Container Terminals
An Initial Domain Analysis & Description Sketch

Dines Bjørner

Fredsvej 11, DK-2840 Holte and DTU, DK-2800 Kgs. Lyngby, Denmark.

e-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~dibj

. November 10, 2018: 09:36 am

The ECNU November 2018 Course Project

1

Abstract

We present a recording of stages and steps of a development of a domain analysis & description of an answer to he question: *what, mathematically, is a container terminal?*

Caveats: The present, . **November 10, 2018: 09:36 am**, version of this document is "vastly" incomplete". It is being distributed only to a limited circle of people so that they can see that my ECNU course project proposal is one of substance. Being incomplete, it is rich on incomplete formulas and poor on explanatory text; and can only be understood, i.e., appreciated, that is, requires non-trivial knowledge of:

> Dines Bjørner.
> Manifest Domains: Analysis & Description.
> Formal Aspects of Computing, 29(2):175–225,
> Online: July 2016. DOI 10.1007/s00165-016-0385-z
> Springer, BCS copyright 2016

Limited Circulation:

This document constitutes my preparation for a possible student project at ECNU in November 2018.
The students are themselves to analyse & describe a domain of container terminals.
Therefore, please, do not circulate this document!

Contents

Abstract

This is a report on an experiment. At any stage of development, and the present draft stage is judged 2/3 “completed” it reflects how I view an answer to the question *what is a container terminal port ?* mathematically speaking.

1 Introduction 3

TO BE WRITTEN

1.1 Survey of Literature on Container-related Matters 4

[1, A Container Line Industry Domain, 2007]

[2, A-Z Dictionary of Export, Trade and Shipping Terms]

[3, Portworker Development Programme: PDP Units]

[4, An interactive simulation model for the logistics planning of container operations in seaports,1996]

[5, Stowage planning for container ships to reduce the number of shifts, 1998]

[6, Container stowage planning: a methodology for generating computerised solutions, 2000]

[7, Container ship stowage problem: complexity and connection to the coloring of circle graphs, 2000] 5

[8, Container stowage pre-planning: using search to generate solutions, a case study, 2001]

[9, A genetic algorithm with a compact solution encoding for the container ship stowage problem, 2002]

[10, Multi-objective ... stowage and load planning for a container ship with container rehandle ..., 2004]

[11, Container terminal operation and operations research - a classification and literature review, 2004]

[12, Online rules for container stacking, 2010]

2 Some Pictures 6

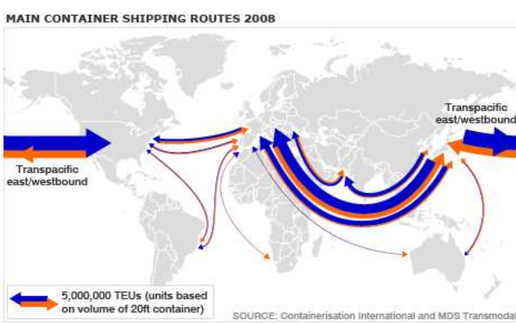
2.1 Terminal Port Container Stowage Area

7 Analysis of the above picture:

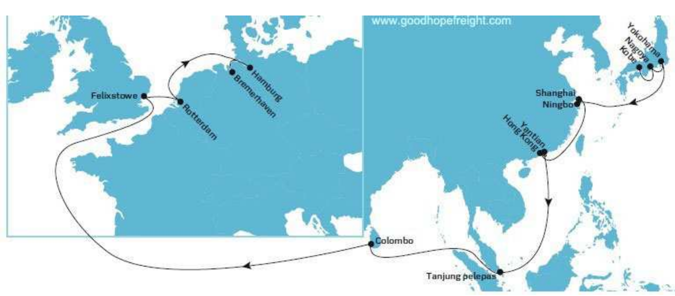
- The picture shows a *terminal*.
- At bottom we are hinted (through shadows) at *quay cranes* serving (unshown) *vessels*.
- Most of the picture shows a *container stowage area*, here organsed as a series of columns, from one side of the picture to the other side, e.g., left-to-right, sequences (top-to-bottom) of [blue] *bays* with *rows* of *stacks* of *containers*.
- Almost all columns show just one *bay*.
- Three “rightmost” columns show many [non-blue] *bays*.
- Most of the column “tops” and “bottoms” show *stack cranes*.
- The four leftmost columns show *stack cranes* at *bays* “somewhere in the middle” of a column.

2.2 Container Stowage Area and Quay Cranes 8

2.3 Container Vessel Routes 9



10
11



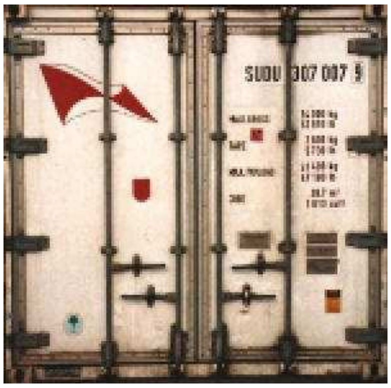
12

2.4 Containers 13

2.4.1 40 and 20 Feet Containers

14  

2.4.2 Container Markings 15



2.5 Container Vessels 16

17  

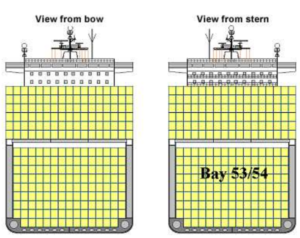
18

Quay cranes and *vessel* showing *row* of *aft (rear) bay*.

2.6 Container Stowage Area: Bays Rows, Stacks and Tier 19

20

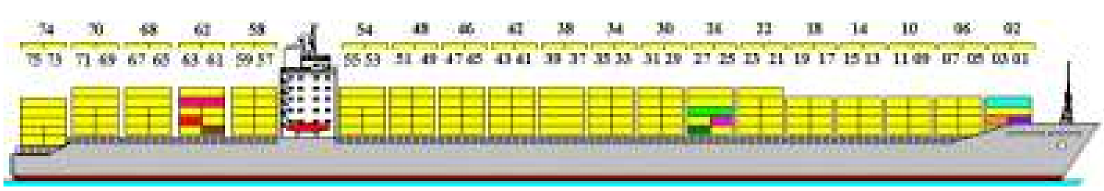
Bay, Row, Tier Numbers. Row Numbers 21



22

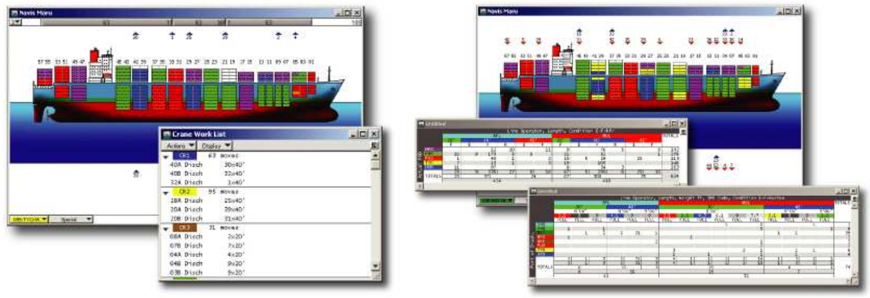
Cross section of a Bay. Tier Numbers.

23



Bay Numbering

2.7 Stowage Software



2.8 Quay Cranes

2.9 Container Stowage Area and Stack Cranes

2.10 Container Stowage Area 30

31

2.11 Quay Trucks 32

33

2.12 Map of Shanghai and YangShan 34

3 SECT 35

- *Shanghai East Container Terminal*

  - is the joint venture terminal of
  - *APM Terminals* and
  - *Shanghai International Port Group*
  - in *Wai Gao Qiao* port area of *Shanghai*.

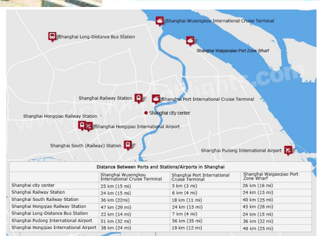
- No.1 Gangjian Road, Pudong New District, Shanghai, China

36

37

38

39



Distance Between Ports and Stations/Airports in Shanghai

	Shanghai Wusongkou International Cruise Terminal	Shanghai Port International Cruise Terminal	Shanghai Waigaoqiao Port Zone Wharf
Shanghai city center	25 km (15 mi)	5 km (3 mi)	26 km (16 mi)
Shanghai Railway Station	24 km (15 mi)	6 km (4 mi)	24 km (15 mi)
Shanghai South Railway Station	36 km (22mi)	18 km (11 mi)	40 km (25 mi)
Shanghai Hongqiao Railway Station	47 km (29 mi)	24 km (15 mi)	45 km (28 mi)
Shanghai Long-Distance Bus Station	22 km (14 mi)	7 km (4 mi)	24 km (15 mi)
Shanghai Pudong International Airport	51 km (32 mi)	56 km (35 mi)	36 km (22 mi)
Shanghai Hongqiao International Airport	38 km (24 mi)	19 km (12 mi)	40 km (25 mi)

4 Main Behaviours

40

41

- From consumer/origin to consumer/final destination:
  - **container loads** onto **land truck**;
  - **land truck travels** to **terminal stack**;
  - **container unloads** by means of **terminal stack crane** from **land truck** onto **terminal stack**.
  - **Container moves** from **stack** to **vessel**:
    - **terminal stack crane moves container** from **terminal stack** to **quay truck**,
    - **quay truck moves container** from **terminal stack** to **quay**,
    - **quay crane moves container** to **top of a vessel stack**;
  - **Container moves** on **vessel** from **terminal** to **terminal**:
    - Either container is unloaded at a next terminal port to a stack and from there to a container truck
    - or: container is unloaded at a next terminal port to a stack and from there to a next container vessel.

4.1 A Diagram 42

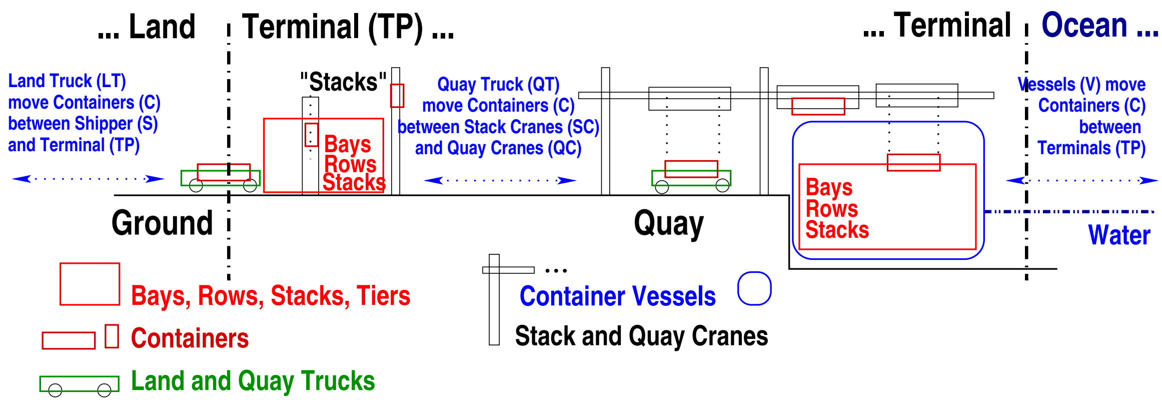


Fig. 1: Container Terminal Ports, I
A "from the side" snapshot of terminal port activities

4.2 Terminology - a Caveat 43

Bay [1]: contains indexed set of *rows* (of stacks of containers).

Container : smallest unit of central (i.e., huge) concern !

Container Stowage Area : An area of a vessel or a terminal where containers are stored, during voyage, respectively awaiting to be either brought out to shippers or onto vessels.

Crane :

- **Stack Crane** : moves *containers* between *land* or *terminal trucks* and *terminal stacks*.
- **Quay Crane** : moves *containers* between [*land* or] *terminal trucks* and *vessels*. 44

Land : ... as you know it ...

Ocean : ... as you know it ...

Shipper : arranges shipment of containers with container lines

Quay : area of terminal next to vessels (hence water).

[1] The terms introduced in this section are mine. They are most likely not the correct technical terms of the container shipping and stowage trade. I expect to revise this section, etc.

Row : contains indexed set of *stacks* (of containers).

Stack : contains indexed set of *containers*.

We shall also, perhaps confusingly, use the term stack referring to the land-based bays of a terminal.

Terminal : area of land and water between land and ocean equipped with container
45 stowage area, and stack and quay cranes, etc.

Truck :

- **Land Truck** : privately operated truck transport *containers* between *shippers* and *stack cranes*.
- **Quay Truck** : terminal operated special truck transport *containers* between *stack cranes* and *quay cranes*.

Tier : index of *container* in *stack*.

Vessel : contains a *container stowage area*.

4.3 Assumptions 46

Without loss of generality we can assume that there is exactly one stack crane per land-basederminal stack; quay cranes each serve exactly one bay on a vessel; there are enough quay cranes to serve all bays of any berthed vessel; quay trucks may serve any (quay and stack) crane; land trucks may serve more than one terminal; et cetera.

5 Endurants

47

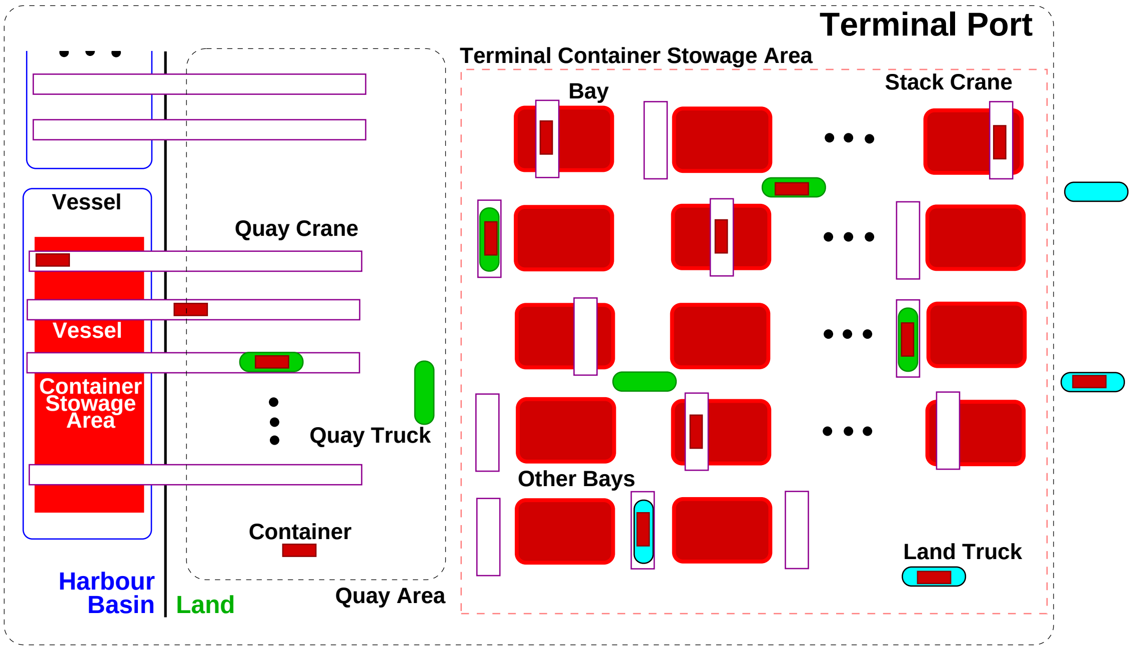


Fig. 2: Container Terminal Ports, II

A "from above" snapshot of terminal port activities

We refer to [13, Sects. 3., 4., and 5.].

Our model focuses initially on parts, that is, manifest, observable phenomena. Our choice of these is expected to be subject to serious revision once we ... MORE TO COME ...

5.1 Parts

48

We refer to [13, Sect. 3.3].

Our model has, perhaps arbitrarily, focused on just some of the manifest, i.e., observable parts of a domain of container terminal ports. We shall invariable refer to container terminal ports as either container terminals, or terminal ports, tp:TP, or just terminals. We expect revisions to the decomposition as shown as we learn more from professional stakeholders, e.g., *APM Terminals/SECT*, Shanghai.

