A Container Line Industry Domain

DRAFT

Dines Bjørner[*]
DTU Informatics[†]
DK-2800 Kgs. Lyngby, Denmark[‡]
bjorner@gmail.com, www.imm.dtu.dk/~db

May 13, 2007. Compiled: June 25, 2007, 14:36

Abstract

We shall sketch a domain description of the (or at least a) container line industry with containers, container vessels, container stowage, container terminal ports with quays, container vehicles, quay cranes, stacks, stack cranes, transfer areas, etc., and with nets of sea lanes, container lines, bills of materials, logistics, etc.

A domain description, according to [1], is a precise informal, say English narrative of the domain, the universe of discourse, as it is, with no reference to what the domain stake holders might wish it to be, let alone requirements to improvement of business processes and supporting IT systems. A proper domain description has its informal narrative be supplemented by a comprehensive terminology (an ontology) and a (mathematical) formalisation (allegedly) of the narrative.

A completed domain description can serve as a basis for business process re-engineering (BPR), or as a basis for developing any number of requirements for computing (and communication) systems to support one or another container line process. A container line domain description can also serve as a basis for the research & development of a container line theory. A container line domain description can finally serve in two additional ways: as a basis for the development of systematic and comprehensive educational material for container line staff and as a basis for the international standardisation of container line industry process interfaces.

The process of developing a reasonably complete domain description can help spot undesirable or inadequate business processes.

The current description is a draft sketch. It is carried out according to the abstraction and modelling principles and techniques for systems as covered in my book: [2, 3, 1]. The presentation is self-contained. That is: the formalisation (in the formal specification language RSL [4] [of the RAISE (Rigorous Approach to Industrial Software Engineering) [5]]) is explained as it is used: along the way, in clearly framed footnotes and larger formalisations are extensively annotated. The aim is to ensure highest possible trust in the alleged correspondence of the narrative and "its" formalisation.

[*]Prof. Emeritus

[†]formerly: Computer Science and Engineering, Informatics and Mathematical Modelling, Technical University of Denmark

[‡]Home address: Fredsvej 11, DK-2840 Holte, Denmark

Contents

0 A Prelude

0.1 Status and Background

The status of the June 25, 2007 version of this domain description is tentative draft sketch: the text has not been reviewed for "misunderstandings" nor for typographical or formula errors; and the model is far from reasonably complete.

More specifically these are some of the more glaring inefficiencies:

- There is no guarantee, yet, that we cover the domain of container line industry evenly and adequately. For this to be ensured we need a larger scale effort.
- The presentation — pls. recall that the present report is the effort of one person, part time in in the period May 13, 2007 to June 25, 2007 — the presentation is not "even". For example:
  - The formal notation should be footnote explained fully systematically.
  - The annotations should likewise be presented more systematically.
  - Many formalisations are missing and some are not as abstract as I would like to see it.
  - Etcetera.
- More work has to be done on the logistics section.
- Etcetera.

Two important methodological aspects that are not covered adequately are:

- the principles and techniques of domain engineering — for those we refer to [1]; and
- the relation between domain descriptions and requirements prescriptions — for those we also refer to [1].

  What we describe is the domain, as it is, not as we, or someone else, say a container line, would like it to be — the latter is a matter of requirements !

This report is being co-written with lecture notes for the July 9–13, 2007 Lipari Summer School: http://lipari.cs.unict.it/LipariSchool/CS/ for which lectures it shall serve as a "running", a reference example.

I have long, since perhaps 18 years or, wished to have a domain model constructed for container shipping. Denmark has — in just one shipping company[1] — the world's largest container shipping fleet[2]. And: the computing systems challenge (that automation of the processes of container shipping offers) is fertile ground, not just for ordinary IT, but certainly still, for years to come, for Informatics: the confluence of computing science, software engineering, mathematics, operations research, etc.

> *In my book, Vol. 3 [1], I give many container industry examples spread over many chapters; I suggest that student projects (as outlined in the book) be associated either with container terminal ports or with container logistics (see Items 4 and 7, Pages 45 and 46, Sect. 1.7) or Exercises 19.1, 19.7, 19.8, 19.9, 19.10, 19.15 and 19.16 (Sect. 19.10.2 Pages 475–478). Sections 19.2–19.3 (Pages 390–403) contains (sic !) three large examples, Example 19.1–.4 (Pages 391–403) devoted entirely to issues concerning container shipping.*

What you find below represents less than four weeks of part time work: I am now in retirement; it is the months of May and June: the garden needs lots of attention after basically 3 years of our absence (in Singapore and in Japan), and the house needs likewise being sorted out: containers (sic!) of goods from the Far East still need unpacking and our many rooms rearranged.

[1] The A.P. Møller – Mærsk
[2] The Maersk Line

0.2 Purposes of This Domain Description

In a recent interview with the Singapore Port Times, Spring 2007, one of the then presidents of Maersk Lines opined that *"alignment and automation of processes across the transport chain"* could lead to a form of standardisation of container line industry processes and hence to improvements of shipper/carrier relations, better service, etc.

The current author rather strongly believes that any attempt at business and operational process improvements, including mergers with other established container lines — each with their several generations old process culture — must be based on a clear understanding of that which is to be improved, i.e., the domain.

As shown, extensively, in [1] business process engineering and re-engineering (BPR) can be done systematically and effectively once clear and comprehensive domain descriptions exist.

We are very skeptic as to whether any container line has such a clear and comprehensive domain description of its own industry.

A clear and comprehensive description of the container line industry domain is a serious affair — as it is to successfully manage and operate such an industry — such as evidently done by Maersk Line. The domain description documentation is a serious affair — so far not mastered by other than perhaps some hundred or so software engineers cum computing scientists world wide.

The current document is a contribution to Maersk and other container lines' desire to further improve their processes. It cannot be done only by asking operations researchers to optimise their processes. Such optimisations must be based on clear and comprehensive domain descriptions. The formalisation component (i.e., the mathematics) is an indispensable part of a professional domain description — as the use of mathematics is unavoidable in all professional engineering tasks.

If such process improvement work is already, in the thinking of the industry, or some of its members, so well advanced that it is too late to construct a proper domain description, then we shall again be witnessing the usual "scandals": missed deadlines, incompatibilities, assumptions which fail to hold, processes that fail to cover the complexities of the domain, etc., etc. Add to this software that might be procured for the support of business processes and the "calamities" will only be propagated — and at great expense.

The container line, we can safely predict, whose BPR builds on clear and comprehensive domain descriptions will win in the ever tougher and competitive industry.

This document — despite its status as an incomplete draft sketch — shows us all how a real, professional domain description may look, and the report implicitly indicates what it will take to achieve such a description.

If the container line industry is to agree on common standards for some of the container line industry processes then a domain description must be made. How else can that industry agree on what they commonly refer to unless it is written down, concisely, in English and supported mathematically ?

1 Overview of The Container Line Industry

1.1 Essential Concept and Terms

We consider the following phenomena and concepts to be basic to the container line industry: the acceptance of containers, sea transport of containers over large distances, possibly by means of more than one voyage, possibly with temporary, transshipment storage of containers 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) (inland) transfer area, (iv.5) quay, stack and transfer area cranes, (iv.6) vehicles, (iv.7) ("inland") trucks, trains and barges, (v) net of sea lanes, (vi) container lines, (vii) bill of ladings and (viii) logistics.

1.2 Some Photos and Diagrams

Figure 1: The Emma Maersk Container Vessel

Emma Maersk and her five sister ships are today, summer 2007 the largest container vessels in the world: 397 meters long and with a capacity of 11,000 TEUs (twenty foot equivalent container units).

Figure 2: A.P. Møller Maersk and Emma Maersk

Figure 3: Maersk Cornelius and Maersk Regina

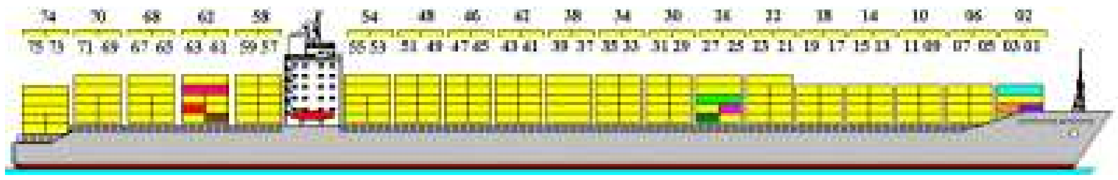

Figure 4: All Bays of a Vessel

Cellularised container vessels stow containers below and above (on) deck. Stowage is organised in bays.

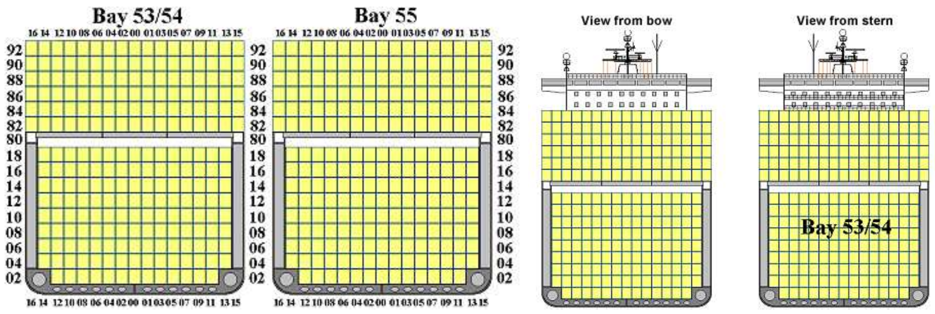

Figure 5: Cross Sections Showing Bays

Bays are numbered, as shown, and are organised into rows and tiers, also as shown.
Here is another way of showing bay, row and tier numbering.

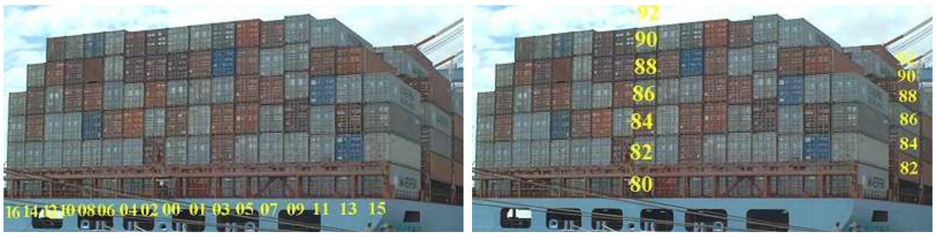

Figure 6: Bay, Rows and Tiers

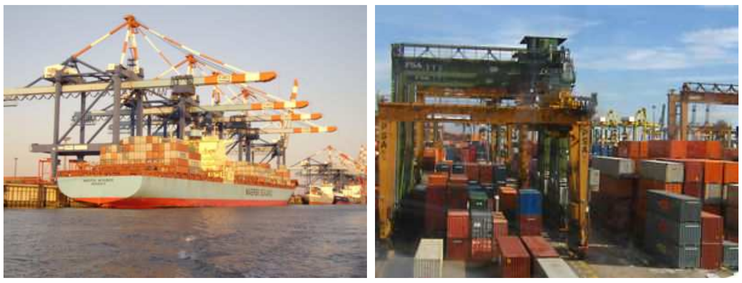

Figure 7: Left: One Quay, Four Cranes and One Vessel. Right: One Stack and Many Cranes

Left: Vessel served by four quay cranes. Right (left part): Part of a stack with three stack cranes in one bay.

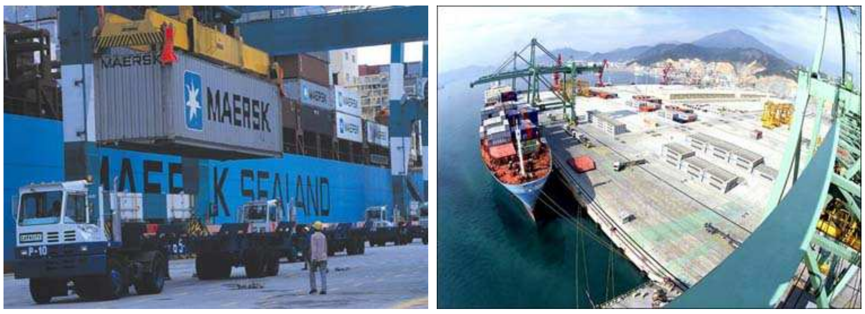

Figure 8: Quay and Stack Vehicle; Yantian (China) CTP

Vehicles move cranes between quay cranes and stack cranes, or between quay cranes and transfer area. The new Yantian container terminal port. One vessel, two quay cranes, one quay, stack and transfer area.

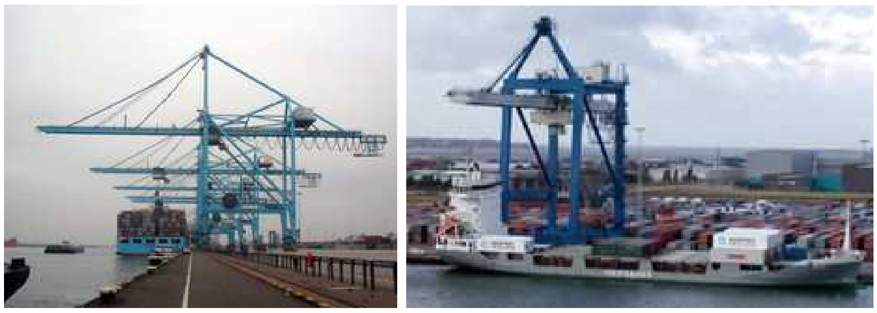

Figure 9: Quay Cranes

Left an oceangoing vessel. Right a feeder (coaster) vessel.

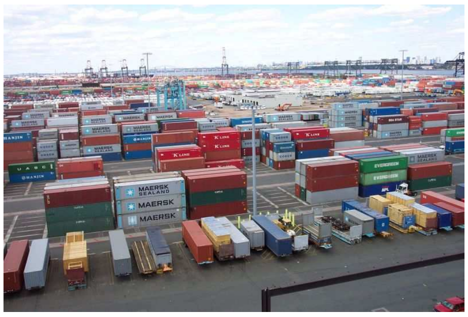

Figure 10: Quays, Stack and Transfer Area

In the background the ocean and quays, then the stack area and, in the foreground, the transfer area.

