Transport
A Domain Description

Dines Bjørner

Technical University of Denmark
Fredsvej 11, DK-2840 Holte
bjorner@gmail.com – www.dtu.dk/~db

June 12, 2025: 16:47

A First "Final" Draft

Transport
A Domain Description

Dines Bjørner

June 12, 2025: 16:47

- **Warning:** Many formulas need being type checked, etc., etc. !

- I began writing this document February 22, 2025.
- In my 87th year !
- I think about it and write "on" it, every day 7/7.
- Often twice a a day, 1-2 hours.
- More than that – and I get tired.

- **A first draft June 10, 2025.**
  - Now, as from June 10, 2025, I will
    - Go through the entire document.
    - Check that all index references to formulas are "correct".
    - Etcetera, et cetera !
  - I will release this and forthcoming versions to the Internet:
    https://www.imm.dtu.dk/~dibj/2025/transport/main.pdf

- Section A.3 (pages 133–141) presents an index to all formulas !
- You may find, in the version of this report, that You are now perusing, that there are some "mysterious" vertical [i.e., line] spacing.

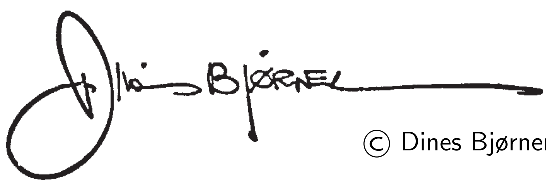
  They are there in order for the index entries to refer to pages (π) where both the (item ι) enumerated narrative and formal entries are on the same page !

- Pls. see Sect. 26.4 on page 126.
- Pls. refer to Appendix Chapter B on page 143 for a summary of main formal entities.

Prelude

This is an engineering report.

We analyze and describe a conceptual domain of *transport* in all its forms: passenger and goods, road, rail, water (navigable rivers and lakes as well as the open sea), and air. From the basis of an abstract notion of *graphs* with *labeled nodes* and *edges,* we define a notion of *routes of graphs:* sequences of node and edge labels. Nodes are then interpreted a street intersections, bus stops, railway stations, harbours and airports and edges as links between neighbouring nodes: street segments, bus routes, rail lines, sea lanes, and air routes. And from there it goes! We expand the treatment to cover customers, [sending and receiving] merchandises, conveyor companies and logistics companies.



June 12, 2025: 16:47

Contents

Chapter 1

Introduction

The Triptych Dogma

In order to *specify software*,
we must understand its requirements.
In order to *prescribe requirements*,
we must understand the *domain*.
So we must *study, analyze* and *describe* domains.

This is one of a series, [10, 16, 15, 14, 8], of domain studies of such infrastructure components as government, public utilities, banking, transport, insurance, health care, etc. The current, this 'Introduction' chapter is common to these study reports.

1.1 On A Notion of 'Infrastructure'

Central to our effort of studying "man-made" domains is the notion of *infrastructure*[1]. The *infrastructure* can be characterized as follows: *the basic physical and organizational structures and facilities (e.g. buildings, roads, power supplies) needed for the operation of a society or enterprise*, "`the social and economic infrastructure of a country`". We interpret the "for example, e.g.," to include, some already mentioned above: government structure: legislative, executive & judicial units, transport: roads, navigable rivers and lakes, the open sea, banking, educational system, health care, utilities: water, electricity, telecommunications (e.g. the `Internet) gas, `, etc.,[2] Also: Winston Churchill is quoted to have said in the House of Commons: *"The young Labour speaker we have just listened to wants clearly impressing his constituency with the fact that he went to Eton and Oxford since he now uses such modern terms as 'infrastructure' ".*

1.2 Domain Models

We rely on [12, 9, 7, 6, 4]. They provide a scientific foundation for modelling domains in the style of this report.

1.2.1 Some Characterizations

Domain: By a *domain* we shall understand a *rationally describable* segment of a *manifest*[3], *discrete dynamics* fragment of a *human assisted* reality: the world that we daily observe – in which we work and act, a reality made significant by human-created entities. The domain embody *endurants* and *perdurants*.

Endurants: By *endurants* we mean those quantities of domains that we can observe (see and touch), in *space*, as "complete" entities at no matter which point in *time* – "material" entities that persists, endures – capable of enduring adversity, severity, or hardship [Merriam Webster].

[1] https://en.wikipedia.org/wiki/Infrastructure

[2] According to the World Bank, 'infrastructure' is an umbrella term for many activities referred to as 'social overhead capital' by some development economists, and encompasses activities that share technical and economic features (such as economies of scale and spill-overs from users to non-users). We take a more technical view, and see infrastructures as concerned with supporting other systems or activities. Software for infrastructures is likely to be distributed and concerned in particular with supporting communication of data, people and/or materials. Hence issues of openness, timeliness, security, lack of corruption and resilience are often important.