1 In the container line industry, CLI, we can observe

2 a structure, TPS, of all terminal ports, and from each such structure, an indexed set, TPs, of two or more container *terminal* ports, TP;

3 a structure, VS, of all container vessels, and from each such structure, an indexed set, Vs, of one or more container *vessels*, V; and

4 a structure, LTS, of all land trucks, and from each such structure, a non-empty, indexed set, LTs of land trucks, LT;

49

```
type
1   CLI
2   TPS,  TPs = TP-set, TP
3   VS,    Vs = V-set, V
9   LTS,   LTs = LT-set, LT
value
2   obs_TPS: CLI → TPS, obs_TPs: TPS → TPs
3   obs_VS:   CLI → VS,   obs_Vs: VS → Vs
9   obs_LTS:  CLI → LTS, obs_LTs: LTS → LTs
axiom
2   ∀ cli:CLI•card obs_TPs(obs_TPS(cli))≥2
3      ∧ card obs_Vs(obs_VS(cli))≥1
9      ∧ card obs_LTs(obs_LTS(cli))≥1
```

5.1.1 Terminal Ports 50

In a terminal port, tp:TP, one can observe

5 a [composite] container stowage area, csa:CSA;

6 a structure, sqc:SQC, of quay cranes, and from that, a non-empty, indexed set, qcs:QCs, of one or more quay cranes, qc:QC;

7 structure, sqt:SQT, of quay trucks, and from that a non-empty, indexed set, qts:QTs, of quay trucks, qt:QT;

8 a structure, Scs:SCS, of stack cranes, and from that a non-empty, indexed set, scs:SCs, of one or more stack cranes, sc:SC;

9 a[n atomic] quay[2], q:Q[3]; and

10 a[n atomic] terminal port monitoring and control center, mcc:MCC.

51

[2]We can, without loss of generality, describe a terminal as having exactly one quay (!) – just as we, again without any loss of generality, describe it as having exactly one container stowage area.

[3]*Quay:* a long structure, usually built of stone, where boats can be tied up to take on and off their goods.

Pronunciation: key.

Thesaurus: berth, jetty, key, landing, levy, slip, wharf

type
5 CSA
6 SQC, QCs = QC-**set**, QC
7 SQT, QTs = QT-**set**, QT
8 SCS, SCs = SC-**set**, SC
9 Q
10 MCC
value
5 obs_CSA: TP → CSA
6 obs_SQC: TP → SQC, obs_QCs: SQC → QCs
7 obs_SQT: TP → SQT, obs_QTs: SQT → QTs
8 obs_SCS: TP → SCS, obs_SCs: SCS → SCs
9 obs_Q: TP → Q
10 obs_MCC: TP → MCC
axiom
6 ∀ sqc:SQC•**card** obs_QCs(sqc)≥1
7 ∀ sqt:SQT•**card** obs_QTs(sqt)≥1
8 ∀ scs:SCS•**card** obs_SCs(scs)≥1

5.1.2 Quays 52

Although container terminal port quays can be modelled as composite parts we have chosen to describe them as atomic. We shall subsequently endow the single terminal port quay with such attributes as quay segments, quay positions and berthing[4].

5.1.3 Container Stowage Areas: Bays, Rows and Stacks 53

11 From a container stowage area one can observe a non-empty indexed set of bays,

12 From a bay we can observe a non-empty indexed set of rows.

13 From a row we can observe a non-empty indexed set of stacks.

14 From a stack we can observe a possibly empty indexed set of containers.

54

type
11 BAYS, BAYs = BAY-**set**, BAY
12 ROWS, ROWs = ROW-**set**, ROW

[4] Berth: Sufficient space for a vessel to maneuver; a space for a vessel to dock or anchor; (whether occupied by vessels or not). Berthing: To bring (a vessel) to a berth; to provide with a berth.

```
13  STKS, STKs = STK-set, STK
14  CONS, CONs = CON-set, CON
value
11  obs_BAYS: CSA → BAYS, obs_BAYs: BAYS → BAYs
12  obs_ROWS: BAY → ROWS, obs_ROWs: ROWS → ROWs
13  obs_STKS: ROW → STKS, obs_STKs: STKS → STKs
14  obs_CONS: STK → CONS, obs_CONs: CONS → CONs
axiom
11  ∀ bays:BAYs • card bays > 0
12  ∀ rows:ROWs • card rows > 0
13  ∀ stks:STKs • card stks > 0
```

5.1.4 Vessels 55

From (or in) a vessel one can observe

15 [5] a container stowage area

16 and some other parts.

```
type
5   CSA
16  ...
value
5   obs_CSA: V → CSA
16  ...
```

5.1.5 Functions Concerning Container Stowage Areas 56

17 One can calculate

18 the set of all container storage areas:

19 of all terminal ports together with those

20 of all container lines.

```
value
17  cont_stow_areas: CLI → CSA-set
18  cont_stow_areas(cli) ≡
19      {obs_CSA(tp)|tp:TP•tp ∈ obs_TPs(obs_TPS(cli))}
20    ∪ {obs_CSA(cl)|cl:CL•cl ∈ obs_CLs(obs_CLS(cli))}
```

One can calculate the containers of

21 a stack,

22 a row,

23 a bay, and

24 a container stowage area.

```
value
21  extr_cons_stack: STK → CONs
21  extr_cons_stack(stk)≡obs_CONs(obs_CONS(stk))
22  extr_cons_row: ROW → CONs
22  extr_cons_row(row) ≡
22     {obs_CONs(obs_CONS(stk))|stk:STK•stk∈obs_STKs(obs_STKS(stk))}
23  extr_cons_bay: BAY → CONs
23  extr_cons_bay(bay) ≡
23     {obs_CONs(obs_CONS(row))|row:ROW•row∈obs_ROWs(obs_ROWS(bay))}
24  extr_cons_csa: CSA → CONs
24  extr_cons_csa(csa) ≡
24     {obs_CONs(obs_CONS(bay))|bay:BAY•bay∈obs_BAYs(obs_BAYS(csa))}
```

5.1.6 Axioms Concerning Container Stowage Areas 58

25 All rows contain different, i.e. distinct containers.

26 All bays contain different, i.e. distinct containers.

27 All container stowage areas contain different, i.e. distinct containers.

```
value
25  ∀ cli:CLI •
25     ∀ csa,csa′:CSA•{csa,csa′}⊆cont_stow_areas(cli) •
25        ∀ row,row′:ROW •
25           {row,row′}⊆obs_ROWs(obs_ROWS(csa))∪obs_ROWs(obs_ROWS(csa′))⇒
25           extr_cons_row(row) ∩ extr_cons_row(row′) = {} ∧
26        ∀ bay,bay′:BAY •
26           {bay,bay′}⊆obs_ROWs(obs_ROWS(csa))∪obs_ROWs(obs_ROWS(csa′))⇒
26           extr_cons_bay(bay) ∩ extr_cons_bay(bay′) = {} ∧
27           extr_cons_csa(csa) ∩ extr_cons_csa(csa′) = {}
```


5.1.7 Stacks 59

An aside: We shall use the term 'stack' in two senses: (i) as a component of container storage area bays; and (ii) to refer to the collection of stacks in a bay of a terminal
60 container storage area.

28 Stacks are created empty, and hence stacks can be *empty*.

29 One can *push* a *container* onto a *stack* and obtain a *non-empty stack*.

30 One can *pop* a *container* from a *non-epmpty stack* and obtain a pair of a *container* and a possibly empty *stack*.

```
value
28  empty: () → STK, is_empty: STK → Bool
29  push: CON × STK → STK
30  pop: STK →̃ (CON × STK)
axiom
28  is_empty(empty()), ∼is_empty(push(c,stk))
29  pop(push(c,stk)) = (c,stk)
30    pre pop(stk),pop(push(c.stk)): ∼is_empty(stk)
30  pop(empty()) = chaos
```

5.2 Terminal Port Command Centers 61

5.2.1 Discussion

We consider terminal port monitoring & control command centers to be atomic parts. The purpose of a terminal port command center is to monitor and control the allocation and servicing (berthing) of any visiting vessel to quay positions and by quay cranes, the allocation and servicing of vessels by quay cranes, the allocation and servicing of quay cranes by quay trucks, the allocation and servicing of quay trucks to quay cranes, containers and terminal stacks, the allocation and servicing of land trucks to containers and
62 terminal stacks, This implies that there are means for communication between a terminal command center and vessels, quay cranes, stack cranes, quay trucks, land trucks, terminal stacks and containers.

5.2.2 Justification 63

We shall justify the concept of terminal monitoring & control, i.e., command centers. First, using the *domain analysis & description* approach of [13], we know that we are going, through a transcendental deduction, to model certain parts as behaviours. These

parts, we decide, after some analysis that we forego, to be vessels, quay cranes, quay trucks, stack cranes stacks, land trucks, and containers. Behaviours are usually like actors:they can instigate actions. But we decide, in our analysis, that some of these behaviours, quay cranes, quay trucks, stack cranes and stacks, are *"passive" actors:* are
behaviourally not endowed with being able to initiate "own" actions. Instead, there- 65
fore, of all these behaviours, being able to communicate directly, pairwise, as loosely indicated by the figures of Pages 43 and 49, we model them to communicate *via* their terminal command centers.

This is how we justify the introduction of the concept of terminal command centers.
They are an abstraction. In *"ye olde days"* you could observe, not one, but, perhaps, 66
a hierarchy of terminal port offices, staffed by people, [each office, each group of staff] with its set of duties: communicating (by radio-phone) with approaching [and departing] vessels; scheduling quay positions, quay cranes and quay trucks; managing the operation of cranes and trucks; and, on a large scale, calculating stowage: on
vessels and in terminals. Today, *"an age of ubiquitous computing"*, most of these offices 67
and their staff are replaced by electronics: sensors, actuators, communication and computing, and with massive stowage data processing: where should containers be stowed on board vessels and in terminals so as to near-optimise all operations.

5.3 Unique Identifications 68

We refer to [13, Sect. 5.1].

31 Vessels have unique identifiers.

32 Quay cranes have unique identifiers.

33 Quay trucks have unique identifiers.

34 Stack cranes have unique identifiers.

35 Bays ("Stacks") of terminal container stowage areas have unique identifiers, cf. Item 39.

36 Land trucks have unique identifiers.

37 Terminal port command centers have unique identifiers.

38 Containers have unique identifiers.

39 Bays of container stowage areas have unique identifiers.

40 Rows of a bay have unique identifiers.

41 Stacks of a row have unique identifiers.

42 The part unique identifier types are mutually disjoint.

69

type
31 VI
32 QCI
33 QTI
34 SCI
35 TBI
36 LTI
37 MCCI
38 CI
39 BI
40 RI
41 SI

value
31 uid_V: V → VI
32 uid_QC: QC → QCI
33 uid_QT: QT → QTI
34 uid_SC: SC → SCI
34 uid_TBI: BAY → TBI
35 uid_LT: LT → LTI
37 uid_MCC: MCC → MCCI
37 uid_CON: CON → CI
34 uid_BAY: BAY → BI
35 uid_ROW: ROW → RI
36 uid_STK: STK → SI

axiom
42 VI, QCI, QTI, SCI, TBI, LTI, MCCI, CI, RI and SI mutually disjoint
42 TBI $\subset$ BI

5.3.1 Unique Identifiers: Distinctness of Parts 70

43 If two containers are different then their unique identifiers must be different.

axiom
43 $\forall$ con,con′:CON • con $\neq$ con′ $\Rightarrow$ uid_CON(con) $\neq$ uid_CON(con′)

The same distinctness criterion applies to stacks, rows, bays, container storage areas, terminal ports, cranes, vessels, etc.

5.3.2 Unique Identifiers: Two Useful Abbreviations 71

Container positions within a container stowage area can be represented in two ways:

44 by a triple of a bay identifier, a row identifier and a stack identifier, and

45 by these three elements and a tier position (i.e., position within a stack).

44 BRS = BI × RI × SI
45 BRSP = BI × RI × SI × **Nat**
axiom
45 ∀ (bu,ri,si,n):BRSP • n>0

5.3.3 Unique Identifiers: Some Useful Index Set Selection Functions 72

46 From a container stowage area once can observe all bay identifiers.

47 From a bay once can observe all row identifiers.

48 From a row once can observe all stack identifiers.

49 From a virtual container storage area, i.e., an icsa:iCSA, one can extract all the unique container identifiers.

73

value
46 xtr_BIs: CSA → BI-**set**
46 xtr_BIs(csa) ≡ {uid_BAY(bay)|bay:BAY•bay ∈ xtr_BAYs(csa)}

46 xtr_RIs: BAY → RI-**set**
47 xtr_RIs(bay) ≡ {uid_ROW(bay)|row:ROW•row ∈ obs_ROWs(bay)}

46 xtr_SIs: ROW → SI-**set**
48 xtr_SIs(row) ≡ {uid_STK(row)|stk:STK•stk ∈ obs_STKs(row)}

49 xtr_CIs: iCSA → CI-**set**
49 xtr_CIs(icsa) ≡
49   ... [to come] ...

5.3.4 Unique Identifiers: Ordering of Bays, Rows and Stacks 74

The bays of a container stowage area are usually ordered. So are the rows of bays, and stacks of rows. Ordering is here treated as *attributes* of container stowage areas, bays and stacks. We shall treat *attributes* further on.

5.4 States, Global Values and Constraints 75

5.4.1 States

50 We postulate a container line industry cli:CLI.

From that we observe, successively, all parts:

51 the set, cs:C-**set**, of all containers;

52 the set, tps:TPs, of all terminal ports;

53 the set, vs:Vs, of all vessels; and

54 the set, lts:LTs, of all land trucks.

value
50 cli:CLI
51 cs:C-**set** = obs_Cs(obs_CS(cli))
52 tps:TP-**set** = obs_TPs(obs_TPS(cli))
53 vs:V-**set** = obs_Vs(obs_VS(cli))
54 lts:LTs = obs_LTs(obs_LTS(cli))

76

We can observe

55 $csas$:CSA-**set**, the set of all terminal port container stowage areas of all terminal ports;

56 $bays$:BAY-**set**, the terminal port bays of all terminals;

57 the set, qcs:QC-**set**, of all quay cranes of all terminals;

58 the set, qts:QT-**set**, of all quay trucks of all terminal ports; and

59 the set, scs:SC-**set**, of all terminal (i.e., stack) cranes of all terminal ports.

value
55 $csas$:CSA-**set** = {obs_CSA(tp)|tp:TP•tp $\in$ tps}
55 $bays$:BAY-**set** = {obs_BAY(csa)|csa:CSA•csa $\in$ $csas$}
57 qcs:QC-**set** = {obs_QCs(obs_QCS(tp))|tp:TP•tp $\in$ tps}
58 qts:QT-**set** = {obs_QTs(obs_QTS(tp))|tp:TP•tp $\in$ tps}
59 scs:SC-**set** = {obs_SCs(obs_SCS(tp))|tp:TP•tp $\in$ tps}

5.4.2 Unique Identifiers 77

Given the generic parts outlined in Sect. we can similarly define generic sets of unique identifiers.

60 There is the set, *c_uis*, of all container identifiers;

61 the set, *tp_uis*, of all terminal port identifiers;

62 the set, *mcc_uis*, of all terminal port command center identifiers;

63 the set, *v_uis*, of all vessel identifiers;

64 the set, *qc_uis*, of quay crane identifiers of all terminal ports;

65 the set, *qt_uis*, of quay truck identifiers of all terminal ports;

66 the set, *sc_uis*, of stack crane identifiers of all terminal ports;

67 the set, *stk_uis*, of stack identifiers of all terminal ports;

68 the set, *lt_uis*, of all land truck identifiers; and

69 the set, *uis*, of all vessel, crane and truck identifiers.