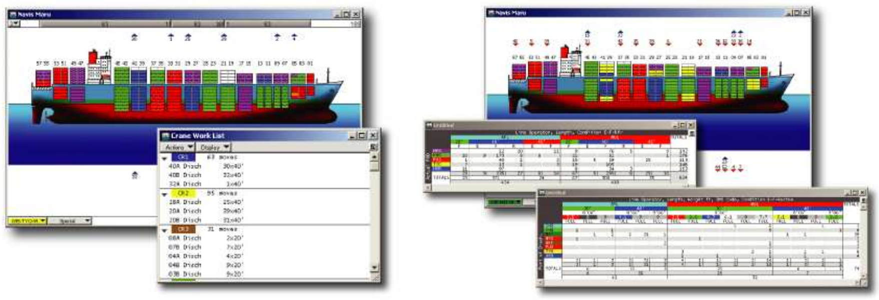

Figure 11: A Commercial Stowage Software System

The commercial “pictures” diagram and tabularise states of stowage planning.

Figure 12: Marchen Maersk in the Panama Canal; Panamax Cranes

Figure 13: 20 and 40 Feet Containers

2 Containers

By a **container** we understand an item of equipment as defined by the International Standardisation Organisation (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[3],
- 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. **Container**s have **weights** (from 0 up!).
3. **Container**s have same **width**s and **height**s, but any one of a few “standard” **length**s. We shall consider only 20′, 40′, and 45′ containers, usually measuring
   (a) 20′ length (TEU)[4]: 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**, an **open top**, a **fantainer**[5], 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.

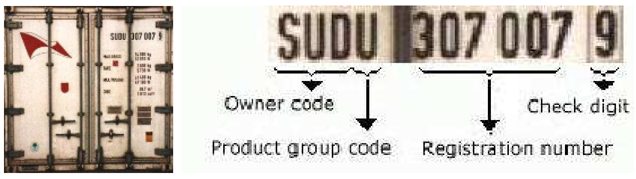

Figure 14: Container Markings, 1

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

[3]Reloading: of container contents
[4]TEU: Twenty Foot Equivalent Unit
[5]Fantainer: general purpose container with special arrangement for possible refrigeration or, at least, ventilation

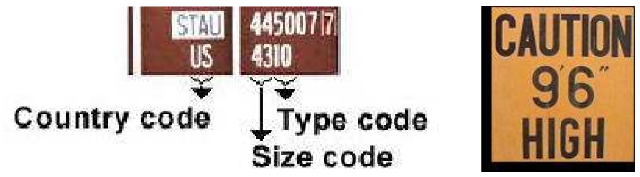

Figure 15: Container Markings, 2

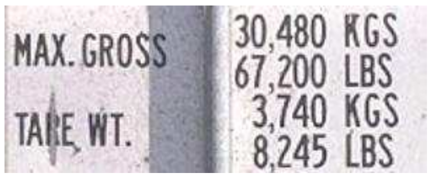

Figure 16: Container Markings, 3

```
    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[6] 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.

6

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).

3 Container Vessels

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**.

3.2 Container Bays, Rows, Tiers and Cells

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

6. We shall restrict ourselves to **cellular vessel**s, i.e., vessels specially designed and equipped for the carriage of containers.
7. A **cell**, then, is a **location** on board a container vessel where one container can be **stowed**.
8. A **cell position** is the location of a cell on board a container vessel identified by a code for successively the bay, the row, and the tier (i.e., the tier index).
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.
10. A **bay** consists of one or more **row**s.
11. A **bay plan** is stowage plan which shows the locations of all the containers on the vessel.
12. 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 **tier**s (containing a fixed number of tier **cell**s).
13. 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.
14. A **cell** is either **empty** or **stow**s a **container**.

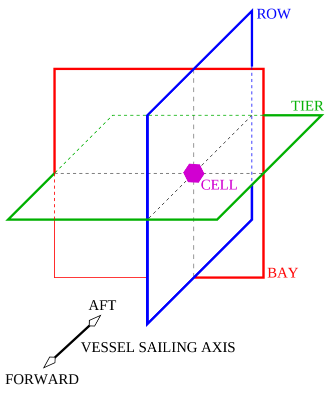

Figure 17: Bay – Row – Tier Plane Intersection Cell

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 bay and row) of a number of **indexed tiers**. That is, an identified bay and, within it, an identified row, and within it, an indexed tier defines a container **cell**. **Container vessels** possess the following overall static properties (attributes): a container vessel **unique name** (CVNm), a **velocity profile** which maps the load profile onto a mean velocity, and possibly many other things. **Container vessels** also possess 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, the current **position** on the high seas or in some harbour, and possibly many other things. Three important quantities to be considered during load planning[7] are the **GM**: distance between the ship's center of gravity[8] and the meta-center position[9]; 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[10], in excess of design drag[11].

We shall build on the below formalisation as the presentation unfolds.[12]

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

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

```
type
   CVNm
```

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

[8]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.

[9]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.

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

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

[12]

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.

[13]

Referring to the axiom on Page 15:

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.

```
    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 and row identifiers and tier indexes as arbitrary (albeit finite, enumerable) sets of further unidentified tokens. This modelling choice allows us to implement these identifiers and indexes 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 identified bays and rows.[14]

```
axiom
    ∀ b,b′:Bt • b≠b′ ⇒ obs_Bi(b)≠obs_Bi(b′)
    ∀ (b,r),(b,r′):B×R • r≠r′ ⇒ obs_Ri(r)≠obs_Ri(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}
axiom
    ∀ bs:B-set,rs:R-set •
        card bs=card xtr_Bis(bs) ∧ card rs=card xtr_Ris(rs)
```

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 celll = xtr_R(ri,b) in
        if ti ∉ inds celll then false else true end
        end end end end end end

select_C: LOC → CV ~→ C | null
select_C(bi,ri,ti)(cv) ≡
    let bis = obs_Bis(cv) in
    if bi ∉ bis then chaos else
    let b = xtr_B(bi)(cv) in
    let ris = obs_Ris(b) in
    if ri ∉ ris then chaos else
    let celll = xtr_R(ri,b) in
    let tis = inds celll in
    if ti ∉ tis then chaos else
    let cell = celll(ti) in
    case cell of mkC(c) → c, _ → null end
    end end end end end end end end end
```

We[15] can now define a predicate which determines whether a cell, designated by a given location, is occupied or not.

14

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**.)

15

```
value
    is_occupied: LOC → CV $\xrightarrow{\sim}$ Bool
    is_occupied(l)(cv) ≡ select_C(l)(cv)≠null
        pre is_LOC_of_CV(l)(cv)
```

3.3 Vessels: Berths, Berth Positions, &c.

A CTP harbour, for short: CTP, consists of an ordered list of berth positions. We assume, without loss of generality, all berth positions to "fill" some volume, i.e., be of same depth, same width and some (not necessarily same) length. A vessel at a specific harbour requires a (minimum) number of such berth positions. 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[16] "read" the above formulas.

- (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.

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.

16

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 $\mathcal{P}$.

- (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.

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 **arrive**s at ports and **depart**s 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 **grant**s 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). And eventually the vessel is berthed.

3.4.1 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.

3.4.2 Vessel and CTP Interactions: Processes.

```
type
   CVNm, CV$\Sigma$, CTP$\Sigma$
value
   vns:CVNm-set
   cv$\sigma$s:(CVNm $\overrightarrow{m}$ CV$\Sigma$)
   ctp$\sigma$:CTP$\Sigma$
axiom
   vns = dom cv$\sigma$s
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 × CV$\Sigma$ → in,out ves_ctp[ cvn ]  Unit
   vessel(cvn)(cv$\sigma$) ≡
           ...
     ⌈⌉ if is_at_sea(cv$\sigma$)
           then
              (vessel(cvn)(next_cv$\sigma$(cv$\sigma$)) ⌈⌉ vessel_arrives(cvn)(cv$\sigma$))
           else
              vessel(cvn)(next_cv$\sigma$(cv$\sigma$)) end
     ⌈⌉ if is_at_CTP(cv$\sigma$)
```

```
            then
               (vessel(cvn)(next_cv$\sigma$(cv$\sigma$)) $\sqcap$ vessel_departs(cvn)(cv$\sigma$))
            else
               vessel(cvn)(next_cv$\sigma$(cv$\sigma$)) end
      $\sqcap$ ...

   next_cv$\sigma$: CV$\Sigma$ $\rightarrow$ CV$\Sigma$
```

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, $\sqcap$, 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 $\times$ CV$\Sigma$ $\rightarrow$ in,out ves_ctp[cvn] Unit
   vessel_arrives(cvn)(cv$\sigma$) $\equiv$
1.     (let time = estimate_arrival_time(cv$\sigma$) in
2.     ves_ctp[cvn]!mkReqBerth(time,obs_berthing_Info(cv$\sigma$));
3.     let m = ves_ctp[cvn]? in
4.     case m of
5.        mkBertAsgn(jn,bpl) $\rightarrow$ vessel(cvn)(upd_berth_cv$\sigma$(jn,bpl)(cv$\sigma$)),
6.        mkPlsWait(jn,est) $\rightarrow$ vessel(cvn)(upd_wait_cv$\sigma$(jn,est)(cv$\sigma$)),
7.        _ $\rightarrow$ chaos
          end end end)

8.  obs_berthing_Info: CV$\Sigma$
9.  estimate_arrival_time: CV$\Sigma$ $\rightarrow$ Time
10  upd_berth_cv$\sigma$: JobNm $\times$ BPosL $\rightarrow$ CV$\Sigma$ $\rightarrow$ CV$\Sigma$
11. upd_wait_cv$\sigma$: JobNm $\times$ Time $\rightarrow$ CV$\Sigma$ $\rightarrow$ CV$\Sigma$
```

Annotations:

- (1.) An estimate[17] 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.

[17] 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[18] 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
```

[18] The **case** a **of** pattern$_1$ → $\mathcal{C}$(p$_1$), pattern$_2$ → $\mathcal{C}$(p$_2$), . . . , pattern$_n$ → $\mathcal{C}$(p$_n$) **end** construct examines the value a. If it "fits" pattern$_1$ then the value of clause $\mathcal{C}$(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.)[19] 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 harbour 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, pertinent 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 advised:
  - (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.

[19] 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".

```
value
    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 ∗/
```

3.5 “Below” and “Above” Deck, Vessel Hold, Hatchways and Covers

Vessels have a **deck**. A vessel deck separates container bays, rows and tiers in two physically separate parts. There are the ‘**below**’ **deck** bays, rows and tiers. And there are the ‘**above**’ **deck** bays, rows and tiers. The below deck bays etc. are referred to as the **hold**. When sailing the bays etc. below deck are isolated from the above deck by **hatch covers**. A **hatch cover** is a watertight means of closing the hatchway of a vessel. A **hatchway** is an opening in the deck of a vessel through which cargo is, i.e., the containers are loaded into, or discharged from the hold and which is closed by means of a hatch cover.

• • •

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!

4 Container Vessel Stowage

This section, Sect. 4, is currently the least developed section of this report — while its topic is such that this section is perhaps that of most interest to operations researchers and, to some extent, also to their customers, the container lines — as it is here monies can be saved and safe sailing guaranteed.

The general container stowage problem is that of planning the location of containers on board a vessel while avoiding impossible, dangerous or costly stowage.

That is, one can, for pragmatic reasons, consider the following three classes of container stowage problems: (i) physically **impossible** stowage, see Sect. 4.2, (ii) physically **dangerous** stowage, see Sect. 4.3, and (iii) operationally **costly** stowage, see Sect. 4.4.

(i) By physically impossible stowage we mean that laws of physics prevents such stowage.

(ii) By physically dangerous stowage we mean that such stowage may lead to explosions, to contamination of containers, to an undesirable vessel list or vessel trim, or to too heavy a container load in general.

(iii) By operationally costly stowage we mean stowage that either results in too few on board containers or in shifting.

4.1 Background

Containers[20] *on board a container ship are placed in container cells, that is, at locations made up from bay and row identifiers and tier index. Since the access to the containers is only from the top of the bay/row column of (tiered) cells, 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.*

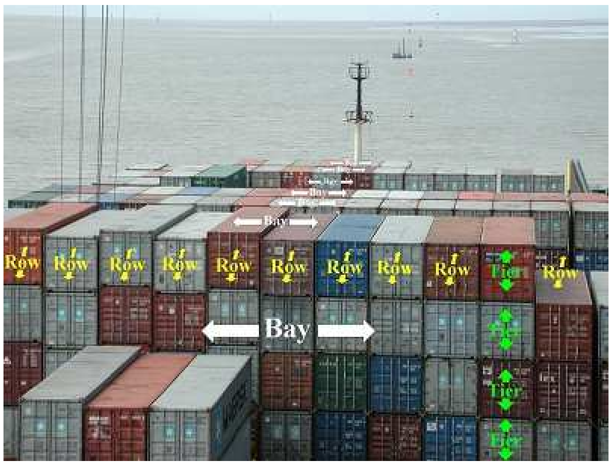

Figure 18: Stowage: Bays and Rows

4.2 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

[20] This *slanted* quote is edited from [6].

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:

15. In any bay/row column of cells (i.e., 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 15.

16. There is a physical limitation of the height of any bay/row column of cells, both below deck and above deck.
    (a) Below deck the maximum number of tiered columns of cells is fixed by the physical height of the hold.
    (b) Above deck the maximum number of tiered columns cells is fixed by physical considerations.

4.3 Stowage Properties: Dangerous Stowage

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:

17. Heavier containers must not be stowed above lighter containers.

18. A container, c, at location c_ℓ must not have a contents which “disagrees” with the contents of “nearby” containers.

19. Heavier containers should be stowed close to the ship center of gravity.

20. 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 17–20) 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)

MORE TO COME

4.4 Costly Stowage: Shifting

4.4.1 The Shifting Problem.

In Sect. 4.1 we outlined the 'shifting problem': 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".

4.4.2 The Vessel Stowage Planning Process, I.

The act of planning the stowage of containers on a vessel consists of many steps, each determining the location of a container to eventually being placed at that vessel cell location.

Each such step of that planning process must therefore satisfy an **invariant**: If the placement of containers before the step entailed no shifting, then, after have planned a location, the new placement of containers mist entail no shifting. We shall now develop a more precise description of this, as wee shall call it, **vessel stowage invariant**.

4.4.3 The Vessel Stowage Invariant.

To define the vessel stowage invariant let us analyse (i) vessel containers and the (ii) vessel (i.e., container) routes. (i) Vessel containers are located in bay/row columns of cells, one on top of another, with many bay/row columns; for, i.e., from, each container one can observe the **next CTP** (i.e., harbour) at which it is to be unloaded, i.e., for which it is destined. (ii) Each vessel sails according to a route visiting one CTP (i.e., harbour) after another; that is, for each vessel, we can speak of the **next CTP visit**, and for every stowed container of the final destination CTP at which it is to be unloaded.

Now we can express the vessel stowage invariant that should be satisfied. For each container aboard a vessel, located at some tier (i) $[i-1]$ in some bay/row column, it must be the case that the port at which it must be unloaded is the **'same as'**, or **'after that'** (CTP) of the container possibly located above it $(i+1)$ in the same bay/row column, and is the **'same as'**, or **'before that'** (CTP) of the container possibly located below it $[i]$ in the same bay/row column, and for any on board container, their CTP destination is on the current vessel route — wrt. which the **'same as'**, **'before that'** and **'after that'** predicates are defined.

