_Clause(mkChk(cex))(σ)(tf)(ρ,α,μ,ℓ) ≡
    (σ,case cex of
            CisAReg(c) → $\sigma(c) \in \mathbf{dom}\ \rho$,
            AisReg(a,c) → $\sigma(c) \in \mathbf{dom}\rho \wedge \sigma(\sigma(a)) \in \rho(\sigma(c))$,
            AhasP(a,p) → $\alpha(\sigma(a)) \geq \sigma(p)$,
            CisMReg(c) → $\sigma(c) \in \mathbf{dom}\ \mu$,
            MisReg(m,c) → $\sigma(c) \in \mathbf{dom}\mu \wedge \sigma(m) \in \mu(\sigma(c))$,
            Mhas0(m) → $\ell(\sigma(m))=0$
        end,(ρ,α,μ,ℓ))
```

There are a number of ways of adding amounts, designated in p, to accounts and mortgages:

- mkAN(a): to account named in a
- mkMN(m): to mortgage named in m
- bank_i: to the bank’s own interest account
- bank_c: to the bank’s own capital account

```
Int_Clause(mkA(p,t))(σ)(tf)(ρ,α,μ,ℓ) ≡
    case t of
        mkAN(a) → $(\sigma,\mathbf{true},(\rho,\alpha\dagger[\,a\mapsto\alpha(\sigma(a))+\sigma(p)\,],\mu,\ell))$
        mkMN(m) → $(\sigma,\mathbf{true},(\rho,\alpha,\mu,\ell\dagger[\,\sigma(m)\mapsto\ell(\sigma(m))+\sigma(p)\,]))$
        bank_i → $(\sigma,\mathbf{true},(\rho,\alpha\dagger[\,a_i\mapsto\alpha(a_i)+\sigma(p)\,],\mu,\ell))$
        bank_c → $(\sigma,\mathbf{true},(\rho,\alpha\dagger[\,a_\ell\mapsto\alpha(a_\ell)+\sigma(p)\,],\mu,\ell))$
    end
```

The case, as above for adding, also holds for subtraction.

```
Int_Clause(mkS(p,t))(σ)(tf)(ρ,α,μ,ℓ) ≡
    case t of
        mkAN(a) → $(\sigma,\mathbf{true},(\rho,\alpha\dagger[\,\sigma(a)\mapsto\alpha(\sigma(a))-\sigma(p)\,],\mu,\ell))$
        mkMN(m) → $(\sigma,\mathbf{true},(\rho,\alpha,\mu,\ell\dagger[\,\sigma(m)\mapsto\ell(\sigma(m))-\sigma(p)\,]))$
        bank_i → $(\sigma,\mathbf{true},(\rho,\alpha\dagger[\,a_i\mapsto\alpha(a_i)-\sigma(p)\,],\mu,\ell))$
        bank_c → $(\sigma,\mathbf{true},(\rho,\alpha\dagger[\,a_\ell\mapsto\alpha(a_\ell)-\sigma(p)\,],\mu,\ell))$
    end
```

And it holds as for subtraction, but subtracting two amounts, of values designated in p and i.

```
Int_Clause(mk2S(p,i,t))($\sigma$)(tf)($\rho$,$\alpha$,$\mu$,$\ell$) $\equiv$
    **let** pi = $\sigma$(p)$-\sigma$(i) **in**
    **case** t **of**
        mkAN(a) $\to$ ($\sigma$,**true**,($\rho$,$\alpha$†[$\sigma$(a)$\mapsto\alpha(\sigma$(a))$-$pi],$\mu$,$\ell$))
        mkMN(m) $\to$ ($\sigma$,**true**,($\rho$,$\alpha$,$\mu$,$\ell$†[$\sigma$(m)$\mapsto\ell(\sigma$(m))$-$pi]))
        bank_i $\to$ ($\sigma$,**true**,($\rho$,$\alpha$†[$a_i\mapsto\alpha(a_i)$$-$pi],$\mu$,$\ell$))
        bank_c $\to$ ($\sigma$,**true**,($\rho$,$\alpha$†[$a_\ell\mapsto\alpha(a_\ell)$$-$pi],$\mu$,$\ell$))
    **end end**
```

To delete an account is to remove it from both the account register and the accounts.