[3] The term 'manifest' is used in order to distinguish between these kinds of domains and those of computing and data communication: compilers, operating systems, database systems, the Internet, etc.

Perdurants: By *perdurants* we mean those quantities of domains for which only a fragment exists, in *space*, if we look at or touch them at any given snapshot in *time* [Merriam Webster].

Domain Description: By a *domain description* we shall here mean a syntactic entity, both narrative and formal, describing the domain. That is, a domain description is a structured text, such as shown in Sects. 2–18 (pages 7–116).

Domain Model: By a *domain model* we shall here mean the mathematical meaning, the semantics as denoted the domain description.

1.2.2 Purpose of Domain Models

The *Triptych* dogma (above) expresses a relation of domain models to software. But domain models serve a wider role. Mathematical models of, say, physics, are primarily constructed to record our understanding of some aspects of the world – only secondarily to serve as a basis for engineering work. So it is with manifest models of infra structure components such banking, insurance, health care, transport, etc. In this, and a series of papers, [15, 14], we shall therefore present the result of infra structure studies. We have, over the years, developed many domain models: [3].

1.2.3 Domain Science & Engineering

A series of publications [4, 6, 7, 9, 13] reflects scientific insight into and an engineering methodology for analyzing and describing manifest domains.

1.3 A Dichotomy

1.3.1 An Outline

As citizens we navigate, daily, in a *God-given* and a *Man-made world.* The God-given world can be characterized, i.e., "domain described", as having natural science properties. The laws that these natural science properties obey are the same – all over the universe ! The Man-made world can be characterized, i.e., "domain described", as having infrastructure components[4]. The "laws" that these properties obey are not necessarily quite the same around our planet !

1.3.2 The Dichotomy

For our society to work, we are being educated (in primary, secondary, tertiary schools, colleges and at universities). We are taught to to read, write and [verbally] express ourselves, recon and do mathematics, languages, history and the sciences: physics (mechanics, electricity, chemistry, biology, botany's, zoology, geology, geography, ...), but we are not taught about most of the infrastructure structures[5]. That is the dichotomy.

1.4 The Dichotomy Resolved

So there it is:

- first *study* a or several domains;
- then *analyze, describe* and *publish* infrastructure domains;
- subsequently *prepare educational texts* "over" these;
- finally introduce *'an infrastructures'* school course.

1.5 A [Planned] Series of Infrastructure Domain Models

So this *domain science & engineering* paper – on banking – is one such infrastructure domain description. In all we are and would like to work on these infrastructure domains:

---

[4]state, regional and local government: executive, legislative and judicial, banking, insurance, health care (hospitals, clinics, rehabilitation, family physicians, pharmacies, ...), passenger and goods transport (road, rail, sea and air), manufacturing and sales, publishing (newspapers, radio, TV, books, journals, ...), shops (stores, ...),

[5]See footnote 4.

- **Transport**[6] [16]
- **Banking**[7] [10]
- **Insurance**[8] [15]
- **Health Care**[9] [14]
- etc.

A report on *double-entry bookkeeping* [8] relates strongly to most of these infra-structure component domains[10].

[6]https://www.imm.dtu.dk/ dibj/2025/infra/main.pdf
[7]https://www.imm.dtu.dk/ dibj/2025/infra/banking.pdf
[8]https://www.imm.dtu.dk/ dibj/2025/infra/insurance.pdf
[9]https://www.imm.dtu.dk/ dibj/2025/infra/healthcare.pdf
[10]`http://www.imm.dtu.dk/ dibj/2023/doubleentry/dblentrybook.pdf`

Part I

A SIMPLE BEGINNING

Chapter 2

Kinds of Transports

Contents

2.1 Informal Outline

The transport we have in mind consists of a common transport net, in the following modelled as a graph of uniquely labeled, bi-directed edges and likewise labeled nodes. The transport net is [“intentional pull”] complemented, cf. Sect. 6 on page 25, by a set of conveyors.