```
type
    C, CV, CR, CTPNm, Bay, Bid, Row, Rid
value
    obs_CTP_dest: C → CTPNm, obs_nxt_CTP: CV → CTPnm
    obs_Bids: CV → Bid-set, obs_Rids: CV × Bid → Rid-set
    obs_C−column: CV × Bid × Rid → C*
    before,after: CTPNm × CTPNm → CR → Bool

    vessel_stowage_invariant: CV → Bool
    vessel_stowage_invariant(cv) ≡
        ∀ bid:Bid,rid:Rid•bid ∈ obs_Bids(cv)∧rid ∈ obs_Rids(cv,bid) in
            let cl = obs_C−column(cv,bid,rid) in
            ∀ i:Nat • {i,i+1}⊆inds cl ⇒
                obs_CTP_dest(cl(i)) = obs_CTP_dest(cl(i+1)) ∨
                before(obs_CTP_dest(cl(i+1)),obs_CTP_dest(cl(i)))(cl) ∨
                after(obs_CTP_dest(cl(i)),obs_CTP_dest(cl(i+1)))(cl) end
```

4.4.4 The Vessel Stowage Planning Process, II.

The usually non-deterministic process of planning stowage of containers aboard a vessel takes place in the following information context — the vessel: its current position along its container route, its layout of bays, rows and tiers, their initial stowage of containers, i.e., its current stowage plan which is assumed to satisfy the vessel stowage invariant, the (input) BoLs of containers to be stowed. The completion of stowage planning results in: updated information about the vessel: a new stowage plan, taking care of all or most input BoLs, such that the new stowage plan satisfies the vessel stowage invariant, and information about all the BoLs: whether successfully planned for or not.

The stowage planning process, one of the most critical cost savings (i.e., profit-oriented) container line planning processes, can roughly be characterised as exhibiting the following usually non-deterministic behaviour: (i) it iterates, with a indefinite number of iterations; (ii) each iteration disposes of a finite, small number of BoL-designated containers, say m; (iii) each iteration results in the placement of $m - n$ of these BoL-designated containers; (iv) the beginning of each iteration after having considered a next, usually non-deterministically chosen "batch" of m' BoLs to be considered may move some of the BoL-designated containers handled in an earlier iteration off the stowage plan and back into the input BoLs — in the belief that that may result in better overall stowage. (v) The iteration will then proceed to find locations such that any placement maintains the vessel stowage invariant, and such that appropriate other constraints are satisfied: no impossible stowage and no dangerous stowage. (vi) The full behaviour disposes of most, if not all of the input BoLs; (vii) The iterations end when the stowage planners decide that an optimum of the combination of cost benefit, degree of safe placement, number of containers, and other stowage criteria ("just-in-time", etc.) has been achieved.

This, the "The Vessel Stowage Planning Process" section is very tentative. It need be far more studied, in actual reality and in operations research papers.

● ● ●

We show only this fragment of modelling logistics here. We clearly need a far more in-depth description of container line logistics. Only a rather full-blown, funded and managed research & development (R&D) project can achieve a more satisfying coverage.

5 Container Terminal Ports (CTPs)

A container terminal port (CTP), sometimes referred to as just a container terminal, or just a port, is[21] *a facility where cargo containers are transshipped between different transport vehicles, for onward transportation. The transhipment may be between ships and land vehicles, for example trains or trucks, in which case the terminal is described as a maritime container terminal. Alternatively the transhipment may be between land vehicles, typically between train and truck, in which case the terminal is described as an inland container terminal.*

Maritime container terminals tend to be part of a larger port, and the biggest maritime container terminals can be found situated around major harbours. Inland container terminals tend to be located in or near major cities, with good rail connections to maritime container terminals.

Both maritime and inland container terminals usually also provide storage facilities for both loaded and empty containers. Loaded containers are stored for relatively short periods, whilst waiting for onward transportation, whilst unloaded containers may be stored for longer periods awaiting their next use. Containers are normally stacked for storage, and the resulting stores are known as container stacks.

Figure 19: PTP: Port of Tanjung Pelepas, Malaysia, near Singapore

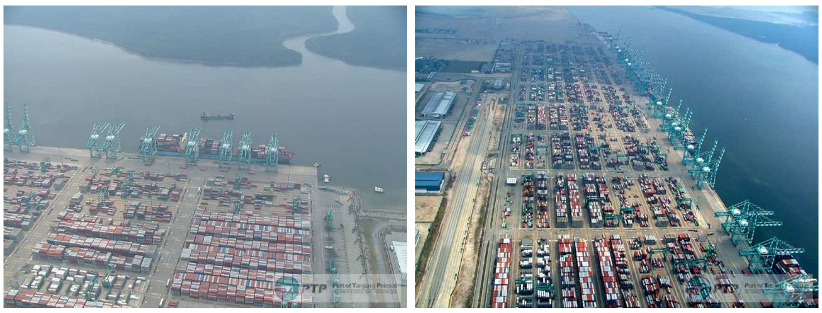

Figure 20: PTP: Quays and Stack Area

• • •

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.

We shall only deal with maritime terminals.

[21]http://en.wikipedia.org/wiki/Container_terminal

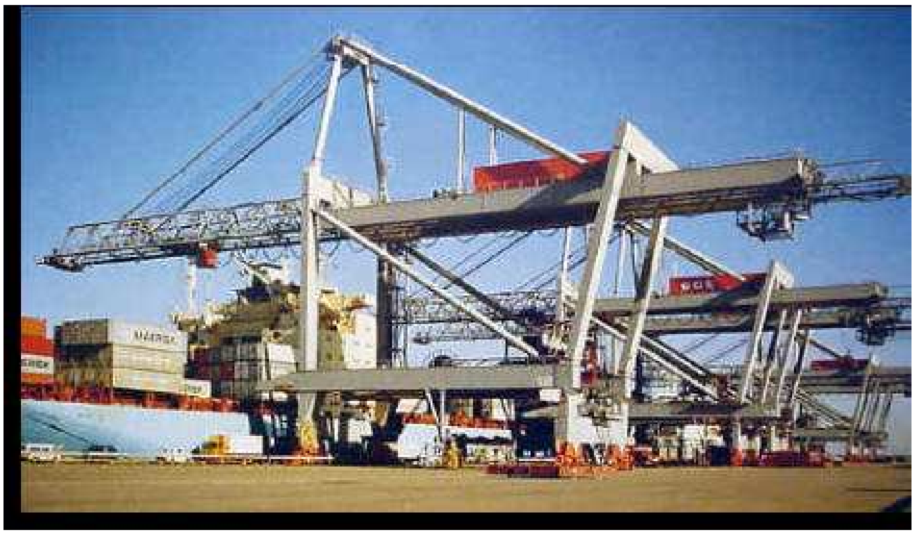

Figure 21: Quay Cranes

5.1 Informal Rough Sketch cum Narrative Presentation

21. A **container terminal** consists of
    (a) a **quay area**[22] where a varying number of ships can be **berth**ed, with quay(s) "sandwiched" between the ocean and the quay area,
    (b) a therefrom physically separated container **stack area**, which consists of a fixed number of one or more **container group**s where containers can be stored (stowed),
    (c) 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.
    (d) 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.
22. Ships
    (a) **arrive** from and **depart** for the **ocean**
    (b) and **dock**, respectively **un-dock**
    (c) at **berth**s which are positioned along the quay.
23. Further **container terminal** entities and some operations involve:
    (a) **Quay crane**s which **move** along the quay(s),
        i. which **position** themselves at a bay position of a berthed vessel,
        ii. which **lift** (**unload**) containers from a berthed ship,
        iii. and which **drop** (**load**) them to **container vehicle**s, respectively
        iv. **lift** (**unload**) containers from these vehicles
        v. and which **drop** (**load**) them onto a berthed ship.
        vi. We shall model the combined lift/drop as a composite **transfer** operation.
    (b) CTP **container vehicle**s (mentioned above)

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

i. which **move** horizontally, two-dimensionally
ii. along the quay(s) and in the quay and stack areas
iii. between ships and stacks.

(c) **Stack crane**s

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 **tier**s (i.e., group, bay, row columns of cells).
iv. We shall model the combined lift/drop as a composite **transfer** operation.

(d) Each stack **group** is organised into one or more **bays**, each bay with one or more **rows**, each row with one or more columns of **tiers**, and each tier with a maximum number of containers (i.e., **cells**).

(e) Other (possibly different kinds of) **container vehicle**s

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 **transfer**red 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 **gate**s, and the inland.

(h) **Gate**s separate the container terminal port from the inland

(i) just a **ship berth**s separate the the container terminal port from the **ocean**.

(j) Finally we may introduce the explicit vehicle operation of **wait**ing

i. at a CTP location
ii. for a specified interval of time,
iii. or until a specified clock time,
iv. or indefinitely.

24. A **container** undergoes the following four **behaviour**s:

(a) The **ship to stack container behaviour**:

i. first a **lift (unload)** — by a **crane** — from a **ship bay/row tier (cell)**
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 **lift**ed (**unload**ed) — by a **crane** — from the vehicle
v. and **drop**ped (**load**ed) — by the **crane** — onto a **group/bay/row tier (cell)**.

So a ship bay/row tier, a crane, a vehicle, another crane, and a stack group/bay/row tier (henceforth, for short: tier) was involved in the stack to ship 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 group/bay/row tier** (i.e., a group/bay/row column of cells)
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 **lift**ed (**unload**ed) — by a **crane** — from the vehicle
v. and **drop**ped (**load**ed) — by the **crane** — onto a **ship bay/row tier**.

So a ship bay/row tier (henceforth, for short: tier), a crane, a vehicle, another crane, and a stack group/bay/row tier (henceforth, for short: 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 group/bay/row 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 **transfer**red — by a **crane** — from the vehicle
v. to an inland **truck, train** or **barge**.

Thus a stack group/bay/row 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 **lift**ed — by a **crane** — from the vehicle
v. followed by a **drop** — by the **crane** — onto a **stack group/bay/row tier**.

Thus a stack group/bay/row 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 **buffer**s. In such cases we need augment the four container behaviours with an additional two (or four) such.[23] 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.

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 quay**s. (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

[23]The additional behaviours are:

1. stack to transfer area,
2. transfer area to stack,
3. quay area to transfer area,
4. and transfer area to quay area.

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 **group**s 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 **gate**s "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 **container**s 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 **container**s may be **lifted from**, **dropped onto**, or **transferred between stocks**.

Let us analyse the notions of crane and vehicle operations.

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

So there are two conceptual notions of **(CTP) route**s and **location**s.

28. 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]

[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 $A_j \setminus L_{ij}$.

29. Thus the quays, quay area, stack and transfer area can be said to be represented (also) by bordering point sets.
30. 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.
31. 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.
32. A **(CTP) transport route** is a (CTP) route
    (a) such that the pair of first and last element of the route designates
    (b) quay area, stack area, or transfer area 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.

33. A **crane operation** is
    (a) either a container **transfer** operation,
    (b) or a crane **move**ment operation.
34. A **vehicle operation** is
    (a) either a **move** operation.
    (b) or a **wait** operation.
35. 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 **vessel container location** embodies a bay and a row identifier as well as a tier index.
    (e) A **stack container location** embodies a group, a bay and a row identifier as well as a tier index.
    (f) A **transfer area container location** is
        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.
36. The crane **transfer** operation has the following operation **signature**:

(a) as input arguments:
   i. **crane** referred[25] to by crane name,
   ii. **container** referred to by 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 (a semantic entity)
   v. and the CTP state (including the designated berthed vessel, and as a real, i.e., the semantic thing);

(b) and as result values:
   i. a possibly changed CTP state (including the designated berthed vessel, and as the real semantic thing)
   ii. and a termination time (still a semantic entity).

(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** [absolute] **time**, t;
      B. they "last", i.e., **take some time**, $\tau\iota$ (a time duration);
      C. and they therefore end, or **terminate at some** [absolute] **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
    add: T × TI → T, add: TI × TI → TI
    sub: T × T → TI, sub: TI × TI → TI
    mpy: TI × Real → TI, div: TI × TI ~→ Real
```

[25] This and the next references as well as the CTP state, see Item 36(a)v, provide access to the crane, container, etc.

