Domain Engineering*

Dines Bjørner†
Department of Computer Science and Engineering,
Institute of Informatics and Mathematical Modelling,
Technical University of Denmark,
DK-2800 Kgs. Lyngby, Denmark‡
bjorner@gmail.com, www.imm.dtu.dk/~db

June 1, 2007

Abstract

These lecture notes are primarily about domain modelling and only secondarily how to transform domain models into domain and interface requirements. The following facets of domain modelling will be covered: business processes, intrinsics, support technologies, management and organisation, rules and regulations, scripts, and human behaviour. The lectures will exemplify excerpts of models of container shipping, financial services, etcetera. We shall relate two (of three) parts of requirements models to domain models: the domain requirements - which are the requirements that can be expressed solely in terms of the terms of the domain models, and the interface requirements - which are those requirements that can only be expressed using terms both of the domain and the machine: the hardware and software being required.

For domain requirements we briefly sketch the domain-to-requirements "algebra" of projection, instantiation, determination, extension and fitting.

For interface requirements we briefly sketch the domain & machine issues of shared entities (bulk data input and refreshments), shared functions, shared events, and shared behaviours.

1 Introduction

A domain is a universe of discourse.

Examples of domains are: airport, air traffic, container line industry, financial service industry, health care, the market and transportation.

By a domain description we understand a precise specification, informal as well as formal of the domain: what there is, not what we would like it to be, not requirements to software to serve in the domain, and certainly not a specification of the software, i.e., the code.

We shall discuss domains and domain engineering in more detail in Sects. 3 — so that we now can first overview the structure of the lecture notes and then "jump right into" a large example of a domain description

We structure the rest of the paper as follows:

In On The Example: The Container Line Industry, Sect. 2, we discuss, briefly, the extensive example presented in Appendix A. We assume that the reader has first studied this appendix. In Domain Engineering Issues, Sect. 3, we outline a number of domain engineering stages: identification of stakeholders, domain acquisition, domain analysis, domain modelling, verification, validation and domain theory. We then cover a number of Domain Facets: Intrinsicsin Sect. 4.3, Support

*Lecture notes for the 19th International School for Computer Science Researchers, Lipari, Italy, July 9–3, 2007 http://lipari.cs.unict.it/LipariSchool/CS/

†Prof. Emeritus

‡Fredsvej 11, DK-2840 Holte, Denmark

Technology in Sect. 4.4, Management & Organisation in Sect. 4.5, Rules & Regulations in Sect. 4.6 Scripts in Sect. 4.7 and Human Behaviour in Sect. 4.8. Then follows a brief summary of principles and techniques of Domain and Interface Requirements in Sect. 5. We close with a Review of Research Issuesin Sects. 6–8.

2 On the Example: The Container Line Industry

Appendix A brings a relatively large example of a domain description. We assume, in the following, that the reader has studied this example.

The purpose of that example is to serve as a background for the rest of this paper. In discussing several, if not most aspects of what we cover of domain engineering, we shall tacitly relate to this example. The purpose of the example is also to indicate, to the reader, that domain descriptions are usually rather large.

The example, as noted in Appendix Sect. A.1, covers a sizable and representative part of, in this case, the domain of the container line industry.

2.1 Overview of The Container Line Industry

We shall consider only the following phenomena and concepts of a basically shipping and logistics industry concerned with — the acceptance, transport over large distances, possibly by means of more than one voyage, and then possibly with temporary storage and final delivery — of containers.

We shall consider major phenomena and concepts of this industry to include (i) containers, (ii) container vessels incl. container stowage, (iii) container terminal ports with (iii.1) quays, (iii.2) container stacks, (iii.3) transfer area, (iii.4) cranes and (iii.5) vehicles, (iv) nets of sea lanes, (v) container lines, (vi) container bill-of-ladings and (vii) container logistics – the allocation and scheduling of containers to container ships and container terminal ports.

2.2 Some Observations and Remarks

We shall here prefix your study of Appendix A with the following remarks and observations:

1. The domain description is experimental. It is far from being a finished domain description. It presents some concepts and some abstractions that are only suggested. As an experiment we think that the domain description serves its purpose well. Not only as a background for the rest of this paper, but also as indicative of what a domain description might contain, and of the style of presentation, of alternation between informal narrative and formal specification, as well as many other things commented on later in this paper.

2. The domain description, to the novel reader, may seem large and complex. Be that as it may. It is at least approaching something believable. We claim that it is worthwhile publishing such a domain description: there are not very many reasonably sized domain descriptions around; and the present one may be used as a reference to enterprises within the container line industry for them to see what can be done — say for purposes of an industry standard, for example for “this” or “that” aspect of the container processes; etc.

3. The domain description encompasses several what we might wish to call sub-domains. One could claim that the following were such sub-domains: containers, container vessels, stowage of containers on vessels and in stacks, container terminal ports, net of sea lanes, i.e., “the high seas”, container lines, bill of ladings, container transport logistics, and the customer interface to the container line industry.

4. When placed in the context of other domain descriptions, in fact just domains, one may see that for example that part of the present domain which we refer to as ‘net of sea lanes’ shares almost all its “features” (i.e., phenomena and concepts) with, for example, the transportation

domain mentioned briefly (late on). This is unavoidable: that seemingly separate domains share "interfaces".

5. The importance of such a possible decomposition (cf., Item 3 on the preceding page) of the container line industry into separate sub-domains could be the following: different groups of one or more domain engineers could then work on separate sub-domains, in parallel, with possibly overlapping groups of stakeholders, provided that clear interfaces between the sub-domains have first been established. Such interfaces can first be tentatively established one a large scale experiment, such as reported in Appendix A has been carried out. The quality of a set of such sub-domain interfaces can be "measured" by the frequency with which an interface has to be adjusted: a low frequency seems to show a high quality.

3 Encircling Some Domain Engineering Issues

3.1 The Triptych Dogma

3.1.1 The Dogma

First the dogma: Before software can be designed its requirements must be understood. Before requirements can be prescribed the application domain must be understood.

3.1.2 The Consequences

Then the "idealised" consequences: In software development we first describe the domain, then we prescribe the requirements, and finally we design the software. As we shall see: major parts of requirements can be systematically "derived"[1] from domain descriptions. In engineering we can accommodate for less idealised consequences, but in science we need investigate the "ideals".

3.1.3 The Triptych Verification

A further consequence of this triptych development is that

$$\mathcal{D}, \mathcal{S} \models \mathcal{R},$$

which we read as: in order to prove that $\mathcal{S}$oftware implements the $\mathcal{R}$equirements the proof often has to make assumptions about the $\mathcal{D}$omain.

3.1.4 Full Scale Development: A First Suggested ℜesearch Topic

Again, presupposing much to come we can formulate a first research topic.

ℜ 1. **The $\mathcal{D}, \mathcal{S} \models \mathcal{R}$ Relation**: Assume that there is a formal description of the $\mathcal{D}$omain, a formal prescription of the $\mathcal{R}$equirements and a formal specification of the $\mathcal{S}$oftware design. Assume, possibly, that there is expressed and verified a number of relations between the $\mathcal{D}$omain description and the $\mathcal{R}$equirements prescription. Now how do we express the assertion: $\mathcal{D}, \mathcal{S} \models \mathcal{R}$ — namely that the software is correct? We may assume, without loss of generality, that this assertion is in some form of a pre/post condition of $\mathcal{S}$ — and that this pre/post condition is supported by a number of assertions "nicely spread" across the $\mathcal{S}$oftware design (i.e., the code). The research topic is now that of studying how, in the pre/post condition of $\mathcal{S}$ (the full code) and in the (likewise pre/post condition) assertions "within" $\mathcal{S}$, the various components of $\mathcal{R}$ and $\mathcal{D}$ "appear", and of how they relate to the full formal pre- and descriptions, respectively.

[1] By "derivation" we here mean one which is guided by humans (i.e., the domain and requirements engineers in collaboration with the stakeholders).

3.2 Preliminary Discussion of Domain Engineering

3.2.1 Archetypal Examples

Lest we loose contact with reality it is appropriate here, however briefly, to give some examples of (application) domains.

Air Traffic: A domain description includes descriptions of the entities, functions, events and behaviours of aircraft, airports (runways, taxi-ways, apron, etc.), air lanes, ground, terminal, regional, and continental control towers, of (national [CAA, CAAC, FAA, SLV, etc.] and international [JAA, CAO]) aviation authorities, etc.

Airports: A domain description includes descriptions of the flow of people (passengers, staff), material (catering, fuel, baggage), aircraft, information (boarding cards, baggage tags) and control; of these entities, of the operations performed by or on them, the events that may occur (cancellation or delay of flights, lost luggage, missing passenger), and, hence, of the many concurrent and intertwined (mutually "synchronising") behaviours that entities undergo.

Container Line Industry: A domain description includes descriptions of containers, container ships, the stowage of containers on ships and in container stacks, container terminal (ports), the loading and unloading of containers between ships and ports and between ports and the "hinterland" (including cranes, port trucking and feeder trucks, trains and barges), the container bills of lading (or way bills), the container transport logistics, the (planning and execution, scheduling and allocation) of voyages, the berthing (arrival and departure) of container ships, customer relations, etc.

Financial Service Industry: A domain description includes descriptions of banks (and banking: [demand/deposit, savings, mortgage] accounts, [opening, closing, deposit, withdrawal, transfer, statements] operations on accounts), insurance companies (claims processing, etc.), securities trading (stocks, bonds, brokers, traders, exchanges, etc.), portfolio management, IPOs, etc.

Health care: A domain description includes descriptions of the entities, operations, events and behaviours of healthy people, patients and medical staff, of private physicians, medical clinics, hospitals, pharmacies, health insurance, national boards of health, etc.

The Internet: The reader is encouraged to fill in some details here!

Manufacturing: Machining & Assembly: The reader is encouraged to also fill in some details here!

"The" Market: A domain description includes descriptions of the entities, operations, events and behaviours of consumers, retailers, wholesalers, producers, the delivery chain and the payment of (or for) merchandise and services.

Transportation: A domain description includes descriptions of the entities, functions, events and behaviours of transport vehicles (cars/trucks/busses, trains, aircraft, ships), [multimodal] transport nets (roads, rail lines, air lanes, shipping lanes) and hubs (road intersections [junctions], stations, airports, harbours), transported items (people and freight), and of logistics (scheduling and allocation of transport items to transport vehicles, and of transport vehicles to transport nets and hubs). Monomodal descriptions can focus on just air traffic or on container shipping, or on railways.

The Web: The reader is encouraged to "likewise" fill in some details here!

There are many "less grand" domains: railway level crossings, the interconnect cabling between the oftentimes dozens of "boxes" of some electronic/mechanical/acoustical measuring set-up, a gas burner, etc. These are all, rather one-sidedly, examples of what might be called embedded, or real-time, or safety critical systems.

We can refer to several projects at UNU-IIST which have produced domain specifications for railway systems (China), ministry of finance (Vietnam), telephone systems (The Philippines), harbours (India), etc.; and to dozens of MSc projects which have likewise produced domain specifications for airports, air traffic, container shipping, health care, the market, manufacturing, etc. I give many, many references in [1]. I also refer the reader to http://www.railwaydomain.org/ for documents, specifically http://www.railwaydomain.org/book.pdf for domain models of railway systems.

3.2.2 Some Remarks

A point made by listing and explaining the above domains is the following: They all display a seeming complexity in terms of multitude of entities, functions, events and interrelated behaviours; and they all focus on the reality of "what is out there": no mention is (to be) made of requirements to supporting computing systems let alone of these (incl. software).

3.2.3 Domains: Suggested ℜesearch Topics

From the above list we observe that the 'transportation item' "lifts" those of 'air traffic' and 'container shipping'. Other examples could be shown. This brings us, at this early stage where we have yet to really outline what domain engineering is, to suggest the following research topics:

ℜ2. **Lifted Domains and Projections**: We observe, above, that the 'transportation' domain seems to be an abstraction of at least four more concrete domains: road, rail, sea and air transportation. We could say that 'transportation' is a commensurate "lifting" of each of the others, or that these more concrete could arise as a result of a "projection" from the 'transportation' domain. The research topic is now to investigate two aspects: a computing science cum software engineering aspect and a computer science aspect. The former should preferably result in principles, techniques and tools for choosing levels of "lifted" abstraction and "projected" concretisation. The latter should study the implied "lifting" and "projection" operators.

ℜ3. **What Do We Mean by an Infrastructure ?** We observe, above, that some of the domains exemplify what is normally called infrastructure[2] components. According to the World Bank: *'Infrastructure' is an umbrella term for many activities referred to as 'social overhead capital' by some development economists, and encompasses activities that share technical and economic features (such as economies of scale and spillovers from users to nonusers).* The research is now to study whether we can reformulate the sociologically vague World Bank definition in precise mathematical terms.

ℜ4. **What Is an Infrastructure Component ?** We observe, above, that not all of the domains exemplified are what is normally called infrastructure components.[3] The research is now to study whether we can formulate and formalise some "tests" which help us determine whether some domain that we are about to model qualifies as part of one or more infrastructure components.

We bring these early research topic suggestions so that the reader can better judge whether domain engineering principles and techniques might help in establishing a base for such research. Throughout the lecture notes we shall "spice it" with further suggestions of research topics.

• • •

We do not cover the important methodological aspects of stakeholder identification and liaison, domain acquisition and analysis, domain model verification and validation. For that we refer to Vol. 3 Chaps. 9–10 and 12–14 [1].

[2] Winston Churchill is quoted to have said, during a debate in the House of Commons, in 1946: ... *The young Labourite speaker that we have just listened to, clearly wishes to impress upon his constituency the fact that he has gone to Eton and Oxford since he now uses such fashionable terms as 'infra-structures'. [I have recently been in communication with the British House of Commons information office enquiries manager, Mr. Martin Davies in order to verify and, possibly pinpoint, this statement. I am told that "as the Hansard debates in question are not available electronically, it could only be found via a manual search of hard copy Hansard". So there it stands.]*

[3] 'Manufacturing' and 'The Market' appear, in the above list to not be infrastructure components, but, of course, they rely on the others, the infrastructure components.

3.2.4 How Is It in Other Branches of Engineering ?

Classical Engineers Practice Their Domains !. A reputable telecommunications company, hiring engineers for the design of mobile telephony radio communications equipment would hire a person who was well acquainted with Maxwell's Equations — and expect that engineering to conduct design in the context of those equations! The point being made above is that electro magnetic wave propagation is the domain for the radio communications engineer. Similar for aeronautics engineers and aerodynamics, for electronics chip designer and plasma physics. Etcetera.

Do Software Engineers Practice Their Application Domains ?. Does the software engineer who is designing software for the container line industry know much about container shipping ? We doubt, and we doubt very much so ! Does the software engineer who is designing software for the railway industry know much about railways ? We doubt, and we doubt very much so ! Does the software engineer who is designing software for the financial service sector know much about banking, securities instruments, the stock exchange, etc. ? We doubt, and we doubt very much so !

Is this acceptable ? Well, it is not professional, cf. $\mathcal{D}, \mathcal{S} \models \mathcal{R}$!

3.3 Stages of The Domain Engineering Phase

By domain development we mean a process, consisting of a number of reasonably clearly separable stages which when properly conducted leads to a domain description, i.e., a domain model. We claim that the following are meaningful and necessary domain development stages of development, each with their attendant principles, techniques and tools: (i) identification of stakeholders, (ii) rough sketching, (iii) domain acquisition, (iv) analysis of rough domain description units, (v) domain modelling, (vi) domain verification, (vii) domain validation and (viii) domain theory formation. We shall focus on domain modelling emphasising the modelling concept of domain facets.

3.3.1 Domain Development Stages: Suggested ℜesearch Topic:

ℜ5. **Sufficiency of Domain Development Stages**: We suppose that the reader is aware of the "contents" of each of these stages. Is the set of domain development stages listed above sufficient ? Could the composition of eight stages be done differently, with fewer or more "orthogonal" stages ? To perform such a study and to answer such questions, in our mind, requires that a number of reasonably distinct domain developments are first undertaken.

3.3.2 Stakeholders

By a domain stakeholder we mean a person, or a group of persons, or an enterprise (private or public institution), or a group of enterprises united in — for all practical purposes — a common interest in the domain such that other stakeholders have discernably different interests while (obviously) sharing some, but not all such concerns.

Example. 1 – ***Stakeholders:*** *For the domain of, for example, railway systems the following stakeholders are included amongst those of that domain: owners (stock owners), board (etc.), executive management, tactical management, operational management, (blue collar) workers, passengers (and their family), suppliers, national railway board, transport safety board, ministry of transport and politicians "at large".* •

The Pragmatics:. The idea of stakeholders is to involve "an as full complement" of these in the subsequent domain development stages (i.e., processes). Involvement means that each stakeholder is liaised with regularly. This liaison and the particulars of the domain development stages shall help the domain engineer identify and model the separate but commensurate interests of all stakeholders.

Domain Stakeholders: Suggested ℜesearch Topic:.

ℜ6. **Stakeholder Separation**: Given that each stakeholder view is reflected in reasonably separable parts of a formal domain description can we arrive at a formal characterisation of "stakeholder-hood" ?

3.3.3 Rough Sketching

Based on literature studies, talks with a few stakeholders, perhaps WWW studies, the domain engineer should be able to rough sketch a domain description. The main example of these lecture notes in fact represents such a rough sketc. A rough sketch enables better planning of susequent stages, communication with domain stakeholders, specifically the preparation of questionnaire material for domain acquisition. A rough sketch is like a proper domain description but it is not claimed final, it may not have desirable abstractions, and it will certainly not be compelete. Other than these cursory remarks we shall not cover 'rough sketching' further.

3.3.4 Domain Acquisition

By domain acquisition we mean the gathering, by domain engineers, of domain description units from the literature (Web etc.), own physical study of the domain, stakeholders and possibly from existing descriptions

The "new" thing here is the concept of a domain description unit. Vol. 3 [1] presents a number of principles and techniques, but no tools, for domain acquisition. We are skeptic about such tools (also the "corresponding" requirements acquisition "tools").

The Reality:. But the reality remains: the domain engineers, together with the stakeholders, must express "bit and pieces" of domain knowledge informally, and we call such "tidbits" domain description units.

Domain Acquisition: Suggested ℜesearch Topic:.

ℜ7. **Domain Description Units (I)**: It may be that the current author has reservations about a domain specific language of domain description units. Be that as it may. Assume n of set domain description units, $\mathcal{D}_{S_1}, \ldots, \mathcal{D}_{S_n}$ in natural language $\mathcal{L}$. Assume existence of $1+n$ formalisable sub-languages of $\mathcal{L}$: $\mathcal{L}_{\mathcal{M}}$ and $\mathcal{L}_{\mathcal{D}_i}$ (for $n \in \{1..n\}$) such that the sets of domain description units $\mathcal{D}_{S_1}, \ldots, \mathcal{D}_{S_n}$ can be expressed in respective sub-languages. What of $\mathcal{D}_i$ cannot be described in $\mathcal{L}_{\mathcal{M}} + \mathcal{L}_{\mathcal{D}_i}$? Is what cannot be described sufficiently small, tiny, so as to warrant a mechanisation of these sub-languages, even for $n = 1$? That's the research topic !

3.3.5 Domain Analysis

By domain analysis we mean a study of a set of domain description units with the aim of discovering inconsistencies, undesirable incompleteness and suitable abstractions: concepts whose use may improve a domain description.

Why Domain Analysis ?. Well, the above explains it. We do not want to create domain models which are inconsistent nor have unwanted gaps, and we want domain models which build on pleasing abstractions.

How (to Perform) Domain Analysis ?. Proper domain (description unit) analysis can, in our view, only be performed if the set of domain description units can be formalised. And then the analysis can be carried out by now more-or-less "standard" formal verification techniques and tools.

Domain Analysis: Suggested ℜesearch Topic:.

ℜ8. **Domain Description Units (II)**: One research topic would be to investigate whether existing techniques for analysing requirements prescription units apply to the analysis of domain description units.

3.3.6 Domain Modelling

By domain modelling we mean the construction of both an informal, narrative and a formal domain description.

We claim that the following identified facets (i.e., "steps") (later to be briefly explained) are necessary parts of the domain modelling process: (i) intrinsics, (ii) support technologies, (iii) management and organisation, (iv) rules and regulations, (v) scripts and (vi) human behaviour. Ideally speaking one may proceed with these "steps" in the order listed. Engineering accommodates for less ideal progressions. Each "step" produces a partial domain description. Subsequent "steps" 'extend' partial descriptions into partial or even (relative) complete descriptions.

Sect. 4 will cover domain modelling in some detail.

3.3.7 Domain Verification

Why Verification ? There are two answers to this question: (i) We really cannot claim to have understood a domain unless we have a domain theory, with proven propositions, lemmas and theorems. (ii) We must make sure we are getting the domain description right, as opposed to the right description, see 'validation' next, that is, that what is described possesses the right properties.

How Verification ? The answer to this question is: First we must assume that there is also a formalisation of our domain description, not just an informal, yet precise narrative. Then we assume that the formal specification language has a proof system — and proof tool support. With that we can — hopefully (and laboriously) — prove properties of the formal descriptions and claim these to be properties of the domain !

We return to the issue of domain verification when we cover the notion of 'domain theory', see below.

3.3.8 Domain Validation

Why Validation ? We must make sure we are getting the right domain description, as opposed to getting the description right (see 'verification' just above), that is, what is described is what the stakeholders can accept.

How Validate ? Since what the stakeholders can relate to must necessarily be the informal description — one cannot assume that they in general can read formal descriptions there is a double validation problem: (i) First there is stakeholder validation: the careful "walk through", by domain engineers and stakeholders, reading and agreeing on every line of the narrative description. (ii) Then there is the domain engineer validations: the careful "walk through", by mutually "buddy checking" domain engineers, reading and agreeing that every line of the narrative description has been properly formalised, and that every line of the formalisation is covered by the narrative.

Domain Validation: Suggested ℜesearch Topic.

ℜ9. **Domain Validation**: The research called for here is more of the engineering and methodology kind than of the computer science kind ! To do meaningful and relevant research in this area it seems that a number of issues must first be settled. These could, illustratively, be:An informal language, $\mathcal{L}$, for expressing domain description units, need probably be identified, and, for it, two formalisable subsets, $\mathcal{L}_{\mathcal{D}_i}$ and $\mathcal{L}_{\mathcal{M}}$ — such as suggested in ℜesearch Topic 7.

Then a number of case studies, for distinct domains, $\mathcal{D}_i$, $\mathcal{D}_j$, ..., $\mathcal{D}_k$ can be made of how the two kinds of validation outlined above actually proceeds. From such studies one might possibly be able to draw some general conclusions as to engineering practices etc. !