```
Int_Clause(mkDA(c,a))($\sigma$)(tf)($\rho$,$\alpha$,$\mu$,$\ell$) $\equiv$
    ($\sigma\backslash$\{a\},**true**,($\rho$†[$\sigma$(c)$\mapsto\alpha(\sigma$(c))$\backslash$\{$\sigma$(a)\}],$\alpha\backslash$\{$\sigma$(a)\},$\mu$,))
```

Similarly, to delete a mortgage is to remove it from both the mortgage register and the mortgages.

```
Int_Clause(mkDM(c,m))($\sigma$)(tf)($\rho$,$\alpha$,$\mu$,$\ell$) $\equiv$
    ($\sigma\backslash$\{m\},**true**,($\rho$,$\alpha$,$\mu$†$\sigma$(c)[$\mapsto\mu(\sigma$(c))$\backslash$\{$\sigma$(m)\}],$\ell\backslash$\{$\beta$(m)\}))
```

To compute a special function requires a place, i, to put, i.e., to store, the resulting, the yielded, value. It also requires the name, fn, of the function, and the actual argument list, aal, i.e., the list of values to be applied to the named function, fct. As an example we illustrate the “built-in” function of computing the interest on a loan, a mortgage.

```
Int_Clause(mkCP(i,fn,aal))($\sigma$)(tf)($\rho$,$\alpha$,$\mu$,$\ell$) $\equiv$
    **let** fct = $\sigma$(fn) **in**
    **let** val = **case** fn **of**
                    "interest" $\to$
                        **let** $\langle$m,d$\rangle$ = aal **in** fct($\langle\mu(\sigma$(m)),d?$\rangle$) **end**
                    ... $\to$ ...
                **end in**
    ($\sigma$†[$\sigma$(i)$\mapsto$val],**true**,($\rho$,$\alpha$,$\mu$,$\ell$)) **end end**
```

This ends the last stage of the development of a script language. ■

11.7.2 Methodological Consequences

We have already covered techniques for, and principles of describing (i.e., modelling) rules and regulations (Sects. 11.6.2 and 11.6.4). These carry over, but in stricter forms, to the description (incl. modelling) of scripts. Designing script languages is basically like designing small programming languages. Vol. 2, Chaps. 3, 6–9 and 16–19 outlined a long series of principles, techniques and tools for designing such languages, including specifying their syntax, semantics and pragmatics.

11.7.3 Reminder + More

We remind the reader of the principle stated at the outset of this section on domain scripts.

Principle. *Describing the Domain Script Facets:* When describing a domain analyse it with respect to its script phenomena and concepts. Focus on possibly describing these separately, and make sure that descriptions of other described domain facets are commensurate with possibly multiple, alternative descriptions of domain scripts. ▪

Techniques. *Domain Scripts:* To properly develop domain scripts, the full force of the semiotic concepts of pragmatics, semantics and syntax, and the techniques of language definition as covered extensively in Vol. 2, apply. ▪

Tools. *Domain Scripts:* Many tools exist for language design and compiler implementation. Some deal with analysis of syntactic and semantic descriptions. Others deal with the automatic generation of lexical scanners, error-correcting parser generators, and yet others with interpreter and compiler generation. We refer to standard textbooks on compiler implementation [6,14,372]. Search the Internet and you will find many references to downloadable compiler construction tools. ▪

11.8 Domain Human Behaviour