26

37. The crane **move** operation has the following operation **signature**:
    (a) as input arguments:
        i. the **crane** referred to by crane name,
        ii. the crane 'from' **position** (designator) along the quay, and
        iii. the crane 'to' **position** (designator) along the quay,
        iv. the start time,
        v. and the current CTP state;
    (b) and as result values:
        i. the end CTP state
        ii. and the termination time.

```
type
    CraP
value
    obs_CraPs: Quay → CraP-set
    move: CNm × CraP × CraP → T → CTP ~→ CTP × T
```

27

38. The vehicle **move** operation has the following operation **signature**:
    (a) the input arguments:
        i. a vehicle referred to by vehicle name,
        ii. a route designation,
        iii. and an initial CTP state (the real, i.e., the semantic thing);
    (b) and as result values:
        i. a result CTP state
        ii. and a termination time.
39. The vehicle **wait** operation has the following operation **signature**:
    (a) the input arguments:
        i. vehicle referred to by vehicle name,

26

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'.

27

RSL.25: The signature f: A × B × C → D → E $\tilde{\rightarrow}$ 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 $\tilde{\rightarrow}$ E × G, respectively f(a,b,c,d,e).)

   ii. location designator at which to wait, and
   iii. optional waiting time (for ex., as a time interval designator).

(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 $\times$ Route $\rightarrow$ CTP $\rightarrow$ T $\stackrel{\sim}{\rightarrow}$ CTP $\times$ T
  wait: VeNm $\times$ Loc $\times$ OptWait $\rightarrow$ T $\rightarrow$ CTP $\stackrel{\sim}{\rightarrow}$ CTP $\times$ Time

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

40. The crane container **xfer** (transfer) operation:

  value
    xfer: CId $\times$ CraNm $\times$ ToFro $\rightarrow$ T $\rightarrow$ CTP $\stackrel{\sim}{\rightarrow}$ CTP $\times$ T
    xfer(ci,cn,(ℓ,ℓ'))(t)(σ) **as** (σ',t')

  can[28] 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 (t).
  (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 **pre**-conditions do not hold 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[29] 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 bay/row column of cells;

[28] 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).

[29] including transfer area trucks, trains and barges

v. if ℓ or ℓ' intends to designate a container position in a stack group or a transfer area buffer,
   A. and there is no such stack group (respectively transfer area buffer) container location;
   B. or, if there is, that location is not a top of a bay/row column of cells location;

(d) If the above implied **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 tier position,
   B. or a stack group/bay/row tier position,
   C. or a transfer area buffer (bay/row tier) location,
vii. and the other location is a vehicle name.

41. The crane **move** operation:

value
move: CNm $\times$ CraPos $\times$ CraPos $\to$ T $\to$ CTP $\overset{\sim}{\to}$ CTP $\times$ 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'.

Figure 22: Crane and Vehicle at Work

(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 implied **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'.

42. The vehicle **move** operation:

value
  move: VeNm × Route → T → CTP $\overset{\sim}{\to}$ 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 implied **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'.

43. The vehicle **wait** operation:

value
  wait: VeNm × Loc × OptWait → T → CTP $\overset{\sim}{\to}$ 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 implied **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.

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.

44. 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) column of cells then that container is now on top of that column. Reversely

iii. if it was lifted from (the top of a ship or stack or buffer) column of cells then that container is no longer on that column, 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.

45. 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.

46. 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.

47. 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 !

5.4 A Resolution on Modelling CTPs and CTP Operations

In this section we make 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.

48. We re-formulate entities of the CTP into behaviours — with (entity) states — 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 there is a separately described behaviour.
  - ii. For every separately stack crane-accessible (for example) stack group and bay there is a separately described behaviour.
  - iii. For every separately transfer area crane-accessible (for example) buffer bay 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.
- (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.

49. Each behaviour "possesses" an own state (thought of as the CTP entities in Items. 36 on page 33, 37 on page 35 and 38 on page 35):
    - (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.
50. Two behaviours might need to **synchronise** and **communicate**.
51. 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".

52. The behaviours will be modelled in terms of **CSP-like process**es.

53. The synchronisation and communication then takes place via and over **CSP-like channel**s.

5.5 Sketches of Behaviour Formalisations

In this section (Sect. 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 !

Appendix C, Pages **??–??**, very briefly explains the RSL/CSP concept of processes. It might be useful to read that appendix before "tackling" the formulas of this section.

5.5.1 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 state reflects for the fixed bay and all relevant row identifiers a list of either cells that are either empty or with containers, and the set of quay positions "from which" a crane can "reach" the bay, row and tier.[30]

type
  Quay_Pos
  $\mathrm{CV\_Bay\_\Sigma} = \mathrm{Bi} \times (\mathrm{Ri} \underset{m}{\rightarrow} \mathrm{Cell}^*) \times \mathrm{Quay\_Pos}$-**set** $\times$ ...
  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_Cra_Pos
  $\mathrm{Quay\_Cra\_\Sigma} = \mathrm{Quay\_Cra\_Pos} \times \mathrm{optC} \times \mathrm{optC} \times$ ...
  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
  Veh_Pos
  $\mathrm{Veh\_\Sigma} = \mathrm{Veh\_Pos} \times \mathrm{optC} \times \mathrm{optC} \times$ ...

[30]
RSL.27: The type definition $\mathsf{A} = \mathsf{B} \underset{m}{\rightarrow} \mathsf{C}$ 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 $\_\Sigma$ is chosen (as a pragmatics) to indicate that A_Σ designates a state component.

5.5.2 CTP, Ship, Quay Crane and Vehicle Process Signatures.

We shall present and discuss the signatures of four behaviours: (i) the CTP behaviour, (ii) ship behaviours indexed by vessel name and a bay identifier — where that index shall indicate that there may be other vessel behaviours "covering" other (but same vessel) bay identifiers[31]; (iii) crane behaviours indexed by crane name; and (iv) 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: CTP$\Sigma \rightarrow$ **Unit**
  ctp(ctpσ) $\equiv$ (... ctp(ctpσ'))

  quay_crane: CraNm $\rightarrow$ Cra$\Sigma \rightarrow$ **Unit**
  quay_crane(cn)(cσ) $\equiv$ (... quay_crane(cn)(cσ'))

  vehicle: VehNm $\rightarrow$ Veh$\Sigma \rightarrow$ **Unit**
  vehicle(vn)(vσ) $\equiv$ (... vehicle(vn)(vσ'))

  stack_crane: StkCraNm $\rightarrow$ Cra$\Sigma \rightarrow$ **Unit**
  stack_crane(scn)(scσ) $\equiv$ (... stack_crane(scn)(scσ'))

  ship_bay: CVNm $\times$ Bid $\rightarrow$ SBR$\Sigma \rightarrow$ **Unit**
  ship_bay(vn,bi)(sσ) $\equiv$ (... ship_bay_row(vn,bi)(sσ'))

We[32] 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.

5.5.3 CTP, Ship, Quay and Stack Crane, and Vehicle Channels.

The idea is to **model synchronisation** and **communication** between CTP and ships, cranes and vehicles. The communication is informing — as possibly requested by — them of details of their next actions. The behaviours include ship and crane processes (when lifting [dropping] containers) and crane and vehicle processes (when dropping [resp. lifting] containers). The behaviours synchronise and communicate by means of **'messages'** sent across **channel**s 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

[31] 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 !

[32]

RSL.29: The **Unit** literal in the function f signature f: A $\rightarrow \Sigma \rightarrow$ **Unit** designates that the function f is a process that never terminates.

(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.[33]
- (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) [34] 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.
- (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.

5.5.4 Quay Crane-related Channel Messages.

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 (d.) and receives (e.) information from the CTP as to whether a container transfer is (e.) 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

33

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

34

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.

to a vehicle (f.) then a crane (h.) requests and (j.) receives permission from a ship bay to "lift" a container from the designated tier; then the crane, (h.) having obtained the container by applying its spreader to the top of the designated bay/row column, (l.) requests and receives permission (m.,j.) to "drop" that container to the designated vehicle; and finally (l.) the crane places the container on the vehicle. Similar for transfers from vehicles to ships (also (f.)). This analysis gives rise to the following channel message types:

```
type
a.  JobNm
b.  BRT = Bid×Rid
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 loosely 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 nay/row identifiers of Bay and Row columns of cells.
- (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).
- (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 interactions 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.

5.5.5 Ship, Quay 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.
- Lines 1–7 defines an auxiliary function, ctp_cra_fct which is invoked in line 8., the main loop.
- (Line 8.) Non-deterministically ([]) the CTP expresses willingness to engage with any crane, eventually receiving a message from some crane i.
- (Line 8.) Then the CTP performs some actions in preparing and possibly delivering a response to the interacting crane.
- (Line 8.) These actions are expressed in terms of the function invocation ctp_fct(i)(m).
- The crane is either requesting a job (Line 3.), or is informing that a job has been completed (Line 6.).
- If the crane is requesting a job then the CTP inquires its state for a next job for that crane (Line 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 (Line 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 (Line 4.).
- The CTP informs the crane of its response (Line 5. first part).
- And the CTP reverts to "itself" — i.e., making itself ready to engage with other behaviours (Line 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" (Line 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 message 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 reverts 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 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 identified) 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, respectively because the vehicle is expecting another container.

```
value
  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) ≡ /∗ exercise for the reader ∗/
```

MORE TO COME

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

MORE TO COME

5.5.6 Vehicle Behaviour.

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 — when there is such a 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 !

5.5.7 Vehicle-related Channel Messages.

```
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 45
    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)
```

5.5.8 CTP and Vehicle Process Definitions.

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
```

MORE TO COME

5.6 Container Stack Stowage

MORE TO COME

5.7 Other Terms

Above we have used some more-or-less standard terms. But there are aliases for these terms. Below we cover some of them.

The Ships' area / Container Berth: This comprises a quay line where the container vessels are berthed. Usually the modern container terminals are provided with gantries, which are heavy cranes required for handling containers. The gantry is generally mounted on rails and moves to and fro along the entire length of the container vessel. The gantry is usually fitted with automatic spreader for faster handling of containers.

Marshaling Yards: The rear position of the ship's area is known as marshaling yard and is used to pre-stack a limited number of import containers after being discharged from vessels and prior to their removal to container stacking yard.

Stacking Yard or Container Yard: This is the area where the import containers are transferred from marshal-ling yard and stored until they are taken to Container Freight Station (CFS), Inland Container Depot (ICD), Consignee's Warehouse, etc. Similarly, this is the area where the export containers are brought from ship operator's warehouse, ICD, CFS, etc. prior to being moved to marshaling yard/container berth for being loaded on board a vessel. Container Yard is also used for stacking empties.

Container Freight Station (CFS): Containers are stuffed (packed) and "destuffed" (un-packed) here. One of the important functions of a CFS is to consolidate smaller shipments of LCL cargo into large units. A CFS can minimize transportation costs by exploiting the economy of movements in FCLs. CFS is normally connected to the nearest ICD by roadways. CFS may be near a port. Many large shipping companies have own CFSs.

6 The High Seas

6.1 Nets and Sea Lanes

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

type
1. N, CTPNm, L, Len, Bezier, L_Misc

value
2. obs_CTPNms: (N|L) $\rightarrow$ CTPNm-**set**
3. obs_Ls: N $\rightarrow$ L-**set**
4. obs_Position: L $\rightarrow$ Bezier*
5. obs_L_Misc: L $\rightarrow$ L_Misc

axiom
6. $\forall$ l:L • **card** obs_CTPNms(l) = 2,
7. $\forall$ n:N, l:L • l $\in$ obs_Ls(n) $\Rightarrow$ obs_CTPNms(l)$\subseteq$obs_CTPNms(n)

Annotations:

- (1.) We postulate classes of (sea lane) net, N, container terminal port (CTP) names, CTPNm, sea lanes, L, between CTPs, and a measure, Len, og length of sea lanes.
- (2.) From a net one can observe the names of all the container terminal ports of that net and from a line one can observe the (two distinct) names of the container terminal ports that the line connects.
- (3.) From a net one can observe all the lines of the net.
- (4.) A list of Bezier (coordinates) is a means for plotting an actual, though approximate curve modelling the geographical position of the sea lane.
- (5.) L_Misc is a means for modelling a number of, in this (June 25, 2007) version of the present document, foreseen, but as yet not determined attributes (i.e., properties) of sea lanes.
- (6.) For any line it is the case that one can observe exactly two CTP names (hence of distinct CTPs).
- (7.) For all net and lines if the line is a line of the net then the CTP names observed from the line are CTP names of the net.

6.2 Sea Routes

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

```
type
57   SeaRt = {|sr:CTPNm*•len sr≥2|}
value
57   is_SeaRt: SeaRt → N → Bool
57   is_SeaRt(sr)(n) ≡
57b     ∀ i:Nat • {i,i+1} ∈ inds sr ⇒
57c       ∃ l:L • l ∈ obs_Ls(n) ∧ {sr(i),sr(i+1)}=obs_CTPNms(l)
```

Given a net and a pair of CTP names one may be able to observe the sea lane connecting the designated CTPs.

```
value
   xtr_L: CTPNm × CTPNm → N ~→ L
   xtr_L(p1,p2)(n) ≡
      if ∃ l:L • l ∈ obs_Ls(n) ∧ obs_CTPNms(l) = {p1,p2}
         then let l:L • l ∈ obs_Ls(n) ∧ obs_CTPNms(l) = {p1,p2} in l end
         else chaos
```

The length of a sea route is the sum of the lengths of between the

```
value
   sum: Len × Len → Len

   length: SeaRt → N ~→ Len
   length(sr)(n) ≡
      case sr of
         ⟨⟩ → 0,
         ⟨p1,p2⟩^sr′ → sum(obs_Len(xtr_L(p1,p2)(n)),length(sr)(n)),
         _ → chaos
      end
```

6.3 Sea Routes of a Net

(1.-2.) A sea lane net gives rise to a set, srs, of sea routes: (3. first part) such that all sea route are sea routes of the net (3. last part) iff (≡ if and only if) they are in the set srs.