3.3.9 Domain Theories

- *By a* **domain theory** *we shall understand a domain description together with lemmas, propositions and theorems that may be proved about the description — and hence can be claimed to hold in the domain.*

To create a domain theory the specification language must possess a proof system. It appears that the essence of possible theorems of — that is, laws about — domains can be found in laws of physics. For a delightful view of the law-based nature of physics — and hence possibly also of man-made universes we refer to Richard Feynman's Lectures on Physics [2].

Example Theorem of Railway Domain Theory. Let us hint at some domain theory theorems: **Kirchhoff's Law for Railways:** Assume regular train traffic as per a modulo κ hour time table. Then we have, observed over a κ hour period, that the number of trains arriving at a station minus the number of trains ending their journey at that station plus the number of trains starting their journey at that station equals the number of trains departing from that station.

Why Domain Theories ? Well, it ought be obvious ! We need to understand far better the laws even of man-made systems.

Domain Theories: Suggested ℜesearch Topics:.

ℜ10. **Domain Theories**: We need to experimentally develop and analyse a number of suggested theorems for a number of representative domains in order to possibly 'discover' some meta-theorems: laws about laws !

4 Domain Modelling

By domain modelling we mean the construction of both informal, narrative, and formal domain descriptions. Many aspects of modelling are usually of concern to the domain engineer: expressing the appropriate entities, functions. events and behaviours of the domain, and expressing the appropriate temporal and static, contextual and state, dimensionality, and many, many other attributes. We will, in these lecture notes only look at the issue of domain facets. My book [3, 4, 1] covers all of this and more !

4.1 Business Processes

To guide the stakeholders and the domain engineers into proper domain facet modelling we suggest that rough sketches (as well as terminology construction) of the domain business processes (with their entities, functions and events) precede proper domain facet modelling. By a 'business process' we shall understand a sequence of domain actions (i.e., function applications) and domain events involving domain entities. where that business process is a characteristic of the domain.

From a set of such rough sketches of business processes whose construction resulted from the analysis of a number of domain description units one is now in a better situation to systematically construct a domain terminology and systematically describe the domain facets.

4.1.1 Two Examples: Some Container Shipping Business Processes

An Example: A Harbour Visit. A container vessel announces (an event) its imminent arrival at a container terminal port to that ports harbour master. The harbour master allocates (an action) a berth at a container quay to that vessel and informs the vessel (another action). The vessel docks (an event) at the allocated berth. First the vessel may undergo an iterative unloading process: all import containers are transferred via cranes to quay and stack area vehicles, and these vehicles move and transfer the import containers to stacks (or, in some cases to transfer area trucks, trains or barges). Several quay cranes, and hence several quay and stack area vehicles may be involved in the usually highly interleaved set of simultaneous ship to stack behaviours. Then the vessel may undergo an iterative loading process: basically the reverse of the above transfer and move & transfer stack to ship behaviours After completion of loading (or just unloading, if no loading) the vessel interacts with the harbour master and leaves the container terminal port.

An Example: Container Shipping. A customer interacts with a shipper: requesting the shipment of a (or more) container(s), hands over this or these container(s), and receives, in return, a copy of the appropriate bills of lading. The shipper, who has interacted with a container line concerning this shipment while also interacting with the customer, hands over this or these container(s) and to the designated container terminal port of origin, at its transfer area from where it is stacked. A container vessel is loaded with the container(s). The container vessel sails to the port of destination. (We ignore possible intermediate harbour visits. And we simplify the container shipping "story": no intermediate transfers from one vessel via stacks to other vessels.) The container(s) is (are) unloaded. They may temporarily be stacked. And finally they are moved and transferred to the transfer area where they are delivered to the receiver of the container(s).

4.2 The Facets

We claim that the following identified facets (i.e., "steps") (later to be briefly explained) are necessary parts of the domain modelling process: (i) intrinsics, (ii) support technologies, (iii) management and organisation, (iv) rules and regulations, (v) scripts and (vi) human behaviour. Ideally speaking one may proceed with these "steps" in the order listed. Engineering accommodates for less ideal progressions. Each "step" produces a partial domain description. Subsequent "steps" 'extend' partial descriptions into partial or even (relative) complete descriptions.

4.3 Intrinsics

4.3.1 Introduction

By the intrinsics of a domain we shall understand those phenomena and concepts, that is, those entities, functions, events and behaviours in terms of which all other facets are described.

The choice as to what constitutes the intrinsics of a domain is often determined by the views of the stakeholders. Thus it is a pragmatic choice, and the choice cannot be formalised in the form of an **is_intrinsics** predicate that one applies to phenomena and concepts of the domain.

4.3.2 Intrinsics Example: Railway Units

We may consider rail units as atomic entities from which rail nets are composed by connecting the units. For simplicitiy we need only think of linear, switch and possibly switchable crossover units. From a rail unit we can observe its connectors: linear units have two, switches have three and crossovers have four connectors. A path (that a train may travel through a unit) can be conceptualised as a pair of suitable connectors of the unit.

type
U, C
P = {|(c,c′):C×C • c≠c′|}

value
obs_Cs: U → C-**set**, obs_Ps: U → P-**set**
is_LinU, is_SwiU, is_CroU: U → **Bool**
axiom
∀ u:U • **let** n = **card** obs_Cs(u) **in**
n=2 ⇒ is_LinU(u) ∧ n=3 ⇒ is_SwiU(u) ∧ n=4 ⇒ is_CroU(u) **end** ∧
is_SwiU(u) ⇒
∃ c,c′,c″:C • obs_Ps(u)={(c,c′),(c,c″),(c′,c),(c″,c)} ∧ ...

In switch and crossover units not all pairings of their connectors designate proper paths A state of a unit is the set of paths of that unit open for train travel. Figure 1 illustrates the 12 possible states of a normal switch unit.

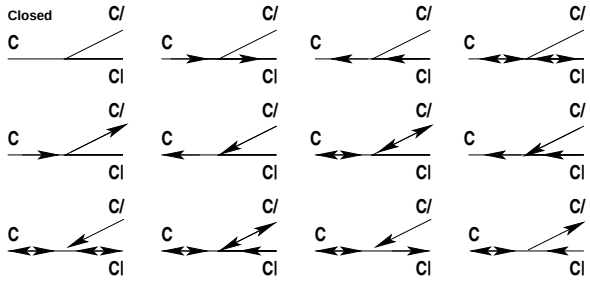


Figure 1: Possible states of a rail switch

.

type
Σ = P-**set**, Ω = Σ-**set**
value
obs_Σ: U → Σ, obs_Ω: U → Ω
axiom
∀ u:U • obs_Σ(u)⊆obs_Ps(u) ∧ obs_Σ(u) ∈ obs_Ω(u)

The state space, here called ω, of a unit is the set of all possible states of that unit.

4.3.3 Intrinsics Example: Container Shipping

We give an number of alternative suggestions of intrinsics for the container shipping domain and as seen from different domain stakeholders.

Customer Intrinsics. (i) Seen from the point of view of customers who is having a container shipped these are the intrinsic entities: customers, containers, shippers, possibly container lines, container terminal ports, i.e., harbours, and bills of lading. We say 'possibly' as all the customer really needs are shippers at ports of origin and destination. The customers rely on shippers, we assume, to deliver and receive containers to — respectively at — ports of origin, respectively ports of destination.

Container Line Intrinsics. (ii) Seen from the point of view of container lines these are the intrinsic entities: containers, container lines, shippers, container vessels and stowage plans, bills of lading, and container terminal ports with quays, quay cranes, and stacks. One can argue whether the container line needs to be concerned with transfer areas, container terminal port vehicles, and stack cranes.

Container Terminal Port Intrinsics. (iii) Seen from the point of view of container terminal ports these are the intrinsic entities: containers, container vessels, container lines, the(ir) container terminal port: quays, stacks, and transfer areas, quay, stack, and transfer area cranes, vehicles, ship and stack stowage planes, crane split plans, in-port container transport job plans, as well as bills of lading, transfer area trucks, trains and barges, truck, train and barge owners, and shippers.

4.3.4 Compositionality of 'Intrinsics' Models

Thus, dependent on the scope and span of a domain one may choose a narrow span, or one may choose an intermediate span, or one may choose a widest reasonable span — where 'one' is a combination of stakeholders and domain engineers. In any case the domain engineer need be careful in choosing appropriate means of modularity so that 'widest possible' models can be reasonably easily presented to different stakeholders as 'narrow' or 'intermediate' models.

4.3.5 Intrinsics: Suggested ℜesearch Topic

ℜ11. **Intrinsics**: We refer to Sect. 11.3 in [1]. The study here is of at least two kinds. (i) There is a programming methodological study of principles and techniques for "merging" different stakeholder views of intrinsics, that is, of possible modularisation principles and techniques. (ii) And there is a study of a more fundamental kind namely of what "really" do we mean by 'intrinsics'.

4.4 Support Technology

By a support technology of a domain we shall understand either of a set of (one or more) alternative entities, functions, events and behaviours which "implement" an intrinsic phenomenon or concept. Thus for some one or more intrinsic phenomena or concepts there might be a technology which supports those phenomena or concepts.

4.4.1 Example Rail Switch Technology

(i) In "ye olde" days rail switches were "thrown" manually. (ii) Later, but it was a long time ago, mechanics allowed rail cabin "throwing" of switches via wires and pullers. (iii) Later the rail cabin "throwing" of switches was supported by electro-mechanics. (iv) And later again this ((iii)) was supported by electronics. (v) Today groups of switches (and switchable crosseovers) are controlled by electronics in what is called 'interlocking'.

4.4.2 Sampling Behaviour of Support Technologies

Let us consider **i**ntrinsic **A**ir **T**raffic as a continuous function (→) from **T**ime to **F**light **L**ocations:

type
  T, F, L
  iAT = T → (F $\overrightarrow{m}$ L)

But what is observed, by some support technology, is not a continuous function, but a discrete sampling (a map $\overrightarrow{m}$):

  sAT = T $\overrightarrow{m}$ (F $\overrightarrow{m}$ L)

There is a support technology, say in the form of **radar** which "observes" the intrinsic traffic and delivers the sampled traffic:

value
  radar: iAT → sAT

4.4.3 Probabilistic cum Statistical Behaviour of Support Technologies

But even the radar technology is not perfect. Its positioning of flights follows some probabilistic or statistical pattern:

type
  P = {|r:**Real** • 0≤r≤1|}
  ssAT = P $\overrightarrow{m}$ sAT-**infset**
value
  radar′: iAT $\xrightarrow{\sim}$ ssAT

The radar technology will, with some probability produce either of a set of samplings, and with some other probability some other set of samplings, etc.[4]

An Example Probabilistic Rail Switch. Figure 2 intends to model the probabilistic (erroneous and correct) behaviour of a switch when subjected to settings. A switch may go to the switched state from the direct [switched] state (d, resp. s) when subjected to a 'switch' setting with probability psd [pss]. A switch may go to the direct state from the switched [direct] state when subjected to a 'direct' setting with probability pds [pdd]. A switch may go to a "hung up" erroneous state, e, when otherwise directed (with probabilities ess, esd, edd, eds). A switch may remain in its present state when signalled to change (with probabilities 1-psd-ess, 1-pdd-edd, 1-pss-ess, 1-pds-eds).

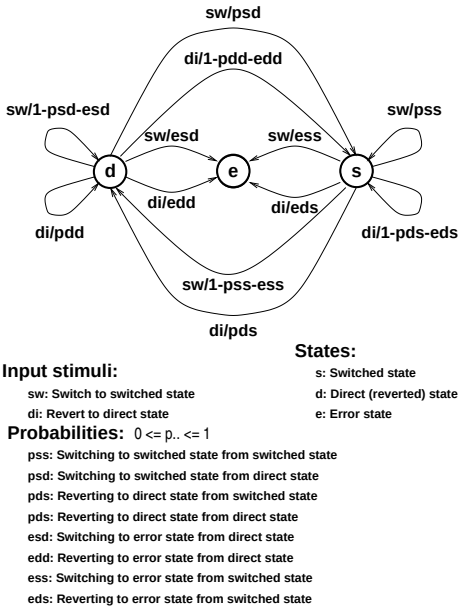


Figure 2: Probabilistic state switching

Example Container Shipping Support Technologies. We mention a few support technologies relevant to our "running" example.

(i) Seen from the point of view of customer or shipper stateholders container terminal cranes and vehicles are support technologies.

(ii) Seen from the point of view of container terminal port management automatically guided vehicles (AGVs) is a support technology.

[4]Throughout this paper we omit formulation of type well-formedness predicates.

(iii) Seen from the point of view of container line management GPS systems, wherever otherwise located, is a support technology helping them to track containers.

4.4.4 Support Technology Quality Control, a Sketch

How can we express that a given technology delivers a reasonable support ? One approach is to postulate intrinsic and technology states (or observed behaviours), Θ_i, Θ_s, a support technology τ and a "**close**ness" predicate:

```
type
  Θ_i, Θ_s
value
  τ: Θ_i → P Θ_s-infset
  close: Θ_i × Θ_s → Bool
```

and then require that an experiment can be performed which validates the support technology.

The experiment is expressed by the following axiom:

```
value
  p_threshhold:P
axiom
  ∀ θ_i:Θ_i •
    let pθ_ss = τ(θ_i) in
    ∀ p:P • p>p_threshhold ⇒
      θ_s:Θ_s • θ_s ∈ pθ_ss(p) ⇒ close(θ_i,θ_s) end
```

The p_threshhold probability has to be a-priori determined as one above which the support technology renditions of the intrinsic states (or behaviours) are acceptable.

4.4.5 Support Technologies: Suggested ℜesearch Topics

ℜ12. **Probabilistic and/or Statistical Support Technologies**: Some cases should be studied to illuminate the issue of probability versus statistics. More generally we need more studies of how support technologies "enter the picture", i.e., how "they take over" from other facet. And we need to come up with precise modelling concepts for probabilistic and statistical phenomena and their integration into the formal specification approaches at hand.

ℜ13. **A Support Technology Quality Control Method**: The above sketched a 'support technology quality control' procedure. It left out the equally important 'monitoring' aspects. Develop experimentally two or three distinct models of domains involving distinct sets of support technologies. Then propose and study concrete implementations of 'support technology quality monitoring and control' procedures.

4.5 Management & Organisation

By the management of an enterprise (an institution) we shall understand a (possibly stratified, see 'organisation' next) set of enterprise staff (behaviours, processes) authorised to perform certain functions not allowed performed by other enterprise staff (behaviours, processes) and where such functions involve monitoring and controlling other enterprise staff (behaviours, processes). By organisation of an enterprise (an institution) we shall understand the stratification (partitioning) of enterprise staff (behaviours, processes) with each partition endowed with a set of authorised functions and with communication interfaces defined between partitions, i.e., between behaviours (processes).

4.5.1 Examples: Container *&c.* Management & Organisation

We give some examples of management & organisation from the "running" main example of container line industry.

(i) Seen from the view of a container line the management & organisation concerns include: the strategic management of buying up other container industry enterprises, of fleet renewal, and of long term relations to container terminal ports; the tactical management of setting rates, of deciding on vessel routes and frequency, and of optimising relevant container transport and storage processes; and the operational management of monitoring & controlling all profit/loss container transport and storage processes, i.e., of issues related to day-to-day scheduling & allocation. The organisational concerns follow the above decomposition into a usually geographically widely spread strategic, tactical and operational management.

(ii) Seen from the view of a container terminal port the management & organisation concerns include: the strategic management of relations to container lines and port workers, and of port layout: up- or downgrading of port facilities; the tactical management (planning) of stowage optimisation in stacks, of crane splits, and of container loading and unloading jobs; the operational management of monitoring & controlling the actual loading and unloading of vessels and of stack usage. The organisational concerns follow the above decomposition into a usually locally, yet spread strategic, tactical and operational management.

(iii) Seen from the view of customers sending (or receiving containers) the management & organisation concerns include: the operational management of the interface between the customer and the shipper, and the geographical organisation into ports of origin and destination.

4.5.2 An Abstraction of Management Functions

Let **E** designate some enterprise state concept, and let **stra_mgt, tact_mgt, oper_mgt, wrkr** and **merge** designate strategic management, tactical management, operational management and worker actions on states such that these actions are "somehow aware" of the state targets of respective management groups and or workers. Let **p** be a predicate which determines whether a given target state has been reached, and let **merge** harmonise different state targets into an agreeable one. Then the following behaviour reflects some aspects of management.

```
type
  E
value
  stra_mgt, tact_mgt, oper_mgt, wrkr, merge: E×E×E×E → E
  p: E* → Bool
  mgt: E → E
  mgt(e) ≡
    let e′ = stra_mgt(e,e″,e‴,e⁗),
        e″ = tact_mgt(e,e″,e‴,e⁗),
        e‴ = oper_mgt(e,e″,e‴,e⁗),
        e⁗ = wrkr(e,e″,e‴,e⁗) in
    if p(e,e″,e‴,e⁗)
      then skip
      else mgt(merge(e,e″,e‴,e⁗))
    end end
```

The recursive set of $e^{\prime..\prime}$ = *f(e, e″, e‴, e⁗)* equations are "solved" by iterative communication between the management groups and the workers. The arrangement of these equations reflect the organisation and the various functions, **stra_mgt, tact_mgt, oper_mgt** and **wrkr** reflect the management. The frequency of communication between the management groups and the workers help determine a quality of the result.

The above is just a very crude, and only an illustrative model of management and organisation.

We could also have given a generic model, as the above, of management and organisation but now in terms of, say, CSP processes. Individual managers are processes and so are workers. The enterprise state, $e : E$, is maintained by one or more processes, separate from manager and worker processes. Etcetera.

4.5.3 Process Model of Manager-Staff Relations

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 $s\sigma$(i).

```
type
   Msg, Ψ, Σ, Sx
   SΣ = Sx m→ Σ
channel
   { ms[ i ]:Msg | i:Sx }
value
   sσ:SΣ, ψ:Ψ

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

In this system the manager, mgr, (1) either broadcasts messages, 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 thetwo things (1 or 2) to do by a so-called nondeterministic internal choice ($\sqcap$).

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

   mgr_out: Ψ → Ψ × MSG,
   mgr_in: Sx × MSG → Ψ → Ψ
```

And in this system, staff i, stf(i), (1) either is willing to receive a message, msg, from the manager, and then to change, 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 thetwo "things" (1 or 2) to do by a nondeterministic internal choice ($\sqcap$).

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

   stf_in: MSG → Σ → Σ,
   stf_out: Σ → Σ × 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 [5, 6, 7, 8]) processes.

4.5.4 Management and Organisation: Suggested ℜesearch Topics

ℜ 14. **Strategic, Tactical and Operation Management**: We made no explicit references to such "business school of administration" "BA101" topics as 'strategic' and 'tactical' management. Study Example 9.2 of Sect. 9.3.1 of Vol. 3 [1]. Study other sources on 'Strategic and Tactical Management'. Question Example 9.2's attempt at delineating 'strategic' and 'tactical' management. Come up with better or other proposals, and/or attempt clear, but not necessarily computable predicates which (help) determine whether an operation (above they are alluded to as 'stra' and 'tact') is one of strategic or of tactical concern.

ℜ 15. **Modelling Management and Organisation: Applicatively or Concurrently**: The abstraction of 'management and organisation' on Page 15 was applicative, i.e., a recursive function — whose auxiliary functions were hopefully all continuous. Suggest a CSP rendition of "the same idea" ! Relate the applicative to the concurrent models. **Hint:** Perhaps a notion of non-interference may be useful in achieving a congruence proof between applicative and concurrent models. Non-interference is satisfied if two concurrently operating behaviours access distinct, non-overlapping parts of a system state.

4.6 Rules & Regulations

Rules guide staff and govern equipment behaviour. Regulations advice on what to do when rules have been found violated.

Some rules apply to humans and other rules apply to equipment. In the following we focus on human-oriented rules & regulations.

4.6.1 Example Container Stowage Rules and Regulations

Some simple rules are: (i) heavier containers must not be stowed on top of lighter containers, (ii) heavier containers must be stowed near the vessel center of gravity, and (iii) reefer containers must be stowed near electric outlets. A corresponding simple regulation may be: (iv) when the above rules (i-ii-iii) have indeed been found violated, then this must be reported.

4.6.2 Example Railway Rules and Regulations

(i) In the Chinese Railway system there is the rule governing trains arrivals at train stations that at most one train may arrive in any three minuter interval, and the regulation that if this rule is found violated that the perpetrators be punished !

(ii) Around the world there is the rule governing train travel along segmented lines between stations that there — typically — must be an empty segment between any two trains along the line, and the regulation that if this rule is found violated that an auditor (a commission) be set up to decide who was at fault, and then, for those found at fault, the regulations stipulate appropriate administrative actions.

4.6.3 Definition of What Are Rules & Regulations

By a rule of an enterprise (an institution) we understand a syntactic piece of text whose meaning apply in any pair of actual present and potential next states of the enterprise, and then evaluates

to either true or false: the rule has been obeyed, or the rule has been (or will be, or might be) broken. By a regulation of an enterprise (an institution) we understand a syntactic piece of text whose meaning, for example, apply in states of the enterprise where a rule has been broken, and when applied in such states will change the state, that is, "remedy" the "breaking of a rule".

4.6.4 Abstraction of Rules and Regulations

Stimuli are introduced in order to capture the possibility of rule-breaking next states.

type
  Sti, Rul, Reg
  RulReg = Rul × Reg
  Θ
  STI = Θ → Θ
  RUL = (Θ × Θ) → **Bool**
  REG = Θ → Θ
value
  meaning: Sti → STI, Rul → RUL, Reg → REG
  valid: Sti × Rul → Θ → **Bool**
  valid(sti,rul)θ ≡ (meaning(rul))(θ,meaning(sti)θ)
axiom
  ∀ sti:Sti,(rul,reg):RulReg,θ:Θ •
    ∼valid(sti,rul)θ ⇒ meaning(rul)(θ,meaning(reg)θ)

4.6.5 Quality Control of Rules and Regulations

The axiom above presents us with a guideline for checking the suitability of (pairs of) rules and regulations in the context of stimuli: for every proposed pair of rules and regulations and for every conceivable stimulus check whether the stimulus might cause a breaking of the rule and, if so, whether the regulation will restore the system to an acceptable state.

4.6.6 Rules and Regulations Suggested ℜesearch Topic:

ℜ16. **A Concrete Case**: The above sketched a quality control procedure for 'stimuli, rules and regulations'. It left out the equally important 'monitoring' aspects. Develop experimentally two or three distinct models of domains involving distinct sets of rules and regulations. Then propose and study concrete implementations of procedures for quality monitoring and control of 'stimuli, rules and regulations'.

4.7 Scripts

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.

4.7.1 An Example Script Language

A Casually Described Bank Script. 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.

The State of a High Street Bank. 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.