Let us consider the staff of any enterprise, any place of work, whether private or public. Some go about doing their job conscientiously: diligently carrying out tasks as expected. Other staff unconsciously sometimes forget: are sometimes a bit sloppy in the dispatch of duties. Yet other staff set themselves lower standards for the pursuit of their assignments: they are slovenly delinquent in completing their work. Finally it may be that some staff are outright criminal in doing their work: They misappropriate funds or steal from the warehouse, etc. A whole spectrum of quality thus characterises human work.

Principles. *Describing the Domain Human Behaviour Facets:* When describing a domain, analyse it with respect to its human behaviour phenomena and concepts. Focus on possibly describing these separately. Make sure that descriptions of other described domain facets are commensurate with possibly multiple, alternative descriptions of domain human behaviours. ▪

11.8.1 Overall Principles

Characterisation. By domain *human behaviour* we shall understand any of a quality spectrum of carrying out assigned work: from *careful, diligent* and *accurate,* via *sloppy* dispatch, and *delinquent* work, to outright *criminal* pursuit. ■

In describing a domain it is important to try capture salient features of what it means to be a human worker: being *careful, diligent* and *accurate,* being unintentionally *sloppy,* being intentionally *delinquent,* being outright *criminal* and, if describable, any shade in-between.

How one describes that, and how one, i.e., the software developer, utilises such descriptions are covered in more detail below.

Example 11.26 *Banking — or Programming — Staff Behaviour:* Let us assume a bank clerk, "in ye olde" days, when calculating, say mortgage repayments, as illustrated in Example 11.21: 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 along the lines illustrated in Example 11.21: 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 11.27 *Shopping — Overall Consumer Behaviour:* A consumers goods market consists of consumers, retailers, wholesalers, producers and delivery services. We focus just on possible consumer behaviours: (i) a consumer inquires, with a retailer, as to availability, price, and delivery terms, of some merchandise. (ii) The retailer responds with zero, one or more offers. (iii) The consumer may decide to ignore the offers, or the consumer may select one of the offers, or the consumer may order something that was not in the set of offers. (iv) The retailer may confirm an order, whereupon delivery takes place and an invoice is sent. (v) The consumer may decide to return the merchandise unpaid, or even paid! (vi) Or the consumer may keep the merchandise and may ignore the invoice, or may pay it, or may pay some other "fictive" (i.e., nonexisting) invoice. (vii) The consumer may then decide to return the merchandise for repair or for claims.

Formal Presentation: Shopping — Overall Consumer Behaviour

We formalise the above. The .. parts indicate "open" parts of the specification, that is, those parts which we believe can be left schematised without loss of basic understanding on the part of the reader.