78

value
60 *c_uis*:CI-**set** = {uid_C(c)|c:C•c∈*cs*}
61 *tp_uis*:TPI-**set** = {uid_TP(tp)|tp:TP•tp∈*tps*}
62 *mcc_uis*:TPI-**set** = {uid_MCC(obs_MCC(tp))|tp:TP•tp∈*tps*}
63 *v_uis*:VI-**set** = {uid_V(v)|v:V•v∈*vs*}
64 *qc_uis*:QCI-**set** = {uid_QC(qc)|qc:QC•qc∈*qcs*}
65 *qt_uis*:QTI-**set** = {uid_QT(qt)|qt:QT•qt∈*qts*}
66 *sc_uis*:SCI-**set** = {uid_SC(sc)|sc:SC•sc∈*scs*}
67 *stk_uis*:BI-**set** = {uid_BAY(stk)|stk:BAY•stk∈*stks*}
68 *lt_uis*:LTI-**set** = {uid_LL(lt)|lt:LT•lt∈*lts*}
69 *uis*:(VI|QCI|QTI|SCI|BI|LTI)-**set** = *v_uis*∪*qc_uis*∪*qt_uis*∪*sc_uis*∪*stk_uis*∪ *lt_uis*

79

70 the map, *tpmcc_idm*, from terminal port identifiers into the identifiers of respective command centers;

71 the map, *mccqc_idsm*, from command center identifiers into the set of quay crane identifiers of respective ports;

72 the map, *mccqt_idsm*, from command center identifiers into the identifiers of quay trucks of respective ports;

73 the map, *mccsc_idsm*, from command center identifiers into the identifiers of quay trucks of respective ports; and

74 the map, *mccbays_idsm*, from command center identifiers into the set of bay identifiers (i.e., "stacks") of respective ports;

80

value

70 *tpmcc_idm*:(TI $\overrightarrow{m}$ MCCI) = [uid_TP(tp)$\mapsto$uid_MCC(obs_MCC(tp))|tp:TP•tp $\in$ *tps*]
71 *mccqc_idsm*:(MCCI $\overrightarrow{m}$ QCI-**set**)
71 = [*tpmcc_uim*(uid_TP(tp)) $\mapsto$ { uid_QC(qc)
71 | qc:QC • qc $\in$ obs_QCs(obs_QCS(tp)) } | tp:TP•tp $\in$ *tps*]
72 *mccqt_idsm*:(MCCI $\overrightarrow{m}$ QTI-**set**) =
72 = [*tpmcc_uim*(uid_TP(tp)) $\mapsto$ { uid_QT(qt)
72 | qt:QT • qt $\in$ obs_QTs(obs_QTS(tp)) } | tp:TP•tp $\in$ *tps*]
73 *mccsc_idsm*:(MCCI $\overrightarrow{m}$ SCI-**set**)
73 = [*tpmcc_uim*(uid_TP(tp)) $\mapsto$ { uid_SC(sc)
73 | sc:SC • sc $\in$ obs_SCs(obs_SCS(tp)) } | tp:TP•tp $\in$ *tps*]
74 *mccbays_idsm*:(MCCI $\overrightarrow{m}$ BI-**set**)
74 = [*tpmcc_uim*(uid_TP(tp)) $\mapsto$ { uid_B(b)
74 | b:BAY•b $\in$ obs_BAYs(obs_BAYS(obs_CSA(tp)))} | tp:TP•tp $\in$ *tps*]

5.4.3 Some Axioms on Uniqueness 81

TO BE WRITTEN

5.5 Mereology 82

We refer to [13, Sect. 5.2].

5.5.1 Physical versus Conceptual Mereology

We briefly discuss a distinction that was not made in [13]: whether to base a mereology on *physical connections* or on *functional* or, as we shall call it, *conceptual relations*. We shall, for this domain model, choose the conceptual view. The physical mereology view can be motivated, i.e. justified, from the figures on pages 43 and 49. The conceptual view is chosen on the basis of the justification of the terminal command centers, cf. Sect. on Page 63. We shall model physical mereology as attributes.[5]

5.5.2 Vessels 83

Physical Mereology:

75 Vessels are physically "connectable" to quay cranes of any terminal port.

type
75 Phys_V_Mer = QCI-**set**
value
75 attr_Phys_V_Mer: V → Phys_V_mer

84

Conceptual Mereology:

76 Container vessels can potentially visit any container terminal port, hence have as [part of] their mereology, a set of terminal port command center identifiers.

type
76 V_Mer = MCCI-**set**
value
76 mereo_V: V → V_Mer
axiom
76 ∀ v:V • v ∈ vs ⇒ mereo_V(v) ⊆ mcc_uis

[5]Editorial note: Names of physical and of conceptual mereologies have to be "streamlined". As now, they are a "mess" !

5.5.3 Quay Cranes 85

Physical Mereology: In modelling the physical mereology, though as an attribute, of quay cranes, we need the notion of quay positions.

77 Quay cranes are, at any time, positioned at one or more adjacent quay positions of an identified segment of such.

```
type
77   Phys_QC_Mereo = QPSId × QP*
value
77   attr_Phys_QC: QC → Phys_QC_Mereo
```

86

78 The quay positions, qcmereo = (qpsid,qpl):QCMereo, must be proper quay positions of the terminal,

79 that is, the segment identifier, qpsid, must be one of the terminal,

80 and the list, qpl, must be contiguously contained within the so identifier segment.

```
axiom ∀ tp:TP,
78 let q = obs_Q(tp), qcs = obs_QCs(obs_QCS(tp)) in
79   ∀ q:Q • q ∈ qcs ⇒
79       let (qpsid,qpl) = obs_Mereo(q), qps = attr_QPSs(q) in
79       qpsid ∈ dom qps
80    ∧ ∃ i,j:Nat • {i,j} ∈ inds qpl ∧ ⟨(qps(qpsi))[k]|i≤k≤j⟩ = qpl
78 end end
```

87

Conceptual Mereology: The conceptual mereology is simpler.

81 Quay cranes are conceptually related to the command center of the terminal in which they are located.

```
type
81   QC_Mer = MCCI
value
81   mereo_QC: QC → QC_Mer
```


5.5.4 Quay Trucks 88

Physical Mereology:

82 Quay trucks are physically “connectable” to quay and stack cranes.

type
82 Phys_QT_Mer = QCI-**set** × QCI-**set**
value
82 attr_Phys_QT_Mer: QT → Phys_QT_Mer

Conceptual Mereology:

83 Quay trucks are conceptually connected to the command center of the terminal port of which they are a part.

type
83 QT_Mer = MCCI
value
83 mereo_QT: QT → QT_Mer

5.5.5 Stack Cranes 89

Physical Mereology:

84 Terminal stack cranes are positioned to serve one or more terminal area bays, one or more quay trucks and one or more land trucks.

85 The terminal stack crane positions are indeed positions of their terminal

86 and no two of them share bays.

type
84 Phys_SCmereo = s_bis:BI-**set** × s_qtis:QTI-**set** × s_ltis:LTI-**set**
axiom
84 ∀ (bis,qtis,ltis):Phys_SCmereo•bis≠{} ∧ qtis≠{} ∧ ltis≠{}
value
84 Phys_SCmereo: SC → Phys_SCmereo
axiom
84 ∀ tp:TP •
84 **let** csa=obs_CSA(tp), bays=obs_BAYs(obs_BAYS(csa)), scs=obs_SCs(obs_SCS(tp)) **in**
85 ∀ sc:SC•sc ∈ scs ⇒ Phys_SCmereo(sc) ⊆ xtr_BIs(csa)
86 ∧ ∀ tp′,tp″:TP•{tc′,tc″}⊆tcs ∧ tc′≠tc″
86 ⇒ s_bis(Phys_SCmereo(tc′)) ∩ s_bis(Phys_SCmereo(tc″))={} **end**

90

Conceptual Mereology: The conceptual stack crane mereology is simple:

87 Each stack is conceptually related to the command center of the terminal at which it is located.

type
87 SC_Mer = MCCI
value
87 mereo_SC: SC $\rightarrow$ SC_Mer

5.5.6 Container Stowage Areas 91

Bays, Rows and Stacks: The following are some comments related to, but not defining a mereology for container stowage areas.

88 A bay of a container stowage area

  a. has either a predecessor
  b. or a successor,
  c. or both (and then distinct).
  d. No row cannot have neither a predecessor nor a successor.

89 A row of a bay has a predecessor and a successor, the first stack has no predecessor and the last stack has no successor.

90 A stack of a row has a predecessor and a successor, the first stack has no predecessor, and the last stack has no successor.

92

value
88 BAY_Mer: BAY $\rightarrow$ ({|'nil'|}|BI) $\times$ (BI|{|'nil'|})
89 ROW_Mer: ROW $\rightarrow$ ({|'nil'|}|RI) $\times$ (RI|{|'nil'|})
90 STK_Mer: STK $\rightarrow$ ({|'nil'|}|SI) $\times$ (SI|{|'nil'|})
axiom
88 $\forall$ csa:CSA $\bullet$ **let** bs = obs_BAYs(obs_BAYS(csa)) **in**
88 $\forall$ b:BAY $\bullet$ b $\in$ bs $\Rightarrow$
88 **let** (nb,nb$'$) = mereo_BAY(b) **in**
88 **case** (nb,nb$'$) **of**
88a. ('nil',bi) $\rightarrow$ bi $\in$ xtr_BIs(csa),
88b. (bi,'nil') $\rightarrow$ bi $\in$ xtr_BIs(csa),
88d. ('nil','nil') $\rightarrow$ **chaos**,

88c. $\quad\quad (\mathsf{bi},\mathsf{bi}') \rightarrow \{\mathsf{bi},\mathsf{bi}'\} \subseteq \mathsf{xtr\_BIs(csa)} \wedge \mathsf{bi}{\neq}\mathsf{bi}'$
88 $\quad$ **end end end**
89 as for rows
90 as for stacks

5.5.7 Bay Mereology 93

Physical Vessel Bay Mereology:

91 A vessel bay is topologically related to the vessel on board of which it is placed and to the set of all quay cranes of all terminal ports.

type
91 Phys_VES_BAY_Mer = VI × QCI-**set**

Conceptual Vessel Bay Mereology:

92 A vessel bay is conceptually related to the set of all command centers of all terminal ports.

type
92 V_BAY_Mer = MCCI-**set**

Physical Terminal Port Bay (cum Stack) Mereology:

93 A terminal bay (cum stack) is topologically related to the stack cranes of a given terminal port and all land trucks.

type
93 Phys_STK_Mer = SCI-**set** × LTI-**set**

Conceptual Terminal Port Bay (cum Stack) Mereology:

94 A terminal port bay is conceptually related to the command center of its port.

type
94 T_BAY_Mer = MCCI

5.5.8 Land Trucks 94

Physical Mereology:

95 Land trucks are physically “connectable” to stack cranes – of any port.

type
95 Phys_LT_Mer = SCI-**set**
value
95 attr_Phys_LT_Mer: LT → Phys_LT_Mer

Conceptual Mereology:

96 Land trucks are conceptually connected to the command centers of any terminal port.

type
96 LT_Mer = MCCI-**set**
value
96 mereo_LT: LT → LT_Mer

5.5.9 Command Center 95

Command centers are basically conceptual quantities. Hence we can expect the physical mereology to be the conceptual mereology.

97 Command centers are physically and conceptually connected toall vessels, all cranes of the terminal port of the command center, all quay trucks of the terminal port of the command center, all stacks (i.e., bays) of the terminal port of the command center, and all land trucks, and all containers.

96

type
97 MCC_Mer = VI-**set**×QCI-**set**×QTI-**set**×SCI-**set**×BI-**set**×LTI-**set**×CI-**set**
value
97 mereo_MCC: MCC → MCC_Mer
axiom
97 ∀ tp:TP • tp ∈ *tps* •
97 **let** qcs:QC-**set** • qcs = obs_QCs(obs_QCS(tp)),
97 qts:QT-**set** • qts = obs_QTs(obs_QTS(tp)),
97 scs:SC-**set** • scs = obs_SCs(obs_SCS(tp)),

```
97       bs:iBAY-set • bs = obs_Bs(obs_BS(obs_CSA(tp))) in
97    let vis:VI_set • vis = {uid_VI(v)|v:V•v ∈ vs},
97       qcis:QCI_set • qcis = {uid_QCI(qc)|qc:QC•qc ∈ qcs},
97       qtis:QTI_set • qcis = {uid_QTI(qc)|qt:QT•qt ∈ qts},
97       scis:SCI-set • scis = {uid_SCI(sc)|sc:SC•sc ∈ scs},
97       bis:iBAY-set • bis = {uid_BI(b)|b:iBAY•b ∈ bs},
97       ltis:LTI-set • ltis = {uid_LTI(lt)|lt:LT•lt ∈ lts},
97       cis:SCI-set • cis = {uid_CI(c)|c:C•c ∈ cs} in
97    mereo_MCC(obs_MCC(tp)) = (vis,qcis,scis,sis,bis,ltis,cis) end end
```

5.5.10 Conceptual Mereology of Containers 97

The physical mereology of any container is modelled as a container attribute.

98 The conceptual mereology is modelled by containers being connected to all terminal command centers.

```
type
98  C_Mer = MCCI-set
value
98  mereo_C: C → C_Mer
axiom
98  ∀ c:C • mereo_C(c) = mcc_uis
```

5.6 Attributes 98

We refer to [13, Sect. 5.3].

5.6.1 States

By a state we shall mean one or more parts such that these parts have *dynamic* attributes, in our case typically *programmable* attributes.

5.6.2 Actions

Actions apply to states and yield possibly updated states and, usually, some result values. 99

We shall in this section, Sect. , on attributes, outline a number of *simple* (usually called *primitive*) actions of states. These actions are invoked by some behaviours either at their own volition, or in response to events occurring in other behaviours. The action outcomes are simple enough, but calculations resulting in these outcomes are not. Together the totality of the actions performed by the terminal's monitoring & control of vessels, cranes, trucks and the container stowage area, reflect the complexity of stowage handling.

5.6.3 Attributes: Vessels 100

99 A vessel is

   a. either at sea, at some *programmable* geographical location (longitude and latitude),

   b. or in some *programmable* terminal port – designated by the identifier of its command center and its quay position.

100 We consider the "remainder" of the vessel state as a programmable attribute – which we do not further define.

   The remainder includes all information about all containers, their bay/row/stack/tier positions, their bill-of-ladings, etc.

101 There may be other vessel attributes.

101

type
99 V_Pos == AtSea | InPort
99a. Longitude, Latitude
99a. AtSea :: Longitude × Latitude
99b. InPort :: MCCI × QPOS


```
100    VΣ
101    ...
value
99     attr_V_Pos: V → V_Pos
101    attr_VΣ: V → VΣ
101    attr_...: V → ...
axiom
99b.   ∀ mkInPort(ti):InPort • ti ∈ tp_uis
```

5.6.4 Attributes: Quay Cranes 102

102 At any one time a quay crane may *programmably* hold a container or may not. We model the container held by a crane by the container identifier.

103 At any one time a quay crane is *programmably* positioned in a quay position within a quay segment.

104 Quay cranes may have other attributes.

```
type
102   QCHold == mkNil('nil') | mkCon(ci:CI)
103   QCPos = QSId × QP
104   ...
value
102   attr_QCHold: QC → QCHold
103   attr_QCPos: QC → QCPos
104   ...
```

5.6.5 Attributes: Quay Trucks 103

105 At any one time a land truck may *programmably* hold a container or may not. We model the container held by a quay truck by the container identifier.

106 Quay trucks may have other attributes.

Note that we do not here model the position of quay trucks.

```
type
105   QTHold == mkNil('nil') | mkCon(ci:CI)
106   ...
value
105   attr_QTHold: QT → QTHold
106   ...
```

5.6.6 Attributes: Terminal Stack Cranes 104

107 At any one time a stack crane may *programmably* hold a container or may not. We model the container held by a crane by the container identifier.

108 Stack cranes are *programmably* positioned at a terminal bay.

109 Stack cranes may have other attributes.

type
107 SCHold == mkNil('nil') | mkCon(ci:CI)
108 SCPos = BI
108 ...
value
107 attr_SCHold: SC $\rightarrow$ SCHold
108 attr_SCPos: SC $\rightarrow$ SCPos
109 ...

5.6.7 Attributes: Container Stowage Areas 105

110 Bays of container storage areas *statically* have total order.

111 Rows of bays *statically* have total order.

112 Stacks of rows *statically* have total order.

We abstract orderings in two ways.

type
110 BOm = BI $\overrightarrow{m}$ **Nat**, BOl = BI*
111 ROm = RI $\overrightarrow{m}$ **Nat**, ROl = RI*
112 SOm = SI $\overrightarrow{m}$ **Nat**, SOl = SI*
axiom
110 $\forall$ bom:BOm•**rng** bom={1:**card dom** bom}, $\forall$ bol:BOl•**inds** bol={1:**len** bol}
111 $\forall$ rom:ROm•**rng** rom={1:**card dom** rom}, $\forall$ rol:ROl•**inds** rol={1:**len** rol}
112 $\forall$ som:SOm•**rng** som={1:**card dom** som}, $\forall$ sol:SOl•**inds** sol={1:**len** sol}
value
110 attr_BOm: CSA $\rightarrow$ BOm, attr_BOl: CSA $\rightarrow$ BOl
111 attr_ROm: BAY $\rightarrow$ ROm, attr_ROl: BAY $\rightarrow$ ROl
112 attr_SOm: ROW $\rightarrow$ SOm, attr_SOl: ROW $\rightarrow$ SOl

CSAs, BAYs, ROWs and STKs have (presently further) ***static*** *d*escriptions[6] and terminal and vessel container stowage areas have definite numbers 106

113 of bays,

114 and any one such bay a definite number of rows,

115 and any one such row a definite number of stacks,

116 and any one such stack a maximum loading of containers.