The bank state is now described: there are clients (c:C), account numbers (a:A), mortgage numbers (m:M), account yields (ay:AY) and mortgage interest rates (mi:MI). The bank registers, by client, all accounts (ρ:A_Register) and all mortgages (μ:M_Register). To each account number there is a balance (α:Accounts). To each mortgage number there is a loan (ℓ:Loans). To each loan is attached the last date that interest was paid on the loan.

type
  C, A, M
  AY′ = **Real**, AY = {| ay:AY′ • 0<ay≤10 |}
  MI′ = **Real**, MI = {| mi:MI′ • 0<mi≤10 |}
  Bank′ = A_Register × Accounts × M_Register × Loans
  Bank = {| β:Bank′ • wf_Bank(β)|}
  A_Register = C $\overrightarrow{m}$ A-**set**
  Accounts = A $\overrightarrow{m}$ Balance
  M_Register = C $\overrightarrow{m}$ M-**set**
  Loans = M $\overrightarrow{m}$ (Loan × 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.

Wellformedness of the Bank State.

value
  ay:AY, mi:MI

  wf_Bank: Bank → **Bool**

wf_Bank(ρ,α,μ,ℓ) $\equiv$ $\cup$ **rng** ρ = **dom** α $\wedge$ $\cup$ **rng** μ = **dom** ℓ
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.

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.

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′))(ρ,α,μ,ℓ) $\equiv$
let (b,d) = ℓ(m) **in**
if α(a)≥p
then
let i = interest(mi,b,d′−d),
ℓ' = ℓ † [m↦ℓ(m)−(p−i)]
α' = α † [a↦α(a)−p,a_i↦$\alpha(a_i)$+i] **in**
((ρ,α',μ,ℓ'),ok) **end**
else
((ρ,α',μ,ℓ),nok)
end end
pre c ∈ **dom** μ $\wedge$ m ∈ μ(c)

Derived Bank Script: Loan Payment Transaction. From 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.

```
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
```

4.7.2 More on Script Development

This ends our example development of a script language. Details of the full — and further — development is given over 20 pages in Vol. 3, Chap. 11, Sect. 11.7.1 of [1].

4.7.3 Script Methodology: Suggested ℜesearch Topics

ℜ17. **License Languages**: We refer to a draft document available from the Internet: A Family of License Languages, incimplete R&D notes, March 4, 2007 at www.imm2.dtu.dk/~db/5-lectures/license-languages.pdf. Here is a fascinating fertile area of research.

4.8 Human Behaviour

By human behaviour we understand a "way" of representing entities, performing functions, causing or reacting to events or participating in behaviours. As such a human behaviour may be characterisable on a per phenomenon or concept basis as lying somewhere in the "continuous" spectrum from (i) diligent: precise representations, performances, event (re)actions, and behaviour interactions; via (ii) sloppy: occasionally imprecise representations, performances, event (re)actions, and behaviour interactions; and (iii) delinquent: repeatedly imprecise representations, performances, event (re)actions, and behaviour interactions; to (iv) criminal: outright counter productive, damaging representations, performances, event (re)actions, and behaviour interactions.

4.8.1 An Example: Ideal versus Human Behaviour

The bank script language of Sect. 4.7.1 gave us a semantics to the mortgage calculation request (i.e., command) as would a diligent bank clerk be expected to perform it. To express, that is, to model, how sloppy, delinquent, or outright criminal persons (staff?) could behave we must modify the int_Cmd(mkPM(c,a,m,p,d′))(ρ,α,μ,ℓ) definition (see Page 20).

```
int_Cmd(mkPM(c,a,m,p,d′))(ρ,α,μ,ℓ) ≡
    let (b,d) = ℓ(m) in
    if q(α(a),p) /∗ α(a)≤p∨α(a)=p∨α(a)≤p∨... ∗/
      then
        let i = f1(interest(mi,b,d′−d)),
            ℓ′ = ℓ † [ m↦f2(ℓ(m)−(p−i)) ],
            α′ = α † [ a↦f3(α(a)−p),a_i↦f4(α(a_i)+i),
```

```
                                a“staff” ↦ f“staff”(α(a_i)+i) ] in
            ((ρ,α′,μ,ℓ′),ok) end
        else ((ρ,α′,μ,ℓ),nok)
    end end
    pre c ∈ dom μ ∧ m ∈ μ(c)
  q: P × P ~→ Bool
  f1,f2,f3,f4,f“staff”: P ~→ P
```

The predicate q and the functions f_1, f_2, f_3, f_4 and $f_{\text{“staff”}}$ are deliberately left undefined. They are being defined by the "staffer" when performing (incl., programming) the mortgage calculation routine.

The point of this example is that one must first define the mortgage calculation script precisely as one would like to see the diligent staff (prgrammer) to perform (incl., correctly program) it before one can "pinpoint" all the places where lack of diligence may "set in". The invocations of q, f_1, f_2, f_3, f_4 and $f_{\text{“staff”}}$ designate those places.

4.8.2 Abstraction of Human Behaviour

We extend the formalisation of rules and regulations.

Human actions (**ACT**) lead from a state (Θ) to any one of possible successor states (Θ**-infset**) — depending on the human behaviour, whether diligent, sloppy, delinquent or having criminal intent. The **hum**an **int**erpretation of a rule (**Rul**) usually depends on the current state (Θ) and can be any one of a possibly great number of semantic rules (**RUL**). For a delinquent (...) user the rule must yield truth in order to satisfy "being delinquent (...)".

```
type
  ACT = Θ → Θ-infset
value
  hum_int: Rul → Θ → RUL-infset
  hum_behav: Sti × Rul → ACT → Θ → Θ-infset
  hum_behav(sti,rul)(α)(θ) as θs
    post θs = α(θ) ∧
         ∀ θ′:Θ • θ′ ∈ θs ⇒
             ∃ se_rul:RUL • se_rul ∈ hum_int(rul)(θ) ⇒ se_rul(θ,θ′)
```

Human **behav**iour is thus characterisable as follows: It occurs in a context of a **sti**mulus, a **rul**e, a present state (θ) and (the choice of) an action (α**:ACT**) which may have either one of a number of outcomes (θ**s**). Thus let θ**s** be the possible spread of diligent, sloppy, delinquent or outright criminal successor states. For each such successor states there must exist a rule interpretation which satisfies the pair of present an successor states. That is, it must satisfy being either diligent, sloppy, delinquent or having criminal intent and possibly achieving that!

4.8.3 Human Behaviour Suggested ℜesearch Topics:

Section 11.8 of Vol. 3 [1] elaborates on a number of ways of describing (i.e., modelling) human behaviour.

ℜ 18. **Concrete Methodology**: Based on the abstraction of human behaviour given earlier, one is to study how one can partition the set, $\alpha(\theta)$, of outcomes of human actions into 'diligent', 'sloppy', 'delinquent' and 'criminal' behaviours — or some such, perhaps cruder, perhaps finer partitioning — and for concrete cases attempt to formalise these for possible interactive "mechanisation".

ℜ 19. **Monitoring and Control of Human Behaviour**: Based on possible solutions to the previous research topic one is to study general such interactive "mechanisation" of the monitoring and control of human behaviour.

4.9 Domain Modelling: Suggested ℜesearch Topic

ℜ 20. **Sufficiency of Domain Facets**: We have covered five facets: intrinsics, support technology, management and organisation, rules and regulations and human behaviour. The question is: are these the only facets, i.e., views on the domain that are relevant and can be modelled? Another question is: is there an altogether different set of facets, "cut up", so-to-speak, "along other lines of sights", using which we could likewise cover our models of domains?

One might further subdivide the above five facets (intrinsics, support technology, management and organisation, rules and regulations and human behaviour) into "sub"-facets. A useful one seems to be to separate out from the facet of rules and regulations the sub-facet of scripts.

5 From Domains to Requirements

5.1 What Is Required ? — The Machine !

Requirements prescribe what "the machine", i.e., the hardware + software is expected to deliver. In Vol. 3, Part V, Chaps. 17–24 we present a through coverage of requirements engineering, and Chap. 19, Sects. 19.4–19.5 we show how to construct, from a domain description, in collaboration with the requirements stakeholders, the domain (i.e., functional) requirements, and the interface (i.e., user) requirements.

5.2 Three Kinds of Requirements

The way I see it, there are only three kinds of requirements: (i) domain requirements — which are those requirements which can be expressed sôlely using ordinary language and professional terms from the domain; (ii) interface requirements — which are those requirements which can be expressed sôlely using ordinary language and professional terms from the domain and from the machine (i.e., both); and (iii) machine requirements — which are those requirements which can be expressed sôlely using ordinary language and professional terms from the machine — no terms, ideally speaking, from the domain ! We shall only cover domain and interface requirements — since only they involve domain descriptions.

5.3 Domain Requirements

To construct the domain requirements is, logicaly speaking, very straightforward. There are basically five steps in the process of "turning" a domain description into a requirements prescription. All steps are conducted jointly by the domain engineer together with the requirements stakeholders. In all steps they together "read" the domain description, "line-by-line" — performing, in five rounds, these domain-to-requirements transformations, i.e., these requirements engineering steps:

1. projection — should a what is described in a description line become a property "carried over" into the prescription, i.e., of the required software;
2. instantiation — should a generality of what might be described be instantiated to special cases, and, if so, then refine possible types, functions and axioms;
3. determination — should what may be nondeterministically described be made mor deterministic, and, if so, then refine possible types, functions and axioms;
4. extension — are there possible phenomena or concepts that were excluded from the domain description because they were infeasible in the domain which are now computationally tractable; and
5. fitting — are there developments of other requirements with which the present one can be "interfaced", i.e., fitted ?

5.3.1 Example Projections: Container Line Industry

We suggest two distinct projections. (p1) **Bill of Lading Requirements:** net of sea lanes, container liners, bill of ladings and logistics. (p2) **Ship Stowage Requirements:** container vessel stowage and bill of ladings.

5.3.2 Example Instantiations: Container Line Industry

We suggest two distinct instantiations. (i1) **Ship Stowage Requirements:** To be instantiated to Maersk Line: container vessel stowage, bill of ladings and container lines. (i2) **Container Vessel Scheduling Requirements:** To be instantiated to Maersk Line: container vessels, container lines, bill of ladings and net of sea lanes instantiated to Europe and Asia.

5.3.3 Example Determinations: Container Line Industry

We suggest two distinct determinations. (d1) **Ship Stowage Requirements:** bay, row and tier identifiers are natural numbers and — via a map — designate ordered physical vessel locations; and preference is to be given to stowage priority of "favoured" customers and then stowage priority of intra-Asia transports. (d2) **Crane Split Requirements:** Preference is to be given to: crane allocation to largest container vessels and then crane allocation to two-continent transports.

5.3.4 Example Extensions: Container Line Industry

We suggest two distinct extensions. (e1) **Ship Stowage Requirements:** near optimal stowage with minimum interaction with human stowage planners. (e2) **Vehicle Job Task Planning Requirements:** near optimal stowage with minimum interaction with human job task planners.

5.3.5 Example Fittings: Container Line Industry

We suggest two distinct fittings. (f1) **Database Requirements:** shared database for all the container related requirements mentioned so far. (f2) **Crane-splitting and Vehicle Job Task Planning Requirements:** obvious !

5.4 Interface Requirements

By 'interface' we shall understand that which is shared between the domain and the machine: entities, functions, events and behaviours.

So in constructing the interface requirements we perform the following requirements engineering steps:

1. entity sharing: which is the information (i.e., the date) that need be initially input to and regularly refreshed by, or in, the machine (and how) — thus leading to prescriptions of bulk data initialisation and more-or-less regular data refreshment;
2. function sharing: which are the functions that need be performed both by man and machine (and how) — leading to prescriptions of man-machine dialogues for interactive computing;
3. event sharing: which are the external, non-machine generated events that need be communicated to the machine and which are the internal, machine generated events that need be communicated to environment (and how); and
4. behaviour sharing: which are the domain behaviours that are to be 'mimicked' by the machine (and how).

5.4.1 Example: Shared Container Line Industry Entities

We can trivially identify the following shared entities — depending, of course, on the specific requirements being established: the net of sea lanes, container vessels, containers, etcetera.

5.4.2 Example: Shared Container Line Industry Functions

We can trivially identify the following shared functions — depending, of course, on the specific requirements being established: transfer of containers, whether between ship and vehicle or between vehicle and stack, etc., shipping of a container: sending off and delivery, etcetera.

5.4.3 Example: Shared Container Line Industry Events

We can trivially identify the following shared events — depending, of course, on the specific requirements being established: arrival and departure of ship to, respectively from a container terminal port, the beginnings and ends of unloading and loading, etcetera.

5.4.4 Example: Shared Container Line Industry Behaviours

We can trivially identify the following shared events — depending, of course, on the specific requirements being established: the behaviour of a ship while at port, the behaviour of a container from being unloaded from a ship till being transferred to an inland truck, etc. etcetera.

6 Requirements-Specific Domain Software Development Models

A long term, that one: 'requirements-specific domain software development models' ! The term is explained next.

6.1 Software "Intensities"

One can speak of 'software intensity'. Here are some examples. Compilers represent 'translation' intensity. 'Word processors', 'spread sheet systems', etc., represent "workpiece" intensity. Databases represent 'information' intensity. Real-time embedded software represent 'reactive' intensity. Data communication software represent connection intensity. Etcetera.

6.2 "Abstract" Developments

Let $''\mathcal{R}''$ denote the "archetypal" requirements for some specific software 'intensity'. Many different domains $\{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_i, \ldots, \mathcal{D}_j, \ldots\}$ may be subject to requirements $''\mathcal{R}''$-like prescriptions. For each such a set of possible software may result. The "pseudo-formula" below attempts, albeit informally, to capture this situation:

$$\left\{ \begin{array}{c} \mathcal{D}_1 \\ \mathcal{D}_2 \\ \ldots \\ \mathcal{D}_i \\ \ldots \\ \mathcal{D}_k \\ \ldots \end{array} \right\} \sim ''\mathcal{R}'' \mapsto \left[\begin{array}{c} \{\mathcal{S}_{1_1}, \mathcal{S}_{1_2}, \ldots, \mathcal{S}_{1_{j_1}}, \ldots\} \\ \{\mathcal{S}_{1_1}, \mathcal{S}_{1_2}, \ldots, \mathcal{S}_{1_{j_2}}, \ldots\} \\ \ldots \\ \{\mathcal{S}_{i_1}, \mathcal{S}_{i_2}, \ldots, \mathcal{S}_{i_{j_i}}, \ldots\} \\ \ldots \\ \{\mathcal{S}_{k_1}, \mathcal{S}_{k_2}, \ldots, \mathcal{S}_{k_{j_k}}, \ldots\} \\ \ldots \end{array} \right]$$

Several different domains, to wit: road nets and railway nets, can be given the "same kind" of (road and rail) maintenance requirements leading to information systems. Several different domains, to wit: road nets, railway nets, shipping lanes, or air lane nets, can be given the "same kind" of (bus, train, ship, air flight) monitoring and control requirements (leading to real-time embedded systems). But usually the specific requirements skills determine much of the requirements prescription work and especially the software design work.

6.3 Requirements-Specific Devt. Models: Suggested ℜesearch Topics

ℜ 21_j. **Requirements-Specific Development Models,** $\mathcal{RSDM}_j$: We see these as grand challenges: to develop and research a number of requirements-specific domain (software) development models $\mathcal{RSDM}_j$.

The "pseudo-formal" $\prod(\sum_i^{\cdots} \mathcal{D}_i)$ $''\mathcal{R}''$ $\sum_{i,j}^{\cdots} \mathcal{S}_{i_j}$ expression attempts to capture an essence of such research: The $\prod$ "operator" is intended to project (that is, look at only) those domains, $\mathcal{D}_i$, for which $''\mathcal{R}''$ may be relevant. The research explores the projections $\prod$, the possible $''\mathcal{R}''$s and the varieties of software $\sum_{i,j}^{\cdots} \mathcal{S}_{i_j}$.

7 On Two Reasons for Domain Modelling

Thus there seems to be two entirely different, albeit, related reasons for domain modelling: one justifies domain modelling on engineering grounds, the other on scientific grounds.

7.1 An Engineering Reason for Domain Modelling

In an e-mail, in response, undoubtedly, to my steadfast, perhaps conceived as stubborn insistence, on domain engineering, Sir Tony Hoare summed up his reaction, in summer of 2006, to domain engineering as follows, and I quote[5]:

"There are many unique contributions that can be made by domain modelling.

1. The models describe all aspects of the real world that are relevant for any good software design in the area. They describe possible places to define the system boundary for any particular project.
2. They make explicit the preconditions about the real world that have to be made in any embedded software design, especially one that is going to be formally proved.
3. They describe[6] the[7] whole range of possible designs for the software, and the whole range of technologies available for its realisation.
4. They provide a framework for a full analysis of requirements, which is wholly independent of the technology of implementation.
5. They enumerate and analyse the decisions that must be taken earlier or later in any design project, and identify those that are independent and those that conflict. Late discovery of feature interactions can be avoided."

All of these issues are dealt with, one-by-one, and in some depth, in Vol. 3 [1] of my three volume book.

7.2 A Science Reason for Domain Modelling

So, inasmuch as the above-listed issues of Sect. 7.1, so aptly expressed in Tony's mastery, also of concepts (through his delightful mastery of words), are of course of utmost engineering importance, it is really, in our mind, the science issues that are foremost: We must first and foremost understand. There is no excuse for not trying to first understand. Whether that understanding can be "translated" into engineering tools and techniques is then another matter. But then, of course, it is nice that clear and elegant understanding also leads to better engineering. It usually does.

[5] E-Mail to Dines Bjørner, CC to Robin Milner et al., July 19, 2006
[6] read: imply
[7] read: a

7.3 Domains Versus Requirements-Specific Development Models

Sir Tony's five statements are more related, it seems, to the concept of requirements-specific domain software development models than to merely the concept of domain models. His statements help us formulate the research programme ℜ 21 of requirements specific domain software development models. When, in his statements, you replace his use of the term 'models' with our term 'requirements-specific development models *based on domain models*', then "complete harmony" between the two views exists.

8 Conclusion

8.1 What Has Been Achieved ?

I set out to focus on what I consider the crucial modelling stage of describing domain facets and to identify a number of their research issues. I've done that. Cursorily, the topic is "near-holistic", so an overview is all that can be done. The issue is that of that of a comprehensive methodology. Hence the "holism" challenge.

8.2 What Needs to Be Achieved ?

Well, simply, to get on with that research. There are two sides to it: the 21 research topics mentioned above, and the ones mentioned below. The latter serves as a carrier for the former research.

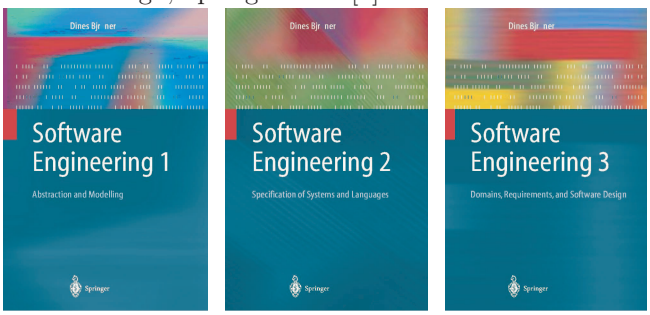
8.3 Domain Theories: Grand Challenge ℜesearch Topics

The overriding research topic is that of:

ℜ 22_i. **Domain Models:** $\mathcal{D}_i$: We see this as a set of grand challenges: to develop and research a family of domain models $\mathcal{D}_i$.

8.4 What Have We Not Covered ?

Many things. Hence study Vol. 3 of my three volume book: Software Engineering, Domains, Requirements and Software Design, Springer 2006 [1].



8.5 Acknowledgemenents

Thanks are due to Prof. Alfredo Ferro (University of Catania) and Prof. Egon Boerger (University of Pisa), the co-chairs of this summer school for inviting me. Thanks are due to all the patient summer school participants who "sat through" the "flasing" of many slides, etc.

9 Bibliographical Notes

References

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

[2] Richard Feynmann, Robert Leighton, and Matthew Sands. *The Feynmann Lectures on Physics*, volume Volumes I–II–II. Addison-Wesley, California Institute of Technology, 1963.

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

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

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

[6] A. W. Roscoe. *Theory and Practice of Concurrency*. C.A.R. Hoare Series in Computer Science. Prentice-Hall, 1997.

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

[8] Tony Hoare. Communicating Sequential Processes. Published electronically: `http://www.-usingcsp.com/cspbook.pdf`, 2004. Second edition of [5]. See also `http://www.usingcsp.-com/`.

[9] Mordecai Avriel, Michal Penn, and Naomi Shpirer. Container ship stowage problem: complexity and connection to the coloring of circle graphs. *Discrete Applied Mathematics*, 103(1–3):271–279, 15 July 2000. Faculty of Industrial Engineering and Management, Technion, Israel Institute of Technology, Haifa 3200, Israel. This paper deals with a stowage plan for containers in a container ship. Since the approach to the containers on board the ship is only from above, it is often the case that containers have to be shifted. Shifting is defined as the temporary removal from and placement back of containers onto a stack of containers. Our aim is to find a stowage plan that minimizes the shifting cost. We show that the shift problem is NP-complete. We also show a relation between the stowage problem and the coloring of circle graphs problem. Using this relation we slightly improve Unger's upper bound on the coloring number of circle graphs.

[10] Mordecai Avriel, Michal Penn, Naomi Shpirer, and Smadar Witteboon. Stowage planning for container ships to reduce the number of shifts. *Annals of Operations Research*, 76(9):55–71, January 1998. This paper deals with a stowage plan for containers in a container ship. Containers on board a container ship are placed in vertical stacks, located in many bays. Since the access to the containers is only from the top of the stack, a common situation is that containers designated for port J must be unloaded and reloaded at port I (before J) in order to access containers below them, designated for port I. This operation is called shifting. A container ship calling at many ports may encounter a large number of shifting operations, some of which can be avoided by efficient stowage planning. In general, the stowage plan must also take into account stability and strength requirements, as well as several other constraints on the placement of containers. In this paper, we only deal with stowage planning in order to minimize the number of shiftings, without considering stability and several other constraints. First, we briefly present a 0-1 binary linear programming formulation that can find the optimal solution for stowage planning. However, finding the optimal solution using

this model is quite limited because of the large number of binary variables and constraints needed for the formulation. Moreover, in [9] the stowage planning problem is shown to be NP-complete. For these reasons, the Suspensory Heuristic Procedure was developed.

[11] Bureau Export. A-Z Dictionary of Export, Trade and Shipping Terms. www.exportbureau.-com/trade_shipping_terms/dictionary.html, 2007.

[12] International Labour Organisation. Portworker Development Programme. www.ilo.org/public/english/dialogue/sector/sectors/pdp/index.htm, April 2002.