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

A (sea) route may be "linear", i.e., not visiting a port twice, cyclic, i.e., first and last port is the same, all other ports are distinct and different from the first (i.e., last) port. "folded", i.e., first n ports. p_1, p_2, ..., p_n are distinct, and last $n-1$ ports. p_{n+1}, p_{n+2}, ..., p_{2n-1} are the reverse of the first $n-1$ ports, i.e., "folded", i.e., $p_{n+1} = p_{n-1}$, $p_{n+2} = p_{n-2}$, ..., $p_{2n-1} = p_1$. Or a route is otherwise characterised.

We shall not define predicates which "tests" for linearity, cyclicity, foldedness, or otherwise. Container vessel routes may typically be folded or cyclic. To impose axioms on sea lane routes that they be so would be wrong: *"in the domain all is possible"*. When a container line's routes are, for example, folded, then we shall define predicate function which expresses that.

6.4 Connected CTPs

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

```
value
1. is_connected_CTPp: CTPNm × CTPNm → N → Bool
2. is_connected_CTPp(hf,ht)(n) ≡
3.    hf≠ht ∧ ∃ sr:SeaRt • sr ∈ sea_routes(n) ∧ hd sr=hf ∧ sr(len sr)=ht
```

Annotations:

- (1.-3.) A pair of CTPs (hf,ht), of a net n, known by their distinct names, are connected
- (3.) if there exists a sea route, sr, of the net, who first CTP name is the first CTP name of the (argument) pair, and whose last CTP name is the second CTP name of the (argument) pair.

Given a net and a pair of CTP names of connected CTPs we can find a shortest sea route between the CTPs.

```
value
   shortest: CTPNm × CTPNm → N → SeaRt
   shortest(p1,p2)(n) ≡
      if is_connected_CTPp(p1,p2)(n)
         then
            let srs = sea_routes(n) in
            let sr:SeaRt • sr ∈ srs ⇒
               ∼∃ sr′:SeaRt • sr′ ∈ srs ⇒ length(sr′)(n)<length(sr)(n) in
            sr end end
         else chaos end
```

6.5 Connected Nets

A net is connected if any pair of CTPs of the net is connected.

```
value
   connected_Net: N → Bool
   connected_Net(n) ≡
      ∀ n,n′:CTPNm • {n,n′}⊆obs_CTPNms(n) ⇒ is_connected_CTPp(n,n′)(n)
```

6.6 Disjoint Nets

Two nets are disjoint if they have no CTPs in common.

```
value
   disjoint_Nets: N×N → Bool
   disjoint_Nets(n,n′) ≡ obs_CTPNms(n) ∩ obs_CTPNms(n′) = {}
```

```
Lemma
   disjoint_Nets(n,n′) ≡ obs_Ls(n) ∩ obs_Ls(n′) = {}
```

- **Lemma:** Two disjoint nets also have no lines in common.

A set of nets is disjoint (i.e., is a disjoint set of nets) if any pair of distinct nets in the set is disjoint.

```
value
   disjoint_Nets: N-set → Bool
   disjoint_Nets(ns) ≡
      ∀ (n,n′):Net×Net • n≠n′ ∧ {n,n′}⊆ns ⇒ disjoint_Nets(n,n′)
```

(2.) Given a net the set of disjoint nets of the net (3.) is the maximum set of nets such that (4.) the union of all their CTPs (known by their distinct names) is the CTPs of the net and (5.) the union of all their lines is the lines of the net,

```
value
1.  disjoint_Nets: N → N-set
2.  disjoint_Nets(n) as ns
3.    post disjoint_Nets(ns) ∧
4.      obs_CTPNms(n) ∪{obs_CTPNms(n′)|n′:N•n′ ∈ ns} ∧
5.      obs_Ls(n) = ∪{obs_Ls(n′)|n′:N•n′ ∈ ns}
```

A net can be decomposed if it consists of two or more disjoint connected nets.

```
value
    can_be_decomposed: N → Bool
    can_be_decomposed(n) ≡ card disjoint_Nets(n) ≥ 2
```

```
Lemma
   connected_Net(n) ≡ ∼can_be_decomposed(n)
```

A connected net cannot be decomposed.

6.7 Composing Nets

Two or more nets can be composed into one net.

```
value
   compose: N-set → N
   compose(ns) as n
      post
         obs_CTPNms(n) = ∪ { obs_CTPNms(n′) | n′:Net • n ∈ ns }
         obs_Ls(n) = ∪ { obs_Ls(n′) | n′:Net • n ∈ ns }
```

If a set of nets are disjoint then their composition will be a disjoint net. The resulting net will be disjoint if there is at least one net in the set which is disjoint with all other nets in the set.

• • •

We show only this fragment of modelling sea nets 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!

7 A Container Line Industry

7.1 Container Line Enterprises

58. A **container line** (CL) is an enterprise
    (a) which operates (owns or [lease] rents) a number of **container vessel**s (CV),
    (b) where these vessels regularly, according to more-or-less fixed **time table**s (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.

By a **carrier** we shall, in the context of containers, understand the same as a **container line**.

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

   is_connected: NW → Bool
   is_connected(nw) ≡ is_connected(obs_Net(nw))
```

59. 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.

7.2 Container Routes and CTP Visits

60. A container line route can be designated by a sequence of two or more container port visits.

61. 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).

62. A container line route further identifies a name of the container line and the name of a container vessel.

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

7.2.1 Time Ordering.

63. There is a before relation between any pair of times,

```
value
    before: Time × Time → Bool
axiom
    ∀ t,t′:Time •
        before(t,t′) ≡ t≠t′ ∧ before(t,t′) ≡ ∼before(t′,t) ∧
        (before(t,t′) ∨ t=t′ ∨ before(t′,t))
```

7.2.2 Well-ordering of Container Routes.

The container routes of any container line all "carry" the same container line identification, identify distinct container route names and if two or more distinctly named container routes identify the same container vessel then the route time intervals of these container routes do not overlap. We now assume that the time axis is a set of time points.

A next_time operator yields the next discrete time

```
value
    next_time: Time → Time
axiom
    ∀ t:Time
        before(t,next_time(t)) ∧
        ∼∃ t′:Time • before(t,t′) ∧ before(t′,next_time(t))
value
    obs_times: Time × Time → Time-set
    obs_times(t,t′) as ts
        post t ∈ ts ∧ t′ ∈ ts ∧
        ∀ t″:Time •  before(t,t″) ∧  before(t″,t′) ⇒ t″ ∈ ts ∧
        ∀ t‴:Time • before(t‴,t) ∨ before(t′,t″) ⇒ t‴ ∉ ts

    rt_time_invl: CR → Time × Time-set
    rt_time_invl(_,_,_,⟨(t,_,_)⟩⁀cvil⁀⟨(_,_,t′)⟩) ≡ obs_times(t,t′)

    overlap: CR × CR → Bool
    overlap(cr,cr′) ≡ rt_time_invl(cr) ∩ rt_time_invl(cr′) ≠ {}
```

7.2.3 Auxiliary Concepts.

A sea route is a sequence of two or more CTP names. A ports_visited assembles the set of CTP names mentioned in a container route.

```
type
    SeaRt = {| nl:CTPNm* • len nl≥2 |}
value
    xtr_SeaRt: CR → SeaRt
    xtr_SeaRt(_,_,_,cvil) ≡
        ⟨n|i:[1..len cvil]•let (a,n′,d)=vil(i) in n=n′ end⟩

    ports_visited: CR → CTPNm-set
    ports_visited(_,_,_,cvil) ≡ elems xtr_SeaRt(cvil)
```

7.3 Container Line Business

64. 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 may be 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
      is_connected(compose({obs_Net(obs_NW(cl))|cl′:CL•cl′ ∈ cls∧cl=cl′}))
      end
```

7.3.1 Distinctness of Container Line Businesses.

We make some assumptions about the names of container lines (CLNm), container vessels (CVNm) and container routes (CRNm). Recall: **type** CR = CLNm × CVNm × CRNm × CViL.

65. From a container route name one can observe the names of the container vessel "plying" the route and the container line managing the route (and vessel).

66. From a container vessel name one can observe the name of the container line managing the vessel.

```
value
   obs_CLVNm: CR → CLNm × CVNm
   obs_CLNm: CVNm → CLNm
```

For a container line it mist be the case that all of its route and vessel names refer to the appropriate line and route names.

```
axiom
   ∀ cl:CL •
      let crs = obs_CRs(cl) in
      ∀ (cln,cvn,crn,_):CR • (cln,cvn,crn,_) ∈ crs ⇒
         cln=obs_CLNm(cl) ∧ cln=obs_CLNm(cvn) ∧ obs_CLVNm(crn)=(cln,cvn) end
```

• • •

We show only this fragment of modelling the container line business and container line 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!

8 Bill of Ladings

8.1 Basics

To explain the **container line operations** of the **accept**ance 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).

67. A **carrier** is here taken to be the same as a **container line**.
68. A **shipper** (or **forwarder**) is ([7]) 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
a. Shipper, ShprNm, BoL
value
b. obs_ShprNm: Shipper → ShprNm
axiom
c. $\forall$ sh,sh′:Shipper • sh≠sh′ ⇒ obs_ShprNm(sh)≠obs_ShprNm(sh′)

Annotations:

- (a.) We model by the sorts Shipper, ShprNm, BoL and BoL_Info the classes of shippers, shipper names, bills of lading (BoL), and BoL information.
- (b.,d.) Every shipper has a unique name.
- (b.) Every container embodies all the information needed to make a bill of lading.

69. A **bill of lading** (BoL) is document which evidences a contract of carriage by sea [7].

    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.

70. At the moment 3 different kinds of bill of ladings are used [7]:

    (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 70e.

8.2 Transshipment

Transshipment is the shipment of goods to an intermediate destination, and then from there to yet another destination.

One possible reason for transshipment is to change the means of transport during the journey (for example from ship transport to road transport), known as intermodal freight transport. Transshipment usually takes place in transportation hubs. Much international transshipment also takes place in designated customs areas, thus avoiding the need for customs checks or duties, otherwise a major hindrance for efficient transport.

Transshipment is normally fully legitimate and an everyday part of the world's trade. However, it can also be a method used to disguise intent, as is the case with illegal logging, smuggling or grey market goods. transshioment—)

8.3 An Extended Classic Marine BoL

71. We shall generalise the notion of a classic marine BoL in order to accomodate for transshipment, cf. Sect. 8.2. Instead of designating
    (a) one voyage with one container vessel
    (b) we allow for more than one such voyage
    (c) involving possibly several container line companies.
72. A BoL has two parts:
    (a) A BoL has a header, common to all voyages. The BoL header includes
        i. the customer and shipper names, and
        ii. information about the container.
    (b) And a BoL has a sequence of one or more voyages.
        A container voyage identifies
        i. the container line,
        ii. the container vessel,
        iii. first port of the voyage and the departure time from that port, and
        iv. the arrival time to the last port of the voyage and its name.
    (c) Some constraints on extended BoLs:
        i. For pairwise adjacent voyages, of an extended BoL,
        ii. the last port name of the first of the pair
            is the same as the first port name of last of the pair.
        iii. Otherwise all port names are distinct (no cyclic voyages).

```
type
           C_Info, CustNm
72         BoL′ = Hdr × Voyage*
72a        Hdr = CustNm × ShprNm × C_Info
72b        Voyage = CLNm × CVNm × CRNm × (CTPNm × Time) × (Time × CTPNm)
72c        BoL = {| bol:BoL′ • wf_BoL(bol) |}
value
           wf_BoL: BoL′ → Bool
           wf_BoL(_,vol) ≡
72b            len vol≥1 ∧
72(c)i         ∀ i:Nat • {i,i+1}⊆inds vol ⇒
72(c)ii                let (_,_,_,(n,t),(t′,n′))= crl(i), (_,_,_,(n″,t″),(t‴,n‴)) = crl(i+1) in
72(c)ii                n≠n′ ∧ n′=n″ ∧ n″≠n‴ ∧ before(t,t′) ∧ before(t′,t″) end
72(c)iii        ∀ i:Nat • {i,j}⊆inds crl ∧ i≠j ⇒
72(c)iii                let (_,_,_,n′)= crl(i), (_,_,n″,n‴_) = crl(i+1) in n′≠n″ end
```

8.4 BoL Constraints, I

A BoL has to harmonise with the container route offerings of the designated container line. The container line business concept embodies container lines, their vessels and timetables. The shipping concept embodies BoLs with their voyages.

```
type
   CLB, CL, CLNm, NW, CV, CVNm, CRNm, CTPNm, Time
   CR = CRNm × CVNm × CLNm × CViL
   CViL = {| cvil:CTP_Visit* • is_wf_CViL(cvil′)|}
   CTP_Visit = Time × CTPNm × Time
   Voyage = CLNm × CVNm × CRNm × (CTPNm × Time) × (Time × CTPNm)


value
   obs_CLs: CLB → CL-set, obs_CLNm: CL → CLNm,
   obs_NW: CL → NW, obs_CRs: NW → CR-set

   xtr_CL: CLB × CLNm ~→ CL
   xtr_CL(clb,clnm) ≡
      let cls = obs_CLs(clb) in
      let cl:CL • cl ∈ cls ∧ obs_CLNm(cl)=clnm in cl end end
      pre ∃ cl:Cl • cl ∈ obs_CLs(clb) ∧ obs_CLNm(cl)=clnm

ports: CTP_Visit* → CTPNm-set
   ports(cvil) ≡
      let pvs=elems cvil in
      {p|p:CTPNm • (_,p′,_)∈ pvs ∧ p′=p} end
```

73. The container port names (CTPNm) of a voyage (Voyage) must be
    (a) CTP names of the CTP visits (CTP_Visit)
    (b) of the named container route (CRNm)
    (c) of the named container vessel (CVNm)
    (d) of the named container line (CLNm)
74. The times (and port names [as already epressed (Item 73a)])
    (a) mentioned in the voyage must be those
    (b) mentioned appropriately in the CTP visits of the designated route etc.

```
value
   wf_Voyage: Voyage → CLB → Bool
74a  wf_Voyage(cln,cvn,crn,(pn1,t1),(pn2,t2))(clb) ≡
73d    ∃ cl:CL • cl ∈ obs_CLs(clb) ∧ obs_CLNm(cl)=cln ⇒
73b       ∃ cr:CR • cr ∈ obs_CRs(obs_NW(cl)) ⇒
73c          let (crn′,cvn′,cln′,cvil) = cr in crn′=crn ∧ cvn′=cvn ∧ cln′=cln ∧
             ∃ i,j:Nat • {i,j}⊆inds cvil ∧ i<j ⇒
73a,74          let (_,p,t)=cvil(i), (t′,p′,_)=cvil(j) in p=p1 ∧ t=t1 ∧ t′=t2 ∧ p′=p2 end end
```