107

type
113 CASd
114 BAYd
115 ROWd
116 STKd
value
113 attr_CSAD: CSA → BI $\xrightarrow[m]{}$ CSAd
114 attr_BAYD: BAY → RI $\xrightarrow[m]{}$ BAYd
115 attr_ROWD: ROW → SI $\xrightarrow[m]{}$ ROWd
116 attr_STKD: STK → (**Nat** × STKd)

5.6.8 Attributes: Land Trucks 108

117 At any one time a land truck may *programmably* hold a container or may not. We model the container held by a land truck by the container identifier.

118 Land trucks also possess a further undefined *programmable* land truck state.

119 Land trucks may have other attributes.

Note that we do not here model the position of land trucks.

type
117 LTHold == mkNil('nil') | mkCon(ci:CI)
118 LTΣ
119 ...
value
117 attr_LTHold: LT → LTHold
118 attr_LTΣ: LT → LTΣ
119 ...

[6]Such descriptions include descriptions of for what kind of containers a container stowage area, a bay, a row and a stack is suitable: flammable, explosives, etc.

5.6.9 Attributes: Command Center 109

120 The `syntactic description`[7] of the spatial positions of quays, cranes and the container storage area of a terminal, TopLogDescr, is a *static* attribute.

121 The `syntactic description`[8] of the terminal state, i.e., the actual positions and deployment of vessels at quays, quay and stack cranes, quay and land trucks, and the actual container "contents" of these, TermΣDescr, is a *programmable* attribute.

type
120 TopLogDescr
121 MCCΣDescr
value
120 attr_TopLogDescr: MCC → TopLogDescr
121 attr_TermΣDescr: MCC → TermΣDescr

5.6.10 Attributes: Containers 110

122 A Bill-of-Lading[9] is a *static* container attribute. [10]

type
122 BoL
value
122 attr_BoL: C → BoL

111

[7] A `syntactic description` describes something, i.e., has some `semantics`, from which it is, of course, different.

[8] The `syntactic description` of the terminal state is, of course, not that state, but only its description. The terminal state is the combined states of all cranes, trucks and the container storage area.

[9] https://en.wikipedia.org/wiki/Bill_of_lading: A bill of lading (sometimes abbreviated as B/L or BoL) is a document issued by a carrier (or their agent) to acknowledge receipt of cargo for shipment. In British English, the term relates to ship transport only, and in American English, to any type of transportation of goods. A bill of Lading must be transferable, and serves three main functions: it is a conclusive receipt, i.e. an acknowledgment that the goods have been loaded; and it contains or evidences the terms of the contract of carriage; and it serves as a document of title to the goods, subject to the nemo dat rule. Bills of lading are one of three crucial documents used in international trade to ensure that exporters receive payment and importers receive the merchandise. The other two documents are a policy of insurance and an invoice. Whereas a bill of lading is negotiable, both a policy and an invoice are assignable. In international trade outside of the USA, Bills of lading are distinct from waybills in that they are not negotiable and do not confer title. The **nemo dat rule:** that states that the purchase of a possession from someone who has no ownership right to it also denies the purchaser any ownership title.

[10] For waybills see https://en.wikipedia.org/wiki/Waybill: A waybill (UIC) is a document issued by a carrier giving details and instructions relating to the shipment of a consignment of goods. Typically it will show the names of the consignor and consignee, the point of origin of the consignment, its destination, and route. Most freight forwarders and trucking companies use an in-house waybill called a house bill. These typically contain "conditions of contract of carriage" terms on the back of the form. These terms cover limits to liability and other terms and conditions

123 At any one time a container is positioned either

a. in a stack on a vessel: at sea or in a terminal, or
b. on a quay crane in a terminal port, being either unloaded from or loaded onto a vessel, or
c. on a quay truck to or from a quay crane, i.e., from or to a stack crane, in a terminal port, or
d. on a stack crane in a terminal port, being either unloaded from a quay truck onto a terminal stack or loaded from a terminal stack onto a quay truck, or
e. on a stack in a terminal port, or
f. on a land truck, or
g. idle.

A container position is a *programmable* attribute.

124 There are other container attributes. For convenience we introduce an aggregate attribute: CAttrs for all attributes.

112

type
123 CPos == onV | onQC | onQT | onSC | onStk | onLT | Idle
123a. onV :: VI $\times$ BRSP $\times$ VPos
123a. VPos == AtSea | InTer
123a. AtSea :: Geo
123a. InTer :: QPSid $\times$ QP$^+$
123b. onQC :: MCCI $\times$ QCI
123c. onQT :: MCCI $\times$ QTI
123d. onSC :: MCCI $\times$ SCI
123e. onStk :: MCCI $\times$ BRSP
123f. onLT :: MCCI $\times$ LTI
123g. Idle :: {|"idle"|}
124 CAttrs
value
123 attr_CPos: C $\rightarrow$ CPos
124 attr_CAttrs: C $\rightarrow$ CAttrs

6 Perdurants 113

We refer to [13, Sect. 7].

6.1 A Modelling Decision

In the *transcendental interpretation* of parts into behaviours we make the following modelling decisions: all atomic and all composite parts become separate behaviours such that vessels and terminal stacks are treated as "atomic" behaviours and such that containers that are elements of container stowage areas on vessels and in terminal stacks are not behaviours embedded in the behaviours of vessels and terminal stacks. 114

This modelling decision entails that container stowage areas, CSAs, of vessels and terminal stacks are modelled by replacing the [physical] containers of these CSAs with descriptions of these: their unique identifiers, their mereology, and their attributes. Thus we replace CSAs with vir_CSAs, see Sect. on Page 120.

6.2 Virtual Container Storage Areas 115

In our transition from endurants to perdurants we shall need a notion of container stowage areas which, for want of a better word, we shall call *virtal CSAs*. Instead of stacks embodying containers, they embody

125 container information: their unique identifier, mereology and attributes.

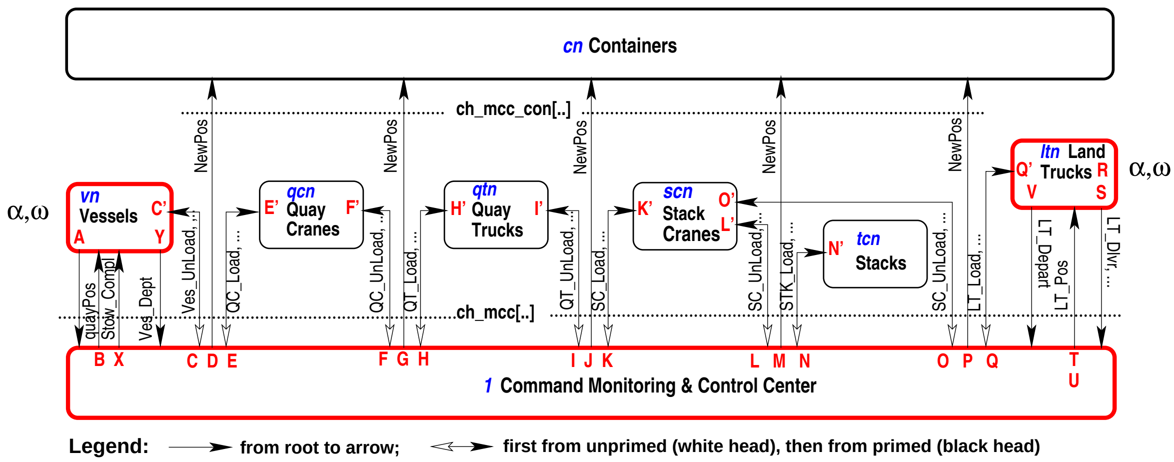
We must secure that no container is referenced more than once across the revised-model;

126 that is, that all ci:CIs are distinct.

116

type
5′ vir_CSA
11′ iBAYm = BI $\overrightarrow{m}$ vir_BAY, vir_BAY
12′ iROWm = RI $\overrightarrow{m}$ vir_ROW, vir_ROW
13′ iSTKm = SI $\overrightarrow{m}$ vir_STK, vir_STK
14′ iCONI = CInfo*
125 CInfo = CI $\times$ CMereo $\times$ CAttrs
value
5′ attr_vir_CSA: CSA $\rightarrow$ vir_CSA
11′ attr_iBAYm: vir_CSA $\rightarrow$ vir_BAYm
12′ attr_iROWm: vir_BAY $\rightarrow$ vir_ROWm
13′ attr_iSTKm: vir_ROW $\rightarrow$ vir_STKm
14′ attr_iCONI: vir_STK $\rightarrow$ vir_CONI
axiom
126 [all CIs of all CSAs are distinct]

6.3 Basic Model Parts 117



118 **Fig. 3: The Container Terminal Behaviours**[11]

There are c_n container behaviours, where c_n is the number of all containers of the system we are modelling. For each terminal port there is 1 controller behaviour, v_n vessel behaviours, where v_n is the number of vessels visiting that terminal port, qc_n quay crane behaviours, where qc_n is the number of quay cranes of that terminal port, qt_n quay truck behaviours, where qt_n is the number of quay trucks of that terminal port, lt_n land truck behaviours, where lt_n is the number of land trucks (of that terminal port), and tb_n terminal stack behaviours, where ts_n is the number of terminal bays of
119 that terminal port.

The vessel, the land truck and the terminal monitoring & control [command] center behaviours are *pro-active:* At their own initiative (volition), they may decide to communicate with other behaviours. The crane, quay truck, stack and container behaviours are *passive:* They respond to interactions with other behaviours.

6.4 Actions, Events, Channels and Behaviours 120

We refer to [13, Sect. 7.1].

In building up to the behavioral analysis & description of the terminal container domain we first analyse the actions and events of that domain. These actions and
121 events are the building blocks of behaviours.

Actions, to remind the reader, are explicitly performed by an actor, i.e., a behaviour, calculates some values and, usually, effect a state change.

Events *"occur to"* actors (behaviours), that is, are not initiated by these, but usually effect state changes.

[11] The labeling **A, B, C, D, ..., X, Y** may seem arbitrary, but isn't!

6.5 Actions 122

We refer to [13, Sects. 7.1.5, 7.3.1].

The unloading of containers from and the loading of container onto container stowage areas are modelled by corresponding actions on virtual container stowage areas. Vessels, land trucks and terminal monitoring & control centers, i.e., command centers, are here modelled as the only entities that can *initiate* actions.

6.5.1 Command Center Actions 123

Motivating the Command Center Concept: We refer to the **[A,B,...,U]** labeled arrows of the figure on Page 122.

Imagine a terminal port. It has several vessels berthed along quays. It also has quay space, i.e., positions, for more vessels to berth. Berthed vessels are being serviced by several, perhaps many quay cranes. The totality of quay cranes are being serviced by [many more] quay trucks. The many quay trucks service several terminal bays, i.e., 124
stacks. Land trucks are arriving, attending ***stacks*** and leaving. Quite a "busy scene". So is the case for all container terminal ports. 125

The concept of a *monitoring & control, i.e., a command* **center**, is an abstract one; the figure on Page 122 does not show a part with a **... center** label. The *actions* of vessels and trucks, and the ***events*** of cranes, terminal stacks and trucks are either hap-hazard, no-one interferes, they somehow "just happen", or they are somehow co-ordinated.

Whether "free-wheeling" or "more-or-less coordinated" we can think of a *command center* as somehow *monitoring and controlling actions and events*. 126

Terminal *monitoring & control centers*, also interchangeably referred to as *command centers*, are thus where the logistics of container handling takes place.

You may think of this command center as receiving notices from vessels and land trucks as to their arrival and with information about their containers; thus building up awareness, i.e., a state, of the containers of all incoming and arrived vessels and land trucks, the layout of the terminal and the state of its container stowage area, the current whereabouts of vessels, cranes and trucks. Quite a formidable "state". 127

We shall therefore model the "comings" and "goings" of vessels, trucks, cranes and stacks as if they were monitored and controlled by a command center, In our modelling we are not assuming any form of efficiency; there is, as yet no notion of optimality, nor of freedom from mistakes and errors. Our modelling – along these lines – is "hidden" in action pre- and post-conditions and thus allows for any degree of internal non-determinism. 128

Calculate Next Transaction: The *core* action of the command center is `calc_nxt_transaction`. We shall define `calc_nxt_transaction` only by its signature and

a pair of **pre/post** conditions. In this way we do not have to consider *efficiency, security, safety, etc.*, issues. These, i.e., the efficiency, security, safety, etc., issues can "always" be included in an *requirements engineering* implementation of `calc_nxt_transaction`. Basically the `calc_nxt_transaction` has to consider which of a
129 non-trivially large number of possible actions have to be invoked. They are listed in Items 128 to 135 below. The `calc_nxt_transaction` occurs in time, and occur repeatedly, endlessly, i.e., "ad-infinitum", At any time that `calc_nxt_transaction` is invoked the monitoring and control command center (mcc) is in some state. That state changes as the result of both monitoring actions and control actions. The `calc_nxt_transaction` therefore non-deterministically-internally chooses one among several
130 possible alternatives. If there is no alternative, then a **skip** action is performed.

The command center, mcc, models the following actions and events: **[A]** the `update` of the mcc state, mccσ, in response to the vessel action that inform the mcc of the vessel arrival.

127 The result of a `calc_nxt_transaction` is an transaction designator, MCCTrans and a state change. There are several alternative designators. We mention some:

128 **[B]:** the `calc`ulation of `ves`sel `pos`itions for [their] arrivals;

129 **[CDE]:** the `calc`ulation of `vess`el to `quay` `crane` container transfers;

131 130 **[FGH]:** the `calc`ulation of `quay` `crane` to `quay` `truck` container transfers;

131 **[IJK]:** the `calc`ulation of `quay` `truck` to `stack` `crane` container transfers;

132 **[LMN]:** the `calc`ulation of `stack` `crane` to `stack` container transfers;

133 **[OPQ]:** the `calc`ulation of `land truck` to `stack` `crane` container transfers;

134 **[T]:** the `calc`ulation of

135 **[X]:** the `calc`ulation that stowage, for a given vessel, has completed; and

136 the calculation that there is no next transaction that can be commenced.

137 The signature of the `calc_nxt_transaction` involves the unique identifier, mereology, static and programmable attributes, i.e., the state of the command center, and indicates that a command center transaction results and a next state "entered".

138 For this, the perhaps most significant action of the entire container terminal port operation, we "skirt" the definition and leave to a pair pf **pre/post** conditions that of characterising the result and next state.

132

type
127 MCCTrans == QayPos | VSQC_Xfer | QCQT_Xfer | QTSC_Xfer
127 | SCSTK_Xfer | SCLT_Xfer | LT_Dept | VS_Dept | Skip
128 **[B]:** QuayPos :: VI $\times$ QPos
129 **[CDE]:** VSQC_Xfer :: VI $\times$ BRS $\times$ CI $\times$ QCI
130 **[FGH]:** QCQT_Xfer :: QCI $\times$ CI $\times$ QTI
131 **[IJK]:** QTSC_Xfer :: QTI $\times$ CI $\times$ SCI
132 **[LMN]:** SCSTK_Xfer :: SCI $\times$ CI $\times$ BRS
133 **[OPQ]:** SCLT_Xfer :: SCI $\times$ CI $\times$ LTI
134 **[T]:** LT_Dept :: LTI
135 **[X]:** VS_Dept :: VI
136 Skip :: nil
value
137 calc_nxt_transaction: MCCI$\times$mereoMCC$\times$statMCC$\to$MCC$\Sigma\to$MCCTrans$\times$MCCΣ
137 calc_nxt_transaction(mcci,mccmereo,mmstat)(mccσ) **as** (mcctrans,mccσ')
138 **pre**: $\mathcal{P}_{calc\_nxt\_trans}$((mcci,mccmereo,mccstat)(mccσ))
138 **post**: $\mathcal{Q}_{calc\_nxt\_trans}$((mcci,mccmereo,mccstat)(mccσ))(mcctrans,mccσ')

133

The above mentioned actions are invoked by the command center in its endeavour to see containers moved from vessels to customers. A similar set of actions affording movement of containers customers to vessels, i.e., in the reverse direction: from land trucks to stack cranes, from stacks to quay trucks, from quay trucks to quay cranes, and from quay cranes to vessels, round off the full picture of all command center actions. 134

Command Center Action [A]: `update_mcc_from_vessel`:

139 Command centers

140 upon receiving arrival information, v_info, from arriving vessels, v_i, can update their state "accordingly".