[13] International Labour Organisation. Portworker Development Programme: PDP Units. The PDP unit titles are listed next. Internet access to these are coded as: www.ilo.org/public/english/dialogue/sector/sectors/pdp/l-δ-δ'.htm where the letter (l) and the digits (δ and δ') are shown as ($l.\delta.\delta'$) below:

Container terminal operations (c.1.1)
Container ship loading and discharging operations (c.1.2)
The container terminal quay transfer operation (c.1.3)
The container yard: the storage operation (c.1.4)
The container terminal receipt/delivery operation (c.1.5)
Container freight station operations (c.1.6)
Container ship construction (c.2.1)
Container ship stowage plans (c.2.2)
Container securing systems (c.2.3)
Container ship loading/discharge lists and workplans (c.2.4)
Container construction (c.3.1)
Container numbering and marking (c.3.2)
Container inspection (c.3.3)
Packing of goods in containers: 1. Principles and planning (c.3.4)
Packing of goods in containers: 2. Working practices (c.3.5)
Safe working on container terminals (c.4.1)
Safe working aboard container vessels (c.4.2)
The container terminal and international trade (c.6.1)
Measuring container terminal performance (c.6.2)
Analysis and review of container terminal performance (c.6.3)
Handling dangerous cargoes in ports (p.3.1)
The port supervisor: organizational status (s.1.1)
The port supervisor: tasks and duties (s.1.2)
The port supervisor: supervisory skills (s.1.3)
The port supervisor: personal attributes (s.1.4)
Supervision of container ship discharge and loading (s.2.1)
Supervision of the container terminal quay side transfer operation (s.2.2)
Supervision of container yard operations (s.2.3)
Supervision of the container terminal receipt/delivery operation (s.2.4)
Supervision of container freight stations (s.2.5)

, April 2002.

[14] Mordecai Avriel and Michal Penn. Exact and approximate solutions of the container ship stowage problem. In C. Patrick Koelling, editor, *Proceedings of the 15th annual conference on Computers and industrial engineering*, pages 271–274, Elmsford, NY, USA, September 1993. Pergamon Press, Inc. Also published in: Computers and Industrial Engineering Volume 25 , Issue 1-4 Sept. 1993. This paper deals with a stowage plan for containers in a container ship. Containers on board a container ship are placed in stacks, located in many bays. Since the access to the containers is only from the top of the stack, a common situation is that contianers designated for port J must be unloaded and reloaded at port I (before J) in order to access containers below them, designated for port I. This operation is called shifting. A container ship calling many ports, may encounter a large number of shifting operations, some of which can be avoided by efficient stowage planning. In general, the stowage plan must also

take into account stability and strength requirements, as well as several other constraints on the placement of containers. In this paper we deal with stowage planning in order to minimize the number of shiftings, without considering stability constraints. First, a 0-1 binary linear programming formulating is presented that can find the optimal solution for stowage in a single rectangular bay of a vessel calling a given number of ports, assuming that the number of constainers to ship is known in advance. This model was successfully implemented using the GAMS software system. It was found, however, that finding the optimal solution using this model is quite limited, because of the large number of binary variables needed for the formulation. For this reason, several alternative heuristic algorithms were developed. The one presented here is based on a reduced transportation matrix. Containers with the same source and destination ports are stowed in full stacks as much as possible, and only the remaining containers are allocated by the binary linear programming model. This approach often allows the stowage planning of a much larger number of containers than using the exact formulation.

[15] Opher Dubrovsky, Gregory Levitin, and Michal Penn. A genetic algorithm with a compact solution encoding for the container ship stowage problem. *Journal of Heuristics*, 8(6):585–599, November 2002. The purpose of this study is to develop an efficient heuristic for solving the stowage problem. Containers on board a container ship are stacked one on top of the other in columns, and can only be unloaded from the top of the column. A key objective of stowage planning is to minimize the number of container movements. A genetic algorithm technique is used for solving the problem. A compact and efficient encoding of solutions is developed, which reduces significantly the search space. The efficiency of the suggested encoding is demonstrated through an extensive set of simulation runs and its flexibility is demonstrated by successful incorporation of ship stability constraints.

[16] I.D. Wilson and P.A. Roach. Container stowage planning: a methodology for generating computerised solutions. *Journal of the Operational Research Society*, 51(11):1248–1255, 1 November 2000. Palgrave Macmillan. University of Glamorgan, UK. The container stowage problem concerns the suitable placement of containers in a container-ship on a multi-port journey; it requires consideration of the consequences each placement has on decisions at subsequent ports. A methodology for the automatic generation of computerised solutions to the container stowage problem is shown; objective functions that provide a basis for evaluating solutions are given in addition to the underlying structures and relationships that embody this problem. The methodology progressively refines the placement of containers within the cargo-space of a container ship until each container is specifically allocated to a stowage location. The methodology embodies a two stage process to computerised planning, that of a generalised placement strategy and a specialised placement procedure. Heuristic rules are built into objective functions for each stage that enable the combinatorial tree to be explored in an intelligent way, resulting in good, if not optimal, solutions for the problem in a reasonable processing time.

[17] I.D. Wilson, P.A. Roach, and J. A. Ware. Container stowage pre-planning: using search to generate solutions, a case study. *Knowledge-Based Systems*, 14(3–4):137–145, June 2001. Container-ships are vessels possessing an internal structure that facilitates the handling of containerised cargo. At each port along the vessel's journey, containers destined for those ports are unloaded and additional containers destined for subsequent ports are loaded. Determining a viable arrangement of containers that facilitates this process, in a cost-effective way, constitutes the deep-sea container-ship stowage problem. This paper outlines a computer system that generates good sub-optimal solutions to the stowage pre-planning problem. This is achieved through an intelligent analysis of the domain allowing the problem to be divided into sub-problems: a generalised placement strategy and a specialised placement procedure. This methodology progressively refines the arrangement of containers within the cargo-space of a container ship until each container is specifically allocated to a stowage location. Good, if not optimal, solutions for the problem are obtained in a reasonable processing time through the use of heuristics incorporated into objective functions for each stage.

[18] Akio Imai, Kazuya Sasaki, Etsuko Nishimura, and Stratos Papadimitriou. Multi-objective simultaneous stowage and load planning for a container ship with container rehandle in yard stacks. *European Journal of Operational Research*, 171:373–389, 2006. imai@kobe-u.ac.jp. Received 14 December 2002; accepted 27 July 2004. (1) Faculty of Maritime Sciences, Kobe University, Fukae, Higashinada, Kobe 658-0022, Japan and (2) World Maritime University, P.O. Box 500, S-201 24 Malmo, Sweden, and (3) c Department of Maritime Studies, University of Piraeus, 80 Karaoli and Dimitriou Street, GR-185 32 Piraeus, Greece.
The efficiency of a maritime container terminal primarily depends on the smooth and orderly process of handling containers, especially during the ship's loading process. The stowage and associated loading plans are mainly determined by two criteria: ship stability and the minimum number of container rehandles required. The latter is based on the fact that most container ships have a cellular structure and that export containers are piled up in a yard. These two basic criteria are often in conflict. This paper is concerned with the ship's container stowage and loading plans that satisfy these two criteria. The GM, list and trim are taken into account for the stability measurements. The problem is formulated as a multi-objective integer programming. In order to obtain a set of noninferior solutions of the problem, the weighting method is employed. A wide variety of numerical experiments demonstrated that solutions by this formulation are useful and applicable in practice.

[19] K.V. Ramani. An interactive simulation model for the logistics planning of container operations in seaports. *SIMULATION*, 66(5):291–300, 1996. Indian Institute of Management Ahmedabad, India 380015. Today, more than 90 percent of international cargo moves through seaports, and 80 percent of seaborne cargo moves in containers. It has thus become imperative for major seaports to manage their container operations both effec tively and efficiently. We have designed and developed an interactive computer simula tion model to support the logistics planning of container operations. Logistics planning of container operations deals with the assign ment and coordination of port equipments such as quay cranes, prime movers, and yard cranes in the transportation of containers between the ship's bay and the container storage yards. This model provides estimates for port performance indicators such as berth occupancy, ship outputs, and ship turnaround time for various operating strategies in the logistics planning of container operations. The main objective of port management is to reduce the ship turnaround time by optimally utilizing the port resources. Reduced turn around time encourages trade and improves the competitiveness of the port to provide efficient and effective services at low cost.

[20] Dirk Steenken, Stefan Voß, and Robert Stahlbock. Container terminal operation and operations research - a classification and literature review. *OR Spectrum*, 26(1):3–49, January 2004. In the last four decades the container as an essential part of a unit-load-concept has achieved undoubted importance in international sea freight transportation. With ever increasing containerization the number of seaport container terminals and competition among them have become quite remarkable. Operations are nowadays unthinkable without effective and efficient use of information technology as well as appropriate optimization (operations research) methods. In this paper we describe and classify the main logistics processes and operations in container terminals and present a survey of methods for their optimization.

[21] US DoD. Stability and Buoyancy Lessons: Surface Officer Warfare School Documents. Course notes: www.fas.org/man/dod-101/navy/docs/swos/dca/index.html. Course: www.fas.org/man/dod-101/navy/docs/swos/dca/stg4-01.html covers Principles of ship stability.

A An Example: A Container Line Industry

A.1 Overview of The Container Line Industry

We consider the following phenomena and concepts to be basic to the container line industry: the acceptance, transport over large distances, possibly by means of more than one voyage, and then possibly with temporary storage and final delivery of containers.

We shall consider major phenomena and concepts of this industry to include (i) containers, (ii) container vessels, (iii) container vessel stowage, (iv) container terminal ports which includes (iv.1) (berthed) vessels, (iv.2) quays, (iv.3) stacks, (iv.4) quay and stack cranes, (iv.5) vehicles, (iv.6) transfer area and (iv.7) ("inland") trucks, trains and barges, (iv) nets of sea lanes, (v) container lines, (vi) bill-of-ladings and (vii) logistics.

A.2 Containers

By a container we understand an item of equipment as defined by the International Organisation for Standardisation (ISO) for transport purposes. It must be of:

- a permanent character and accordingly strong enough to be suitable for repeated use,
- specially designed to facilitate the carriage of goods, by one or more modes of transport without intermediate reloading,
- fitted with devices permitting its ready handling, particularly from one mode of transport to another,
- so designed as to be easy to fill and empty,
- having an internal volume of $1m^3$ or more. The term container includes neither vehicles nor conventional packing.

This definition is now "sharpened" to reflect current container shipping practices:

1. With any container we associate a unique identifier.
2. Containers have weights (from 0 up!).
3. Containers have same widths and heights, but any one of a few "standard" lengths. We shall consider only $20'$, $40'$, and $45'$ containers, usually measuring
   (a) $20'$ length: 5898 mm, width 2352 mm (W), and height 2393 mm (H), or
   (b) $40'$ length: 12032 mm, W, H, or
   (c) $45'$ length: 13556 mm, W, H.
4. We currently abstract from whether the container is of kind: a general purpose, a refrigerated, a hanging garment, a reefer, an open top, a fantainer, or a flat rack container.
5. We shall in the following associate many additional properties with containers — such as check digit, lease, load plan, manifest, number, owner, prefix, serial number, size code, size/type, type code, etc.

For our purposes we shall model a container as a value c in C from which all its evolving properties (static or dynamic) can be observed:

type
  C, CId, W, Le, Wi, He
  Kind == gepu | refr | haga | reef | opto | fata | ...
value
  obs_CNm: C → CNm, obs_W: C → W

obs_Le: C → Le, obs_Wi: C → Wi, obs_He: C → He
obs_Kind: C → Kind
axiom
∀ c,c′:C • c≠c′ ⇒
obs_CNm(c)≠obs_CNm(c′) ∧
obs_Wi(c)=obs_Wi(c′) ∧ obs_He(c)=obs_He(c′)

The[8] axiom expresses that no two containers have the same unique container identifier, and that all containers, as for cellular vessels (see below), have same width and height.

We show only this fragment of modelling containers here. Additional fragments will appear later in this modelling of the domain of container shipping.

A.3 Container Vessels

A.3.1 Basics

By a container vessel we understand a ship that has own locomotive force, and which can move (i.e., transport) freight (i.e., containers] across the open sea, from harbours (container terminal ports) to harbours, through canals, along rivers, and across lakes, i.e., from location to location.

A.3.2 Container Bays, Rows, Tiers and Cells

With container ships we will in addition wish to associate a number of properties:

6. We shall restrict ourselves to cellular vessels, i.e., vessels specially designed and equipped for the carriage of containers.
7. A cell, then, is a location on board of a container vessel where one container can be stowed.
8. A cell position is the location of a cell on board of a container vessel identified by a code for successively the bay, the row, the tier and tier index position), indicating the position of a container on that vessel.
9. A bay is a vertical division of a cellular vessel from stem to stern, used as a part of the indication of a stowage place for containers. The numbers run from stem to stern; odd numbers indicate a 20 foot position, even numbers indicate a 40 foot position. A bay consists of one or more rows.
10. A bay plan is stowage plan which shows the locations of all the containers on the vessel.
11. A row is a vertical division of a vessel from starboard to port side, used as a part of the indication of a stowage place for containers. The numbers run from midships to both sides. A row consists of one or more tiers (containing a fixed number of tier cells.

8

We shall, laboriously, explain, by "reading", the formal, mathematical specification language notation:
RSL.1: **type** A introduces a sort, i.e., a class of — at the moment — further undefined entities (though 'of type A').
RSL.2: **type** B == nm1 | nm2 | ... | nmn introduces a variant class (of name B) whose atomic and distinct elements appears to the right of the ==, i.e., nm1, nm2, ..., nmn.
RSL.3: **value** obs_Y: X → Y introduces an observer function which applies to entities, x, of type X and yields entities of type Y. These latter are said to be either attributes of or sub-entities contained in x.
RSL.4: **axiom** $\mathcal{P}$ expresses that a certain property, $\mathcal{P}$ holds over the entities mentioned in $\mathcal{P}$.
RSL.5: ∀ x:X,y:Y,...,z:Z • p(x,y,...,z) expresses that the predicate p(x,y,...,z) is expected to hold for all entities x of type X, entities y of type Y, etc.
RSL.6: p ∧ q expresses that both predicates p and q are expected to hold.
RSL.7: p ∨ q expresses that either predicate p is expected to hold or predicate q holds (or both holds).

12. A tier is a horizontal division of a vessel from bottom to top. The tier numbered positions (tier indexes) run from bottom to deck and from deck upwards and are used as a part of the indication of a stowage place for containers.

13. A cell is either empty or stows a container.

We thus abstract a container vessel as consisting, statically: of a number of identified bays, (for each identified bay) of a number of identified rows, and (for each identified row) of a number of identified tiers. That is, an identified bay and, within it, an identified row, and within it, an identified tier defines a set of tier positions (or cells), and possessing the following overall static properties (attributes): a container vessel unique name (CVNm), and a velocity profile which maps the load profile onto a mean velocity. and the following overall dynamic properties (attributes): the current load profile: which cells are occupied and then possibly with an estimated (total) weight, the current direction, velocity and acceleration of the vessel and the current position on the high seas or in some harbour. Three important quantities to be considered during load planning[9] are the GM: distance between the ship's center of gravity[10] and the meta-center position[11]; the current list: inclination of a ship to port or starboard caused by eccentric weights such as cargo or ballast (same as 'heel'); and the current trim: the difference between the forward and after drafts[12], in excess of design drag[13].

We shall build on the below formalisation as the presentation unfolds.[14]

```
type
  CV
  B, R, T, Bi, Ri, Ti
  Cell == mkC(c:C) | empty
value
  obs_Bs: CV → B-set, obs_Rs: B → R-set, obs_Ts: R → T-set
  obs_Bi: B → Bi, obs_Ri: R → Ri, obs_Ti: T → Ti
  obs_Ti_Max: T → Nat, obs_Cells: T → Cell*
axiom [ no un−occupied gaps ]
  ∀ t:T • obs_Ti_Max(t)≥1
      ∧ let max=obs_Ti_Max(t), cells=obs_Cells(t) in
          len cells=max ∧ ∃ i:{1..max} •
              cells(i)=empty ⇒ ∀ j:{i+1..max}•cells(j)=empty end
```

[15]We shall not build on the below formalisation as the presentation unfolds.

```
type
  CVNm
```

[9]Load planning: determining which containers, from a container terminal port container stack, goes where on a container vessel.

[10]Center of gravity: Point at which the entire weight of a body may be considered as concentrated so that if supported at this point the body would remain in equilibrium in any position.

[11]As a ship is inclined through small angles of heel (listing), the lines of buoyant force intersect at a point called the meta-center. As the ship is inclined, the center of buoyancy moves in an arc as it continues to seek the geometric center of the underwater hull body. This arc describes the meta-centric radius.The meta-center is a fictitious point. If the meta-centric height is zero or negative, the vessel will heel (list) or capsize.

[12]Draft: The number of feet that the hull of a ship is beneath the surface of the water.

[13]Design drag: A design feature where the draft aft is greater than the draft forward; assume 0.

[14]

RSL.8: A == mkB(b:B) | void expresses that A is a type consisting of the singleton, distinct and atomic element void together with the class of all elements mkB(b) for all entities b in class B. Think of the mkB as "markings" such that if A == mkB1(b:B) | mkB2(b:B) then for any entity b in class B, mkB1(b) is distinct from mkB2(b).
RSL.9: A-set is a type expression. It denotes the class whose elements are finite sets of entities of type A.
RSL.10: A* is a type expression. It denotes the class whose elements are finite lists (sequences) of entities of type A.
RSL.11: **Nat** is a type literal, i.e., a type expression. It denotes the class of all natural numbers.

[15]

VelPro, LdPro, Pos, GM, List, Trim, FwdDrag, AftDrag
value
obs_CVNm: CV → CVNm
obs_LdPro: CV → LdPro
obs_VelPro: CV → VelPro
obs_Pos: CV → Pos
obs_List: CV → List, obs_Trim: CV → Trim,
obs_FwdDrag: CV → FwdDrag, obs_AftDrag: CV → AftDrag
axiom
∀ cv,cv′:CV • cv≠cv′ ⇒ obs_CVNm(cv)≠obs_CVNm(cv′)

We have modelled bay, row and tier identifiers as arbitrary (albeit finite, enumerable) sets of further unidentified tokens. This modelling choice allows us to implement these identifiers in any way we so wish. For example as numbers (integers, or even natural numbers). But the modelling choice begs an answer to the following question. On any one real container vessel these identifiers, cum "numbers", enjoy an ordering relation as well as there being first and last identifiers. The question now is: How to model that ? To answer that we must first assume that all set of bays and set of rows consists of uniquely identifier bays and rows.[16]

axiom
∀ b,b′:Bt • b≠b′ ⇒ obs_Bi(b)≠obs_Bi(b′)
∀ r,r′:R • r≠r′ ⇒ obs_Ri(r)≠obs_Ri(r′)
∀ t,t′:T • t≠t′ ⇒ obs_Ti(r)≠obs_Ti(r′)
value
xtr_Bis: B-**set**→Bi-**set**, xtr_Ris: R-**set**→Ri-**set**, xtr_Tis: T-**set**→Ti-**set**
xtr_Bis(bs) ≡ {obs_Bi(b)|b:B•b ∈ bs}
xtr_Ris(rs) ≡ {obs_Ri(r)|r:R•r ∈ rs}
xtr_Tis(rs) ≡ {obs_Ti(t)|t:T•t ∈ ts}
axiom
∀ bs:B-**set**,rs:R-**set**,ts:T-**set** •
card bs=**card** xtr_Bis(bs) ∧ **card** rs=**card** xtr_Ris(rs) ∧ **card** ts=**card** xtr_Tis(ts)

The axiom above expresses that all bays and rows of respective sets of these are uniquely identified.

value
is_LOC_of_CV: LOC × CV → **Bool**
is_LOC_of_CV(bi,ri,ti)(cv) ≡
let bis = obs_Bis(cv) **in**
if bi ∉ bis **then false else**
let b = xtr_B(bi)(cv) **in**
let ris = obs_Ris(b) **in**
if ri ∉ ris **then false else**
let r = xtr_R(ri,b) **in**
let tis = obs_Tis(r) **in**
if ti ∉ tis **then false else true end**
end end end end end end end

select_C: LOC → CV $\xrightarrow{\sim}$ C | null
select_C(bi,ri,ti)(cv) ≡
let bis = obs_Bis(cv) **in**
if bi ∉ bis **then chaos else**

Referring to the axiom on Page 34:
RSL.12: **let** a = expr1 **in** expr2 **end** introduces a local name a to have the value of expression expr1 such that a can now be used in expression expr2 having that fixed value.
RSL.13: ∃ i:{1..max} • p(i) expresses that there should exist, in this case a natural number within the range of 1 to max such that the predicate p(i) holds.
RSL.14: list(i) expresses that if list is indeed a list and i an index into that list, then the i'th list element is yielded.

16

RSL.15: {f(x)|x:X•p(x)} comprehends the set consisting of all those f(x)'s such that x is of type X and the predicate p(x) holds.
RSL.16: **card** s yields the cardinality, i.e., the number of zero, one or more elements in the finite set s. (If s is infinite then **card** s yields **chaos**.)

```
    let b = xtr_B(bi)(cv) in
    let ris = obs_Ris(b) in
    if ri ∉ ris then chaos else
    let r = xtr_R(ri,b) in
    let tis = obs_Tis(r) in
    if ti ∉ tis then chaos else
    let t = xtr_T(ti,r) in
    case t of mkC(c) → c, _ → null end
    end end end end end end end end end
```

We[17] can now define a predicate which determines whether a cell, designated by a given location, is occupied or not.

```
value
  is_occupied: LOC → CV ~→ Bool
  is_occupied(l)(cv) ≡ select_C(l)(cv)≠null
    pre is_LOC_of_CV(l)(cv)
```

A.3.3 Vessels: Berths, Berth Positions, *&c.*

A CTP harbour consists of an ordered list of berth positions. We assume, without loss of generality, all berth positions to "fill" some area, i.e., be of same depth, width and length. A vessel at a specific harbour requires a (minimum) number of such berths. A vessel, independently has a (static) length, (static) width and (dynamic) depth. Berth positions are, at any time, either free or occupied. Vessels "occupy" consecutive berth positions.

```
type
a.  Vessel, CTP_Harbour, BPos, Berth, Depth, Length, Width
b.  BPosL = BPos*, BPosLs = BPosL-set
value
c.  obs_BPosL: (CTP_Harbour|Vessel) → BPosL
d.  calc_#BPoss: Vessel × CTP_Harbour → Nat
e.  obs_used_BPosLs, obs_free_BPosLs: CTP_Harbour → BPosL*
f.  obs_Len: (Vessel|CTP_Harbour|BPos) → Length
g.  obs_Wid: (Vessel|CTP_Harbour|BPos) → Width
h.  obs_Dep: (Vessel|CTP_Harbour|BPos) → Depth
i.  is_at_Sea, is_at_CTP: Vessel → Bool
axiom
j.  ∀ ctp_h:CTP_Harbour •
k.    (elems obs_used_BPosLs(ctp_h) ∩ elems obs_free_BPosLs(ctp_h) = {} ∧
l.    elems obs_BPosLs(ctp_h) = elems obs_used_BPosLs(ctp_h) ∪ elems obs_free_BPosLs(ctp_h) ∧
m.    ∀ vessel:Vessel •
n.      is_at_Sea(vessel) ~≡ is_at_CTP(vessel) ∧
o.      let nbps=calc_#BPosL(vessel,ctp_h) in obs_Len(vessel)≤nbps*obs_Len(ctp_h)∧
p.      obs_Wid(vessel)≤obs_Wid(ctp_h)∧obs_Dep(vessel)≤obs_Dep(ctp_h) end)
```

Annotations: We[18] "read" the above formulas.