```
type
    Σ
    Choice == inq | ord | acc | ret | pay | cla | ign
    CR == Inq(..)|Ord(..)|Acc(..)|Pay(..)|Cla(..)|Ign(..)
    RC == Ofr(..)|Del(..)|Inv(..)|..
channel
    cr:CR, rc:RC
value
    consumer: Σ → out cr in rc  Unit
    consumer(σ) ≡
c0   (let cho == inq⌈⌉ord⌈⌉acc⌈⌉ret⌈⌉pay⌈⌉cla⌈⌉ign in
c1    let σ′ =
c2      case cho of
c3        inq → let (σ″,i) = .. in cr!i ; σ″ end
c4        ord → let order = .. in cr!order end
c5        acc → if .. then let (σ″,a) .. in cr!a ; σ″ end else σ end
c6        ret → if .. then let (σ″,r) = .. in cr!r ; σ″ end else σ end
c7        pay → if .. then let (σ″,p) = .. in cr!c ; σ″ end else σ end
c8        cla → if .. then let (σ″,c) = .. in cr!c ; σ″ end else σ end
c9        ign → σ
c10     end
c11   consumer(σ′) end end)
      ⌈⌉
s1   (let res = rc ? in
s2    let σ′ =
s3      case res of
s4        Ofr(..) → handle_ofr(res)(σ),
s5        Del(..) → handle_del(res)(σ),
s6        Inv(..) → handle_inv(inv)(σ),
s7        .. → ..
s8      end in
s9   consumer(σ′) end end)
```

We explain the above formalisation, or, to put it differently, we narrate in more detail the informal points (i–vii) above.

The consumer function has two internally nondeterministically chosen alternatives. Either the initiative is on the side of the consumer (i.e., ‘client’ mode, shown using “c” prefixed line labels); or the consumer “passively” awaits response from the retailer (i.e., ‘server’ mode, shown using “s” prefixed line labels).

(c) As a client the consumer nondeterministically internally, i.e., of her own free will,[16] chooses (c0) between doing any of the actions (c3) inquire about merchandise (..), (c4) order merchandise (..), (c5) accept delivery of merchandise (..) believed to have been delivered (hence the **if .. then .. else .. end**), (c6) return merchandise (..) believed to have been delivered (hence the **if .. then .. else .. end**), (c7) pay for merchandise (..) believed to have been delivered (hence the **if .. then .. else .. end**), (c8) claim refund on supposedly faulty merchandise (..) believed to have been delivered (hence the **if, then, else**), or (c9) ignore whatever goes on! Any of these actions (the last is, in effect, a nonaction) does, indeed, leave a side effect, a remembrance, in the mind of the consumer, hence a state change, from state to state′ ((c1)).

(s) As a server the consumer awaits a response from the retailer. If none is forthcoming, the consumer “deadlocks”! This models that the consumer has gotten “stuck” and stubbornly refuses to take her own initiative, just waits and waits. If a response is forthcoming, it is either (s4) an offer, possibly prompted by an earlier consumer inquiry — but not necessarily. It could be an “own initiative” by the retailer, or (s5) a delivery (etc.), (s6) an invoice (etc.), (s7) or other! In any case, a new state (s2) results. The consumer resumes being a consumer in a new state resulting from either her own initiatives, or from externally prompted actions (c11), resp. (s9). ■

In the above example we are deliberately leaving many things unspecified (..). The point is that we are not so much interested — in this section — in those (..) things. We are interested in modelling, in describing, the vagaries of consumers. These uncertainties, these unpredictable wanderings, were fully described by the nondeterministic choice (c0) and by the fact that after the outputs (!) the consumer “recursed” being a consumer without awaiting responses from the retailer. It was also shown in our not defining, yet, the handle_xyz(..) clauses.

Example 11.28 *Shopping — Detailed Consumer Behaviour:* We continue Example 11.27. We left some open points in the earlier example. We shall use these to illustrate other aspects of human behaviour, its informal and formal descriptions.

[16] We tacitly assume that such a concept as “free will” exists in connection with consumer behaviour!

We start by singling out the treatment of a consumer-initiated initiative, like making an inquiry (c3).

Formal Presentation: Shopping — Detailed Consumer Behaviour

c3 inq $\rightarrow$ **let** (σ'',i) = .. **in** cr!i ; σ'' **end**

To (c3) we add the "missing" information about how we form (i.e., "compute") the information (i.e., data) that goes into an inquiry: '..':

Formal Presentation: Shopping — Detailed Consumer Behaviour

c3 inq $\rightarrow$ **let** (σ'',i) = mki(σ) **in** cr!i ; σ'' **end**

and

value
  mki: $\Sigma \rightarrow$ Inq Σ

In the formula above we have referred to the action of human "gathering" the information that goes into an inquiry by the cryptic function name mki. To make an inquiry we assume that the consumer refers to whatever sense impressions that person may have, and we model that ("whatever sense impressions that person may have") as part of that person's state. Hence the gathering action operates on the state and updates it with the fact that the person (whose state it is) has contemplated and formed an inquiry. We leave the description of mki open. Leaving it open also leaves it open to interpretation. Anything is allowed that forms an inquiry and possibly changes the state. This "openness" models the vagaries of human behaviour. The case for all other consumer-initiated actions directed at the retailer is similar to that of the inquiry action in respect of acting upon and communicating information. We now treat the case of retailer-initiated interactions. Let us consider the consumer's reaction to a retailer offer response.

Formal Presentation: Shopping — Detailed Consumer Behaviour

s4 Ofr(..) $\rightarrow$ handle_ofr(res)(σ)

We refer to this reaction by handle_ofr. As for the making of an inquiry (etc.), this action is not being further described, other than saying: It is any action that somehow records, in the consumer's state, i.e., mind, or jotted down on a piece of paper, say stuck to a kitchen notice board, the fact that approximately "such and such" an offer was received.

Formal Presentation: Shopping — Detailed Consumer Behaviour