141 We leave undefined the pre- and post-conditions.

value
139 update_mcc_from_vessel: VSMCC_MSG $\times$ MCC_Σ $\to$ MCC_Σ
140 update_mcc_from_vessel((vs_i,vir_csa,vs_info),mcc_σ) **as** mcc_σ'
141 **pre**: $P_{upd\_mcc\_f\_v}$((vs_i,vir_csa,vs_info),mcc_σ)
141 **post**: $Q_{upd\_mcc\_f\_v}$((vs_i,vir_csa,vs_info),mcc_σ)(mcc_σ')

135

Command Center Action [B]: `calc_ves_pos`:

142 Command centers

143 can calculate, q_pos, the quay segment and quay positions for an arriving vessel, v_i.

144 We leave undefined the pre- and post-conditions.

value
142 calc_ves_pos: MCCI$\times$MCC_mereo$\times$TopLog$\times$MCC$\Sigma\times$VI $\rightarrow$ (QSId$\times$QP*)$\times$MCCΣ
143 calc_ves_pos(mcc_i,mcc_mereo,toplog,mcc_σ,v_i) **as** (q_pos,mcc_σ')
144 **pre**: $P_{calc\_ves\_pos}$(mcc_i,mcc_mereo,toplog,mcc_σ,v_i)
144 **post**: $Q_{calc\_ves\_pos}$(mcc_i,mcc_mereo,toplog,mcc_σ,v_i)(q_pos,mcc_σ')

136

Command Center Action [C-D-E]: `calc_ves_qc`

145 The command center non-deterministically internally calculates a triplet:

146 the bay-row-stack coordinates, brs, from which a top container is to be removed by quay crane qci and a next command center state reflecting that calculation (and that the identified quay crane is being so alerted).

147 We leave undefined the relevant pre- and post-conditions

value
145 calc_ves_qc: MCCΣ $\rightarrow$ BRS $\times$ QCI $\times$ MCCΣ
146 calc_ves_qc(mccσ) **as** (brs,qci,mccσ')
147 **pre**: $\mathcal{P}_{calc\_ves\_qc}$(mcc$\sigma$)
147 **post**: $\mathcal{Q}_{calc\_ves\_qc}$(mcc$\sigma$)(brs,qci,mcc$\sigma$')

137

Command Center Action [F-G-H]: `calc_qc_qt`

148 The command center non-deterministically internally calculates a triplet:

149 the identities of the quay crane and quay truck from, respectively to which the quay crane is to transfer a container, and an update command center state reflecting that calculation (and that the identified quay crane and truck are being so alerted).

150 We leave undefined the relevant pre- and post-conditions

value
148 calc_qc_qt: MCCΣ $\rightarrow$ QCI $\times$ QTI $\times$ MCCΣ
149 calc_qc_qt(mccσ) **as** (qci,qti,mccσ')
150 **pre**: $\mathcal{P}_{calc\_qc\_qt}$(mccσ)
150 **post**: $\mathcal{Q}_{calc\_qc\_qt}$(mccσ)(qci,qti,mccσ')

138

Command Center Action [I-J-K]: `calc_qt_sc`

151 The command center non-deterministically internally calculates a triplet:

152 the identities of a quay crane, a container and a quay truck.

153 We leave undefined the relevant pre- and post-conditions

value
151 calc_qt_sc: MCCΣ $\rightarrow$ (QCI$\times$CI>QTI) $\times$ MCCΣ
152 calc_qt_sc(mccσ) **as** ((qci,ci,qtu),mccσ')
153 **pre**: $\mathcal{P}_{calc\_qt\_sc}$(mccσ)
153 **post**: $\mathcal{Q}_{calc\_qt\_sc}$(mccσ)((qci,ci,qtu),mccσ')

139

Command Center Action [L-M-N]: `calc_sc_stack`

154 The command center non-deterministically internally calculates a pair:

155 a triplet of the identities of a stack crane, a container and a terminal bay/row/stack triplet and a new state that reflects this action.

156 We leave undefined the relevant pre- and post-conditions

value
154 calc_sc_stack: MCCΣ $\rightarrow$ (SCI$\times$CI$\times$BRS)$\times$MCCΣ
155 calc_sc_stack(mccσ) **as** ((sci,ci,brs),mccσ')
156 **pre**: $\mathcal{P}_{calc\_sc\_stack}$(mcc$\sigma$)
156 **post**: $\mathcal{Q}_{calc\_sc\_stack}$(mcc$\sigma$)((sci,ci,brs),mcc$\sigma'$)

140

Command Center Action [N-M-L]: `calc_stack_sc`

157 The command center non-deterministically internally calculates a pair:

158 a triplet of a terminal bay/row/stack triplet and the identities of a container and a stack crane, and a new state that reflects this action.

159 We leave undefined the relevant pre- and post-conditions

value
157 calc_stack_sc: MCC$\Sigma \rightarrow$ (BRS$\times$CI$\times$SCI)$\times$MCCΣ
158 calc_stack_sc(mccσ) **as** ((brs,ci,sci),mccσ')
159 **pre**: $\mathcal{P}_{calc\_stack\_sc}$(mcc$\sigma$)
159 **post**: $\mathcal{Q}_{calc\_stack\_sc}$(mcc$\sigma$)((brs,ci,sci),mcc$\sigma'$)

141

Command Center Action [O-P-Q]: `calc_sc_lt`

160 The command center non-deterministically internally calculates a pair:

161 a triplet of the identities of a stack crane, a container and a land truck, and a new state that reflects this action.

162 We leave undefined the relevant pre- and post-conditions.

value
160 calc_sc_lt: MCC$\Sigma \rightarrow$ (BRS$\times$CI$\times$SCI)$\times$MCCΣ
161 calc_sc_lt(mccσ) **as** ((sci,ci,lti),mccσ')
162 **pre**: $\mathcal{P}_{calc\_sc\_lt}$(mccσ)
162 **post**: $\mathcal{Q}_{calc\_sc\_lt}$(mccσ)((sci,ci,lti),mccσ')

142

Command Center Action [Q-P-O]: `calc_lt_sc`

163 The command center non-deterministically internally calculates a pair:

164 a triplet of the identities of a land truck, a container and a stack crane, and a new state that reflects this action.

165 We leave undefined the relevant pre- and post-conditions.

value
163 calc_lt_sc: MCC$\Sigma \rightarrow$ (BRS$\times$CI$\times$SCI)$\times$MCCΣ
164 calc_lt_sc(mccσ) **as** ((lti,ci,sci),mccσ')
165 **pre**: $\mathcal{P}_{calc\_lt\_sc}$(mccσ)
165 **post**: $\mathcal{Q}_{calc\_lt\_sc}$(mccσ)((lti,ci,sci),mccσ')

143

Command Center: Further Observations Please observe the following: any terminal command, i.e., monitoring & control, center repeatedly and non-deterministically alternates between any and all of these actions. Observe further that: The intention of the very many pre and post-conditions [cf. Items 141, 144, 147, 150, 153, 156, 159, 165, and 162], express requirements to the command center states, $\mathsf{mcc}\sigma$:$\mathsf{mcc}\Sigma$, as to what information about vessels and the terminal port, it must contain and be updated with. Quite a complex state.

6.5.2 Container Storage Area Actions 144

We define two operations on virtual CSAs:

166 one of stacking (loading) a container, referred to by its unique identifier in a virtual CSA,

167 and one of unstacking (unloading) a container;

168 both operations involving bay/row/stack references.

145

type
168 BRS = BI × RI × SI
value
166 load_CI: iCSA × BRS × CI → iCSA
166 load_CI(icsa,(bi,ri,si),ci) **as** icsa′
166 **pre**: $\mathcal{P}_{load}$(icsa,(bi,ri,si),ci)
166 **post**: $\mathcal{Q}_{load}$(icsa,(bi,ri,si),ci)(icsa′)
167 unload_CI: ICSA × BRS $\xrightarrow{\sim}$ CI × iCSA
167 unload_CI(icsa,(bi,ri,si)) **as** (ci,icsa′)
167 **pre**: $\mathcal{P}_{unload}$(icsa,(bi,ri,si))
167 **post**: $\mathcal{Q}_{unload}$(icsa,(bi,ri,si))(ci,icsa′)

146

The Load Pre-/Post-Condtions

169 The virtual iCSA, i.e., icsa, must be wellformed;

170 the ci must not be embodied in that icsa; and

171 the bay/row/stack reference, (bi,ri,si) must be one of the [virtual] container stowage area.

value
166 $\mathcal{P}_{load}$(icsa,(bi,ri,si),ci) $\equiv$
169 well_formed(icsa) cf. 25– 27 on Page 60
170 $\wedge$ ci $\notin$ xtr_CIs(icsa) cf. 49 on Page 75
172 $\wedge$ valid_BRS(bi,ri,si)(icsa)

172 valid_BRS: BRS $\rightarrow$ iCSA $\rightarrow$ **Bool**
172 valid_BRS(bi,ri,si)(icsa) $\equiv$ bi $\in$ **dom** icsa$\wedge$ri $\in$ **dom** icsa(bi)$\wedge$si $\in$ **dom** (icsa(bi))(ri)

147

172 The resulting iCSA, i.e., icsa′, must have the same bqy, row and stack identifications, and

173 except for the designated bays, rows and stacks, must be unchanged.

174 The designated "before", i.e., the stack before loading, must equal the tail of the "after", i.e., the loaded stack, and

175 the top of the "after" stack must equal the "input" argument container identifier.,

148

value
167 $\mathcal{Q}_{load}$(icsa,(bi,ri,si),ci)(icsa′) $\equiv$
172 **dom** icsa = **dom** icsa′
172 $\wedge$ $\forall$ bi′:BI•bi′$\in$ **dom** icsa(bi′)$\Rightarrow$**dom** icsa(bi′)=**dom** icsa′(bi′)
172 $\wedge$ $\forall$ ri′:RI•bi′$\in$ **dom** (icsa(bi′))()$\Rightarrow$**dom** (icsa(bi′))(ri′)=(**dom** icsa′(bi′))(ri′)
172 $\wedge$ $\forall$ si′:BI•bi′$\in$ **dom** icsa(bi′)$\Rightarrow$**dom** ((icsa(bi′))(ri′))(si′)=**dom**((icsa′(bi′))(ri′))(si′)
173 $\wedge$ $\forall$ bi′:BI•bi′ $\in$ **dom** icsa $\setminus$ {bi}$\Rightarrow$icsa $\setminus$ {bi}=icsa′ $\setminus$ {bi}
173 $\wedge$ $\forall$ ri′:RI•ri′ $\in$ **dom** icsa(bi) $\setminus$ {ri}$\Rightarrow$(icsa(bi))(ri′)=(icsa′(bi))(ri′)
173 $\wedge$ $\forall$ si′:SI•si′ $\in$ **dom** (icsa)(ri′) $\setminus$ {si}$\Rightarrow$((icsa)(bi′))(si′)=((icsa′)(bi′))(si′)
174 $\wedge$ **tl**((icsa′)(bi′))(si′)=((icsa′)(bi′))(si′)
175 $\wedge$ **hd**((icsa′)(bi′))(si′)=ci

149

The Unload Pre-/Post-Conditions

176 The virtual iCSA, i.e., icsa, must be wellformed; and

177 the bay/row/stack reference, (bi,ri,si) must be one of the [virtual] container stowage area.

value
166 $\mathcal{P}_{unload}$(icsa,(bi,ri,si)) $\equiv$
176 well_formed(icsa)
177 $\wedge$ valid_BRS(bi,ri,si)(icsa)

178

179

180

181

182

151

value
167 $\mathcal{Q}_{unload}$(icsa,(bi,ri,si))(ci,icsa′) ≡
178 **dom** icsa = **dom** icsa′
179 ∧ ∀ bi′:BI • bi′ ∈ **dom** icsa \ {bi} ⇒ icsa \ {bi} = icsa′ \ {bi}
180 ∧ ∀ ri′:RI • ri′ ∈ **dom** icsa(bi) \ {ri} ⇒ (icsa(bi))(ri′) = (icsa′(bi))(ri′)
181 ∧ ∀ si′:SI • si′ ∈ **dom** (icsa)(ri′) \ {si} ⇒ ((icsa)(bi′))(si′) = ((icsa′)(bi′))(si′)
182 ∧ ((icsa′)(bi′))(si′)=**tl**((icsa′)(bi′))(si′) ∧ **hd**((icsa)(bi′))(si′)=ci

6.5.3 Vessel Actions 152

Vessels (and land trucks) are in a sense, the primary movers in understanding the terminal container domain. Containers are, of course, at the very heart of this domain. But without container vessels (and land trucks) arriving at ports *nothing would happen!* So the actions of vessels are those of actively announcing their arrivals at and departures from ports, and participating, more passively, in the unloading and loading of containers. 153

Action [A]: `calc_next_port`**:**

183 Vessels can calculate, `calc_next_port`, the unique identifier, `mcc_i`, of that ports' monitoring & control center.

184 We do not further define the pre- and post-conditions of the `calc_next_port` action.

value
183 calc_next_port: VI×VS_Mereo×VS_Stat → vir_CSA×VSΣ → MCCI×VSΣ
183 calc_next_port(vs_i,vs_mereo,vs_stat)(vir_csa,vσ) ia (mcc_i,vsσ')
184 **pre**: $\mathcal{P}_{calc-next-port}$(vs$\sigma$,v_mereo,...)
184 **post**: $\mathcal{Q}_{calc-next-port}$(vs$\sigma$,v_mereo,...)(mcc_i,vs$\sigma'$)

154

Vessel Action [B]: `calc_ves_mcc_msg`:

185 Vessels can calculate, `calc_ves_info`, the vessel information, vs_info:VS_Info, to be handed to the next ports' command center.

186 This information is combined with the vessel identifier and its virtual CSA,

187 We leave undefined the pre- and post-conditions over vessel states and vessel information.

type
185 VS_Info
186 VS_MCC_MSG :: VI×vir_CSA×VS_Info
value
185 calc_ves_mcc_msg: VI×VMereo×VStat → vir_CSA×VSΣ → VS_MCC_MSG×VSΣ
185 calc_ves_info(vs_i,vs_mereo,vs_stat)(vir_csa,vsσ) **as** (vs_mcc_msg,vsσ')
187   **pre**: $\mathcal{P}_{calc\_ves\_mcc\_msg}$(vs_i,vs_mereo,vs_stat)(vir_csa,vsσ)
187   **post**: $\mathcal{Q}_{calc\_ves\_mcc\_msg}$(vs_i,vs_mereo,vs_stat)(vir_csa,vσ)(vs_mcc_msg,vsσ')

6.5.4 Land Truck Actions 155

Land trucks can initiate the following actions vis-a-vis a targeted terminal port command center: announce, to a terminal command center, its arrival with a container; announce, to a terminal command center, its readiness to haul a container. Land trucks furthermore interacts with stack cranes – as so directed by terminal command
156 centers.

Land Truck Action [R]: `calc_truck_delivery`:

188 Land trucks, upon approaching, from an outside, terminal ports, calculate

189 the identifier of the next port's command center. We do not define the

190 pre- and

191 post conditions of this calculation.

value
188 calc_truck_delivery: CI × CONΣ → MCCI
189 calc_truck_delivery(ci,conσ) **as** mcci
190   **pre**: $\mathcal{P}_{calc\_truck\_deliv}$(ci,con$\sigma$)
191   **post**: $\mathcal{Q}_{calc\_truck\_deliv}$(ci,con$\sigma$)(mcci)

157

Land Truck Action [T]: `calc_truck_avail`:

192 Land trucks, when free, i.e., available for a next haul, calculate

193 the identifier of a suitable port's command center.

We do not define the

194 pre- and

195 post conditions of this calculation.

value

192 calc_truck_avail: LTI $\times$ LTΣ $\rightarrow$ MCCI
193 calc_truck_avail(lti,ltσ) **as** mcci
194 **pre**: $\mathcal{P}_{calc\_truck\_avail}$(lti,lt$\sigma$)
195 **post**: $\mathcal{Q}_{calc\_truck\_avail}$(lti,lt$\sigma$)(mcci)

6.6 Events 158

We refer to [13, Sect. 7.1.6 and 7.3.2]. Events occur to all entities. For reasons [urely of presentation we separate events into active part initiation events and active part completion events. Active part initiation events are those events that signal the initiation of actions. (Let **[Θ]** designate an action, then **[Θ′]** designates the completion of that action.) Active part completion events are those events that signal the completion of actions. We do not show the lower case [**d, f, g, h, i, j, k, l, m, n, o**] in Fig. 3.