17

RSL.17: **if** test **then** expr1 **else** expr2 **end** expresses the classical if-then-else.
RSL.18: **chaos** expresses the totally undefined value.
RSL.19: **case** expr_0 **of** expr_1 → expr_a, **of** expr_2 → expr_b, ..., _ → expr_final **end** expresses a multiple choice: if an expression expr_0 "matches" the value of an expression expr_1 then the value of expression expr_a is yielded, else ..., and finally the value of expression expr_final is yielded.

18

RSL.20: **type** A, B, ..., C defines not necessarily disjoint classes of values of type A, B, ..., C, respectively.
RSL.21: obs_B: A → B postulates the existence of a (not further defined) observer function which from type A values observer their type B constituent values.
RSL.22: **axiom** ∀ a:A,b:B,...,c:C • $\mathcal{P}$(a,b,...,c) expresses a property that is claimed to always hold for values a,b,...,c such as constrained by the predicate $\mathcal{P}$.

- (a.) Vessels, CTP_Harbours, Berth Positions, Depths, Lengths and Widths are further undefined classes of values — to be constrained by the axioms (i.–p.).
- (b.) A sequence of berth positions form a berth position list.
  Sets of berth position lists helps us model free and occupied berth positions.
- (c.) A harbour defines a sequence of (all) berth positions. (Some may be occupied, some may be free.)
  A vessel, when berthed, likewise defines a sequence of (hence occupied) berth positions.
- (d.) Depending on the CTP harbour a vessel, when berthed, will occupy a certain number of (fixed length) berth positions (of that harbour).
- (e.) At any one time zero, one or more berthed ships define as set of occupied berth position lists and hence a set of free berth position lists.
- (f.-g.-h.) Vessels have lengths, so does the entire sequence of harbour berth positions, and these have lengths. Similar for allowable widths and depths of berthed vessels and of vessels (depth is usually a dynamic attribute).
- (i., n.) A vessel is either at see or berthed at a CTP quay.
- (j.) For all CTP harbours the following predicates hold.
- (k.) The occupied berth positions share no positions with the free berth positions.
- (l.) The harbour berth positions are the same as collection of the occupied and the free berth positions.
- (m.) For all vessels the following predicates hold (in the context of the ranged harbour).
- (o.) The length of a vessel is not larger than the sum of the lengths of the berth positions that it occupies.
- (p.) The vessel width and depth is not larger than those of (i.e., prescribed by) the harbour.

A.3.4 Vessel Arrivals, Berthing and Departures

Container vessels ply the high seas, coastal areas, canals and, in cases, inland rivers. Container vessels sail from container terminal port to port. Container vessels arrives at ports and departs from port. Container vessels announce their imminent arrival to the CTP harbour master requesting permission to enter the port and request a berth position. The harbour master either grants the request to enter and then assigns a berth position to the vessel or informs the vessel that it must wait (say, outside the container terminal port area proper, say at a buoy). Eventually the harbour master will grant the vessel permission to berth (at a specific position).

Vessel and CTP Interactions: Messages.

type
  Ship_CTP_M = ReqBerth | BerthAsgn | PlsWait | ReqDept | OKDept
  ReqBerth == mkReqBerth(est:Time,ℓ:Length)
  BerthAsgn == mkBerthAsgn(jn:JobNm,bpl:BPosL)
  PlsWait == mkPlsWait(jn:JobNm,est:Time)
  ReqDept == mkReqDept(jn:JobNm)
  OKDept == ok

Annotations:

- A ship, cvn, asks permission to enter and requests a berth position: mkReqBerth(cvn,est,ℓ); the vessel estimates a time of arrival, and states its ℓength.

- The CTP harbour (master) acknowledges receipt of the arrival notice and berth request and responds positively, mkBerthAsgn(jn,bpl), by informing the vessel of job name (for harbour visit) and assigning a sequence, bpl, of berth positions (commensurate with the ship ℓength).
- Or the CTP harbour (master) acknowledges receipt of the arrival notice and berth request and responds negatively, mkPlsWait(jn,est), by informing the vessel the vessel of job name (for harbour visit) to wait up to an estimated time.
- A vessel requests permission, mkReqDept(jn), to depart, referring to the harbour visit job number.
- The CTP harbour (master) acknowledges receipt of the departure request and responds positively, oking the departure.
- The CTP harbour (master) may acknowledge receipt of the departure request by responding negatively with a mkPlsWait(jn,est) response.

Vessel and CTP Interactions: Processes.

```
type
  CVNm, CVΣ, CTPΣ
value
  vns:CVNm-set
  cvσs:(CVNm →m CVΣ)
  ctpσ:CTPΣ
axiom
  vns = domcvσs
channel
  {ves_ctp[ i ]|i:vns}:Ship_CTP_M
```

Annotations:

- CVNm designates the class of all vessel names. CVΣ designates the class of all vessel states — and we can model a vessel just by modelling its state. CTPΣ designates the class of all CTP states.
- vns designates a value of arbitrarily chosen vessel names.
- cvσs designates a value which to every arbitrarily chosen vessel name, vn in vns, associates an arbitrarily chosen vessel state.
- ctpσ designates an arbitrarily chosen CTP state value.
- The axiom states that the definition set of cvσs must be the set, vns, of arbitrarily chosen vessel names.
- There are **card**inality vns channels, one for each vessel, whether at high sea or at CTP, between vessels and the CTP.

```
value
  vessel: cvn:CVNm × CVΣ → in,out ves_ctp[ cvn ]  Unit
  vessel(cvn)(cvσ) ≡
        ...
    ⌈⌉ if is_at_sea(cvσ)
          then
            (vessel(cvn)(next_cvσ(cvσ)) ⌈⌉ vessel_arrives(cvn)(cvσ))
          else
            vessel(cvn)(next_cvσ(cvσ)) end
    ⌈⌉ if is_at_CTP(cvσ)
```

```
            then
               (vessel(cvn)(next_cvσ(cvσ)) ⌈⌉ vessel_departs(cvn)(cvσ))
            else
               vessel(cvn)(next_cvσ(cvσ)) end
      ⌈⌉ ...

   next_cvσ: CVΣ → CVΣ
```

Annotations:

- The vessel is here modelled as non-deterministically "wavering" between being on the high seas or in some CTP or ...
- If at sea then the vessel either remains at sea or enters a CTP.
- If at a CTP then the vessel either remains at that CTP or departs.
- There are other vessel behaviours, ..., which we do not describe.
- The liberal use of non-deterministic choice, ⌈⌉, serves to model that the vessel may decide to remain at sea or at harbour, or to do other things.
- The next_cvσ function "increments" the state "one step" (whatever that is).

```
value
   vessel_arrives: cvn:CVNm × CVΣ → in,out ves_ctp[cvn] Unit
   vessel_arrives(cvn)(cvσ) ≡
1.    (let time = estimate_arrival_time(cvσ) in
2.    ves_ctp[cvn]!mkReqBerth(time,obs_berthing_Info(cvσ));
3.    let m = ves_ctp[cvn]? in
4.    case m of
5.      mkBertAsgn(jn,bpl) → vessel(cvn)(upd_berth_cvσ(jn,bpl)(cvσ)),
6.      mkPlsWait(jn,est) → vessel(cvn)(upd_wait_cvσ(jn,est)(cvσ)),
7.      _ → chaos
         end end end)

8.  obs_berthing_Info: CVΣ
9.  estimate_arrival_time: CVΣ → Time
10  upd_berth_cvσ: JobNm × BPosL → CVΣ → CVΣ
11. upd_wait_cvσ: JobNm × Time → CVΣ → CVΣ
```

Annotations:

- (1.) An estimate[19] is made as to when the vessel is expected to actually arrive at the harbour.
- (2.) The ship informs the CTP harbour master of estimated time of arrival and other such information that help determine whether and, and if so, where the ship can berth.
- (3.) The ship receives a response, m, from the CTP harbour master.

19

The **let** a = $\mathcal{E}$ **in** $\mathcal{C}$(a) **end** construct defines a to be bound to the value of expression $\mathcal{E}$ in the body $\mathcal{C}$, that is, a is free in $\mathcal{C}$(a)

- (4.-5.) If[20] the response is an accept to berth then that response will also state a job name, jn, for the ship at harbour and a direction as to to which berth position, bpl, to dock. The ship will then go to dock, i.e., update its state accordingly.
- (6.) If the response is a deferral (i.e., to wait) then the ship will wait, i.e., update its state accordingly.
- (7.) Any other, unexpected response will lead to **chaos**, i.e., is not further described here.
- (8.–10.) The signatures of auxiliary behaviours are given, but the no further definition is given.

```
value
  vessel_departs: cvn:CVNm × CVΣ → in,out ves_ctp[ cvn ] Unit
  vessel_departs(cvn)(cvσ) ≡
1.   (let jn = obs_JobNm(cvσ) in
2.   ves_ctp[ cvn ]!mkReqDept(jn);
3.   let m = ves_ctp[ cvn ]? in
4.   case m of
5.     ok → vessel(cvn)(upd_dept_cvσ(cvσ)),
6.     mkPlsWait(jn,est) → vessel(cvn)(upd_wait_cvσ(jn,est)(cvσ)),
7.     _ → chaos
       end end end)

8. obs_JobNm: CVΣ → JobNm
9. upd_dept_cvσ: CVΣ → CVΣ
10. upd_wait_cvσ: JobNm × × Time → CVΣ → CVΣ
```

Annotations:

- (1.) As the vessel is about to depart it recalls the job name for the current harbour visit
- (2.) and informs the CTP harbour master of its intention to depart.
- (3.) The vessel then awaits the harbour master response.
- (5.) If it is OK, then the vessel leaves, i.e., updates its state accordingly.
- (6.) If it is not OK to leave, but to wait further in harbour, then the vessel waits, i.e., updates its state accordingly.
- (7.) Any other response is not expected.
- (8.–10.) The signatures of auxiliary behaviours are given, but the no further definition is given.

```
value
  ctp: CTPΣ → in,out ves_ctp[ ∗ ] Unit
  ctp(ctpσ) ≡
    ...
1.  ⌈⌉ []{let m = ves_ctp[ cvn ]? in
2.    case m of
3.      mkReqBerth(t,ℓ) → ctp_berth(vn,t,ℓ)(ctpσ),
4.      mkReqDept(jn) → ctp_dept(vn,jn)(ctpσ) end end
```

[20] The **case** a **of** $\mathsf{pattern}_1 \rightarrow \mathcal{C}(\mathsf{p}_1)$, $\mathsf{pattern}_2 \rightarrow \mathcal{C}(\mathsf{p}_2)$, ..., $\mathsf{pattern}_n \rightarrow \mathcal{C}(\mathsf{p}_n)$ **end** construct examines the value a. If it "fits" $\mathsf{pattern}_1$ then the value of clause $\mathcal{C}(\mathsf{p}_1)$ is yielded as the value of the entire s construct, else — and so on.

```
5.      | cvn:CVNm }
6.  ⫿
      ...
```

Annotations:

- (1.,6.) The CTP harbour master decides which next task to work on, i.e., internally non-deterministically alternates between, as shown in (1.)[21] being willing to "listen" to requests from approaching vessels, or, as shown in (6.) to do something else.
- (3.) If an approaching vessel, cvn, requests a berthing then the CTP harbour master will handle that request and then continue to be a harbour master.
- (4.) If a ship at harbour, (still) cvn, requests to depart, then the CTP harbour master will handle that request and then continue to be a harbour master.

```
value
  ctp_berth: cvn:CVNm×Time×Info → CTPΣ → in,out ves_ctp[cvn] Unit
  ctp_berth(cvn,t,ℓ)(ctpσ) ≡
1.  let jn:JobNm • jn ∉ job_names(ctpσ) in
2.  if is_available_BPosL(t,info)(ctpσ)
3.   then
4.    let bpl = allocate_BPosL(t,info)(ctpσ) in
5.    ves_ctp[cvn]!mkBerthAssgn(jn,bpl);
6.    vessel(cvn)(upd_berth_asgn(cvn,jn,bpl)(ctpσ)) end
7.   else
8.    let est = estimate_berth_avail(t,info)(ctpσ) in
9.    ves_ctp[cvn]!mkPlsWait(jn,est);
10.   vessel(cvn)(upd_berth_avail(cvn,jn,est,bpl)(ctpσ)) end
    end end
```

Annotations: The handling of requests, by approaching vessels, to enter barbour and be berthed, are described by this behaviour definition.

- (1.) First the harbour master assigns a job name to the request.
- (2.) If, based on estimated time of arrival and other, ertinent vessel information, the harbour master decides that a sequence of berth positions can be allocated,
- (3.) then allocation and notification is done:
  - (4.) a suitable sequence of berth positions is selected,
  - (5.) the vessel is so informed,
  - (6.) and the harbour master reverts to being a harbour master;
- (7.) else deferral is adviced:
  - (8.) a time is estimated for possible (later) arrival,
  - (9.) the vessel is so informed,
  - (10.) and the harbour master reverts to being a harbour master.

21

The clause []{ ... ch[i]? ... $\mathcal{C}_i$ | i:Idx } expresses externally driven, but still non-deterministic choice as to which other process, indicated by cvn to interact with. Once chosen the behaviour $\mathcal{C}_i$ is "undergone".

job_names: CTPΣ → JobNm
is_available_BPosL: Time × Length → CTPΣ → **Bool**
allocate_BPosL: Time × Length → CTPΣ → BPosL
upd_berth_asgn: CVNm × JobNm × BPosL → CTPΣ → CTPΣ
estimate_berth_avail: Time × Length → CTPΣ → Time
upd_berth_avail: CVNm × JobNm × Time → CTPΣ → CTPΣ

Annotations:

- The signatures of auxiliary behaviours are given, but the no further definition is given.

value
ctp_dept: cvn:CVNm × JobNm → ctpΣ → **in**,**out** ves_ctp[cvn] **Unit**
ctp_dept(cvn,jn)(ctpσ) ≡ /∗ left as an assignment exercise ∗/

•••

We show only this fragment of modelling container vessels here. Additional fragments will appear later in this modelling of the domain of the container line industry.

The above and the following (i.e., the below) formalisations need be harmonised wrt. type names. Above we have used one set. Below we may deviate from this set and, occasionally, use other (synonym) type names. This is clearly not acceptable from a final document!

A.4 Container Vessel Stowage

"Containers[22] *on board a container ship are placed in container cells, that is, at locations made up from bay and row identifiers and tier indexes. Since the access to the containers is only from the top of the tier, a common situation is that containers designated for port J must be unloaded and reloaded at port I (before J) in order to access containers below them, designated for port I. This operation is called "shifting". A container ship calling at many ports may encounter a large number of shifting operations, some of which can be avoided by efficient stowage planning (*). In general, the stowage plan must also take into account stability and strength requirements (*), as well as several other constraints (*) on the placement of containers.*

A.4.1 Physically Impossible Stowage

But let us not get derailed into stowage requirements such as expressed above wrt. avoidance of "shifting" and satisfaction of the (*) marked requirements. In the domain all we have to secure is that certain impossible situations are not represented in any container vessel: First we introduce, as part of the concept of 'stowage', the phenomenon of a cell being "occupied", that is, its location "houses" a container. We have already defined that predicate (**is_occupied**).

Then we must express the following physical impossibility:

14. In any tier (i.e., sequence of cells (tier positions), since containers are stowed from lower positions toward higher positions (and correspondingly unloaded from higher positions toward lower positions), we have that there cannot be any empty cells between adjacent occupied cells.

We have already ruled out the possibility of "empty gaps". This was done in the formal axiom, "no un-occupied gaps", on Page 34.

[22] This *slanted* quote is edited from [10].

A.4.2 Stowage Properties

It may surprise the reader that this is all we need to say at this early stage about container on-open-sea stowage. All other properties of stowage is here seen as requirements to proper storage. Of course we can always, in the domain, speak of proper storage so let us define some predicates that do not necessarily need to be satisfied of any actual stowage. Therefore we express these as defined predicates rather than in the form of axioms. Each property, that is, each desirable form of container stowage, is usually relative to a whole container vessel, and involves the container, i.e., its contents, its absolute cell position as well as its narrower or wider context of other occupied cell positions (and their contents). Informally such properties are illustratively expressed as follows:

15. Heavier containers must not be stowed above lighter containers.
16. A container, c, at location c_ℓ must not have a contents which "disagrees" with the contents of "nearby" containers.
17. Heavier containers should be stowed close to the ship center of gravity.
18. Containers should be stowed so as to minimize trim and list.

The variety of 'disagreements' and notions of 'locations' and 'neighbourhood' is rather large. When abstractly formalising these variations, we therefore choose to not even detail them, but to introduce un-interpreted sets of predicates. Examining the above four examples (Items 15–18) we find that some involve basically one container versus "all other containers" and that others involve "all containers". But we "lift" even this distinction and let our un-interpreted predicates embody this 'one-versus-neighbours', 'one-versus-all-others', 'all', etc.

type
  P = CV → **Bool**
value
  ps:P**-set**
axiom
  ∀ cv:CV,p:P • p ∈ ps ⇒ p(cv)

A.5 Container Terminal Ports

This section will first present a semi-structured narrative. It is in a form somewhere between rough sketches and more "stricter" narratives. Then follows some analysis and sketch formalisations. Based on that we suggest another form of formal modelling. But we do not bring a "strict" narrative — and our formalisation is just sketchy.

A.5.1 Informal Rough Sketch cum Narrative Presentation

19. A container terminal consists of
    (a) a quay[23] where a varying number of ships can be berthed, with quay(s) "sandwiched" between the ocean and the quay area,
    (b) a therefrom physically separated container stack, which consists of a fixed number of one or more container blocks (or container groups) where containers can be stored (stowed),
    (c) a quay area which is physically located properly between and separating the quay(s) from the stack, and

[23] We abstract from whether we speak on one quay of some length, or a number of quays of the same total length.

(d) a land side transfer area which is physically located properly between and separating the stack from, or interfacing the container terminal with an inland from which containers originate or are finally destined.

(e) The reason for the berthing of a varying number of ships is that the ships have possibly differing lengths whereas the quay side length is fixed.

20. Ships
    (a) arrive from and depart for the ocean
    (b) and dock, respectively un-dock
    (c) at berths which are positioned along the quay.
21. Further container terminal entities and some operations involve:
    (a) Quay cranes which move along the quay(s),
        i. lift (unload) containers from a berthed ship,
        ii. and drop (load) them to container vehicles, respectively
        iii. lift (unload) containers from these vehicles
        iv. and drop (load) them onto a berthed ship.
        v. We shall model the combined lift/drop as a composite transfer operation.
    (b) Container vehicles (mentioned above)
        i. which horisontally, two-dimensionally move
        ii. along the quay(s) and in the quay and stack areas
        iii. between ships and stacks.
    (c) Stack cranes
        i. which move within a very restricted area of a stack — usually only within a stack group (or block) or just bay,
        ii. and lift (and drop) containers from (respectively to) container vehicles
        iii. to (respectively from) stack tiers.
        iv. We shall model the combined lift/drop as a composite transfer operation.
    (d) Each stack block (or group) is organised "like on a ship": one or more bays, each bay with one or more rows, each row with one or more tiers, and each tier with a maximum number of containers (i.e., cells).
    (e) Other (possibly different kinds of) container vehicles
        i. likewise land-surface move
        ii. in and between the container stack blocks (or stack groups)
        iii. and the land side transfer area
        iv. where containers may be transferred by transfer cranes
        v. to and from inland trucks, trains or even barges.
    (f) The transfer cranes move only in the transfer area.
    (g) The trucks, trains and barges land surface (water surface) move between the land side transfer area, through gates, and the inland.
    (h) Gates separate the container terminal port from the inland
    (i) just a ship berths separate the the container terminal port from the ocean.
    (j) Finally we may introduce the explicit vehicle operation of waiting
        i. at a CTP location
        ii. for a specified interval of time,

iii. or until a specified clock time,
iv. or indefinitely.

22. A container undergoes the following four behaviours:
    (a) The ship to stack container behaviour:
        i. first a lift (unload) — by a crane — from a ship tier
        ii. followed by a drop (load) — by the crane — to a container vehicle,
        iii. then a transport by the moving vehicle from the quay area to the stack
        iv. where the container is lifted (unloaded) — by a crane — from the vehicle
        v. and dropped (loaded) — by the crane — onto a tier.

        So a ship tier, a crane, a vehicle, another crane, and a stack tier was involved in the ship to stack container behaviour, and in that order.

        Now to a **reverse** behaviour of the above.
    (b) The stack to ship behaviour:
        i. first a lift (unload) — by a crane — from a stack tier
        ii. followed by drop (load) — by the crane — to a container vehicle,
        iii. then a transport by the moving vehicle from the stack area to the quay area
        iv. where the container is lifted (unloaded) — by a crane — from the vehicle
        v. and dropped (loaded) — by the crane — onto a ship tier.

        So a ship tier, a crane, a vehicle, another crane, and a stack tier was involved in the stack to ship container behaviour, but in the reverse order.
    (c) The stack to transfer area (and inland) behaviour:
        i. first a lift (unload) — by a crane — from a stack tier
        ii. followed by drop (load) — by the crane — to a container vehicle,
        iii. then a transport by the moving vehicle from the stack to the transfer area
        iv. where the container is transferred — by a crane — from the vehicle
        v. to an inland truck, train or barge.

        Thus a stack tier, a crane, a vehicle, yet a crane and an inland truck, train or barge were involved in this behaviour.

        Now to a **reverse** behaviour of the above.
    (d) the (inland and) transfer area to stack behaviour:
        i. first a transfer — by a crane — from an inland truck, train or barge
        ii. to a vehicle,
        iii. then a transport by the moving vehicle from the transfer area to the stack area
        iv. where the container is lifted — by a crane — from the vehicle
        v. followed by a drop — by the crane — onto a stack tier.