8.5 BoL Construction

Let us recall that an extended, classical BoL consists of a header with customer name, shipper (forward) name, and information about the container (to be shipped), and a list of one or more voyages, where each voyage concontains of the names of the container line, its carrying container vessel, the container route of the designated vessel, and a pair of pairs of names of container terminal ports and times: the departure time from the first port of the pair and the arrival time to the second port of the pair.

type
  CustNm, ShprNm, C_Info, CLNm, CVNm, CTPNm, Time
  $\text{BoL}' = \text{Hdr} \times \text{Voyage}^*$
  $\text{Hdr} = \text{CustNm} \times \text{ShprNm} \times \text{C\_Info}$
  $\text{Voyage} = \text{CLNm} \times \text{CVNm} \times \text{CRNm} \times (\text{CTPNm} \times \text{Time}) \times (\text{Time} \times \text{CTPNm})$
  $\text{BoL} = \{|\ \text{bol:BoL}' \bullet \text{wf\_BoL(bol)}\ |\}$

An example composite BoL may thus be abstracted as:

```
((cu,sh,ci),
  ⟨(l1,v1,r1,(p1,ta),(tb,p2)),      /* first voyage  */
   (l2,v2,r2,(p2,tc),(td,p3)),      /* second voyage */
   (l3,v3,r3,(p3,te),(tf,p4))⟩)     /* third voyage  */
```

We explain the identifiers of the BoL value abstracted above: *cu*stomer (cu), *sh*ipper, container *i*nformation (ci), *1*st container *l*ine (l1), *1*st *v*essel (v1), *1*st *r*oute (r1), departure *p*ort (p1), departure *t*ime (ta), 1st transshipment arrival *t*ime (tb, tb>ta), 1st transshipment *p*ort (p2), *2*nd container *l*ine (l2, l2=l1), *2*nd *v*essel (v2), *2*nd *r*oute (r2), 1st transshipment *p*ort (again, p2), 1st transshipment departure *t*ime (tc, tc>tb), 2nd transshipment arrival *t*ime (td, td>tc), 2nd transshipment *p*ort (p3), *3*rd container *l*ine (l3, l3≠l2), *3*rd *v*essel (v3), *3*rd *r*oute (r3), 2nd transshipment departure *t*ime (te, te>td), 2nd transshipment *p*ort (again, p3), final destination arrival *t*ime (tf, tf>te), and *4*th destination *p*ort (p4).

Figure 23 diagrams the BoL voyages.

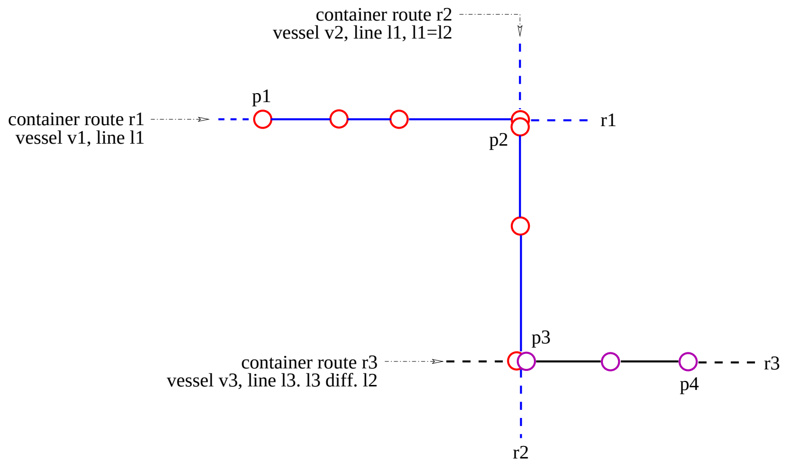

Figure 23: Container transport from port p_1 to p_4 with transshipments at ports p_2 and p_3.
Transport is effectd by three vessel container routes, r_1, r_2 and r_3.
First two by one container line, $\ell_1 = \ell_2$, last by another, ℓ_3.

• • •

A Container Customer - Shipper Scenario: A customer, someone who wishes to have a number of containers transported from port p_{origin} to port $p_{\text{destination}}$ contacts a shipper, someone who offers to arrange for such transports, using either a specific container line, or any number of such lines.

The customer presents the following information: (a) customer (i.e., sender) and receiver names, addresses, etc., (b) necessary and sufficient information about the containers, (c) ports of origin and destination, (d) earliest shipping time, latest delivery time, and whether (e) fast, or low cost transport, whether (f) transport via fewest possible transshipments (0, etc.), etcetera (g, ...),

The shipper is assumed to have the following capabilities: (α) access to the the container routes of one or more container lines, and (ω) ability to arrange, with these container lines, for the confirmed transport of any number of containers, provided they conform to container line criteria (f.ex.: non-explosives, no reefer, or other),

The customer and shipper now arranges for the transport.

The further arrangment may evolve as follows (let's call it the "inquiry phase"): The shipper checks (1) that "their" container lines indeed cover possible transshipment sea routes between customer designated ports of origin and destination; (2) that the relevant vessels accept containers as described by the customer; and (3) that costs and transport times are commensurate with customer stipulations.

The shipper present the result of these checks to the customer.

The customer (x) either accepts these results as the basic parameters for the desired transport; (y) or modifies some or all of the information given to the shipper; (z) or ends the contact to this shipper.

In case (y) above the inquiry phase is repeated.

In case (x) the shipper proceeds as follows: (x_1) A draft BoL is constructed, (x_2) it is OK'ed by designated container lines, and (x_3) it is OK'ed by the customer (i.e., the sender). The BoL, as a contract, (x_4) is signed, (x_5) financial deposits are made, (x_6) shipper arranges for the pick-up of desiganted containers and their delivery to port of origin, (x_7) the shipper regularly informs the customer of the whereabouts of the containers (perhaps in response to the customer trace requests), (x_8) the shipper may request further financial deposits as the transport progresses, and, finally, (x_9) the shipper arranges for delivery to, at port of destination, to the designated receiver (x_9) with final payments due from sender and/or receiver. **End of Scenario.**

We refer to Sect. 9.7 on page 68 in which we sketch a formalisation of the construction of BoLs.

MORE TO COME

8.6 BoL Constraints, II

MORE TO COME

9 Logistics

9.1 Thre Delineations of the Term: ‘Logistics’

9.1.1 A General, Merriam-Webster Delineation.

Logistics is defined in Merriam-Webster as follows: **Main Entry:** logistics. **Pronunciation:** lO-’jis-tiks, l&-. **Function:** noun plural but singular or plural in construction. **Etymology:** French logistique, art of calculating, logistics, from Greek logistikE, art of calculating, from feminine of logistikos of calculation, from logizein to calculate, from logos, reason. **Explanation:** 1 : the aspect of military science dealing with the procurement, maintenance, and transportation of military matriel, facilities, and personnel; 2 : the handling of the details of an operation.

9.1.2 A Specific, Wikipedia Delineation and Discussion.

Logistics[35] is the art and science of managing and controlling the flow of goods, energy, information and other resources like products, services, and people, from the source of production to the marketplace. Logistical support involves the integration of information, transportation, inventory, warehousing, material handling, and packaging. The operating responsibility of logistics is the geographical repositioning of raw materials, work in process, and finished inventories where required at the lowest cost possible.

9.1.3 A Container Line Industry Delineation of the Term: ‘Logistics’.

In the context of the container line industry we shall take ‘logistics’ to include such issues as: (i) a container liners determination of which sea routes to serve; (ii) the allocation and scheduling of container vessels to container routes; (iii) the stowage of containers aboard container vessels; (iv) the stowage of containers in stack at ports; (v) the crane-vehicle-crane split (i.e., allocation and scheduling) at ports, of cranes and vehicles to the movement of containers (either way) between vessels and stacks; and (vi) the determination of which routes and vessels to use for any specific shipping, that is, for any specific extended BoL.

• • •

In the following seven subsections we analyse these logistics functions.

9.2 Which Container Routes to Serve

To analyse which sea routes a container line, ℓ, might wish to serve the following information is required: (i) a net of all sea lanes (and hence a set of all the container terminal ports, CTPs) of the region in which ℓ operates and wishes to operate (ℓ already operates (zero, one or more) sea routes in that region); (ii) a statistics of container “traffic” between any pair of CTPs of the region — whether already transported (say last year), or expected in future, and whether transported by ℓ or by competition; and (iii) the current container routes and vessels of ℓ.

The above container route analysis takes place in a context where a number of such routes are already being served, and such analysis will be repeated, probably annually.

```
type
    N, CTPNm, L, CL, STA, CR_Est
    Num_of_Conts = own_CL × other_CLs_Est × addtl_Est
    other_CLs_Est,own_CL = Nat, addtl_Est = Int
    CR_Est = (CTPNm × CTPNm) → Nat
value
    obs_CTPNms: (N|L) → CTPNm-set, obs_Ls: N → L-set
```

[35]From: http://en.wikipedia.org/wiki/Logistics

obs_CRs: CL → CR-**set**
obs_Num_of_Conts: (CTPNm×CTPNm) → N $\tilde{\to}$ STA $\tilde{\to}$ Num_of_Conts

analyse_CR_Est: N × CL × STA → CR_Est

We model the estimated "number" of containers as a triplet: what the analysing container line, ℓ, currently transports, what other container lines currently transport, and an additional, positive or negative number of containers. The container line numbers are assumed positive.

Caveat: See footnote 36. [36]

9.3 Allocation and Scheduling of Vessels to Container Routes

To analyse the allocation and scheduling of vessels to container routes of a container line, ℓ, the following information is required: (i) an edited version of the sea routes that ℓ wishes to operate, whether already operated or additional ones, or even omitting currently operated routes, (ii) information about the current fleet of container vessels, and (iii) the current container routes.

The above allocation and scheduling of vessels to container routes takes place in a context where a number of such routes are already being served, and such analysis will be repeated, probably monthly or quarterly.

type
Fleet = CV-**set**
value
obs_Fleet: CL → Fleet, obs_CRs: CL → CR-**set**

vessel_alloc_and_sched: CR_Est → CL → CL

MORE TO COME

Repeat Caveat: See footnote 36 replacing analyse_CR_Est with vessel_alloc_and_sched.

9.4 Container Stowage Aboard Vessels

To analyse the stowage of containers onboard vessels of a container route the following information is required: (i) the set of current BoLs tentatively assigned to a container route, (ii) the current stowage of the vessel of the container route, and (iii) preferably current stowage of container in stacks of CTPs of the container route.

The above stowage calculation of any particular vessel takes place in a context where a such vessels are already stowed, and such calculation will be repeated, possibly several times per voyage.

type

value
vessel_stowage:

[36] The "most desirable function" analyse_CR_Est is most likely not computable or its is very hard to calculate. That is, it is not decidable, or it is so-called NP complete. Typically the calculation of a reasonable result requires heuristics and interaction with the analysing staff of the container line ℓ. To further specify analyse_CR_Est would "move" us from describing the domain to possibly prescribing requirements to either business process re-engineering and/or supporting software. Of course, a number of properties of any rendition of analyse_CR_Est could be formalised. We leave that for a future version of the present document.

MORE TO COME

Repeat Caveat: See footnote 36 on the previous page replacing analyse_CR_Est with vessel_stowage.

9.5 Container Stowage in Stacks

To analyse the stowage of containers in the stack of a given CTP the following information is required: (i) the current stowage of containers in the stack, (ii) the import containers of imminent and near-future arrivals of container vessels, (iii) the export containers for imminent and near-future departures of container vessels, (iv) the stack containers to be transferred inland by imminent and near-future trucks, trains and barges, and (v) the containers to be transferred to the stack from the inland (due to the arrival of imminent and near-future trucks, trains and barges).

The above stowage calculation for any particular stack takes place in a context where such stacks are already stowed, and such calculation will be repeated, possibly on a daily base!

type

value
    stack_stowage:

MORE TO COME

Repeat Caveat: See footnote 36 on the preceding page replacing analyse_CR_Est with stack_stowage.

9.6 Crane & Vehicle Split

By **'crane & vehicle split'**[37] we mean the combined (hence the use of '&' instead of 'and') allocation and scheduling of one or more quay cranes, one or more CTP vehicles, and one or more stack cranes for the purposes of moving (either way) containers between vessels and stacks.

By an **import container** we mean a container that arrives by vessel and is then to be stowed in the stack (or sent directly to the transfer area for inland transport). By an **export container** we mean a container that from being stowed in the stack (or being transpprt from inland) is to depart by vessel. By a **transit container** we mean a container that arrives and departs by vessel and which, for a time, is stowed in the stack.

To analyse and calculate the allocation and scheduling of quay cranes, CTP vehicles and stack cranes for the movement of containers (either direction) between vessels and stacks the following information is required: the set of all the pairs (α,ω) of (α) vessel bay/row tier locations of import containers (ω) stack group/row tier locations of containers — or the other way around: (ω,α), where ω then refer to export containers — such that each pair has a time interval stamp as to the desirable unloading (respectively loading) time.

From the above set of time interval stamped pairs of (origin,destination) locations, whether from vessel to stack or from stack to vessel, one can create a set of sequences of such pairs, each sequence pertaining either to a specific vessel bay, or to a specific stack group/bay, or to a specific pair of vessel and group/bay.

The set of all pairs can be calculated from the BoL information (i) of all import and transit containers of all incoming vessels, and (ii) of all transit and export containers of the stack. (iii) We could also include containers currently arriving at the transfer area.