6.6.1 Active Part Initiation Events 159

Vessels:

196 [α_{vessel}] approaching terminal port;

197 [**A**] informing the command center, mcc, of a terminal port, of arrival;

198 [**B**] receiving from an mcc directions as to quay berth positions;

199 [**C**] receiving from an mcc, for each container to be unloaded or loaded, directions as to these unloads and ladings – and these actual unloads/ladings;

200 [**X**] receiving from an mcc directions of completion of stowage (no more unloads/loads);

201 [**Y**] informing the mcc of its departure from terminal port; or

202 [ω_{vessel}] leaving a terminal port.

160 **Land Trucks:**

203 [α_{land_truck}] approaching a terminal port;

204 [**W**] informing its mcc of its arrival;

205 [**V**] being directed, by an mcc, as to the stack (crane) of destination;

206 [**S**] the unloading, to a stack crane, of a container;

207 [**T**] the loading of a container from a stack crane;

208 [**R**] informing its mcc of its departure; or

209 [ω_{land_truck}] leaving a terminal port.

161 **Containers:** the transfers from

210 [**D**] vessel to quay crane;

211 [**d**] quay crane to vessel;

212 [**G**] quay crane to quay truck;

213 [**g**] quay truck to quay crane;

214 [**J**] quay truck to stack crane;

215 [**j**] stack crane to quay truck;

216 [**M**] stack crane to stack;

217 [**m**] stack to stack crane;

218 [**P**] stack crane to land truck; or from

219 [**p**] land truck to stack crane.

Quay Cranes: being informed, by the command center, mcc, of a container to be

220 [**E**] picked-up from a vessel;

221 [**e**] set-down on a vessel;

222 [**F**] set-down on a quay truck; or

223 [**f**] picked-up from a quay truck.

Quay Trucks: being informed, by the command center, mcc, of a container to be

224 [**H**] loaded from a quay crane;

225 [**h**] picked-up by a quay crane;

226 [**I**] picked-up by a stack crane; or

227 [**i**] loaded from a stack crane.

[Terminal] Stack Cranes: being informed, by the command center, mcc, of a container to be

228 [**K**] picked-up from a quay truck;

229 [**k**] loaded on to a quay truck;

230 [**L**] picked-up from a stack;

231 [**l**] loaded on to a stack;

232 [**O**] picked-up from a land truck; or

233 [**o**] loaded on to a land truck.

[Terminal Bay] Stacks: being informed, by the command center, mcc, of a container to be

234 [**N**] set-down, of a container, from a stack crane; or

235 [**n**] picked-up, of a container, by a stack crane.

These events, in most cases, prompt interaction with the terminal command center.

6.6.2 Active Part Completion Events: 165

We do not show, in Fig. 3, the **c', e', h', o', q', t'** events.

236 **[C']**

237 **[E']** 162 163 164

238 **[H']**

239 **[O']**

240 **[Q']**

241 **[T']**

6.7 Channels 166

We refer to [13, Sect. 7.2], and we refer to Sect. and to Fig. 2 on Page 122.

6.7.1 Channel Declarations

There are channels between terminal port monitoring & control command center (mcci) and that command centers and that terminal port's

242 all the **containers** (ci), that might visit the terminal port; ch_mcc_con[mcci,ci][12];

243 **vessels** (vi) that might visit that port, ch_mcc[mcci,vi][13];

244 **quay cranes** (qci) of that port, ch_mcc[mcci,qci][14];

245 **quay trucks** (qti) of that port , ch_mcc[mcci,qti][15];

246 **stack cranes** (sci) of that port, ch_mcc[mcci,sci][16];

247 **stacks [bays]** (stki) of that port, ch_mcc[mcci,stki][17]; and

248 **land trucks** (lti) of, in principle, any port, ch_mcc[mcci,lti][18].

249 We shall define the concrete types of messages communicated by these channels subsequently (Sect.).

167

channel
242 $\{$ch_mcc_con[mcci,ci]|mcci:MCCI,ci:CI•mcci$\in mcc_uis \wedge$ci$\in c_uis\}$:MCC_Con_Cmd
243-248 $\{$ch_mcc[mcci,ui]|mcci:MCCI,ui:(VI|QCI|QTI|SCI|STKI|LTI) • mcci$\in mcc_uis \wedge$ui$\in uis\}$:MCC_Msg
type
249 MCC_Con_Msg, MCC_Msg

6.7.2 Channel Messages 168

We present a careful analysis description, for the channels declared above, of the rather rich variety of messages communicated over channels. All messages "goes to" (a few) or "comes from" (the rest) the command center. Messages from quay cranes, quay trucks, stack cranes, and land trucks – directed at the command center – are all
169 in response to the *events* of their being loaded or unloaded.

[12] cf. Item 98 on Page 101
[13] cf. Item 76 on Page 87
[14] cf. Item 81 on Page 90
[15] cf. Item 83 on Page 91
[16] cf. Item 87 on Page 93
[17] cf. Item 94 on Page 97
[18] cf. Item 96 on Page 98

A,B,X,Y,C′: Vessel Messages

250 There are a number **command center – vessel** and vice-versa messages:

a. **A**: Vessels announce their (forthcoming) arrival to the next destination terminal by sending such information, Ves_Arriv, to its monitoring & control (also referred to as command) center, that enables it to handle those vessels' berthing, unloading and loading (of container stowage).[19]

b. **B**: The terminal command center informs such arriving vessels of their quay segment positions, QS_Pos.

c. **X**: The terminal command center informs vessels of completion of stowage handling, Stow_Compl.

d. **Y**: Vessels inform the terminal of their departure, Ves_Depart. 170

type
250 MCC_Cmd == Ves_Arriv|Quay_Pos|Stow_Compl|Ves_Depart|...
250a. **A**: Ves_Arriv :: VI × vir_CSA
250b. **B**: Quay_Pos :: QSId × QP^+
250c. **X**: Stow_Compl :: MCCI × VI
250d. **Y**: Ves_Depart :: MCCI × VI

171

C,D,E,E′: Vessel/Container/Quay Crane Messages

251 The terminal command center, at a time it so decides, "triggers" the simultaneous transitions, **C,D,E**: VtoQC, of

a. **C**: unloading (loading) from (to) a vessel stack position of a container (surrogate), UnLoad, Load),

b. **D**: notifying the physical, i.e., the actual container that it is being unload (loaded), C_VtoQC (C_QCtoV), and

c. **E**: loading (unloading) the container (surrogate) onto (from) a quay crane, VtoQC (QCtoV).

252 **C′,E′**: The vessel and the quay crane, in response to their being unloaded, respectively loaded with a container "moves" that load, from its top vessel bay/row/stack position to the quay crane and notifies the terminal command center of the completion of that move, VQC_Compl. 172

[19] What exactly that information is, i.e., any more concrete type model of Ves_Info cannot be given at this early stage in our development of *what a terminal is*.

type
250 MCC_Cmd == ... | V_UnLoad | V_Load | C_VtoQC | C_QCtoV | VQC_Compl
251a. V−UnLoad, V_Load :: BRS × CI
251b. C_VtoQC, C_QCtoV :: QCI
251c. VtoQC, QCtoV :: CI
252 VQC_Compl :: V_UnLoad | V_Load

173

F,G,H,H′: Quay Crane/Container/Quay Truck Messages

253 The terminal command center, at a time it so decides “triggers” the simultaneous transitions, **F,G,H**: QCtoQT, of

a. **F**: the removal of the container from the quay crane,

b. **G**: the notification of the physical container that it is now being transferred to a quay truck, and

c. **H**: the loading of that container to a quay truck.

d. **H′**: The quay truck, in response to it being loaded notifies the terminal command center of the completion of that move.

type
250 MCC_Cmd == ... | QCtoQT | ...
253 QCtoQT == F | G | H | QCtoQTCompl
253a. F :: ...
253b. G :: QTI
253c. H :: CI
253d. QCtoQTCompl :: ...

174

I,J,K,K′: Quay Truck/Container/Stack Crane Messages

254 The terminal command center, at a time it so decides “triggers” the simultaneous transitions, **I,J,K**: QTtoSC, of

a. **I**: the removal of a container from a quay truck,

b. **J**: the notification of the physical container that it is now being transferred to a stack crane, and

c. **K**: the loading of that container to a stack crane.

255 **K′**: The stack crane, in response to it being loaded notifies the terminal command center of the completion of that move.

type
254 MCC_Cmd = ... | QCtoSC | ...
254 QCtoSC == I | J | K | QCQTCompl
254a. I :: ...
254b. J :: CI
254c. K :: CI × QCI
255 QCSCCompl :: ...

175

L,M,N,N′: Stack Crane/Container/Stack Messages

256 The terminal command center, at a time it so decides "triggers" the simultaneous transitions, **L,M,N**: SCtoStack, of

a. **L**: the unloading of the container from a stack crane;

b. **M**: the notification of the physical container that it is now being transferred to a stack, and

c. **N**: the loading of that container to a stack.

257 **N′**: The stack, in response to it being loaded, notifies the terminal command center of the completion of that move.

type
256 MCC_Cmd = ... | SCtoStack | ...
256 SCtoStack == L | M | N | SCStkCompl
256a. L :: ...
256b. M :: CI
256c. N :: CI × BRS
257 SCStkCompl :: ...

176

O,P,Q,Q′: Land Truck/Container/Stack Crane Messages

258 The terminal command center, at a time it so decides "triggers" the simultaneous transitions, **O,P,Q**: LTtoSC, of

a. **Q**: the unloading of the container from a land truck to a stack crane;

b. **P**: the notification of the physical container that it is now being transferred to a stack crane, and

c. **O**: the loading of that container to a stack crane.

d. **O′**: The stack crane, in response to it being loaded, notifies the terminal command center of the completion of that move.[20]

177

type
258 MCC_Cmd = ... | LTtoSC | ...
258 LTtoSC == LTtoSCQ | LTtoSCP | LTtoSCO | LTtoSCCompl
258a. LTtoSCQ :: ...
258b. LTtoSCP :: CI
258c. LTtoSCO :: CI × SCI
258d. LTtoSCCompl :: ...

178

R,W: Land Truck Messages

259 a. **R**: Land trucks, when approaching a terminal port, informs that port of its offering to deliver a container to stowage.

b. **S**: Land trucks, when approaching a terminal port, informs that port of its offering to accept a container from stowage.

c. **T**: Land trucks, at a terminal, are informed by the terminal of the bay/row/stack
179 position at which to deliver a named container.

d. **U**: Land trucks, at a terminal, are informed by the terminal of the bay/row/stack position from which to accept a named container.

e. **Q**: Land trucks, at a termial, are informed by the terminal of the stack crane at which to unload a named container.

f. **V**: Land trucks, at a termial, inform the terminal of their departure.

180

type
259 MCC_Cmd = ... | LTCmd | ...
259 LTCmd == LTR | LTW | LTS | LTT | LTV
259b. LTR == LT_del | LT_pup
259b. LT_del :: CI
259b. LT_pup :: CI
259f. LTW :: ...

[20]The **O′** event is “the same” as the **K′** event.

6.8 Behaviours 181

We refer to [13, Sects. 7.1.7, 7.3.3-4-5, and 7.4].

To every part of the domain we associate a behaviour. Parts are in space: there are the manifest parts, and there are the notion of their corresponding behaviours. Behaviours are in space and time. We model behaviours as processes defined in RSL$^+$. We cannot see these processes. We can, however, define their effects. 182

Parts may move in space: vessels, cranes, trucks and containers certainly do move in space; processes have no notion of spatial location. So we must "fake" the movements of movable parts. We do so as follows: We associate with containers the programmable attribute of location, as outlined in Items 123– 123g. on Page 116. We omit, for this model, the more explicit modelling of vessels, cranes and trucks but refer to their physical mereologies. 183

In the model of endurants, cf. Page 55, we modelled vessel and terminal container stowage areas as physically embodying containers, and we could move containers: push and pop them onto, respectively from bay stacks. This model must now, with containers being processes, be changed. The stacks, STACK, of container stowage areas, CAS, now embody unique container identifiers! We rename these stacks into cistack:CiSTACK

6.8.1 Terminal Command Center 184

The terminal command center is at the core of activities of a terminal port. We refer to the figure on Page 122. “Reading” that figure left-to-right illustrates the movements of containers from **[C-D-E]** vessels to quay cranes, **[F-G-H]** quay cranes to quay trucks, **[I-J-K]** quay trucks to stack cranes, **[L-M-N]** stack cranes to stacks, and from **[O-P-**
185 **Q]** land truck to stack cranes. A similar “reading” of that figure from right-to-left would illustrate the movements of containers from **[q-p-o]** stack cranes to land trucks; **[n-m-l]** stacks to stack cranes; **[k-j-i]** stack cranes to quay trucks; **[h-g-f]** quay trucks to quay cranes; and from **[e-d-c]** quay cranes to vessels. We have not show the **[c-d-**
186 **e-f-g-h-i-j-k-l-m-n-o-p-q]** labels, but their points should be obvious (!).

The Command Center Behaviour: We distinguish between the command center behaviour offering to *monitor* primarily vessels and land trucks, secondarily cranes, quay cranes and stacks, and offering to *control* vessels, cranes, trucks and containers.

187

260 The signature of the command center behaviour is a triple of the command center identifier, the conceptual command center mereology and the static command center attributes (i.e., the topological description of the terminal); the programmable command center attributes (i.e., the command center state); and the input/output channels for the command center.

The command center behaviour non-deterministically (externallY) chooses between

261 either monitoring inputs from

262 or controlling (i.e., outputs to)

188 vessels, cranes, trucks, stacks and containers.

```
value
260  command_center:
260     mcci:MCCI×(vis,qcis,qtis,scis,bis,ltis,cis):MCC_Mer×MCC_Stat
260     → MCCΣ →
260        in,out { ch_mcc[ mcci,ui ]n
260                  | mcci:MCCI,ui:(VI|QCI|QTI|SCI|BI|LTI)
260                    • ui∈vis∪qcis∪qtis∪scis∪bis∪ltis }
260        out { ch_mcc_con[ mcci,ci ] | ci:CI•ci ∈ cis }  Unit
260  command_center(mcci,(vis,qcis,scis,bis,ltis,cis),mcc_stat)(mccσ) ≡
261     monitoring(mcci,(vis,qcis,scis,bis,ltis,cis),mcc_stat)(mccσ)
260   []
262     control(mcci,(vis,qcis,scis,bis,ltis,cis),mcc_stat)(mccσ)
```

189

The Command Center Monitor Behaviours: The command center monitors the behaviours of vessels, cranes and trucks: **[A,Y′,C′,E′,F′,H′,I′,K′,L′,N′,O′,Q′]**. The input message thus received is typed:

```
type
    VCT_Info = ...
```

That information is used by the command center to update its state:

```
value
    update_MCCΣ: VCT_Infor → MCCΣ → MCCΣ
```

190

The definition of monitoring is simple.

263 The signature of the monitoring behaviour is the same as the command center behaviour.

264 The monitor non-deterministically externally ([]) offers to accept any input, vct_info, message from any vessel, any land truck and from local terminal port quay trucks and cranes.

265 That input, vct_info, enters the update of the command center state, from mccσ to mccσ'.

266 Whereupon the monitoring behaviour resumes being the command center behaviour with an updated state.

191

```
value
263  monitoring: mcci:MCCI × mis:MCC_Mereo × MCC_Stat
263      → MCCΣ
263      → in,out {chan_mcc[mcci,i] | i ∈ mis} Unit
263  monitoring(mcci,mis,mcc_stat)(mccσ) ≡
264      let vct_info = [] { chan_mcc[mcci,i] ? | i ∈ mis } in
265      let mccσ′ = update_MCCΣ((vct_info,ui))(mccσ) in
266      command_center(mcci,mis,mcc_stat)(mccσ′) end end
```

192

The Command Center Control Behaviours:

267 The command center control behaviour has the same signature as the command center behaviour (formula Items 260).

268 In each iteration of the command center behaviour in which it chooses the control alternative it calculates[21] a next [output] transaction. This calculation is at the very core of the overall terminal port. We shall have more to say about this in Sect. on Page 248.