        Thus a stack tier, a crane, a vehicle, yet a crane and an inland truck, train or barge were involved in this behaviour — but in the reverse order.

There is the possibility of having transfer areas in which containers may be temporarily stored ("stowed"). We shall call such transfer area storage for buffers. In such cases we need augment the four container behaviours with an additional two such. When dealing with a proper, full scale description of the CTP domain we must provide for alternative transfer areas as well as for alternative, or further abstracted any such areas within the CTP. We have not made any distinction between various forms of quay cranes (gantry, single or dual trolley, etc., cranes), various forms of CTP vehicles, and various forms of stack cranes, (rail mounted cranes, rubber tired gantries, overhead bridge cranes, etc.). These rather technology-bound phenomena shall, of course, be further detailed as part of the support technology domain facet. Presently we focus on the intrinsics of cranes and vehicles: their ability to move, lift, drop and transfer, respectively their ability to stock and move.

A.5.2 Analysis of CTP and First Draft Formalisations

We observe that the container terminal port (CTP) can be physically characterised as a composition of a number of sea, land and possibly river/canal/coastal waters areas (i.e., entities). (i) There is the ocean (which is adjacent to and interfacing with many harbours, i.e., CTPs). (ii) The ocean is (for any particular CTP) adjacent to and interfacing with quays. (iii) A(ny) quay (iii.1) is partly part of and partly adjacent to (interfacing with) the harbour basins (iii.2) and partly part of and partly adjacent to (interfacing with) the quay area. (iii.3) That is: quays are partly water partly land based. (iv) The quay area is wholly land based and "sandwiched" between the quays and the stack. (v) The stack area contains container groups as properly "embedded" parts of the stack. (vi) The transfer area which is "sandwiched" between the stack and the inland. (vii) There is the inland which we consider to be outside the container terminal port — as is the ocean. viii) Finally gates "connect" the transfer area and the inland.

type
  Ocean, Onm [ocean name], Inland, Inm [inland name]
  CTP, CTPnm [CTP name], Quay, QuayArea, Stack, Group, TransArea, Gate
value
  obs_CTPs: Ocean → CTP-**set**, obs_CTPnm: CTP → CTPnm
  obs_Onm: CTP → Onm, obs_Inm: CTP → Inm
  obs_Quay: CTP → Quay, obs_QuayArea: CTP → QuayArea,
  obs_Stack: CTP → Stack, obs_TransArea: CTP → TransArea,
  obs_Gates: (CTP|TransArea) → Gate-**set**

There seems to be a conceptual notion of 'stock' "buried" in the above description. By a stock we mean a place where one or more containers may be (however temporarily) stored ("stowed"). Examples of stocks are container vessels (bays, rows, tiers, cells), CTP vehicles (usually one or two containers), stack groups (bays, rows, tiers, cells), transfer area (buffer [bays, rows, tiers, cells]) and transfer area to inland trucks, trains and barges. What characterises stocks is that containers may be lifted from, dropped onto, or transferred between stocks.

Let us analyse the notions of crane and vehicle operations.

23. Container lift (unload), drop (load) and transfer operations can only be performed by cranes.
24. Movement of containers
    (a) (i.e., transport)
    (b) along a (CTP) route,
    (c) between different locations within the CTP
        i. can only be performed
        ii. by container vessels
        iii. and by some stack cranes.
25. Movement of containers
    (a) (i.e., transport)
    (b) along an (ocean) route,
    (c) between different CTPs across the ocean
    (d) can only be performed by container vessels.

So there are two conceptual notions of (CTP) routes and locations.

26. Consider a CTP as characterisable **also**
    (a) in terms of a dense (possibly finitely enumerable) set of points,

(b) that is, a point can be said to be a "neighbour", or "in the neighbourhood" of some other point(s),

(c) and we can speak of two bordering sets of points sharing an interface line of points.[24]

27. Thus the quays, quay area, stack and transfer area can be said to be represented (also) by bordering point sets.

28. A (CTP) location can then either be defined as a point or as a "small" dense set of points
    (a) such that all points
    (b) are proper points of the CTP.

29. A (CTP) route can then be defined as a sequence of (CTP) locations
    (a) such that adjacent elements of the route sequence
    (b) form bordering locations.
    (c) The "size" of locations, i.e., their granularity, determines "smoothness" or a route.

30. A (CTP) transport route is a (CTP) route
    (a) such that the pair of first and last element of the route designates
    (b) quay, stack, or transfer crane positions c_{q_p}, c_{y_p}, or c_{t_p}
    (c) and as follows: (c_{q_p}, c_{y_p}) or (c_{y_p}, c_{t_p})
    (d) or their reverses.

We are now ready to further characterise crane and vehicle operations. We do not model the atomic crane operations of lift and drop. We model, instead the composite crane operations of lift (of the container, by the crane spreader), crane trolley movement (with the container), and drop (of the container, by the crane spreader), and we call this operation a/the transfer operation.

31. A crane operation is
    (a) either a container transfer operation,
    (b) or a crane movement operation.

32. A vehicle operation is
    (a) either a move operation.
    (b) or a wait operation.

33. To explain these operations let us introduce the notions of:
    (a) container ship container location,
    (b) stack container location, and
    (c) transfer area container location.

    They are definable as follows:

    (d) A container ship container location embodies a bay and a row identifier as well as a stack tier (i.e., stack index).
    (e) A stack container location embodies a group (or block), a bay, a row, and a tier identifier as well as a tier index.
    (f) A transfer area container location is

[24] Let A_i and A_j be two such bordering sets of points. Let L_{ij} be the (shared) interface line of points. Then $A_i \setminus L_{ij}$ and $A_j \setminus L_{ij}$ are disjoint sets of points. Some points in $A_i \setminus L_{ij}$ are 'adjacent' to some points in $A_j \setminus L_{ij}$.

i. either a buffer location which embodies a a row identifier and a stack tier (i.e., stack index),
ii. or it is a train, a truck or a barge position.

(g) We leave the characterisation of train, truck and barge positions to the interested reader.

34. The crane transfer operation has the following operation signatures:
    (a) as input arguments:
        i. crane (f.ex. referred to by a crane name),
        ii. container (f.ex. referred to by a container identifier),
        iii. and a to/from designation which is a pair of either
            A. a ship container location and a vehicle name, or
            B. a vehicle name and a ship container location, or
            C. a stack container location and a vehicle name, or
            D. a vehicle name and a stack container location, or
            E. a transfer area container location and a vehicle name, or
            F. a vehicle name and a transfer area container location, and
        iv. a start time
        v. and the CTP state;
    (b) and as result values:
        i. a possibly changed CTP state
        ii. and a termination time.
    (c) Some comments pertinent not just to the transfer operation, but to all container shipping operations are in order.
        i. All operations, also transfer, "take place" in the state of a specific CTP — and potentially change the (CTP) state.
        ii. And all operations take place in time:
            A. They start at some time, t;
            B. they "last", i.e., take some time, $\tau\iota$;
            C. and they therefore end, or terminate at some time, t'.
        iii. The "times" $t, \tau\iota$ and τ' are of types:
            A. t and τ' are absolute times: Year, month, day, hour, minute, etc., while
            B. $\tau\iota$ is an interval time;
            C. for the wait operation $\tau\iota$, however, may be indefinite .
    (d) So absolute and/or interval times have to be added to the signature of all CTP operations.

```
type
  T, TI
  Pt, Loc, Vehicle, Vn [vehicle name], Cra, CraNm [crane name]
  SLOC = LOC, YLOC = Gid × LOC, XLOC
  Route = Loc*
  ToFro = ShVe | VeSh | YaVe | VeYa | VeXf | XfVe
  ShVe == mkSV(sl:SLOC,v:Vn)
  VeSh == mkVS(v:Vn,sl:SLOC)
  YaVe == mkYV(yl:YLOC,vn:Vn)
  VeYa == mkVY(vn:Vn,yl:YLOC)
  XfVe == mkXV(xl:XLOC,vn:Vn)
  VeXf == mkVX(vn:Vn,xl:XLOC)
value
  xfer: CraNm × CId × ToFro → T → CTP ~→ CTP × T
```

25

(a) The crane move operation has the following operation signatures:
  i. as input arguments:
    A. the crane (f.ex. referred to by a crane name),
    B. the crane 'from' position along the quay, and
    C. the crane 'to' position along the quay,
    D. the start time,
    E. and the current CTP state;
  ii. and as result values:
    A. the end CTP state
    B. and the termination time.

type
  CraP
value
  obs_CraPs: Quay → CraP**-set**
  move: CNm × CraP × CraP → T → CTP $\xrightarrow{\sim}$ CTP × T

26

35. The vehicle move operation has the following operation signature:
  (a) the input arguments:
    i. a vehicle (f.ex. referred to by a vehicle name)
    ii. a route,
    iii. and an initial CTP state;
  (b) and as result values:
    i. a result CTP state
    ii. and a termination time.
36. The vehicle wait operation has the following operation signature:
  (a) the input arguments:
    i. vehicle (f.ex. referred to by a vehicle name),
    ii. location at which to wait, and

25

RSL.23: The 'type definitions':

```
type
  A = B|C|...|D
  B == mkX(...), C == mkY(...), ..., D == mkZ(...)
```

defines A to be the disjoint union of types B, C, etc., D. Disjointness is achieved solely through distinctness of all mkX, mkY, etc., mkZ. That is, the ...'s "inside" the mkX(...) may be identical.

RSL.24: The 'type expression' mkE(s1:F1,s2:F2,...,sn:Fn) designates a type of 'records' ('structures') with n 'fields' of respective 'types' Fi whose 'value' in some e: mkE(f1,f2,...,fn) can be 'selected' by applying the 'selector' si to the 'value' e, i.e., si(e). mkE is called the 'constructor'.

26

RSL.25: The signature f: A × B × C → D → E $\xrightarrow{\sim}$ E × G "reads" 'function application' is expressed as some f(a,b,c)(d)(e) and a yielded result can be expressed as some (e',g). (The signature and hence function application could have been expressed in a non-Curried form: f: A × B × C × D × E $\xrightarrow{\sim}$ E × G, respectively f(a,b,c,d,e).)

iii. optional waiting time.

(b) and as result values:

i. a result CTP state

ii. and a termination time.

```
type
  Hour, Min
  Time == mkT(t:T) | Indef
  Wait = Interval | Clock | Indef
  Intvl == mkIntvl(h:Hour,m:Min)
  Clock == mkClock(h:Hour,m:Min)
  Indef == indefinite
  OptWait == empty | mkWait(i:Wait)
value
  move: VeNm × Route → CTP → T ~→ CTP × T
  wait: VeNm × Loc × OptWait → T → CTP ~→ CTP × Time
```

We shall now discuss the meaning of the lift, drop, transfer, move and wait operations.

37. The crane container xfer (transfer) operation:

```
value
  xfer: CId × CraNm × ToFro → T → CTP ~→ CTP × T
  xfer(ci,cn,(ℓ,ℓ′))(t)(σ) as (σ′,t′)
```

can[27] be roughly described as follows:

(a) The transfer (xfer) operation

i. is performed in some initial CTP state σ,

ii. and starts at some time.

(b) The result of performing a transfer operation

i. is a possibly new CTP state σ',

ii. and a termination time t'.

(c) The transfer (xfer) operation ends in **chaos**, that is, is undefined if one or more of the following holds in state σ:

i. there is no container of identity *ci* at either location ℓ or ℓ';

ii. there is no crane of name *cn*;

iii. if ℓ or ℓ' designates a vehicle and there is no such named vehicle[28] located at the identified crane;

iv. if ℓ or ℓ' intends to designate a container position on a ship

A. and either there is no such ship at the crane position,

B. or the ship which is there has no such container location,

C. or, if there is, that container location is not on top of a tier;

v. if ℓ or ℓ' intends to designate a container position in a stack group (or block) or a transfer area buffer,

27

RSL.26: The 'function definition' f(a,...,b) **as** r or f(a,...,b) **as** (c,...,d) "reads" as follows: Application f(a,...,b) yields a result which can be expressed as r (or as a grouping (c,...,d).

28 including transfer area trucks, trains and barges

A. and there is no such stack group (respectively transfer area buffer) container location;
B. or, if there is, that location is not a tier top location;

(d) If the above listed **pre** conditions are satisfied then proper interpretation, i.e., crane container transfer operation can be commenced.
i. The identified crane "grabs" (lifts), with its spreader,
ii. the identified and properly located container from that location (ℓ),
iii. moves the crane trolley appropriately, and
iv. the crane spreader "releases" (drops) the container
v. onto the identified and properly identified location (ℓ').
Of the two locations
vi. one is a tier location:
A. either a ship bay/row/stack/tier and tier index position,
B. or a stack group/bay/row/stack/tier and tier index position,
C. or a transfer area buffer (bay/row/stack/tier position and tier index) location,
vii. and the other location is a vehicle name.

38. The crane move operation:

value
move: CNm × CraPos × CraPos → T → CTP $\xrightarrow{\sim}$ CTP × T
move(cn,cp,cp′)(t)(σ) **as** (σ',t′)

can be roughly described as follows:

(a) The crane move operation
i. is performed in some initial CTP state σ
ii. and starts at some time t.

(b) The result of performing a crane move operation
i. is a possibly next CTP state σ'
ii. and some termination time t'.

(c) The crane move operation ends in **chaos**, i.e., is undefined, if one or more of the following holds in initial state σ:
i. there is no crane of name *cn*,
ii. there is no quay position *cp*,
iii. there is no quay position *cp*′, and/or
iv. the crane of name *cn* is not, in state σ, in position *cp*.

(d) If the above listed **pre** conditions are satisfied then proper interpretation, i.e., the crane move operation can be commenced.
i. The crane, named *cn* starts moving from quay position *cp*,
ii. the crane, for some interval of time, continues moving "towards" quay position *cp*′,
iii. and the crane finally halts (ends it move) at quay position *cp*′.

39. The vehicle move operation:

value
move: VeNm × Route → T → CTP $\xrightarrow{\sim}$ CTP × T
move(vn,rt)(t)(σ) **as** (σ',t′)

can be roughly described as follows:

(a) The vehicle move operation
  i. is performed in some initial state σ
  ii. and starts at some time t.

(b) The result of performing a vehicle move operation
  i. is a possibly next CTP state σ'
  ii. and some termination time t'.

(c) The vehicle move operation ends in **chaos**, i.e., is undefined, if one or more of the following holds in initial state σ:
  i. there is no vehicle of name vn,
  ii. the route rt is not well-fined within the CTP.

(d) If the above listed **pre** conditions are satisfied then proper interpretation, i.e., the vehicle move operation can be commenced.
  i. The vehicle starts moving, from its current location,
  ii. that is the origin,
  iii. which is the first element location of the prescribed route,
  iv. along the prescribed route,
  v. until it reaches the destination location
  vi. which is the last element location of the prescribed route.
  vii. The time interval, τ, that it has taken to perform the entire move is added to the absolute initial time t to yield the termination time t'.

40. The vehicle wait operation:

value
  wait: VeNm × Loc × OptWait → T → CTP $\xrightarrow{\sim}$ CTP × T
  wait(vn,loc,owt)(t)(σ) **as** (σ',t′)

can be roughly described as follows:

(a) The vehicle wait operation
  i. is performed in some initial state σ
  ii. and starts at some time t.

(b) The result of performing a vehicle wait operation
  i. is a possibly next CTP state σ'
  ii. and some termination time t'.

(c) The vehicle wait operation ends in **chaos**, i.e., is undefined, if one or more of the following holds in initial state σ:
  i. there is no vehicle of name vn and/or
  ii. there is no proper location loc within the CTP.

(d) If the above listed **pre** conditions are satisfied then proper interpretation, i.e., the vehicle wait operation can be commenced.
  i. If the location loc is different from the current location of the vehicle,
    A. then a vehicle move operation is first performed.
  ii. Having possibly first had to properly move to location loc
  iii. and assuming that the move has taken some time (interval) τ'

iv. (τ' could be 0 if no move was necessary),
v. the vehicle stays at location *loc* till either of the following occurs:
   A. either the wait has been prescribed as a relative interval mkIntvl(τ) in which case the vehicle stays at location *loc* for $\tau - \tau'$ — which, if negative, means no wait and hence an abnormal termination (which we have yet to properly describe),
   B. or if the wait has been prescribed as a definite time interval, τ'', and τ' is less than τ'' then the vehicle stays at location *loc* till time $t + \tau'' - \tau'$,
   C. or if the wait has been prescribed as a definite time interval then the vehicle stays at location *loc* indefinitely. What further happens is presently left undefined.

A.5.3 Analysis of Draft Operation Descriptions

We now comment on the above informal and formal descriptions of the CTP operations.

(i) The descriptions are mostly idealised. We do define proper **pre** conditions for all operations, but we mostly neglect unforeseen adversary events: (i.1) breakdown of crane trolleys, (i.2) breakdown of vehicles, (i.3) collision between two or more CTP vehicles "on the move" during overlapping time intervals, etcetera; and we have not detailed (i.4) what happens if wait times are in conflict, (i.5) what happens if the wait goes on indefinitely, that is, why the vehicle has to wait, etc.

(ii) The descriptions focus on just one particular operation. No consideration is given to the simultaneity of two or more CTP operations involving two or more cranes, or two or more vehicles, or combinations of one or more cranes and one or more vehicles during overlapping time intervals. Such as the above operation descriptions are given no allowance is made for two or more CTP operations to occur during overlapping time intervals and that is certainly contrary to "the real" domain !

(iii) The descriptions omitted detailing in which way the CTP states were updated. We now remedy these omissions.

41. The final state after successful execution of a crane xfer operation records
    (a) that a container has been transferred.
        i. If it was lifted from vehicle then that container is no longer on that vehicle.
        ii. If it was dropped onto a (ship or stack or buffer) tier then that container is now on top of that tier. Reversely
        iii. if it was lifted from (the top of a ship or stack or buffer) tier then that container is no longer on that tier, and
        iv. if it was dropped onto a (presumably empty) vehicle then that container is on that vehicle
    (b) (we do not specify what happens (to the state) if the vehicle is a two or more container vehicle [the reader should be able to fill in such details]);
    (c) and the time which it has taken
        i. to perform the lowering of ,"grabbing" by, and raising of the crane spreader,
        ii. to move the crane trolley,
        iii. to perform the lowering of, "release" by, and raising of the crane spreader,
        iv. on to move the crane trolley to an initial trolley position —

        that time is reflected in the result time.
42. The final state after successful execution of a crane move operation records
    (a) that the crane has been moved:

i. from one crane position along the quay
ii. to another crane position along the quay,

(b) and that the time it has taken to perform that move is reflected in the result time.

43. The final state after successful execution of a vehicle move operation records
    (a) that the vehicle has been moved:
        i. from one CTP location
        ii. to another CTP location,
    (b) and that the time it has taken to perform that move is reflected in the result time.
44. The final state after successful execution of vehicle wait operation records
    (a) that the vehicle has possibly been moved, as for the vehicle move operation, to a wait location,
    (b) that is, that the vehicle now is in that wait location
    (c) and that the time it move time plus the (possibly adjusted) wait time is reflected in the result time.

Many other comments could be put forward.

The gist of these comments is that we cannot proceed with the draft formalisation as shown. The current model basis was one of an applicative model for all CTP operations. Simultaneity of many (thus concurrent) CTP operations means that state changes from different CTP operations must be "merged". We must formulate an altogether different model basis. It seems that a model based on concurrency and shared state components is more appropriate. Let us try ! The above negatively critical comments apply only to the draft formalisations not to the informal operation descriptions — they are still valid !

A.5.4 A Resolution on Modelling CTPs and CTP Operations

In this section some modelling decisions. These are illustrative in the sense that other decompositions into process (crane, vehicle, ship, and stack) behaviours could be shown. The ones shown are OK, but typically such modelling choices as we show should be the outcome of far more experimentation than we can afford in a presentation such as ours. So, without much "further ado" we put forward a more realistic model basis.

45. We decompose entities of the CTP into behaviours as follows.
    (a) For every ship there is a separately described behaviour
        i. whether in harbour, at quay,
        ii. or on the high seas.

        That is, our model is going to assume a very large, fixed number of ship processes.
    (b) For every CTP vehicle there is a separately described behaviour.
    (c) For every quay crane in a CTP there is a separately described behaviour.
    (d) For every stack crane in a CTP there is a separately described behaviour.
    (e) For every transfer area crane in a CTP there is a separately described behaviour.
    (f) For every (other) state component there are separately described behaviours:
        i. For every separately quay crane-accessible (for example) ship bay, row and possibly also tier there is a separately described behaviour.
        ii. For every separately stack crane-accessible (for example) stack group, bay, row and possibly also tier there is a separately described behaviour.

iii. For every separately transfer area crane-accessible (for example) buffer bay, row and possibly also tier there is a separately described behaviour.

(g) Thus we suggest
   i. one behaviour for each container vessel and
   ii. one separate, "embedded" behaviour for each separately quay crane-accessible (for example) ship bay and row.

(h) We could suggest the same for a quay:
   i. as one overall behaviour
   ii. composed from a number of "embedded" behaviours,
   iii. one for each quay crane, whether in use or idle.

   We will not do so presently. But we may have to do that later !

(i) We shall, in later sections add additional CTP processes.

46. Each behaviour "possesses" an own state:
   (a) The state of ship behaviours include information about the ship, including overall topological information about bays, rows and tiers.
   (b) The state of ship bay/row behaviours include the local state of all tiers within the scope of the bay/row behaviour.
   (c) The state of stack group/bay/row behaviours include the local state of all tiers within the scope of the group/bay/row behaviour.
   (d) The state of transfer area buffer behaviours include the local state of all tiers within the scope of the transfer area buffer behaviour.
   (e) The state of CTP vehicle behaviours include the local state of the vehicle: its current position, the zero, one or usually at most two containers that it might be transporting.
   (f) Etcetera. We leave it to the reader to complete, if necessary, the description of the state of the decomposed behaviours.
47. Two behaviours might need to synchronise and communicate.
48. Examples are
   (a) the quay crane and ship bay/row behaviours,
   (b) the quay crane and vehicle behaviours,
   (c) the vehicle and stack crane behaviours,
   (d) the stack crane and stack group/bay/row behaviours, and
   (e) the stack crane and transfer area either
      i. the transfer area buffer or behaviours, or
      ii. the transfer area truck, train or barge behaviours.
   (f) Their synchronisation and communication takes place when containers are being "handed over".
49. The behaviours will be modelled in terms of CSP-like processes.
50. The synchronisation and communication then takes place via and over CSP-like channels.