Edges, nodes and conveyors are “of kind”: **"road"**, **"rail"**, **"sea"**, and **"air"**; these are literal values[11]. A conveyor is of one kind. Conveyors of kind **"road"** include taxis, buses, trucks and the like. Conveyors of kind **"rail"** include passenger trains, freight trains, etc. Conveyors of kind **"sea"** include sail boats, river and canal barges, fishing vessels, line and ramp freighters, passenger liners, etc. Conveyors of kind **"air"** include helicopters, freight and passenger planes. An edge is of one kind. Edges of kind **"road"** are called automobile roads. Edges of kind **"rail"**, **"sea"** and **"air"** are called rail tracks, sea lanes and air lanes. A node may be of one or more kinds. Nodes of kind **"road"** are called street point (street crossings, street ends, bus stops). Nodes of kind **"rail""** are called train stations. Nodes of kind **"sea"** are called harbours. Nodes of kind **"air"** are called airports.

2.2 Narrative & Formalization

1. There are four kinds of transportation: **"road, rail, sea"** and **"air"**.

type

1. Kind = **"road"**|**"rail"**|**"sea"**|**"air"**

People are not conveyors, so they are no “of a kind” ! People may be merchandises.

• • •

That is: transport, in this report, is all about moving goods – here referred to as merchandises – around. By what/whichever means: on roads, rails, sea and/or by air – possibly combining two or more of these: moving from (road) trucks to (air) freight and/or by (sea) freighter– whether line or tramps[12], or in some other order ! We omit considering people as conveyors.

We divide the first formal presentation into five [further] segments: *Overall Transport Endurants, Graph Endurants, Conveyor Endurants, Intentional Pull* and *Perdurants.*

By an overall traffic domain we mean that of a graph[13] and a conveyor[14] sub-domain.

A relation between graphs and conveyors is expressed in the intentional pull section.

[11] – as are **true** and **false**

[12] a boat or ship engaged in the tramp trade is one which does not have a fixed schedule, itinerary nor published ports of call, and trades on the so-called spot market [https://en.wikipedia.org/wiki/Tramp_trade.

[13] https://en.wikipedia.org/wiki/Graph_(discrete_mathematics)

[14] Conveyor: anything that conveys, transports or delivers. [Words are a conveyor of meaning] [https://en.wiktionary.org/wiki/conveyor]

The “co-operation” of graphs and conveyors is expressed in the perdurant section.

By a graph we mean a set of nodes and edges: nodes are then interpreted as road intersections (hubs); train stations; river, canal and sea harbours; and airports. A node may be one or more of these. Edges are accordingly interpreted as either street (or road) links, irail tracks, sailing or air routes. An edges can be only one of these. Hence there may be many edges between any two [neighbouring] nodes.

By conveyors we mean *buses*, *trains*, *boats*, *ships*, and *aircraft*.

The presentation follows the domain analysis & description ontology of Fig. 2.1.