193 Items, 269a.–269j. represent 10 alternative transactions.

269 They are "selected" by the **case** clause (Item 269).

So for each of these 10 alternatives there the command center offers a communication. For the **[CDE, FGH, IJK, LMN, OPQ, opq]** cases there is the same triple of concurrently synchronised events. For the **[B,T,X]** clauses there are only a single synchronisation effort. The command center events communicates:

a. **[B]** the quay positions to arriving vessels,

   the transfer of containers

b. **[CDE]** from vessel stacks to quay cranes,

194 c. **[FGH]** quay cranes to quay trucks,

d. **[IJK]** quay trucks to stack cranes,

e. **[LMN]** stack cranes to stacks,

f. **[OPQ]** stack cranes to land trucks, and

g. **[opq]** land trucks to stack cranes.

   We also illustrate

h. **[T]** the bays to which a land truck is to deliver, or fetch a container, and

i. **[X]** the "signing off" of a vessel by the command center.

j. For the case that the next transaction cannot be determined [at any given point in time] there is nothing to act upon.

195

270 After any of these alternatives the command center control behaviour resumes being the command center behaviour with the state updated from the next transaction calculation.

[21] For `calc_nxt_transaction` see Items 127 – 138 on Page 138

```
value
267   control: mcci:MCCI×(vis,qcis,qtis,scis,bis,ltis,cis):MCC_Mer×MCC_Stat → MCCΣ →
260       in,out {ch_mcc[ mcci,ui ]|mcci:MCCI,ui:(VI|QCI|QTI|SCI|BI|LTI)•ui∈vis∪qcis∪qtis∪scis∪bis∪ltis}
260       out { ch_mcc_con[ mcci,ci ] | ci:CI•ci ∈ cis }  Unit
267   control(mcci,(vis,qcis,scis,bis,ltis,cis),mcc_stat)(mccσ) ≡
268     let (mcc_trans,mccσ′) = calc_nxt_transaction(mcci,mcc_mereo,mcc_stat)(mccσ) in
269     case mcc_trans of
269a. [B]     (vi,mkQuayPos(qp)) → ch_mcc[ mcci,vi ] ! mkQuayPos(qp),
269b. [CDE]  (vi,mkVSQC_Xfer(brs,ci,qci)) →
269b. [C]             ch_mcc[ mcci,vi ] ! mkVes_UnLoad(ci,brs)
269b. [D]             ‖ ch_mcc_con[ mcci,ci ] ! mkNewPos(mcci,qci)
269b. [E]             ‖ ch_mcc[ mcci,qci ] ! mkQC_Load(ci),
269c. [FGH]  mkQCQT_Xfer(qci,ci,qti) →
269c. [F]             ch_mcc[ mcci,qci ] ! mkQC_UnLoad(ci)
269c. [G]             ‖ ch_mcc_con[ mcci,ci ] ! mkNewPos(mcci,qti)
269c. [H]             ‖ ch_mcc[ mcci,qti ] ! mkQT_Load(ci),
269d. [IJK]    mkQTSC_Xfer(qti,ci,sci) →
269d. [I]             ch_mcc[ mcci,qci ] ! mkQT_UnLoad(ci)
269d. [J]             ‖ ch_mcc_con[ mcci,ci ] ! mkNewPos(mcci,sci)
269d. [K]             ‖ ch_mcc[ mcci,qti ] ! mkSC_Load(ci),
269e. [LMN]  mkSCSTK_Xfer(brs,ci,sci,sti) →
269e. [L]             ch_mcc[ mcci,sci ] ! mkSC_UnLoad(ci)
269e. [M]             ‖ ch_mcc_con[ mcci,ci ] ! mkNewPos(mcci,brs)
269e. [N]             ‖ ch_mcc[ mcci,stki ] ! mkSTK_Load(ci,brs),
269f. [OPQ]  mkSCLT_Xfer(sci,ci,lti) →
269f. [O]             ch_mcc[ mcci,sci ] !  mkSC_UnLoad(ci)
269f. [P]             ‖ ch_mcc_con[ mcci,ci ] ! mkNewPos(mcci,lti)
269f. [Q]             ‖ ch_mcc[ mcci,lti ] ! mkLT_Load(ci),
269g. [opq]    mkLTSC_Xfer(sci,ci,lti) →
269g. [o]             ch_mcc[ mcci,sci ] ! mkSC_Load(ci)
269g. [p]             ‖ ch_mcc_con[ mcci,ci ] ! mkNewPos(mcci,cti)
269g. [q]             ‖ ch_mcc[ mcci,lti ] ! mkLT_UnLoad(ci),
269h. [T]      mkLT_Dept(lti) → ch_mcc[ mcci,lti ] ! ″bye-bye″,
269i.  [X]      mkV_Dept(vi) → ch_mcc[ mcci,vi ] ! ″bye-bye″,
269j.           _ → skip
269     end ; command_center(mcci,(vis,qcis,scis,bis,ltis,cis),mcc_stat)(mccσ′) end
```


6.8.2 Vessels

196

271 The signature of the vessel behaviour is a triple of the vessel identifier, the conceptual vessel mereology, the static vessel attributes, and the programmable vessel attributes. [We presently leave static attributes unspecified: ...]

Nondeterministically externally, [], the vessel decides between

272 **[A]** either approaching a port,

273 **[]** or [subsequently] arriving at that port,

or [subsequently] participating in the

274 **[]** unloading and

275 **[]** loading of containers of containers,

276 **[]** or [finally] departing from that port.

197

```
value
270 vessel: vi:VI×mccis:V_Mereo×V_Sta_Attrs → (V_Pos×vir_CSA×VΣ)
270     → in,out {ch_mcc[ mcci,vi ]|mcci:MCCI•mcci∈mccis}  Unit
270 vessel(vi,mccis,...)(vpos,vir_csa,vσ) ≡
271         port_approach(vi,mccis,...)(vpos,vir_csa,vσ)
272     [] port_arrival(vi,mccis,...)(vpos,vir_csa,vσ)
273     [] unload_container(vi,mccis,...)(vpos,vir_csa,vσ)
274     [] load_container(vi,mccis,...)(vpos,vir_csa,vσ)
275     [] port_departure(vi,mccis,...)(vpos,vir_csa,vσ)
```

198

Port Approach

277 The signature of port_approach behaviour is identical to that of vessel behaviour.

278 On approaching any port the vessel calculates the identity of that port's command center.

279 Then, with an updated state, it calculates the information to be handed over to the designated terminal –

280 **[A]** which is then communicated from the vessel to the command center;

281 whereupon the vessel resumes being a vessel albeit with a doubly updated state.

```
value
276 port_approach: vi:VI×mccis:V_Mereo×V_Sta_Attrs→(V_Pos×vir_CSA×VΣ)
276     → in,out {ch_mcc[ mcci,vi ]|mcci:MCCI•mcci∈mccis}  Unit
276 port_approach(vi,mccis,...)(vpos,vir_csa,vσ) ≡
277     let (mcci,vσ′) = calc_next_port(vσ) in
278     let (mkVInfo(vi,vir_csa,vs_info),vσ″) = calc_vessel_info(vσ″) in
279     ch_mcc[ mcci,vi ] ! mkVS_Info(vi,vir_csa,vs_info) ;
280     vessel(vi,mccis,...)(vpos,vir_csa,vσ″) end end
```

200

Port Arrival

282 The signature of port_arrival behaviour is identical to that of vessel behaviour.

283 **[B]** Non-deterministically externally the vessel offers to accept a terminal port quay position from any terminal port's command center.

284 The vessel state is updated accordingly.

285 Whereupon the vessel resumes being a vessel albeit with a state updated with awareness of its quay position.

286 The vessel is ready to receive such quay position from any terminal port.

201

```
value
281 port_arrival: vi:VI×mccis:V_Mereo×V_Sta_Attrs → (V_Pos×vir_CSA×VΣ)
281     → in,out {ch_mcc[ mcci,vi ]|mcci:MCCI•mcci∈mccis}  Unit
281 port_arrival(vi,mccis,...)(vpos,vir_csa,vσ) ≡
282     { let mkQuayPos(qs,qpl) = ch_mcc[ mcci,vi ] ? in
283       let vσ′ = upd_ves_state(mcci,mkQuay_Pos(qs,qpl))(vσ) in
284       vessel(vi,mccis,...)(mkInPort(mcci,mkQuayPos(qs,qpl)),vir_csa,vσ′) end end
285     | mcci:MCCI•mcci∈mccis }
```

202

Unloading of Containers

287 The signature of port_arrival behaviour is identical to that of vessel behaviour.

288 **[C]** The vessel offers to accept, ch_mcc_v[mcci,vi] ?, a directive from the command center of the terminal port at which it is berthed, to unload, mkUnload((bi,ri,si),ci). a container, identified by ci, at some container stowage area location ((bi,ri,si)).

289 The vessel unloads the container – identified by ci′.

290 If the unloaded container identifier is different from the expected **chaos** erupts!

291 The vessel state, vσ', is updated accordingly.

292 **[C’]** “Some time has elapsed since the unload directive, modelling” the completion, from the point of view of the vessel, of the unload operation –

293 whereupon the command center is informed of this completion (**[’]**).

294 The vessel resumes being the vessel in a state reflecting the unload.

203

```
value
286 unload_container: vi:VI × mccis:V_Mereo × V_Sta_Attrs → (V_Pos × iCSA × VΣ) →
286     in,out {ch_mcc[ mcci,vi ]|mcci:MCCI•mcci∈mccis}  Unit
286 unload_container(vi,mccis,...)(vpos,vir_csa,vσ) ≡
287     let mkV_UnLoad((bi,ri,si),ci) = ch_mcc[ mcci,vi ] ? in
288     let (ci′,vir_csa′) = unload_CI((bi,ri,si),vir_csa) in
289     if ci′ ≠ ci then chaos end;
290     let vσ″ = unload_update_VΣ((bi,ri,si),ci)(vir_csa) in
291     wait sometime ;
300     ch_mcc[ mcci,vi ] ! mkCompl(mkV_UnLoad((bi,ri,si),ci)) ;
301     vessel(vi,mccis,...)(vpos,vir_csa′,vσ″) end end end
```

204

Loading of Containers

295 The signature of load_container behaviour is identical to that of vessel behaviour.

296 **[c]** The vessel offers to accept, ch_mcc_v[mcci,vi] ?, a directive from the command center of the terminal port at which it is berthed, to load, mkLoad((bi,ri,si),ci). a container, identified by ci, at some container stowage area location ((bi,ri,si)).

297 The vessel (in co-operation with a quay crane, see later) then unloads the container – identified by ci.

298 The vessel state, vσ', is updated accordingly.

299 **[c’]** “Some time has elapsed since the unload directive, modelling” the completion, from the point of view of the vessel, of the unload operation – whereupon the command center is informed of this completion (**[’]**).

300 and the vessels resumes being the vessel in a state reflecting the load.

```
value
294 load_container: vi:VI×mccis:V_Mereo×V_Sta_Attrs → (V_Pos×vir_CSA×VΣ)
294     → in,out {ch_mcc[ mcci,vi ]|mcci:MCCI•mcci∈mccis}  Unit
294 load_container(vi,mccis,...)(vpos,vir_csa,vσ) ≡
295     let mkV_Load((bi,ri,si),ci) = ch_mcc[ mcci,vi ] ? in
296     let vir_csa′ = load_CI(vir_csa,(bi,ri,si),ci) in
297     let vσ′ = load_update_VΣ((bi,ri,si),ci) in
298     ch_mcc[ mcci,vi ] ! mkCompl(mkV_Load((bi,ri,si),ci)) ;
299     vessel(vi,mccis,...)(vpos,vir_csa′,vσ′) end end end
```

206

Port Departure

301 The signature of port_departure behaviour is identical to that of vessel behaviour.

302 **[Y]** At some time some command center informs a vessel that *stowage*, i.e., the unloading and loading of containers has ended.

303 Vessels update their states accordingly.

304 **[Y']** Vessels respond by informing the command center of their departure.

305 Whereupon vessels resume being vessels.

```
value
300 port_departure: vi:VI×mccis:V_Mereo×V_Sta_Attrs → (V_Pos×vir_CSA×VΣ)
300     → in,out {ch_mcc[ mcci,vi ]|mcci:MCCI•mcci∈mccis}  Unit
300 port_departure(vi,mccis,v_sta)(vpos,vir_csa,vσ) ≡
301     let mkStow_Compl(mcci,vi) ⌷ { ch_mcc[ mcci,vi ] ? | mcci:MCCI•mcci∈mccis } in
302     let vσ′ = update_vessel_state(mkVes_Dept(mcci,vi))(vσ) in
303     ch_mcc[ mcci,vi ] ! mkVes_Dept(mcci,vi) ;
304     vessel(vi,mccis,v_sta)(vpos,vir_csa,vσ′) end end
```


207

•••

The next three behaviours: quay_crane, quay_truck and stack_crane, are very similar. One substitutes, line-by-line, command center/quay crane, quay crane/quay truck, quay truck/stack crane et cetera!

6.8.3 Quay Cranes 208

306 The signature of the quay_crane behaviour is a triple of the quay crane identifier, the conceptual quay crane mereology, the static quay crane attributes, the programmable quay crane attributes – and the 'command center'/'quay crane' channel.

307 The quay crane offers, non-deterministially externally, to

308 either, **[E]**, accept a directive of a '*container transfer from vessel to quay crane*'.

  a. The quay crane then resumes being a quay crane now holding (a surrogate of) the transferred container.

209

309 or, **[F]** accept a directive of a transfer '*container from quay crane to quay truck*'.

  a. The quay crane then resumes being a quay crane now holding (a surrogate of) the transferred container.

value

```
305   quay_crane: qci:QCI × mcci:QC_Mer × QC_Sta → (QCHold×QCPos)
305         → ch_mcc[ mcci,qci ]  Unit
305   quay_crane(qci,mcci,qc_sta)(qchold,qcpos) ≡
307       let mkVSQC(ci) = ch_mcc[ mcci,qci ] ? in
307a.       quay_crane(qci,mcci,qc_sta)(mkCon(ci),qcpos) end
306     []
308       let mkQCVS(ci) = ch_mcc[ mcci,qci ] ? in
308a.       quay_crane(qci,mcci,qc_sta)(mkCon(ci),qcpos) end
```

6.8.4 Quay Trucks 210

310 The signature of the quay_truck behaviour is a triple of the quay truck identifier, the conceptual quay truck mereology, the static quay truck attributes, the programmable quay truck attributes – and the 'command center'/'quay truck' channel.

311 The quay truck offers, non-deterministially externally, to

312 either, **[H]**, accept a directive of a *'container transfer from quay crane to quay truck'*.

   a. The quay truck then resumes being a quay truck now holding (a surrogate of) the transferred container. 211

313 or, **[I]**, accept a directive of a *'container transfer from quay truck to quay crane'*.

   a. The quay truck then resumes being a quay truck now holding (a surrogate of) the transferred container.

```
value
309   quay_truck: qti:QTI × mcci:QC_Mer × QT_Sta → (QTHold×QTPos)
309       → ch_mcc[ mcci,qci ]  Unit
309   quay_truck(qti,mcci,qt_sta)(qthold,qtpos) ≡
311       let mkQCQT(ci) = ch_mcc[ mcci,qti ] ? in
311a.       quay_crane(qti,mcci,qc_sta)(mkCon(ci),qcpos) end
310     ⌷
312       let mkQTQC(ci) = ch_mcc[ mcci,qti ] ? in
312a.       quay_crane(qti,mcci,qc_sta)(mkCon(ci),qcpos) end
```


6.8.5 Stack Crane 212

314 The signature of the stack_crane behaviour is a triple of the stack crane stack crane identifier, the conceptual mereology, the static stack crane attributes, the programmable stack crane attributes – and the 'command center'/'stack crane' channel.

315 The stack crane offers, non-deterministially externally, to

316 either, **[K]**, accept a directive of a *'container transfer from quay truck to stack crane'*.

   a. The stack crane then resumes being a stack crane now holding (a surrogate of) the transferred container.