A.5.5 Sketches of Behaviour Formalisations

In this section (Sect. A.5.5) we shall further analyse the container line industry behaviour, more specifically the ship (in port), quay crane and (CTP) vehicle behaviours. But we shall do so "from the point of view" of abstract modelling ! That is, concerns of formal specification possibilities will now play a not in-significant rôle in our choice also of informal narrative description ! So, dear reader, please accept that considerations of formalisation "creep" into our informal narrative. You should still be able to read just the informal text skipping the formulas !

Ship, Crane and Vehicle States. We model only position and container-related components of respective states.

We leave (ship) quay, crane and vehicle positions undefined.

The container vessel, i.e., the ship, state reflects for every bay, row and tier identifier a list of either empty cells or cells with containers, and the set of quay positions "from which" a crane can "reach" the bay, row and tier.[29]

type
Quay_Pos, Quay_Cra_Pos, Veh_Pos
SBRΣ, = ((Bi×Ri×Ti) $\overrightarrow{m}$ Cell* × Quay_Pos-**set**) × ...
Cell == empty | mkC(c:C)

The quay crane state reflects the current position, along the quay, of the crane and whether it is currently transferring a container (or two).

type
Quay_CraΣ = Quay_Cra_Pos × optC × optC × ...
optC == empty | mkC(c:C)

The vehicle state reflects the current position, "around" the CTP, of the vehicle and whether it is currently transporting a container (or two).

type
VehΣ = Veh_Pos × optC × optC × ...

CTP, Ship, Crane and Vehicle Process Signatures. We shall present and discuss the signatures of four behaviours: the CTP behaviour, ship behaviours indexed by vessel name and a bay identifier — where that index shall indicate that there may be other ship behaviours "covering" other same ship bay identifiers[30]; crane behaviours indexed by crane name; and vehicle behaviours indexed by vehicle name. The CTP behaviour has no index: it serves all ship, crane and vehicle behaviours. All processes "embodies an own" state, σ, here shown as a function (i.e., a function) argument being iteratively passed on in some updated form (σ').

value
ctp: CTPΣ → **Unit**
ctp(ctpσ) ≡ (... ctp(ctpσ′))

quay_crane: CraNm → CraΣ → **Unit**
quay_crane(cn)(cσ) ≡ (... quay_crane(cn)(cσ′))

vehicle: VehNm → VehΣ → **Unit**
vehicle(vn)(vσ) ≡ (... vehicle(vn)(vσ′))

stack_crane: StkCraNm → CraΣ → **Unit**
stack_crane(scn)(scσ) ≡ (... stack_crane(scn)(scσ′))

ship_bay: CVNm × Bid → SBRΣ → **Unit**
ship_bay(vn,bi)(sσ) ≡ (... ship_bay_row(vn,bi)(sσ′))

29

RSL.27: The type definition A = B $\overrightarrow{m}$ C defines A to designate the class of all maps (i.e., finite, enumerable domain functions) from B elements into C elements.
RSL.28: The suffix Σ is chosen (as a pragmatics) to indicate that AΣ designates a state component.

30 Thus we simplify, without loss of generality, a crane to serve an entire bay — but the model allows several cranes to serve the same bay !

We[31] have just very crudely indicated that the "bodies" of the three process definitions "tail recurse" (as in an iterative **while true do** loop), that is, that the CTP processes do not terminate — hence the **Unit** clause.

CTP, Ship, Crane and Vehicle Channels. The idea is to model synchronisation and communication between CTP and ships, cranes and vehicles informing — as possibly requested by — them of details of their next actions, ship and crane processes (when lifting [dropping] containers) and crane and vehicle processes (when dropping [resp. lifting] containers) by means of 'messages' sent across channels between these processes. Actual channels, at this level of exposition of the container line domain, are:

type
  a CVNm, QCraNm, SCraNm, VehNm, Gid, Bid
  b M_CTP_Shp, M_CTP_QCra, M_CTP_Veh,
  c M_Shp_QCra, M_QCra_Veh, M_Veh_SCra, M_SCra_Stk
value
  d ss:(CVNm $\overrightarrow{m}$ Bid-**set**),
  e qcs:QCraNm−Set, vs:VehNm-**set**, scs:SCraNm-**set**
  f stk:(Gid $\overrightarrow{m}$ Bid-**set**)
channel
  g {ctp_shi[i]|i:**dom** ss}:M_CTP_Shp
  h {ctp_qcra[i]|i:qcs}:M_CTP_QCra,
  i {ctp_veh[i]|i:vs}:M_CTP_Veh
  j {ctp_scra[i]|i:qs}:M_CTP_QCra,
  k {shi_cra[i,j,k]|i:**dom** ss,j ∈ ss(i),k:qcs}:M_Shp_QCra,
  l {qcra_veh[i,j]|i:qcs,j:vs}:M_QCra_Veh,
  m {scra_veh[i,j]|i:scs,j:vs}:M_SCra_Veh,
  n {scra_stk[i,j,k]|i:scs,j:**dom** stk,k:stk(j)}:M_Scra_Stk

Annotations:

- (a) Names of container vessels, quay cranes, stack cranes, and vehicles, and identifiers of stack groups and stack bays.
- (b) Types of entities communicated between CTPs and vessels, CTPs and quay cranes, CTPs and vehicles.
- (c) Types of entities communicated between vessels and quay cranes, quay cranes and vehicles, vehicles and stack cranes and stack cranes and stacks.
- (d) There is a value, ss, which to every container vessel associates a set of bays, hence bay identifiers.[32]
- (e) qcs, vs and scs defines a set of quay crane, vehicle, respectively stack crane names.
- (f) The value stk which to every CTP stack group associates a set of bays (known by their identifications). hence bay identifiers.
- (g) [33] There is a set of channels, ctp_shp, which serve as means for synchronisation and communication between CTP and ship (vessel) behaviours. This set is indexed by vessel names.

31

RSL.29: The **Unit** literal in the function f signature f: A → Σ → **Unit** designates that the function f is a process that never terminates.

32

RSL.30: The **value** nm:Type clause defines nm to be an arbitrarily selected (or chose) value of type Type.

33

- (h-i-j) ctp_qcra: CTP quay crane, ctp_veh: CTP vehicle and ctp_scra: CTP stack crane channels.
- (k) The channels shi_cra serve to synchronise and communicate between ship (i.e., vessel) and quay crane behaviours — hence the triple indexing over ship names, their bay identifiers and quay crane names.
- (l-m) qcra_veh: quay crane vehicle, scra_veh: stack crane vehicle channels.
- (n) The channels scra_stk serve to synchronise and communicate between stack crane and stack group bay behaviours — hence the trip indexing over appropriate names.

Channel Messages, I. Let us now analyse the interactions between the CTP, ship bay, crane and vehicle behaviours. We focus on the transfer of containers between ships and vehicles. We formulate this analysis in terms of archetypal behaviours.

First a crane requests and receives information from the CTP as to whether a container transfer is from a ship to a vehicle or the reverse, or there is no job. and with this information follows further, "as appropriate", details. If a crane container transfer is from a ship to a vehicle: then a crane (c) requests and (d) receives permission from a ship bay to "lift" a container from the designated tier; then the crane, having obtained the container by applying its spreader to the top of the designated tier, (e) requests and (f) receives permission to "drop" that container from the designated vehicle; and finally the crane places the container on the vehicle. Similar for transfers from vehicles to ships. This analysis gives rise to the following channel message types:

type
a. JobNm
b. BRT = Bid×Rid×Tid
c. M_CTP_Cra = Cra_to_CTP | CTP_to_Cra | ...
d. Cra_to_CTP == Req_Job(cp:CraPos) | Fin_Job(jn:JobNm)
e. CTP_to_Cra == Job_SV(m:CTP_Cra_M) | Job_VS(m:CTP_Cra_M) | ... | no_job
f. CTP_Cra_M = JobNm×CVNm×QuayPos×BRT×Cn×VehNm
g. M_Shp_Cra = Cra_to_Shp | Shp_to_Cra
h. Cra_to_Shp == Req_Lift(m:Cra_Shp_M,cn:Cn) | Lift(m:Cra_Shp_M,cn:Cn) |
Req_Drop(m:Cra_Shp_M,cn:Cn) | Drop(m:Cra_Shp_M,c:C)
i. Cra_Shp_M = CVNm×QuayPos×BRT
j. Shp_to_Cra == ok_lift | ok_drop | not_ok_lift | not_ok_drop | mkC(c:C)
k. M_Cra_Veh = Cra_to_Veh | Veh_to_Cr
l. Cra_to_Veh == Req_Lift(cn:Cn) | Lift(cn:Cn) | Req_Drop(cn:Cn) | Drop(c:C)
m. Veh_to_Cra = Shp_to_Cra

Annotations: We loosly annotate annotate the above type definitions.

- (a.) JobNm designates a further unidentified class of job names. Each job, i.e., each task assigned by the CTP to either quay cranes, vehicles or stack cranes will be given a unique job name.
- (b.) BRT designates the class of triplet indentifiers of Bays, Roes and Tiers.
- (c.) M_CTP_Cra designates the disjoint classes of quay crane to CTP messages, Cra_to_CTP, and CTP to quay crane messages. CTP_to_Cra.
- (d.) Cra_to_CTP designates the disjoint classes of job requests, Req_Job(cp:CraPos), and job finished notifications, Fin_Job(jn:JobNm).

RSL.31: The definition **channel** ch:M declares ch to be a channel, i.e., a means of synchronisation and communication of messages of type M between processes.
RSL.32: The definition **channel** {ch[i]|i:set}:M declares a number (**card**set) of indexed channels of type M.

- (e.) CTP_to_Cra designates the disjoint classes of container transfer from ship to vehicle job assignments, Job_SV(m:CTP_Cra_M), vehicle to ship assignments, Job_VS(m:CTP_Cra_M), etc.: ..., and no_job assignment.
- (f.) CTP_Cra_M designate the class of groupings (Cartesians) of job names, JobNm, vessel names, CVNm, quay positions, QuayPos (not used in the below model), bay-row-tier locators, BRT, container names, Cn, and vehicle names, VehNm.
- (g.) M_Shp_Cra designates the disjoint classes of crane to ship, Cra_to_Shp, and ship to crane, Shp_to_Cra, communications.
- (h.) The crane to ship, Cra_to_Shp, communications either (1) requests, Req_Lift(m,cn), through the m triplet of vessel name, sn:CVNm, quay position qp:QuayPos (not used in this model), and bay, row and tier locator, brt:BRT, and a container name, cn:Cn, that a container be lifted from the vessel, or (2) the communication: Lift(m,cn) designates the actual lifting of the container from the vessel, or (3) requests, Req_Drop(m:Cra_Shp_M,cn:Cn), through the triplet of vessel name, CVNm, quay position QuayPos (not used in this model), and bay, row and tier locator, BRT, and a container name, cn:Cn, a container to be dropped, i.e., placed, onto the vessel, or (4) the communication: Drop(m,c) designates the actual dropping of the container onto the vessel.
- (i.) The crane to ship (as well as the crane to vehicle) lift and drop messages, Cra_Shp_M, all contain the triplet information: vessel name, CVNm, (unused) quay position, QuayPos, and bay-row-tier locator, BRT.
- (j.) The ship to crane "response", Shp_to_Cra, is either an ok_lift, an ok_drop, a not_ok_lift, a not_ok_drop, or it is the actual container, mkC(c:C). The same response, see item (m.), is also that from vehicles to quay cranes.
- (k.) The interactione between quay cranes and vehicles, M_Cra_Veh, form two disjoint classes of communications: Cra_to_Veh and Veh_to_Cr.
- (l.) The crane to vehicle communications, Cra_to_Veh, either (1) requests, Req_Lift(cn:Cn) (like in (h.1)) that a container be lifted from the vehicle, or (2) that it actually be lifted, Lift(cn:Cn), or (3) requests, Req_Drop(cn:Cn), that a container to be dropped, i.e., placed, onto the vehicle, or (4) that it actually be dropped, Drop(c:C).
- (m.) For Veh_to_Cr see Item (j.) substituting, instead of ship, the term vehicle.

Ship, Crane and Vehicle Process Definitions: Interactions. We show only the CTP and quay crane interaction:

```
value
  ctp: CTPΣ → in,out ctp_cra[∗] Unit
  ctp(ctpσ) ≡
1.  (let ctp_cra_fct(i)(m) =
2.   cases m of
3.     Req_Job(cp) →
4.       let (cra_job,ctpσ′) = next_cra_job(i)(ctpσ) in
5.       ctp_cra[i]!cra_job; ctp(ctpσ′) end,
6.     Fin_Job(jn) →
7.       ctp(upd_cra_ctpσ_fin(jn)(ctpσ)) end in
8.   [] { let m = ctp_cra[i]? in ctp_cra_fct(i)(m) end | i:QCraNm })
9.  [] ...

  next_cra_job: QCraNm → CTPΣ → CTP_to_Cra_M × CTPΣ
  upd_cra_ctpσ_fin: Jn → CTPΣ → CTPΣ
```

Annotations:

- Line 8. expresses the main "loop" of the CTP behaviour wrt. cranes.
- Non-deterministically ([]) the CTP expresses willingness to engage with any crane, eventually receiving a message from some crane i.
- Then the CPT performs some actions in preparing and possibly delivering a response to the interacting crane.
- These actions are expressed in terms of the function invocation ctp_fct(i)(m).
- The crane is either requesting a job (3.), or is informing that a job has been completed (6.).
- If the crane is requesting a job then the CTP inquires its state for a next job for that crane (4.).
- This inquiry yields basically what is expressed in a CTP to crane message: either a ship to vehicle container transfer, or the reverse, or no job (4.).
- The inquiry on the state changes the state recording that an inquiry has been made and that a certain response has likewise been made (4.).
- The CTP informs the crane of its response (5. first part).
- And the CTP reverts to "itself" — i.e., making itself ready to engage with other behaviours (5. last part).
- If the crane is informing that a job has been completed then the CTP records that event in its state — and reverts to "itself" (7.).

```
value
  quay_crane: qcn:QCraNm → CraΣ → out,in ctp_cra[ qcn ]  Unit
  quay_crane(qcn)(cσ) ≡
1.   (ctp_cra[ qcn ]!ReqJob(obs_QuayPos(cσ));
2.    let m = ctp_cra[ qcn ]? in
3.    case m of
4.      Job_SV(job) → fct_sv(qcn)(job)(cσ),
5.      Job_VS(job) → fct_vs(qcn)(job)(cσ),
6.      ... → ...,
7.      no_job → quay_crane(qcn)(cσ)
8.    end end)
9.  []
        ...
```

Annotations: The quay_crane behaviour defines how a quay crane may abstractly interact with the CTP and effect jobs involving container transfers between a ship bay and a vehicle — in either direction.

- (1.) The quay crane behaviour inquires with the CTP: is there a next job for it, and then which.
- (2.) The CTP behaviour responds with a message.
- (3.) If the nessage is
  - (4.) a job description for a container transfer from a vessel to a vehicle then that function, fct_sv is performed; else if it is
  - (5.) a job description for a container transfer from a vehicle to the vessel then that function, fct_vs is performed; else if
  - (6.) ... (other jobs, like "move the crane", etc., are not detailed here); else if the job is

– (7.) a no job message, then the quay crane behaviour "reverts to itself".

The fct_sv and fct_vs operations likewise revets to the quay crane behaviour.

- (9.) The quay crane behaviour may non-deterministically engage in other behaviours — but these are not detailed here.

```
value
  fct_sv: qcn:QCraNm→Cra_Shp_M→CraΣ→out,in shp_cra[sn,∗,∗] out,in cra_veh[qcn,∗] Unit
  fct_sv(qcn)(jn,sn,_,qcn,(bi,ri,ti),vn)(cσ) ≡
a.  shp_cra[sn,bi,qcn]!Req_Lift(qp,(bi,ri,ti),cn);
b.  let cσ′ =
c.    if shp_cra[sn,bi,qcn]?≠ok_lift then chaos end;
d.    shp_cra[sn,bi,qcn]!Lift(qp,(bi,ri,ti),cn);
e.    let c = shp_cra[sn,bi,qcn]? in
f.    cra_veh[qcn,vn]!Req_Drop(qp,cn);
g.    if cra_veh[qcn,vn]?≠ok_drop then chaos end;
h.    cra_veh[qcn,vn]!Drop(c);
i.    ctp_cra[cn]!Fin_Job(jn);
j.    upd_cσ_on_sv_job(jn,sn,_,(bi,ri,ti),cn,vn)(cσ) in
k.  quay_crane(cn)(cσ′) end
```

Annotations: This behaviour describes how a quay crane interacts with a specific ship sn bay (bi,ri,ti) and a specific vehicle vn to transfer a container (identified by vn) from the ship to the vehicle.

- (a) The quay crane informs the ship that it wishes to lift container (identified by cn) from tier (bi,ri,ti). (In this model we do not describe what happens if the quay crane position, qp, does not "match up" with the ship's bay/row/tier position.)
- (b) The entire transfer operation changes the quay crane state (to σ').
- (c) The ship is expected to respond to the lift request. If it does not respond then we do not describe, in this model, what then happens. If it responds, shp_cra[sn,bi,qcn]?, and the response is not an OK to the lift request then **chaos** ensues, that is: we do not specify what happens !
- (d) Otherwise the quay crane operation proceeds with the actual lift.
- (e) The lift is now expected to result in a(n appropriately identifed) container, c.
- (f) The quay crane now moves across and requests permission from the designated vehicle to drop a container.
- (g) If the vehicle responds that it is not OK to drop the container then **chaos** ensues.
- (h) The quay crane then drops the container onto the vehicle,
- (i) and informs the CTP that its current job assignment has finished.
- (j) The quay crane then updates its local state and
- (k) reverts to itself in that updated state.

  The ship or the vehicle may decide to respond with other than OK to lift, respectively drop a located and designated container either because the designated location (i.e., tier) does not have a container of the appropriate identity on it current top, repectively because the vehicle is expecting another container.

```
  fct_vs: qcn:QCraNm→Cra_Shp_M→CraΣ→out,in shp_cra[sn,∗] out,in cra_veh[∗] Unit
  fct_vs(cn)(jn,sn,_,brt,cn,vn)(cσ) ≡ /∗ left to reader ∗/
```

```
value
  ship_bay: CVNm → SBRΣ → Unit
  ship_bay(cvnm)(sbrσ) ≡ /∗ exercise for the reader ∗/
```

Channel Messages, II. Let us now analyse the interactions between the CTP, vehicle and stack behaviours. We focus on the transfer of containers between quay cranes and stacks or there is no job. We formulate this analysis in terms of archetypal behaviours.

First a vehicle requests and receives information from the CTP as to whether a container transfer is from a quay crane to a stack or the reverse, and with this information follows further, "as appropriate", details.

If the transfer is from a quay crane to a stack crane then the vehicle awaits request from a quay crane to drop a container of the right identify and OKs that request whereupon the vehicle accepts the container if it does indeed have the right identity. The vehicle then moves to an appropriately identified stack crane position; requests that crane to lift the properly identified container; and then delivers that container to the stack crane spreader. If any of the conditions implied above fails then we leave it undefined as to what then happens !

```
type
  SCraNm
  M_CTP_Veh = Veh_to_CTP | CTP_to_Veh
  Veh_to_CTP == Req_Job(vp:VehPos) | Fin_Job(jn:JobNm)
  CTP_to_Veh == Job_CS(j:CS_Job) | Job_SC(j:CS_Job) | ... | no_job
  CS_Job = JobNm×QCraNm×QCraPos×Cn×SCraNm×SCraPos
  GBRTid = Gid×Bid×Rid×Tid
  M_Cra_Veh = /∗ defined above ∗/ Page 58
  M_Veh_Stk = Veh_Stk | Stk_Veh
  Veh_Stk == Req_Lift(m:GBRTid,cn:Cn) | Lift(m:GBRTid,cn:Cn) |
             Req_Drop(m:GBRTid,cn:Cn) | Drop(m:GBRTid,c:C)
  Stk_Veh == ok_lift | not_ok_lift | ok_drop | not_ok_drop | mkC(c:C)
```

Crane, Vehicle and Stack Crane Process Definitions: Interactions. For the CTP we show only the CTP and vehicle interaction:

```
value
  ctp: CTPΣ → in,out ctp_veh[∗] Unit
  ctp(ctpσ) ≡
    ...
    ⌈⌉
    (let ctp_veh_fct(i)(m) =
     cases m of
       Req_Job(cp) →
         let (veh_job,ctpσ′) = next_veh_job(i)(ctpσ) in
         ctp_veh[i]!veh_job; ctp(ctpσ′) end,
       Fin_Job(jn) →
         ctp(upd_veh_cptσ_fin(jn)(ctpσ)) end in
     [] { let m = ctp_veh[i]? in ctp_veh_fct(i)(m) end | i:VehNm })
    ⌈⌉ ...

  next_veh_job: CraNm → CTPΣ → CTP_to_Cra_M × CTPΣ
  upd_veh_cptσ_fin: Jn → CTPΣ → CTPΣ
```

```
value
  vehicle: vn:VehNm → VΣ → out,in ctp_veh[ vn ] Unit
  vehicle(vn)(vσ) ≡
    (ctp_veh[ vn ]!Req_Job(obs_VehPos(vσ));
     let m = ctp_veh[ vn ]? in
     case m of
       Job_CS(job) → fct_cs(vn)(job),
       Job_SC(job) → fct_sc(vn)(job),
       ... → ...,
       no_job → vehicle(vn)(vσ)
     end end)
    ⌈⌉
    ...

value
  fct_cs: vn:VehNm → CS_Job → VΣ → in,out veh_cra[ vn,∗ ] Unit
  fct_cs(vn)(jn,qcn,qcp,cn,scn,scp,gbrti)(vσ) ≡
    let vσ′ = move_vehicle(obs_VehPos(vσ),crapos) in
    let m = qcra_veh[ qcn,vn ]? in
    case m of
      Req_Drop(qp,cn′) →
        if qp=qcp ∧ cn′=cn then qcra_veh[ qcn,vn ]!ok_drop else chaos end;
        if obs_Cn(qcra_veh[ qcn,vn ]?)≠cn then chaos end;
        let vσ″ = move_vehicle(crapos,xtr_Pos(stkcran,gbrti)) in
        scra_veh[ scn,vn ]!Req_Lift(cn,gbrti);
        if scra_veh[ scn,vn ]?≠ok_lift then chaos end;
        scra_veh[ scn,vn ]!Lift(cn,gbrti);
        ctp_veh[ vn ]!Fin_Job(jn)
        vehicle(vn)(vσ″) end
      _ → chaos
    end end end
```

A.5.6 Container Stack Stowage

More to come — before July 2007

A.6 Net of Sea Lanes

By a sea lane net we shall understand a set of CTPs and a set of sea lanes. By a sea lane we shall understand the designation of a set of two distinct CTP names, possibly the coordinates of positions on the high sea through which the lane "passes", the length of the sea lane, and possibly other things. A sea lane net must be well-formed: all sea lane CTP name designations must be names of CTPs of the net

```
type
  N, CTPNm, L, Len
value
  obs_CTPNms: (N|L) → CTPNm-set
  obs_Ls: N → L-set
axiom
  ∀ n:N • ∀ l:L • l ∈ obs_Ls(n) ⇒
      card obs_CTPNms(l) = 2 ∧ obs_CTPNms(l)⊆obs_CTPNms(n)
```