```
value
    handle_ofr: Ofr → Σ → Σ
```

No further action is described. In particular, the perhaps expected reaction of the consumer "immediately firing off" an order, or a declination of the offer is not described. Any such possible reaction is modelled by the internal nondeterministic choices of the client actions of the consumer: The consumer may, sooner or later or even never select or choose an order reply. And that order reply may relate, "through" the mko action (c4, not shown), to the Offer response (s4). ▪

11.8.2 Methodological Consequences

Techniques. *Human Behaviour:* (I) We often model the "arbitrariness" of human behaviour by internal nondeterminism. There are two concepts to keep clear of one another: the user choosing to perform an arbitrary action, act_i, from a set Act, of alternative actions, and the interpretation, by the user, or by a system, of that action, b_x, that is, the resulting behaviour.

Formal Explication: Conceptual Model of Human Behaviour, I

```
type
    Act == act_1 | act_2 | ... | act_n | ...
value
    f(...) ≡ ... b_p ⌈⌉ b_s ⌈⌉ b_d ⌈⌉ b_c ...
```

Act denotes a type of action. f defines a function which nondeterministically, under no influence from an, or the, environment (i.e., arbitrarily), selects one of the behaviours b_p, b_s, b_d or b_c. The, possibly deterministic, meaning of each of the alternatives can then be separately described. Proper actions, act_i: some actually perceivable fruitful action, as illustrated in the examples above through the use of the signature-only functions (mk_x and handle_y); and (or versus) action qualities: (i) b_p: professional, (ii) b_s: sloppy, (iii) b_d: delinquent, or (iv) b_c: criminal. We prefer to merge the latter into the former, that is, to assume that the definitions of the actions (mk_x and handle_y) embody both intended actions as well as their quality. ▪

Techniques. *Human Behaviour:* (II) Alternatively we can model human behaviour by the arbitrary selection of elements from sets and of subsets of sets:

Conceptual Model of Human Behaviour, II

```
type
    X
value
    hb_i: X-set ...→... , hb_i(xs,...) ≡ let x:X • x ∈ xs in ... end
    hb_j: X-set ...→... , hb_j(xs,...) ≡ let xs′:X-set • xs′ ⊆ xs in ... end
```

The above shows just fragments of formal descriptions of those parts which reflect human behaviour. Similar, loose descriptions are used when describing faulty supporting technologies, or the "uncertainties" of the intrinsic world. ▪

Techniques. *Human Behaviour (III):* Commensurate with the above, humans interpret rules and regulations differently, and not always "consistently" — in the sense of repeatedly applying the same interpretations.

Our final specification pattern is therefore:

Formal Explication: Conceptual Model of Human Behaviour, III

```
type
    Action = Θ →~ Θ-infset
value
    hum_int: Rule → Θ → RUL-infset
    action: Stimulus → Θ → Θ
    hum_beha: Stimulus × Rules → Action → Θ →~ 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. This is the subject of Part V.

11.8.3 Human Behaviour and Knowledge Engineering

We refer to Sect. 4.1.1 for a first, albeit very brief coverage of the concept of knowledge engineering.