[37] The term 'crane & vehicle split' is our "invention".

We refer to Sects. 5.5.5, 5.5.7, and 5.5.8 for details of the quay crane, CTP vehicle and stack crane behaviours and the entities by means of which these behaviours are coordinated.

```
type
    QuayCraNm, StakCraNm,
    TimeIntv = {|(ti,ti′):Time×Time•before(ti,ti′)|}
    SoTSPs = (TI × FromTo)-set
    Fromto == mkSV(stk_loc:(Gid×BRT×Ti),ves_loc:(BRT×Ti)) |
                mkVS(ves_loc:(BRT×TI),stk_loc:(Gid×BRT×Ti))
    CVC_Job = mkSV(ti:TimeIntv,sft:Fromto,qc:QuayCraNm,ve:CehNm,sc:StakCraNm)
value
    xtr_SoTSPs: TimeIntv × CR-set × CV-set × Stack → SoTSPs
    crane_vehicle_crane_split: TimeIntv × SoTSPs → CTP → CTP × CVC_Job-set
```

Annotations:

- TimeIntv designates the class of time intervals.
- SoTSPs designates the class of pairs of time intervals and pairs of
- Fromto container locations,
  - whether, as in mkSV(sl,vl), from stack to vessel,
  - or, as in mkVS(vl,sl), from vessel to stack.
- CVC_Job designates the class of crane/vehicle/crane job descriptions:
  - in which time interval ti:(Time×Time) the job should (preferably) be done,
  - from and to, sft:Fromto, which container cells, and by means of which
    * quay crane, qc:QuayCraNm,
    * CTP vehicle, ve:CehNm, and
    * stack crane, sc:StakCraNm

    in that order or in the reverse order, the as indicated by the sft:Fromto information.
- The function xtr_SoTSPs applies to
  - a suitably selected time span,
  - a suitably selected set of container routes,
  - a suitably selected set of container vessels, and
  - a stack

  and yields a set, s:SoTSPs, of pairs of time intervals and pairs of Fromto containers locations.
- The function crane_vehicle_crane_split applies to
  - a suitably chosen time interval,
  - a set of pairs of time intervals and pairs of (from,to) containers locations,
  - an entire CTP,

  and yields
  - a possibly changed CTP,
  - and a set of crane-vehicle-crane job descriptions.

MORE TO COME

Repeat Caveat: See footnote 36 on page 65 replacing analyse_CR_Est with crane_vehicle_crane_split.

9.7 BoL Routing

We refer to Sect. 8.5 on page 62 in which we dealt with the construction of BoLs in some detail.

To calculate the routing on one or more container vessels, by a forwarder, i.e., a shipper, the following information is required: (i) the container data: contents, weight, container size, customer (sender), shipper, receiver, from harbour, to harbour, customer or forwarder estimated departure time, customer or forwarder estimated arrivval time, cost estimate, etc., (ii) the complete route net of all relevant container lines with (iii) some indication of availability of vessels for shipping as well as (iv) cost of shipping.

The above calculation of the route for any particular BoL takes place in a context there are many requests for container transports, i.e., requests for calculation of BoLs, that is, such BoL route calculations will be repeated for every BoL. A possibility is that several BoL calculations are performed together, i.e., in groups.

type

value
  bol_route:

More to come

9.8 Review

More to come

10 The Container Line Industry System Model

We shall summarise these lecture notes by presenting a CSP-like model of the entire system of (i) container lines with their (i.0) container line management (**CLNm**), (i.1) container vessels (**CVNm**), and (i.2) container line routes (**CRNm**) served by the vessels; (ii) container terminal ports, CTPs, with their (ii.0) CTP harbour master (**CTPNm**), (ii.1) quay cranes (**QCraNm**), (ii.2) CTP vehicles (**VehNm**), (ii.3) stack cranes (**SCraNm**), (ii.4) stack (**Gid,Bid,Rid**), (ii.5) transfer area (**Bid,Rid**), and (ii.5) transfer area trucks, trains and barges. (iii) shippers (i.e., forwarders) (**ShprNm**) with their (iii.0) management (**ShprNm**), (iii.1) customers (**CustNm**), and (iii.2) “catalogues” of container line container line routes.

10.1 Behaviours

10.1.1 Entity Names, a Technicality.

We have mentioned, in parentheses in the above introduction, the names of classes of entitiy names of the container line industry. We have stipulated earlier that from the names of a container route one can observe the names of the container vessel plying the route and the container line managing (f.ex., owning) the route. This was a mere technicality, but helpful in shortening some of the formalities coming up. We now stipulate that from names of quay cranes, CTP vehicles, stack cranes, stack groups, stack bays, stack rows, transfer cranes and transfer area bays and rowsvone can also observe the names of the container terminal port.

type
  CTPNm, QCraNm, VehNm, SCraNm, XCraNm, Gid, Bid, Rid
value
  obs_CTPNm: (QCraNm|VehNm|SCraNm|XCraNm|Gid|Bid|Rid) $\rightarrow$ CTPNm

As for container line related names we have:

type
  CL, CLNm, CR, CRNm, CV, CVNm, BoL
value
  obs_CLNm: (CL|CR|CRNm|CV|CVNm|BoL) $\rightarrow$ CLNm
  obs_CVNm: (CR|CRNm|CV|BoL) $\rightarrow$ CVNm
  obs_CVNm: (CR|CV|CVNm) $\rightarrow$ CRNm
  obs_CRNml: (CV|CVNm|BoL) $\rightarrow$ CRNm*

Annotations:

- obs_CVNml: From a container vessel, hence its name, and from a bill of lading, one can observe the list of container routes served, from the time of observation, by the vessel, respectively the bill of lading.

10.1.2 Entity Behaviours.

When we say ‘entity behaviours’ we mean that behaviours evolve around a state, i.e., a dynamic entity. Thus we use the terms ‘container line’, ‘crane’, ‘CTP’, ‘vehicle’, and ‘vessel’ in two senses: as entity (i.e., a state), and as a behaviour (with the entity as its evolving, i.e., dynamically changing state).

The container line industry consists of the following behaviours: container line behaviours, one for each enterprise, container vessel behaviours, one for each of several vessels of each enterprise, container bay behaviours, one for each of several bays of each vessel of each enterprise, CTP behaviours, one for each CTP, quay crane behaviours, one for each of several quay cranes of each CTP, CTP vehicle behaviours, one for each of several vehicles of each CTP, stack crane behaviours, one for each of several stack cranes of each CTP, stack group/bay behaviours, one for each of several stack group/bays of each CTP, etcetera.[38]

```
type
    CLΣ, CVΣ, CV_BayΣ, CTPΣ, QCraΣ, VehΣ, SCraΣ, S_BayΣ
value
    cl: CLNm → CLΣ → in ... out ... Unit
    cv: CLNm×CVNm → CVΣ → in ... out ... Unit
    cv_bay: CLNm×CVNm×Bid → CV_BayΣ → in ... out ... Unit
    ctp: CTPNm → CTPΣ → in ... out ... Unit
    qc: CTPNm×QuayCraNm → QCraΣ → in ... out ... Unit
    veh: CTPNm×VehNm → VehΣ → in ... out ... Unit
    sc: CTPNm×StakCraNm → SCraΣ → in ... out ... Unit
    stk_bay: CTPNm×Bid → S_BayΣ → in ... out ... Unit
    ...
```

Annotations:

- The container line, cl, *behaviour maintains a state CLΣ, is willing to synchronise and communicate with other behaviours* ***in ..., out ...*** *(to be detailed below), and otherwise, ideally speaking, does not terminate* ***Unit***. The container line state, CLΣ, embodies knowledge, i.e., information about all container routes and vessels, including their stowage conditions and state. Two or more container lines are distinguished by their container line names, CLNm.
- The container vessel, cv, *behaviour maintains . . . , etc.* The container vessel state, CVΣ, embodies knowledge, i.e., information about its current (and next) container route(s), where it currently is wrt. the current route: on the high sea (and where) or in a CTP harbour, and then at which berth, i.e., quay and quay crane positions. Two or more container vessels are distinguished by their container vessel names, CVNm, which also identifies the container line owner (operator).
- The container bay, cv_bay, *behaviour maintains . . . , etc.* ***Unit***. The container bay state, CV_BayΣ, embodies the current bay, and for all bay rows and their tiers, the container cells, whether empty or occupied, and, for each located container, its BoL, etc. Two or more container bays are distinguished by their bay identifiers, Bid, which also identify the container line owner (operator, by name) and container vessel (by name).
- The CTP, ctp, *behaviour maintains . . . , etc.* ***Unit***. The CTP state, CTPΣ, embodies knowledge, i.e., information about all its cranes, all its vehicles and the quay, stack and transfer area layouts, positions and container cell locations, stack stowage (past, present and within some foreseeable future). Additionally, based on this, the CTP state reflects all vessel visits, whether past, current or within some foreseeable future: possible or actual berth locastions, possibly which quay cranes served the vessels, etc. Two or more CTPs are distinguished by their CTP names, CTPNm.
- The quay crane, qc, *behaviour maintains . . . , etc.* ***Unit***. The quay crane state, QCraΣ, embodies only its current task, i.e., either (i) the container movement (either way) between a bay of a vessel and a CTP vehicle, (ii) or that it is moving from one quay position to another, (iii) or that it is idle. Two or more quay cranes are distinguished by their names, QuayCraNm, which also identify the CTP (by name, CTPNm).

[38] We omit consideration of transfer area bays and cranes.

- The CTP vehicle, veh, *behaviour maintains ..., etc.* ***Unit***. The vehicle state, VehΣ, embodies only its current task, i.e., either (i) the container movement (either way) between a quay crane and a stack crane, (ii) or that it is moving from one stack position to another, (iii) or that it is idle. Two or more CTP vehicles are distinguished by their names, VehNm, which also identify their owning (operating) CTP (by name, CTPNm).
- The stack crane, sc, *behaviour maintains ..., etc.* ***Unit***. The stack crane state, SCraΣ, embodies only its current task, i.e., either (i) the container movement (either way) between a CTP vehuicle and a stack group/bay/row tier, (ii) or that it is moving from one stack position to another, (iii) or that it is idle. Two or more quay cranes are distinguished by their names, StakCraNm, which also identify their owning (operating) CTP (by name, CTPNm).
- The stack group and bay, stk_bay, *behaviour maintains ..., etc.* ***Unit***. The stack group and bay state, S_BayΣ, embodies the current group/bay, and for all bay rows and their tiers, the container cells, whether empty or occupied, and, for each located container, its BoL, etc. Two or more quay cranes are distinguished by their names which also identify their owning (operating) CTP (by name, CTPNm).

10.2 Channels

10.2.1 Channel Justification.

The various behaviours synchronise and communicate.

Container line, *cl*, and container line vessel, *cv*, behaviours synchronise and communicate (i) when the *cl* instructs the *cv* to change route, or (ii) when the *cv* informs of current events and/or requires new instructions from the *cl*. Container line vessel, *cv*, and container terminal port, *ctp*, behaviours synchronise and communicate when (i) the *cv* advices the *ctp* of its early, on schedule (i.e., imminent) or delayed arrival — requesting a berth position, or (ii) when the *cv* advices the *ctp* of its early, on schedule (i.e., imminent) or delayed departure, or (iii) (either way) of unusual events. Container bay, *cv_bay*, and quay crane, *qc*, behaviours synchronise and communicate when (i,ii) the *cq* requests permission to lift or drop a container with the *cv_bay*, or (iii,iv) actually performs that lift or drop, respectively. Quay crane, *qc*, and vehicle, *veh*, behaviours synchronise and communicate when (i) *cq* requests permission to lift or drop a container with the *veh*, or actually performs that lift or drop. Vehicle, *veh*, and stack crane, *sc*, behaviours synchronise and communicate when (i,ii) the *sc* requests permission to lift or drop a container with the *veh*, or (iii,iv) actually performs that lift or drop, respectively. Stack group/bay, *stk_bay*, and stack crane, *sc*, behaviours synchronise and communicate when (i,ii) the *sc* requests permission to lift or drop a container with the *stk_bay*, or (iii,iv) actually performs that lift or drop, respectively. CTP, *ctp*, and the quay crane, *qc*, behaviours synchronise and communicate when (i) the *qc* request the next job from the *ctp*, or when (ii) it is informing the *ctp* of completion of job, or when (iii) the *ctp* informs the *qc* of the next job (whether a container transfer, a crane move, or a 'remain idle' job). CTP, *ctp*, and the vehicle, *veh*, behaviours synchronise and communicate when (i) the *veh* request the next job from the *ctp*, or when (ii) it is informing the *ctp* of completion of job, or when (iii) the *ctp* informs the *veh* of the next job (whether a container transport, a non-container vehicle move, or a 'remain idle' job). CTP, *ctp*, and the stack crane, *sc*, behaviours synchronise and communicate when (i) the *sc* request the next job from the *ctp*, or when (ii) it is informing the *ctp* of completion of job, or when (iii) the *ctp* informs the *cc* of the next job (whether a container transfer, a crane move, or a 'remain idle' job).

We shall not motivate further channels.

10.2.2 Channel Declarations.

```
channel
    {cl_cv[ cln,cvn ]|cln:CLNm,cvn:CVNm}:CL_CV_M
```

$\{$cv_ctp[cvn,ctpn]|cvn:CVNm,ctpn:CTPNm$\}$:CV_CTP_M
$\{$cvbay_qc[bid,qcn]|bid:Bid,qcn:QuayCraNm$\}$:CVBay_QCra_M
$\{$qc_veh[qcn,vehn]|qcn:QuayCraNm,vehn:VehNm$\}$:QCra_Veh_M
$\{$veh_sc[vehn,scn]|,vehn:VehNm,scn:StakCraNm$\}$:Veh_SCra_M
$\{$sc_stkbay[scn,bid]|scn:StakCraNm,bid:Bid$\}$:SCra_SBay_M
$\{$ctp_qc[ctpn,qcn]|ctpn:CTPNm,gcn:QuayCraNm$\}$:CTP_QCra__M
$\{$ctp_veh[ctpn,vehn]|ctpn:CTPNm,vehn:VehNm$\}$:CTP_Veh_M
$\{$ctp_sc[ctpn,scn]|ctpn:CTPNm,scn:StakCraNm$\}$:CTP_SCra_M

We omit definition of the channel messages.