A.6.1 Sea Routes

A sea route of a given sea lane net is a sequence of two or more CTP names, such that pairwise adjacent CTP names of the route correspond to a sea lane of the net.

```
type
  SeaRt = {|sr:CTPNm*•len sr≥2|}
value
  is_SeaRt: SeaRt → N → Bool
  is_SeaRt(sr)(n) ≡
    ∀ i:Nat • {i,i+1} ∈ inds sr ⇒
      ∃ l:L • l ∈ obs_Ls(n) ∧ {sr(i),sr(i+1)}=obs_CTPNms(l)
```

A.6.2 Sea Routes of a Net

A sea lane net gives rise to a set of sea routes: namely all those sequences of two or more CTP names, such that pairwise adjacent CTP names of the route correspond to a sea lane of the net.

```
value
  sea_routes: N → SeaRt-set
  sea_routes(n) as srs
    post ∀ sr:SeaRt • is_SeaRt(sr)(n) ≡ sr ∈ srs
```

A.6.3 Connected CTPs

A pair of CTPs are connected if there is a sea route from one to the other.

```
value
  is_connected_CTPp: CTPNm × CTPNm → B → Bool
  is_connected_CTPp(hf,ht)(n) ≡
    ∃ sr:SeaRt • sr ∈ sea_routes(n) ∧ hd sr=hf ∧ sr(len sr)=ht
```

A.7 Container Lines

51. A container line (CL) is an enterprise
    (a) which operates (owns or [lease] rents) a number of container vessels (CV),
    (b) where these vessels regularly, according to more-or-less fixed time tables (TT), serves a number of container terminal ports (CTPs) along a (container line) route (CR),
    (c) accepting export containers at these CTPs for carriage on their vessels and discharging these containers at other CTPs along their route network,
    (d) where such a network consists of all the (container line) routes.
    (e) One or more line vessels may serve the same route, and then most likely at different times.

```
type
  CL, TT, CR, NW
value
  obs_CVs: CL → CV-set, obs_TT: CL → TT,
  obs_NW: CL → NW, obs_CRs: NW → CR-set
  is_connected:NW→Bool
```

52. The container line route network is usually a connected network, i.e., it is possible to reach any CTP in the network from any other CTP in the same network via one or more container line routes.
53. A container line route can be designated by a sequence of two or more container port visits.
54. A container port visit can be designated by a triple:
    (a) an estimated time of arrival (Time),
    (b) the name of the container terminal port (CTPNm), and
    (c) an estimated time of departure (Time).

```
type
  CR′ = CTP_Visit*
  CR = {| cr:CR′ • is_wf_CR(cr′) |}
  CTP_Visit = Time × CTPNm × Time
value
  is_wf_CR: CR′ → Bool
  is_wf_CR(cr) ≡
    len cr≥2 ∧
    ∀ i:inds cr • {i,i+1}⊆inds cr ⇒
      let (,,dt) = cr(i), (ar′,,dt′) = cr(i+1) in
      before(dt,ar′) ∧ before(ar′,dt′) end

  before: Time × Time → Bool

  ports_visited: CR → CTPNm-set
  ports_visited(cr) ≡ {n|n:CTPNm•∃ (,n′,):CTP_Visit•(,n′,) ∈ elems cr∧n=n′}


value
  is_connected: NW → Bool
  is_connected(nw) ≡
    let crs = obs_CRs(nw) in
    ∀ cr:CR • cr ∈ crs •
      ∀ n:CTPNm • n ∈ ports_visited(cr) ⇒
        ∃ cr′:CR • n ∈  ports_visited(cr′) end
```

55. The container line business consists of one or more container lines.
    (a) Each container line has a unique container line name.
    (b) Container vessels across all container lines have unique container vessel names.
    (c) The container line networks are connected.

```
type
  CLB, CLNm
value
  obs_CLs: CLB → CL-set, obs_CLNm: CL → CLNm
  is_connected_NWs: CLB → Bool
  is_connected_NWs(clb) ≡
    let cls = obs_CLs(clb) in
    ∀ cl:CL • cl ∈ cls •
      let ps = line_ports(cl) on
      ∀ p:CTPn•p ∈ ps ⇒ ∃ cl′:CL•cl′ ∈ cls∧cl≠cl′⇒p′ ∈ line_ports(cl′)
```

end end

line_ports: CL → CTPNm-**set**
line_ports(cl) ≡
let nw = obs_NW(cl) **in let** crs = obs_CRs(nw) **in**
∪{ports_visited(cr)|cr:CR•cr ∈ crs} **end end**

A.8 Bill of Ladings

To explain the container line operations of the acceptance and discharge of a container that is, of receiving from and delivering to a shipper a container (being shipped), we need to first introduce the concepts of carrier, shipper and bill of lading (BoL).

56. A carrier is here taken to be the same as a container line.
57. A shipper is ([11]) the merchant (person) by whom, in whose name or on whose behalf a contract of carriage of goods has been concluded with a carrier or any party by whom, in whose name or on whose behalf the goods are actually delivered to the carrier in relation to the contract of carriage.

type
Shipper, ShNm
value
obs_ShNm: Shipper → ShNm
ship: ShNm × C × CLNm × BoL_Info → BoL

58. A bill of lading (BoL) is document which evidences a contract of carriage by sea [11].

    The document has the following functions:

    (a) A receipt for goods, signed by a duly authorised person on behalf of the carriers, i.e., the container line.
    (b) A document of title to the goods described therein.
    (c) Evidence of the terms and conditions of carriage agreed upon between the two parties: shipper and carrier.

59. At the moment 3 different kinds of bill of ladings are used [11]:

    (d) A document for either Combined Transport or Port to Port shipments depending whether the relevant spaces for place of receipt and/or place of delivery are indicated on the face of the document.
    (e) A classic marine Bill of Lading in which the carrier is also responsible for the part of the transport actually performed by himself.
    (f) Sea Waybill: A non-negotiable document, which can only be made out to a named consignee. No surrender of the document by the consignee is required.

We shall, for illustration, model the classic marine Bill of Lading, Item 59e.

MORE TO COME — BEFORE JULY 2007

A.9 Logistics

More to come — before July 2007

A.10 Customer Interface to the Container Line Industry

More to come — before July 2007

A.11 Etcetera !

More to come — before July 2007

A.12 Open Issues of Requirements

We have modelled many parts of the container line industry. But professionals of that industry will fail to read anything about most of things that are their daily concern. We are referring to such obvious container line industry concerns as optimisation of stowage, optimisation od crane split, optimisation of vehicle allocation to container jobs, the combined optimisation of stowage, crane split, and vehicle allocation to container jobs, etc. Our obvious answer to this is, of course: All those concerns, and many more belong to requirements to business processes as well as to support software. As such it follows, according to our principles that we express each and every one of these concerns after the domain engineering phase and as part of the business process re-engineering and the software requirements development phase.

More to come — before July 2007

A.13 Domain Acquisition

Acquisition of information for the current domain modelling took place over the years. Apart from general observations of containers, container vessels and container terminals ports — it is hard to avoid observing these phenomena when living in or moving in and out, for many years, of such ports (Hong Kong, Singapore) — special insight was most kindly provided by Maersk Line's Stowage Manager at Singapore and through visits to the huge container terminal port: PTP, Port of Tanjung Pelepas, in Malaysia, but quite close to Singapore. Around 2004 I studied:

> P&O Nedlloyd: A–Z Shipping Terms. Electronically, on the Web: `http://www.ponl.com/-topic/home_page/language_en/about_us/useful_information/az_of_shipping_terms`, 2004.

It seems that that URL has been replaced by the equally substantial [11]. We encourage the reader to study [11].

General introductions, seen from the point of view of the training of port workers are provided by the ILO's (International Labour Organisation's) Portworker Development Programme, see [12]. In a number of course units (The ILO PDP Units) guidance is given for the training of port workers worldwide. For a list of these see [13].

The general issues of stowage are briefly covered in [14, 10, 9, 15], in [16, 17] and in [18]. The papers focuses on the requirements (not a domain) issue of optimal stowage. The reader is encouraged to read the abstracts of these papers. See Sect. 9. The literature review, Sect. 2 of

[18], provides what appears to be quite a nice survey of several stowage-related papers including the above referenced.

More general container port issues are covered in [19, 20]. [20] presents a particularly thorough analysis of well-nigh all aspects of container terminal operations; it also provides a very thorough literature review (212 cited papers !). The reader is encouraged to read these papers. See Sect. 9.

We refer the reader to [21] for information about the ship dynamics terms: center of gravity, meta-centric height, list, rime, draft, drag, etc., and their calculations.

B An RSL Syntax & Abstraction Primer

We bring an ultra–short "recap" of RSL: its syntax and some of the abstraction ideas.

The recap is, alas, just an overview of the syntax of main aspects of RSL and an overview of some abstraction, i.e., model choices made possible by, for example, RSL.

For proper explanation of the meaning (i.e., semantics), and, of course, the proper use (i.e., pragmatics) of this syntax, we refer to [3, 4, 1].

B.1 RSL Types

B.1.1 Type Expressions

Let A, B, and C be any type names or type expressions, then:
(save the [i] line numbers) exemply generic type expressions:

1. The Boolean type of truth values **false** and **true.**
2. The integer type on integers ..., -2, -1, 0, 1, 2, ...
3. The natural number type of positive integer values o, 1, 2, ...
4. The real number type of real values, i.e., valuse whose numerals can be written as an integer, followed by a priod ("."), followed by a natural number (the fraction).
5. The character type of character values "a", "b", ...
6. The text type of character string values "aa", "aaa", ..., "abc", ...
7. The set type of finite set values, see below.
8. The set type of infinite set values.
9. The Cartesian type of Cartesian values, see below.
10. The list type of finite list values, see below.
11. The list type of infinite list values.
12. The map type of finite map values, see below.
13. The function type of total function values, see below.
14. The function type of partial function values.
15. In (A) A is constrained to be
    - either a Cartesian $A \times B \times ... \times D$, in which case it is identical to type expression kind 9,
    - or not to be the name of a built–in type (cf., 1–6) or of a type, in which case the parentheses serve as simple delimiters, eg.: $(A \xrightarrow[m]{} B)$, or (A^*)**-set**, or (A**-set**)list, or $(A|B) \xrightarrow[m]{} (C|D|(E \xrightarrow[m]{} F))$, etc.

16. The (postulated disjoint) union of types A, B, ..., and C.
17. The record type of mk_id–named record values mk_id(av,...,bv), where av, ..., and bv, are values of respective types. The distinct identifiers sel_a, etc., designate selector functions.
18. The record type of unnamed record values (av,...,bv), where av, ..., and bv, are values of respective types. The distinct identifiers sel_a, etc., designate selector functions.

B.1.2 Type Definitions

Subtypes: The set of elements b of type B which satisfy the predicate $\mathcal{P}$ is a sub–type (of type B):

Sorts or Abstract Types: Sorts (i.e., abstract types) A, B, ..., C are introduced when specifying:

Concrete Types: Concrete types are introduced when specifying:

BNF Rule Right–hand Sides for Concrete Type Definitions: where a form of [2–3] is provided by the combination:

B.2 The RSL Predicate Calculus

B.2.1 The RSL Proposional Expressions

Let identifiers (or propositional espressions) a, b, ..., c designate Boolean values. Then: are propositional expressions, all having a Boolean value. $\sim$, $\wedge$, $\vee$, $\Rightarrow$, and $=$ are Boolean connectives (i.e., operators) and "read" as not, and, or, if-then (or implies), equal and not-equal.

B.2.2 The RSL Predicate Expressions

Simple RSL Predicate Expressions Let identifiers (or propositional espressions) a, b, ..., c designate Boolean values, and let x, y, ..., z (or term expressions) designate other than Boolean values, and let i, j, ..., k designate number values, then: are simple predicate expressions.

Quantified RSL Expressions Let X, Y, ..., C be type names or type expressions, and let $\mathcal{P}(x)$, $\mathcal{Q}(y)$ and $\mathcal{R}(z)$ designate predicate expressions in which z, y, and z are free. Then: are quantified expressions, are also predicate expressions, and are "read" as: For all x (values in type X) the predicate $\mathcal{P}(x)$ holds; there exists (at least) one y (value in type Y) such that the predicate $\mathcal{Q}(y)$ holds; and: there exists a unique z (value in type Z) such that the predicate $\mathcal{R}(z)$ holds.

B.3 RSL Sets, Cartesians, Lists, and Maps

B.3.1 RSL Set, Cartesian, List, and Map Enumerations

Sets: Let the below as denote values of type A, then the below designate simple set enumerations: The expression, last line below, to the right of the $\equiv$, expresses set comprehension.

comprehensionset@{ ○ | ○ : ○ • ○ }!set comprehension

Cartesians:

Lists: Simple enumerations:
The last line above assumes e_i and e_j to be integer valued expressions. It then expresses the set of intergers from the value of e_i to and including the value of e_j. If the latter is smaller than the former then the list is empty.

The last line below expresses list comprehension. comprehensionlist@⟨ ∘ | ∘ **in** ∘ • ∘ ⟩!list comprehension

Maps: Simple map enumerations:
The last line below expresses map comprehension:
comprehensionmap@[∘ ↦ ∘ | ∘ : ∘ • ∘]!map comprehension

B.3.2 RSL Set Operations

- ∈: The membership operator (*is an element member of a set,* **true** *or* **false***?*); ∈!set membership
- ∉: The non-membership operator (*is an element not a member of a set,* **true** *or* **false***?*);
- ∪: The infix union operator (when applied to two sets expresses the set whose members are in either or both of the two operand sets); ∪!set union
- ∪: The distributed prefix union operator (when applied to a set of sets expresses *the set whose members are in some of the sets of the operand set*);
- ∩: The infix intersection operator (expresses *the set whose members are in both of the two operand sets*); ∩!set intersection
- ∩: The distributed prefix intersection operator (when applied to a set of sets expresses *the set whose members are in all of the sets of the operand set*);
- \: The set complement (or set subtraction) operator (expresses *the set whose members are those of the first operand set which are not in the second operand set*); /!set difference\!set complement
- ⊂: The proper subset operator (*are the members of the first operand set all members of the second operand set, and are there members of the second operand set which are not in the first operands set,* **true** *or* **false***?*); ⊂!proper subset
- ⊆: The subset operator (as for proper subset, but allows equality of the two operand set to be **true**); ⊆!subset
- =(≠): The equal operator (*are the two operand sets the same (different),* **true** *or* **false***?*); and =!set equality≠!set in–equality
- **card**: The cardinality operator (*"counts" the number of elements* **card***!set cardinalityin the presumed finite operand set*).

B.3.3 RSL Cartesian Operations

B.3.4 RSL List Operations

- **hd**: Head: Yield the head (i.e., first) element of non–empty lists. head@**hd**!list head
- **tl**: Tail: tail@**tl**!list tailYield the list of list elements other than the head of the argument list (also only of non–empty lists) .
- **len**: Length: len@**len**!list lengththe length of a finite list.

- **inds**: Indices, or index set: inds@**inds**!list indicesYield the index set, from 1 to the length of the list (which may be empty in which case the index set is also empty, or may be infinite, in which case the result is **chaos**).
- **elems**: Elements: elems@**elems**!list elementsYield the possibly infinite set of all distinct elements of the list.
- $\ell(i)$: Indexing with a natural number, i, larger than 0 into a list ℓ larger than or equal to i yields its i'th element.
- $\widehat{}$: $\widehat{}$!list concatenationConcatenate two operand lists into one list, first the elements of the first, finite length operand list, and then the elements of the second, possibly infinite length operand list, and in their respective order.
- = and $\neq$: equal@=!list equalitynotequal@$\neq$!list inequalityCompare two operand lists for equality, element–by–element, respectively for the occurrence of at least one deviation!

B.3.5 RSL Map Operations

- $\bullet(\bullet)$: Application: map@$\circ(\circ)$!map applicationexpresses that functions and maps can be applied to arguments.
- **dom**: Domain/Definition Set: domain@**dom**!map definition set (domain)denote "taking" the definition set values of a map (the a values for which the map is defined).
- **rng**: Range/Image: range@**rng**!map rangedenote "taking" the range of a map (the corresponding b values for which the map is defined).
- †: Override/Extend: override@†!map overridewhen applied to two operands denote the map which is like an override of the first operand map by all or some "pairings" of the second operand map,
- $\cup$: Merege: union@$\cup$!map unionwhen applied to two operands denote the map which is the merge of two such maps,
- \: Restriction: restriction@\!map restrictionthe map which is a restriction of the first operand map to the elements that are not in the second operand set
- /: Restriction: restriction@/!map restrictionthe map which is a restriction of the first operand map to the elements of the second operand set.
- =, $\neq$: Equal, Not–Equal: equal@=!map equalityequal@$\neq$!map inequalitywhen applied to two maps, compares these for equality, respectively inequality.
- $\circ$: Composition: composition@$\circ$!map compositionThe map from definition set elements of the first, left–operand map, m_1, to the range elements of the second, right–operand map, m_2, such that if a, in the definition set of m_1 and maps into b, and if b is in the definition set of m_2 and maps into c, then a, in the composition, maps into c.

B.4 RSL λ–Calculus and Functions

B.4.1 The λ–Calculus Syntax

B.4.2 Free and Bound Variables

B.4.3 Substitution

B.4.4 α–Renaming and β–Reduction

B.4.5 The RSL λ–Notation

B.4.6 Function Signatures in RSL

B.4.7 Function Definitions in RSL

B.5 Applicative Constructs of RSL

B.5.1 The RSL let Constructs

General: Simple (i.e., non–recursive) let:
is an "expanded" form of:
Recursive let:

Predicative lets: expresses the selection of an a value of type A which satisfies a predicate $\mathcal{P}(\mathsf{a})$ for evaluation in the body $\mathcal{B}(\mathsf{a})$.

Patterns and Wild Cards: Some indicative examples:

B.5.2 The Applicative RSL Conditionals

B.5.3 Common Operator/Operand RSL Constructs

B.6 Imperative Constructs of RSL

B.6.1 Variables, Assignments and the Unit Value

B.6.2 Statement Sequence and skip

B.6.3 The Imperative RSL Conditionals

B.6.4 The Iterative RSL Conditionals

B.6.5 The Iterative RSL Sequencing

B.6.6 RSL Variable Expressions

B.7 RSL-CSP: Parallel Constructs of RSL

B.7.1 Process Channels

Let A, B and KIdx stand for a type of (channel) messages, respectively a (sort–like) index set over channels, then: declare a channel, c, and a set of channels, k[i], which can communicate values of the designated types.

B.7.2 Composition of Processes

Let P and Q stand for names of process functions, i.e., of functions which express willingness to engage in input and/or output events, i.e., in communication over channels.

Let P() and Q(i) stand for process expressions, then:
express the parallel of two processes, respectively the non–deterministic choice between two processes: Either external or internal.

B.7.3 Process Input/Output

Let c, k[i] and e designate a channel, a channel and a type A, resp., type B valued expression. Then:
expresses the willing of a process to engage in an event that reads an input, respectively that writes an output.

B.7.4 Process Signatures and Definitions

The below signatures are just examples. They emphasise that process functions must somehow express, in their signatyure via which channels they wish to engage in input and output events. The process function definitions (i.e., their bodies) express possible events.

B.8 Simple RSL Specifications

Not using schemes, classes and objects — See Chap. **??** — an RSL specification is some sequence one or more below **type,** zero, one or more **variable,** zero, one or more **channel,** one or more **value,** and zero, one or more **axiom** clauses.

C RSL-CSP: More on the Parallel Constructs of RSL

C.1 Processes and Their Parallel Composition

The synchronisation of sender/receiver processes (s, r) and the communication between them is modelled by what is called process input/output clauses.

The idea of these channels is that they serve as "carriers" of information, control or physical entities between processes. We illustrate this conceptually:

type
  M, SΣ, RΣ, Sx, Rx
channel
  ch:M
value
  σ_s:SΣ, σ_r:RΣ, xs:Sx, xr:Rx
  update_sσ: M → SΣ → SΣ, update_rσ: M → RΣ → RΣ

  s: Sx → SΣ **out** ch → **Unit**
  s(sx)(sσ) ≡ (... ch!m ... s(sx)(update_sσ(m)(sσ)))

  r: Rx → RΣ **in** ch → **Unit**
  r(rx)(rσ) ≡ (... **let** m=ch? **in** ... r(rx)(update_rσ(m)(rσ)) **end**)

  system: **Unit** → **Unit**
  system() ≡ s(xs)(σ_s)‖r(xr)(σ_r)

Annotations: The above formalisation is of generic nature. It is not specific to the container line industry example. The generic formulas are intended to illustrate the issue of non-terminating behaviours and how they can be modelled as non-terminating CSP-like processes.

- M, SΣ, RΣ, Sx, and Rx designate messages, sender behaviour states, receiver behaviour states, sender behaviour names and receiver behaviour names.
- ch:M indicates that there is a channel intended, as we shall soon see, to synchronise sender and receive processes and communicate entities between them.

- σ_s:SΣ, σ_r:RΣ, xs:Sx, and xr:Rx designate arbitrarily chosen sender and receiver states (σ), and sender and receiver behaviour names.
- update_sσ: M → SΣ → SΣ, update_rσ: M → RΣ → RΣ expresses the signature of generalised sender and receiver state update functions.
- Sx → SΣ **out** ch → **Unit** expresses[34] the signature of the indexed sender process, s: every iteration of the process "starts" in a state sσ:SΣ, proceeds to output synchronise over channel ch with whoever is willing to input synchronise over that channel, and then proceeds to recurse from an "end" state which captures the message m having been communicated over channel ch.
- Behaviour r (of signature Rx → RΣ **in** ch → **Unit** [35]) is a process which can synchronise and communicate with behaviour s.
- The system behaviour [36] is the parallel composition of the sender and receiver behaviours.

C.2

[34]

RSL.33: The output clause ch!m of process (definition) s expresses that process s is willing to engage (synchronise) with an input process on channel ch and, when such occurs, process s offers the value m to the willing input process.

[35]

RSL.34: The input clause **let** m=ch? **in** ... of process (definition) r expresses that process r is willing to engage with an output process on channel ch and, when such occurs, process r accepts a (i.e., the output) value and names it m.

[36]

RSL.35: The process clause p1‖p2 expresses that processes p1 and p2 are to proceed in parallel, i.e., concurrently — possibly, as indicated above, "now-and-then" synchronising and communicating values.

Contents