213

317 or, **[L]**, accept a directive of a *'container transfer from stack crane to quay truck'*.

   a. The stack crane then resumes being a stack crane now holding (a surrogate of) the transferred container.

value

```
313   stack_crane: sci:SCI × mcci:SC_Mer × SC_Sta → (SCHold×SCPos)
313       → ch_mcc[ mcci,sci ]  Unit
313   stack_crane(sci,mcci,sc_sta)(schold,scpos) ≡
315       let mkQTSC(ci) = ch_mcc[ mcci,sci ] ? in
315a.       stack_crane(sci,mcci,sc_sta)(mkCon(ci),scpos) end
314     []
316       let mkSCQT(ci) = ch_mcc[ mcci,sci ] ? in
316a.       stack_crane(sci,mcci,sc_sta)(mkCon(ci),scpos) end
```

6.8.6 Stacks 214

The stack behaviour is very much like the unload_container container behaviour of the vessel, cf. Items 286 – 290 on Page 211.

318 The signature of the stack behaviour is a triple of the stack, i.e. terminal port bay identifier, the conceptual bay mereology, the static bay attributes, the programmable bay attributes and the 'command center'/'stack' channel. 215

319 The stack offers, **[N]**, to accept directive of a *'container transfer from stack crane to stack'*.

   a. The stack behaviour loads the container, identified by ci′, to the bay/row/stack top, identified by (bi,ri,si).
   b. If the unloaded container identifier is different from the expected **chaos** erupts!
   c. The stack state, bay′, is updated accordingly.
   d. **[N']** "Some time has elapsed since the load directive, modelling" the completion, from the point of view of the vessel, of the unload operation –
   e. whereupon the command center is informed of this completion (**[']**).
   f. The stack then resumes being a stack now holding (a surrogate of) the transferred container.

216

value

```
317   stack: tbi:TBI×mcci:STK_Mer×Stk_Sta_Attrs → (iCSA × Stk_Dir) →
317      in,out {ch_mcc[ mcci,vi ]|mcci:MCCI•mcci∈mccis}  Unit
317   stack(tbi,mcci,stk_sta)(bay,dir) ≡
318      let mkUnload((bi,ri,si),ci) = ch_mcc[ mcci,tbi ] ? in
318a.    let (ci′,bay′) = unload_CI((bi,ri,si),bay) in
318b.    if ci′ ≠ ci then chaos end ;
318c.    let bay″ = unload_update_BAY((bi,ri,si),ci)(bay′) in
318d.    wait sometime ;
318e.    ch_mcc[ mcci,tbi ] ! mkCompl(mkUnload((bi,ri,si),ci)) ;
318f.    stack(tbi,mcci,stk_sta)(bay″,dir) end end end
```


6.8.7 Land Trucks 217

320 The signature of the land_truck behaviour is a triple of the land truck identifier, the conceptual land truck mereology and the static land truck attributes, and the programmable land truck attributes.

321 **R**

a. The land truck calculates the identifier of the next port's command center

b. and communicates with this center as to its intent to deliver a container identified by ci,

c. whereupon the land truck resumes being that.

218

322 **T**

a. The command center informs the land truck of the bay ('stack'), brs, at which to deliver the container,

b. whereupon the land truck resumes being that.

323 **Q**

a. The command center informs the land truck of the delivery of a container from a stack crane,

b. ...,

c. whereupon the land truck resumes being that.

219

324 **V**

a. The land truck informs the command center of its intent to depart from the terminal port,

b. whereupon the land truck resumes by eaving the terminal port.

220

```
value
319  land_truck:
319
319  land_truck(lti,lt_mer,lt_sta)(lt_pos,lt_hold) ≡
320        next_port(lti,lt_mer,lt_sta)(lt_pos,lt_hold)
321     [] stack_location(lti,lt_mer,lt_sta)(lt_pos,lt_hold)
322     [] stack_crane_to_land_truck(lti,lt_mer,lt_sta)(lt_pos,lt_hold)
323     [] land_truck_departure(lti,lt_mer,lt_sta)(lt_pos,lt_hold)
```

```
value
320    next_port(lti,lt_mer,lt_sta)(...,mkHold(ci,cσ)) ≡
320a.      let mcci = calc_truck_delivery(ci,cσ) in
320b.      ch_mcc[ mcci,lti ] ! mkDlvr(ci,cσ) ;
320c.      land_truck(lti,lt_mer,lt_sta)(...,...) end ???
```

```
value
321    stack_location(lti,lt_mer,lt_sta)(...,mkHold(ci,cσ)) ≡
321a.      let  mkLT_Pos(mcci,brs) = { ch_mcc[ mcci,lti ] ? | mcci:MCCI • mcci ∈ mcc_uis }
321b.      land_truck(lti,lt_mer,lt_sta)(...,lt_hold) end ???
```

```
value
322    stack_crane_to_land_truck(lti,lt_mer,lt_sta)(lt_pos,lt_hold) ≡
322a.
322b.
```

```
value
323    land_truck_departure(lti,lt_mer,lt_sta)(...,...) ???
323a.      ch_mcc[ mcci,lti ] ! mkDept(lti) ;
323b.      land_truck(lti,lt_mer,lt_sta)(...,...)  ???
```


6.8.8 Containers 221

In RSL, as with all formal specification languages one cannot "move" values. So we model containers of vessels and of terminal port stacks as separate behaviours and replace their "values", C in vessel and terminal port stacks by their unique identifications, CI.

325 The signature of the container behaviour is simple: the container identifier, its
222 mereology, its static values, its position and state[22], and its input channels.

326 **[D,G,J,M,P]** The container is here simplified to just, at any moment, accepting a new position from any terminal ports command center;

327 whereupon the container resumes being that with that new position.

value

```
324 container: ci:CI×mcci_uis:C_Mer×C_Stat → (CPos×CΣ)
324     → in { ch_mcc_con[ mcci,ci ]
324             | mcci:MCCI • mcci∈mcci_uis }  Unit
324 container(ci,mcci_uis,...)(pos,sσ) ≡
325     let mkNewPos(p) = { ch_mcc_con[ mcci,ci ] ?
325                          | mcci:MCCI•mcci∈mcci_uis } in
326     container(ci,mcci_uis,...)(mkNewPos(p),sσ) end
```

[22] As for state: I need to update the container attribute section, Sect. on Page 115 to reflect a state (for example: the component contents of a container)

6.9 Initial System 223

6.9.1 The Distributed System

We remind ourselves that the container line industry includes a set of vessels, a set of land trucks, a set of containers and a set of terminal ports. We rely on the states expounded in Sect. 's Items 50 on Page 78 – 54 on Page 78. 224

328 The signature of τ_initial_system is that of a function from an endurant endurant container line industry to its perdurant behaviour, i.e., **Unit**.

This behaviour is expressed as

329 the distributed composition of all vessel behaviours in parallel with

330 the distributed composition of all land truck behaviours in parallel with

331 the distributed composition of all container behaviours in parallel with

332 the distributed composition of all terminal port behaviours. 225

value
328 τ_initial_system: CLI $\rightarrow$ **Unit**
328 τ_initial_system(cli) $\equiv$
328 $\quad \|\{$ τ_vessel(v) $|$ v:V $\bullet$ v $\in vs$ $\}$
329 $\| \; \|\{$ τ_land_truck(lt) $|$ lt:LT $\bullet$ lt $\in lts$ $\}$
330 $\| \; \|\{$ τ_container(c) $|$ c:CON $\bullet$ c $\in cs$ $\}$
331 $\| \; \|\{$ τ_terminal_port(tp) $|$ tp:TP $\bullet$ tp $\in tps$ $\}$

6.9.2 Initial Vessels 226

333 The signature of the i_vessel `transalation function` is simple: a τ`ranslator` from endurant vessel `parts` v to perdurant vessel `behaviours`, i.e., **Unit**.

334 The transcendental deduction then consists of obtaining the proper arguments for the vessel behaviour –

335 and invoking that behaviour.

value
332 τ_vessel: V $\rightarrow$ **Unit**
332 τ_vessel(v) $\equiv$
333 **let** v_ui = uid_V(v), v_mer = mereo_V(v),
333 v_sta = attr_V_Sta(v), v_pos = attr_V_Pos(v),
333 v_csa = attr_iCSA(v), vσ = attr_VΣ(v) **in**
334 vessel(v_ui,v_mer,v_sta)(v_pos,v_csa,vσ) **end**

6.9.3 Initial Land Trucks 227

Similarly:

```
τ_land_truck: LT → Unit
τ_land_truck(lt) ≡
      let lt_ui = uid_LT(lt), lt_mer = mereo_LT(lt),
          lt_sta = attr_LT_Sta(lt), lt_pos = attr_LT_Pos(lt),
          lt_hold = attr_LT_Hold(v), ltσ = attr_LTΣ(lt) in
      vessel(lt_ui,lt_mer,lt_sta)(lt_pos,lt_hold,ltσ) end
```

6.9.4 Initial Containers 228

Similarly:

```
τ_container: CON → Unit
τ_container(con) ≡
      let c_ui = uid_CON(con), c_mer = mereo_CON(con),
          c_sta = attr_C_Sta(con), c_pos = attr_C_Pos(con),
          cσ = attr_CONΣ(lt) in
      container(c_ui,c_mer,c_sta)(c_pos,cσ) end
```

6.9.5 Initial Terminal Ports 229

Terminal ports consists of a set of quay cranes, a set of quay trucks a set of stack
230 cranes, and a set of stacks. They translate accordingly:

```
τ_terminal_port: TP → Unit
τ_terminal_port(tp) ≡
      let qcs = obs_QCs(obs_QCS(tp)),
            qts = obs_QTs(obs_QTS(tp)),
            scs = obs_SCs(obs_SCS(tp)),
            stks = obs_STKs(obs_STKS(tp)) in
   ‖ { τ_quay_crane(qc) | qc:QC • qc ∈ qcs } ‖
   ‖ { τ_quay_truck(qt) | qt:QT • qt ∈ qts } ‖
   ‖ { τ_stack_crane(sc) | sc:SC • sc ∈ scs } ‖
   ‖ { τ_stack(stk) | stk:STK • stk ∈ stks } end
```

6.9.6 Initial Quay Cranes 231

τ_quay_crane: QC → **Unit**
τ_(qc) ≡
  let qc_ui = uid_QC(qc), qc_mer = mereo_QC(qc),
    qc_sta = attr_QC_Sta(qc), qc_pos = attr_QC_Pos(qc),
    qcσ = attr_QCΣ(qc) **in**
  quay_crane(qc_ui,qc_mer,qc_sta)(qc_pos,qcσ) **end**

6.9.7 Initial Quay Trucks 232

τ_quay_truck: QT → **Unit**
τ_quay_truck(qt) ≡
  let qt_ui = uid_QT(qt), qt_mer = mereo_QT(qt),
    qt_sta = attr_QT_Sta(qt), qt_pos = attr_QT_Pos(qt),
    qtσ = attr_QTΣ(qt) **in**
  quay_truck(qt_ui,qt_mer,qt_sta)(qt_pos,qtσ) **end**

6.9.8 Initial Stack Cranes 233

τ_stack_crane: SC → **Unit**
τ_stack_crane(sc) ≡
  let sc_ui = uid_SC(sc), sc_mer = mereo_SC(sc),
    sc_sta = attr_SC_Sta(sc), sc_pos = attr_SC_Pos(sc),
    scσ = attr_SCΣ(sc) **in**
  container(sc_ui,sc_mer,sc_sta)(sc_pos,scσ) **end**

6.9.9 Initial Stacks 234

τ_stack: STK → **Unit**
τ_stack(stk) ≡
  let stk_ui = uid_STK(stk), stk_mer = mereo_STK(stk),
    stk_sta = attr_STK_Sta(stk),
    stkσ = attr_STKΣ(stk) **in**
  stack(stk_ui,stk_mer,stk_sta)(stkσ) **end**

7 Conclusion 235

TO BE WRITTEN

7.1 An Interpreation of the Behavioural Description 236

TO BE WRITTEN

7.2 What Has Been Done 237

TO BE WRITTEN

7.3 What To Do Next 238

TO BE WRITTEN

7.4 Acknowledgements 239

This report was begun when I was first invited to lecture, for three weeks in November 2018, at ECNU[23], Shanghai, China. For this and for my actual stay at ECNU, I gratefully acknowledge Profs. He JiFeng, Zhu HuiBiao, Wang XiaoLing and Min Zhang. I chose at the time of the invitation to lead the course students through a major, nontrivial example. Since Shanghai is also one of the major container shipping ports of the world, and since the Danish company Maersk, through its subsidiary, APMTerminals, operates a major container terminal port, see Sect. , I decided on the subject fo this experimental report. I gratefully acknowledge the support the ECNU course received from APMTerminals, through its staff, Messrs Henry Bai and Niels Roed.

[23]ECNU: East China Normal University

8 Bibliography 240

8.1 References

[1] Dines Bjørner. A Container Line Industry Domain. Techn. report, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, Denmark, June 2007. [24].

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

[3] International Labour Organisation. Portworker Development Programme: PDP Units. Enumerate PDP units. , April 2002.

[4] K.V. Ramani. An interactive simulation model for the logistics planning of container operations in seaports. *SIMULATION*, 66(5):291–300, 1996.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[24] http://www2.imm.dtu.dk/~db/container-paper.pdf

[11] 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.

[12] Bram Borgman, Eelco van Asperen, and Rommert Dekker. Online rules for container stacking. *OR Spectrum*, 32:687–716, 19 March 2010.

[13] Dines Bjørner. A Domain Analysis & Description Method – Principles, Techniques and Modelling Languages. Paper submitted for publication, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, Denmark, May 16 2018. [25].

[25] http://www.imm.dtu.dk/~dibj/2018/tosem/Bjorner-TOSEM.pdf

9 Summary of Internal Types 241

9.1 Unique Identifiers

32 pp.70 QCI
33 pp.70 QTI
34 pp.70 SCI
35 pp.70 TBI
36 pp.70 LTI
37 pp.70 MCCI
38 pp.70 CI
39 pp.70 BI
40 pp.70 RI
41 pp.71 SI

9.2 Mereologies

value
76 pp.87 V_Mer = MCCI-**set**
81 pp.90 QC_Mer = MCCI
83 pp.91 QT_Mer = MCCI
87 pp.93 SC_Mer = MCCI
94 pp.97 T_BAY_Mer = MCCI
96 pp.98 LT_Mer = MCCI-**set**
97 pp.99 MCC_Mer =
97 pp.99 VI-**set**×QCI-**set**×QTI-**set**×
97 pp.99 SCI-**set**×TBI-**set**×LTI-**set**×CI-**set**
98 pp.101 C_Mer = MCCI-**set**

9.3 Attributes

type Vessels:
99 pp.105 V_Pos == AtSea | InPort
99a. pp.105 Longitude, Latitude
99a. pp.105 AtSea :: Longitude × Latitude
99b. pp.105 InPort :: MCCI × QPOS
100 pp.105 VΣ

type Quay Cranes:
102 pp.107 QCHold == mkNil('nil')
102 pp.107 | mkCon(ci:CI)
103 pp.107 QCPos = QSId × QP

type Quay Trucks:
105 pp.108 QTHold == mkNil('nil')
105 pp.108 | mkCon(ci:CI)

type Stack Cranes:
107 pp.109 SCHold == mkNil('nil')
107 pp.109 | mkCon(ci:CI)
108 pp.109 SCPos = BI

type Terminal [i.e., Bay] Stacks
110 pp.110 BOm = BI $\overrightarrow{m}$ **Nat**, BOI = BI*
111 pp.110 ROm = RI $\overrightarrow{m}$ **Nat**, ROI = RI*
112 pp.110 SOm = SI $\overrightarrow{m}$ **Nat**, SOI = SI*

type Land Trucks:
117 pp.113 LTHold == mkNil('nil')
117 pp.113 | mkCon(ci:CI)
118 pp.113 LTΣ

type Command Centers:
120 pp.114 TopLogDescr
121 pp.114 MCCΣDescr

type Containers:
122 pp.115 BoL
123 pp.116 CPos == onV | onQC | onQT
123 pp.116 | onSC | onStk | onLT | Idle

concrete **type**s of onV, onQC, onQT, onSC, onStk, onLT and Idle
123a. pp.116 onV :: VI × BRSP × VPos
123a. pp.116 VPos == AtSea | InTer
123a. pp.116 AtSea :: Geo
123a. pp.116 InTer :: QPSid × QP^+
123b. pp.116 onQC :: MCCI × QCI
123c. pp.116 onQT :: MCCI × QTI
123d. pp.116 onSC :: MCCI × SCI
123e. pp.116 onStk :: MCCI × BRSP
123f. pp.116 onLT :: MCCI × LTI
123g. pp.116 Idle :: {|"idle"|}