Annotations: We annotate just two of the channel declarations.

- The cl_cv[cln,cvn] declaration establishes a bundled set of sets of channels: one for each container line and any one of its vessels.
- The sc_stkbay[scn,bid] declaration establishes a bundled set of sets of channels: one for each stack crane (of a CTP) and any one of the group bays of the stack of that CTP. (Recall that from the bay identifier we can observe the group and the CTP (both by name).

10.3 The System, A Sketch

10.3.1 The Narrative.

The container line industry is now the parallel composition of

- (1) the parallel composition of one or more **c**ontainer **l**ines, and, for each line, with
  - (2) the parallel composition of one or more **c**ontainer **v**essels, and, for each vessel, with
    - (3) the parallel composition of one or more **c**ontainer **v**essel **bay**s, with
- (4) the parallel composition of two or more CTPs (**ctp**), and, for each CTP, with
  - (5) the parallel composition of one or more **q**uay **c**ranes, with
  - (6) the parallel composition of one or more **veh**icles, with
  - (7) the parallel composition of one or more **s**tack **c**ranes, with
  - (8) the parallel composition of one or more **sta**c**k bay**s.
  - (9) And possibly etcetera.

10.3.2 A Sketch Formalisation.

value
1. system() $\equiv$
2. $\parallel$ $\{$cl(cln)(clω(cln)) | cln:CLNm$\}$ $\parallel$
3. $\parallel$ $\{$cv(cln,cvn)(cvω(cln,cvn)) | cln:CLNm,cvn:CVNm $\bullet$ $\mathcal{P}_{vessel}$$\}$ $\parallel$
4. $\parallel$ $\{$cv_bay(cln,cvn,bid)(cv_bayω(cln,cvn,bid)) | cln:CLNm,cvn:CVNm,bid:Bid $\bullet$ $\mathcal{P}_{cv\_bay}$$\}$ $\parallel$
5. $\parallel$ $\{$ctp(pn)(ctpω(pn)) | pn:CTPNm$\}$ $\parallel$
6. $\parallel$ $\{$qc(pn,qcn)(qcω(pn,qcn)) | pn:CTPNm,qcn:QuayCraNm $\bullet$ $\mathcal{P}_{qc}$$\}$ $\parallel$
7. $\parallel$ $\{$veh(pn,vehn)(vehω(pn,vehn)) | pn:CTPNm,vehn:VehNm $\bullet$ $\mathcal{P}_{veh}$$\}$ $\parallel$
8. $\parallel$ $\{$sc(pn,scn)(scω(pn,scn)) | pn:CTPNm,scn:StakCraNm $\bullet$ $\mathcal{P}_{sc}$$\}$ $\parallel$
9. $\parallel$ $\{$stk_bay(pn,bid)(stk_bayω(pn,bid)) | pn:CTPNm,bid:Bid $\bullet$ $\mathcal{P}_{stk\_bay}$$\}$ $\parallel$

..

where

```
value:type
   clω:         CLΩ = CLNm →m CLΣ
   cvω:         CVΩ = (CLNm×CVNm) →m CVΣ
   cv_bayω:     CV_BayΩ = (CLNm×CVNm×Bid) →m CV_BayΣ
   ctpω:        CTPΩ = CTPNm →m CTPΣ
   qcω:         QuayCraΩ = (CTPNm×QuayCraNm) →m QuayCraΣ
   vehω:        VehΩ = (CTPNm×VehNm) →m VehΣ
   scω:         StakCraΩ = (CTPNm×StakCraNm) →m StakCraΣ
   stk_bayω:    Stak_BayΩ = (CTPNm×Bid) →m StakΣ
```

Annotations: In the above definitions we both define some types and arbitrarily postulate some values of those types.

- The $X\omega$ values generally map single or compounded names or identifiers into $X\sigma$ states.
- The $\mathcal{P}_\pi$ predicates constrain the compound names or identifiers. Examples are:
  - $\mathcal{P}_{vessel}$ expresses that the container vessel names must be those of container vessels of the identified container line.
  - $\mathcal{P}_{cv\_bay}$ first expresses the same as $\mathcal{P}_{vessel}$ and then that the bay identifier is that of the identified vessel.
- The various values of the above types cannot be completely arbitrarily selected:
  - The container line names of the definiton set of cvω, respectively in the definiton set of cv_bayω must be those of the definition set of clω.
  - The bay identifiers of cv_bayω must be bay identifiers of the designated, cvn, vessel, i.e., obs_CVNm(bid)=cvn.
  - The CTP names of qcω, vehω, scω, and stak_bayω must be those of the definition set of ctpω.
  - The quay crane names, qcn, of qcω must be names of quay cranes of the designated, pn, CTP that is: obs_CTPNm(qcn)=pn.
  - The vehicle names, vehn, of vehω must be names of vehicles of the designated, pn, CTP, that is: obs_CTPNm(vehn)=pn.
  - The stack crane names, scn, of scω must be of the designated, pn, CTP, that is: obs_CTPNm(scn)=pn.
  - The stack bay indentiifers, bid, of stk_bayω must be names of stack bays of the designated, pn, CTP, that is: obs_CTPNm(bid)=pn.
  - The above constraints can all be formalised in a set of axioms.

```
value
   ctpns:CTPNm-set, cvns:CVNm-set
axiom
   dom ctpns=ctpω ∧
   cvns={cv|(cln,cvn):(CLNm×CVNm)•(cln,cvn)∈ dom cvω}
value
   cl: cln:CLNm → CLΣ →
         in,out {cl_cv[ cln,cvn ]|cvn:CVNm•obs_CLNm(cvn)=cln}  Unit
   cv: cln:CLNm × cvn:CVNm → CVΣ →
         in,out cl_cv[ cln,cvn ]
         in,out {cv_ctp[ cvn,pn ]|ctpn:CTPNm•ctpn ∈ ctpns}  Unit
   cv_bay: cln:CLNm × cvn:CVNm × bid:Bid → CV_BayΣ →
         in,out {cvbay_qc[ bid,qc ]|qc:QuayCraNm}  Unit
```

```
ctp: pn:CTPNm → CTPΣ →
        in,out {cv_ctp[ cvn,pn ]|cvn:CVNm•cvn ∈ cvns}
        in,out {ctp_qc[ pn,qcn ]|qcn:QuayCraNm•obs_CTPNm(qcn)=pn}
        in,out {ctp_veh[ pn,vehn ]|vehn:VehNm•obs_CTPNm(vehn)=pn}
        in,out {ctp_sc[ pn,sc ]|sc:StakCraNm•obs_CTPNm(sc)=pn}  Unit
qc: qcn:QuayCraNm → QCraΣ →
        in,out {cvbay_qc[ bid,qcn ]|bid:Bid•∃ cvn:CVNm•cvn ∈ cvns∧obs_CVNm(bid)=cvn}
        in,out {ctp_qc[ pn,qcn ]|pn:CTPNm•obs_CTPNm(qcn)=pn}  Unit
veh: vehn:VehNm → VehΣ →
        in,out {qc_veh[ qcn,vehn ]|qcn:QuayCraNm•obs_CTPNm(qcn)=obs_CTPNm(vehn)}
        in,out {veh_sc[ vehn,scn ]|scn:StakCraNm•obs_CTPNm(scn)=obs_CTPNm(vehn)}
        in,out {ctp_qc[ pn,vehn ]|pn:CTPNm•obs_CTPNm(vehn)=pn}  Unit
sc: scn:StakCraNm → SCraΣ →
        in,out {veh_sc[ vehn,scn ]|vehn:VehNm•obs_CTPNm(scn)=obs_CTPNm(vehn)}
        in,out {ctp_qc[ pn,scn ]|pn:CTPNm•obs_CTPNm(scn)=pn}  Unit
stk_bay: bid:Bid → S_BayΣ →
        in,out {sc_stkbay[ scn,bid ]|scn:StakCraNm•obs_CTPNm(scn)=obs_CTPNm(bid)}  Unit
...
```

10.4 Discussion of the Sketch Formalisation

We have sketched a number of container line industry behaviours: container line, vessel, vessel bay, CTP harbour master, quay crane, CTP vehicle, stack crane, and stack bay. We have not sketched the following behaviours: transfer area, inland truck, train and barge, shipper (forwarder), and customer. In a more complete report such will be done.

We have assembled all of these behaviours into an overall system behaviour which contains multiple instantiations of all of the immediately above-listed behaviours, each with their own initial state, and each with the ability to synchronise and communicate with all relevant other behaviours. We have earlier sketched the definition of some of these behaviours: the CTP behaviour, `ctp`, Pages 46 and 49, the quay crane behaviour, Page 47, and the vehicle behaviour, Page 50.

Thus we have given the more-or-less final signature of most of the import container line industry behaviours and inter-comunication means (i.e., channels) for all of these behaviours. We have to define all behaviours, also the one illustrated earlier (CTP, quay crane and vehicle). We have to define types of all channel messages and behaviour states.

11 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.

12 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 [7]. We encourage the reader to study [7].

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 [8]. 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 [9].

A good terminology is provided by The *A-Z Dictionary of Export, Trade and Shipping Terms:* www.exportbureau.com/trade_shipping_terms/dictionary.html.

The general issues of stowage are briefly covered in [10, 6, 11, 12], in [13, 14] and in [15]. 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. 14. The literature review, Sect. 2 of [15], 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 [16, 17]. [17] 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. 14.

Further references (topics and URLs) are: Container Terminals: en.wikipedia.org/wiki/Container_terminal, Container Handbook: www.containerhandbuch.de/, Transshipment: en.wikipedia.org/wiki/Transshipment, Shipping and Logistics: www.plymouth.ac.uk/pages/view.asp?-page=15379, Supply chain and Logistics: ctl.mit.edu/,

We refer the reader to [18] for information about the ship dynamics terms: center of gravity, meta-centric height, list, rime, draft, drag, etc., and their calculations.

13 Conclusion

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

[3] 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.

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

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

[6] 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 [11] the stowage planning problem is shown to be NP-complete. For these reasons, the Suspensory Heuristic Procedure was developed.

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

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

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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 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 [2, 3, 1].

A.1 RSL Types

A.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 B $\times$ C $\times$ ... $\times$ 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 $\underset{m}{\rightarrow}$ B), or (A*)-**set**, or (A-**set**)list, or (A|B) $\underset{m}{\rightarrow}$ (C|D|(E $\underset{m}{\rightarrow}$ 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.

A.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:

A.2 The RSL Predicate Calculus

A.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.

A.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.

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

A.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.

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.

Maps: Simple map enumerations:
The last line below expresses map comprehension:

A.3.2 RSL Set Operations

- $\in$: The membership operator (*is an element member of a set*, **true** *or* **false***?*);
- $\notin$: The non-membership operator (*is an element not a member of a set*, **true** *or* **false***?*);
- $\cup$: The infix union operator (when applied to two sets expresses the set whose members are in either or both of the two operand sets);
- $\cup$: 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*);
- $\cap$: The infix intersection operator (expresses *the set whose members are in both of the two operand sets*);
- $\cap$: 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*);
- $\backslash$: 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*);
- $\subset$: 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***?*);
- $\subseteq$: The subset operator (as for proper subset, but allows equality of the two operand set to be **true**);
- $=(\neq)$: The equal operator (*are the two operand sets the same (different)*, **true** *or* **false***?*); and
- **card**: The cardinality operator (*"counts" the number of elements in the presumed finite operand set*).

A.3.3 RSL Cartesian Operations

A.3.4 RSL List Operations

- **hd**: Head: Yield the head (i.e., first) element of non–empty lists.
- **tl**: Tail: Yield the list of list elements other than the head of the argument list (also only of non–empty lists) .
- **len**: Length: the length of a finite list.
- **inds**: Indices, or index set: Yield 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: Yield 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.
- $\widehat{\ }$: Concatenate 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 $\neq$: Compare two operand lists for equality, element–by–element, respectively for the occurrence of at least one deviation!

A.3.5 RSL Map Operations

- •(•): Application: expresses that functions and maps can be applied to arguments.
- **dom**: Domain/Definition Set: denote "taking" the definition set values of a map (the a values for which the map is defined).
- **rng**: Range/Image: denote "taking" the range of a map (the corresponding b values for which the map is defined).
- †: Override/Extend: when 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,
- ∪: Merege: when applied to two operands denote the map which is the merge of two such maps,
- \: Restriction: the map which is a restriction of the first operand map to the elements that are not in the second operand set
- /: Restriction: the map which is a restriction of the first operand map to the elements of the second operand set.
- $=, \neq$: Equal, Not–Equal: when applied to two maps, compares these for equality, respectively inequality.
- °: Composition: The 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.

A.4 RSL λ–Calculus and Functions

A.4.1 The λ–Calculus Syntax

A.4.2 Free and Bound Variables

A.4.3 Substitution

A.4.4 α–Renaming and β–Reduction

A.4.5 The RSL λ–Notation

A.4.6 Function Signatures in RSL

A.4.7 Function Definitions in RSL

A.5 Applicative Constructs of RSL

A.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 $\mathcal{P}$(a) for evaluation in the body $\mathcal{B}$(a).

Patterns and Wild Cards: Some indicative examples:

A.5.2 The Applicative RSL Conditionals

A.5.3 Common Operator/Operand RSL Constructs

A.6 Imperative Constructs of RSL

A.6.1 Variables, Assignments and the Unit Value

A.6.2 Statement Sequence and skip

A.6.3 The Imperative RSL Conditionals

A.6.4 The Iterative RSL Conditionals

A.6.5 The Iterative RSL Sequencing

A.6.6 RSL Variable Expressions

A.7 RSL-CSP: Parallel Constructs of RSL

A.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.

A.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.

A.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.

A.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.

A.8 Simple RSL Specifications

Not using schemes, classes and objects 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.

Index