Domain engineering aims at making precise our understanding of the entities, functions, events and behaviours of the observable phenomena and the intellectual concepts of the domain. By *knowledge* we shall, in the narrow context of knowledge engineering, understand that which a human (or a machine, i.e., an agent) knows or believes or assumes or commits with respect to (*knowledge, beliefs, promises* or *commitments* of) another agent. By *knowledge engineering* we shall understand the formulation (whether informal or formal) of such knowledge. Knowledge engineering is thus concerned with understanding relations between two or more agents' knowledge (etcetera) about one another with respect to the following issues: what does an agent know about what another agent knows or believes; which (things) does an agent promise another agent who may then commit or promise other or similar things to yet other agents; and so on. The subject of knowledge engineering is of importance when we model human behaviour but we shall not in this book venture into this very important field of computer and computing science. We refer to the seminal treatise on the subject [98].

11.8.4 Discussion

Please observe the difference between the version of meaning under the rules and regulations facet, Sect. 11.6.2, and the present version. The former reflected the semantics as intended by the stakeholder who issued the rules and regulations. The latter reflects the professional or the sloppy or the delinquent or the criminal semantics as intended by the similarly "qualified" staff which carries out the rule-abiding or rule-violating actions. Please also observe that we do not here exemplify any regulations.

11.8.5 Reminder

We remind the reader of the principle stated at the outset of this section on domain human behaviour:

Principles. *Describing the Domain Human Behaviour Facets:* When describing a domain, analyse it with respect to its human behaviour phenomena and concepts. Focus on possibly describing these separately. Make sure that descriptions of other described domain facets are commensurate with possibly multiple, alternative descriptions of domain human behaviours. ▪

11.9 Other Domain Facets?

We have exemplified and formalised some aspects of human behaviour in the domain. And we have informally and formally described how we model some aspects of some facets (rules and regulations, respectively human behaviour). The latter form some initial contributions to a more proper theory of what we mean by domain facets. The domain facets that we have covered included: intrinsics, support technologies, management and organisation, rules and regulations, domain scripts and human behaviour. The question now is obvious: Are there other domain facets? We refrain, at present, from an answer. But we would be surprised if there were not! In other words, we expect further practice and further exploratory and experimental research to yield additional facets. Thus the reader should be on the look out for whether the facets covered here suffice. More generally we must accept the next principle:

Principle. *Domain Facets:* When modelling, informally or formally, a domain, analyse the domain phenomena with respect to whether one or another, or a combination of currently identified domain facets suffice to model the domain, or whether you, the developer, have to discover, i.e., identify, define and otherwise find a suitable set of one or more principles, techniques and tools with which to model the domain. ▪

11.10 Composition of Domain Models

From the various facet descriptions the domain engineer now has to weave a fabric, and Sect. 11.10.1 is about that. The domain engineer may also have to formalise the full description, and Sect. 11.10.2 is about that.

11.10.1 Collating Domain Facet Descriptions

General

The various domain facets can be described more or less individually. It is a good idea to try identify and describe these separate facets individually — in other words applying the principle of separate concerns. But, in doing so the describer may be repeating some descriptive material unnecessarily. Such duplicate material may differ in details and may thus create inconsistencies as well as doubts in the minds of the readers. But analysing the domain and describing it on a per facet basis may yield insight and lead to discoveries about the domain not otherwise attainable.

A Comprehensive Narrative

Describing the domain on a per facet basis may lead to a fragmented, staccato (abrupt, disjointed) description. To avoid this it may be a good idea to take all the bits and pieces of the various facet descriptions and write them into one whole comprehensive narrative. In merging the various facets into one structured narrative the domain engineer may discover possible inconsistencies — and thus will have an early opportunity to correct such. The possibly revised (for example corrected) "bits and pieces" should not be thrown away. They can serve as possibly clarifying study material.

From Big Lies via Smaller Lies to the Truth

A Golden Rule of Comprehension

Develop your domain understanding — and hence the first round of domain descriptions — by analysing and describing the domain facet-by-facet (including formalisation), then by consolidating this into a more pedagogical and didactical[17] flow of narration (with edited formalisation).

One typical way of structuring a comprehensive narrative, as well as its accompanying formalisations, is to formulate the full narrative as a sequence of narratives. Initially the narratives pretend to cover the entire domain, starting, obviously with some intrinsics. But steps of subsequent narratives enlarge upon the scope, choosing pedagogically further domain aspects — be they of intrinsic, of support technology, of management and operation, or of the nature of some other domain facets. The order chosen is determined by what the writer judges is good didactics and good pedagogics. Many such orders are possible. We can phrase this unfolding of a narrative as follows:

Principles. The principle of *From Big Lies via Smaller Lies to the Truth*. To achieve a smooth, pedagogically and didactically sound presentation of some universe of discourse, start by narrating a suitable lie, call it a big lie, a gross simplification. Proceed by adorning the ("false") narration with smaller lies, that is, with less gross simplifications. In doing this you have to accommodate it so that the smaller lies fit nicely onto the big lie, that is, that you do not have to change anything in your presentation, only, so to speak, "refine" it. Then go on to detail the less gross simplifications, i.e., tell tiny lies while still adhering to the "accommodation principle". Finally you have added so much detail that you have told "the truth", that is, what we abstract of the universe of discourse as our truthful abstraction of that universe. Thus "the limit of all the lies is the truth". ∎

[17] *Pedagogical:* of the art and science of teaching. *Didactic:* intended to convey instruction and information as well as pleasure and entertainment [238].

11.10.2 Technical Issues

We saw, in Sect. 11.3.1, the need for composing intrinsic descriptions from intrinsic description parts. We have now seen, in this chapter, through its coverage of many facets, the need for composing from descriptions of separate facets of a domain a comprehensive and consistent description. As in Sect. 11.3.1, we refer to the use, for example, of `RSL`'s *scheme* facility. We refer to Vol. 2, Chap. 10 (Modularisation) in which we cover the *scheme* concept of `RSL` (Sect. 10.2 (`RSL` Classes, Objects and Schemes) of that volume). Non-intrinsic facet schemes can be expressed by *extending* basic (e.g., intrinsic) schemes *with* additional types, values and axioms. The *hiding* facility of schemes can likewise be used to express different, but commensurate models.

11.11 Exercises

11.11.1 A Preamble

We refer to Sect. 1.7.1 for the list of 15 running domain (requirements and software design) examples; and we refer to the introductory remarks of Sect. 1.7.2 concerning the use of the term "selected topic".

11.11.2 The Exercises

Exercise 11.1 *Intrinsics.* For the fixed topic, selected by you, identify and describe

1. some intrinsic entities,
2. some intrinsic functions,
3. some intrinsic events and
4. some intrinsic behaviours.

Exercise 11.2 *Business Processes.* For the fixed topic, selected by you, identify and describe two ("as different as is reasonable") business processes.

Exercise 11.3 *Support Technologies.* For the fixed topic, selected by you, identify and describe two ("as different as is reasonable") support technologies.

Exercise 11.4 *Management and Organisation.*

1. For the fixed topic, selected by you, identify and describe management entities, functions, events and behaviours.
2. Identify and describe a possible organisational structure of your chosen domain.

Exercise 11.5 *Rules and Regulations.* For the fixed topic, selected by you, identify and describe three to four rules and corresponding regulations.

Exercise 11.6 *Scripts.* For the fixed topic, selected by you, identify and describe a possible script language (hint at a syntax, and rough sketch or narrate a semantics).

Exercise 11.7 *Human Behaviour.* For the fixed topic, selected by you, identify and describe:

1. specifically desirable human behaviours, and
2. specifically undesirable human behaviours.

Exercise 11.8 *A Comprehensive Domain Description.* For the fixed topic, selected by you, collate the descriptions that you have produced in answers to Exercises 11.1–11.7 into one comprehensive domain description.