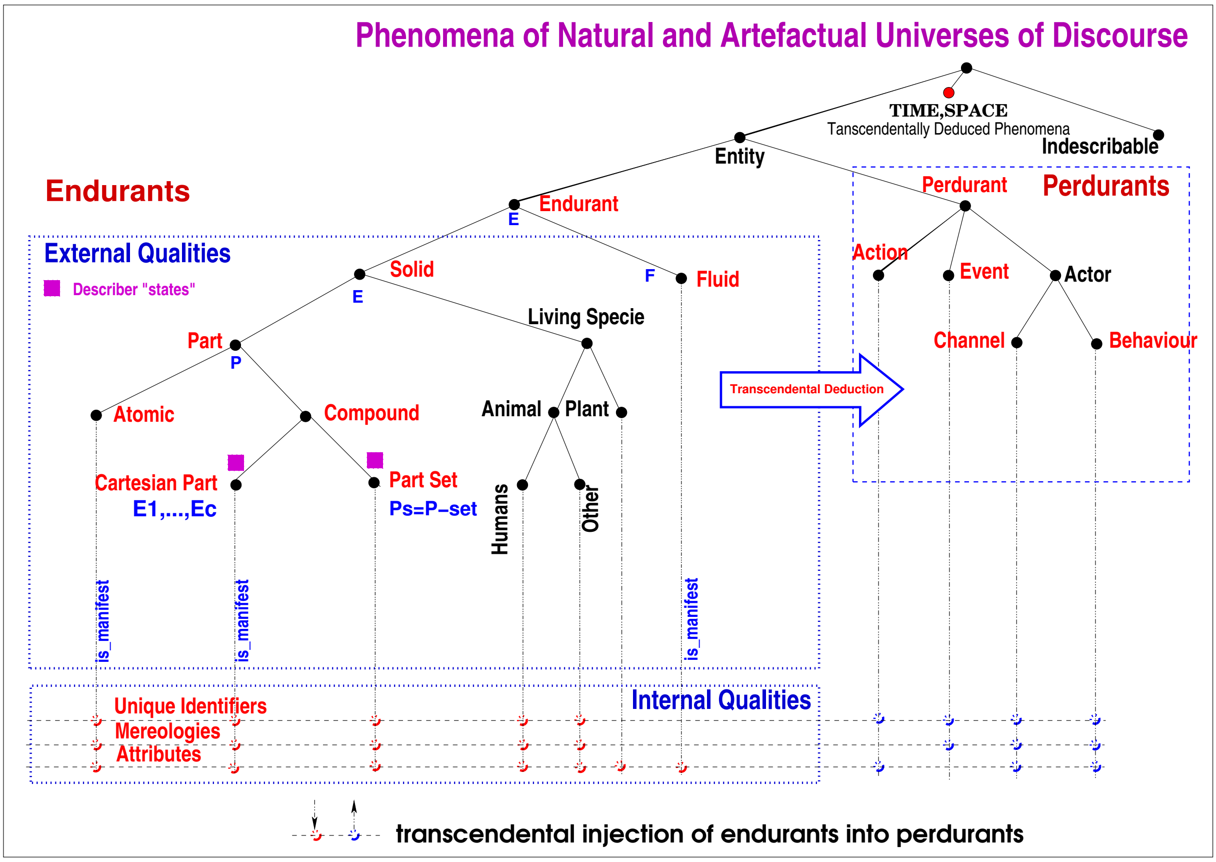


Figure 2.1: Domain Analysis Ontology

Chapter 3

Overall “Single-Mode” Transport Endurants

This early section introduces the, perhaps two most central classes of endurants: `transport nets`, in the abstracted form of `graphs`, and `conveyor aggregates`. Conveyor aggregates embody conveyors. Conveyors “move along” nets, and nets serve to [*intentional pull*[15]] “carry” conveyor traffic.

3.1 Endurant Sorts & Observers

2. There is the domain of transport.
3. From transport endurants we can observe transport nets, i.e., graphs.
4. And from transport endurants we can observe a conveyor aggregate – embodying conveyors.

type
2. T
3. G
4. CA
value
3. **obs**_G: T→G
4. **obs**_CA: T→CA

3.1.1 An Endurant State Notion

We can speak of a transport state.

5. There is given a “global”[16] transport value, t. It contributes to a transport state.
6. From this transport value one can derive another transport state element: a global graph value, g.
7. And from this transport value one can derive another transport state element: a global conveyor aggregate value, ga.
8. We can postulate a transport state to consist of the three endurants: t, g, ca.

value
5. t:T
6. g:G = **obs**_G(t)
7. ca:CA = **obs**_CA(t)
5. $\sigma_t = \{t\} \cup \{g\} \cup \{ca\}$

[15]cf. Sect. 6 on page 25
[16]We shall be using this term: ‘global’ extensively. By double quoting it we intend to express that “global” values are values that can be referred to anywhere in the domain description. We emphasize their “globality” by use this kind of [mathematical] *font* !

3.2 Unique Identification

3.2.1 Unique Identifier Sorts & Observers

9. The transport endurant has a unique identifier.
10. So has the graph, and
11. the conveyor components.

type
9. TI
10. GI
11. CAI
value
9. **uid**_T: T → TI
10. **uid**_G: T → GI
11. **uid**_CA: T → CAI

3.2.2 A Unique Identifier State Notion

We an postulate a "global" transport state value, t.

12. From t we observe its unique identity.
13. Given t we can derive a "global" graph value g, hence its unique identity.
14. And a "global" conveyor aggregate value ca, hence its unique identity..
15. We can therefore postulate an "uppermost" endurant transport state to consist of the three endurants: ti, gi, cai.

value
12. ti:TI = **uid**_T(t)
13. gi:GI = **uid**_G(g)
14. cai:CAI = **uid**_CA(ca)
15. $\sigma_{t_{uis}} = \{ti\} \cup \{gi\} \cup \{cai\}$

3.2.3 Uniqueness

16. The three ["uppermost"] transport endurants are distinct: have distinct unique identifiers.

axiom [Uniqueness of Transport Identifiers]
16. **card** σ_t = **card** $\sigma_{t_{uis}}$ = 3

• • •

It seems that at least the overall transport endurant need not be a manifest one. Hence we leave out treatment of mereology and attributes of the transport endurant.

Chapter 4

Graphs: Transport Nets

In addition to describing the external and internal qualities of transport nets we introduce the concepts or *paths*, i.e., *routes*, through/across a transport net.

4.1 The Endurant Sorts and Observers

External qualities are the endurant sorts of graphs, node and edges aggregates and nodes and edges, their observers and endurant states.

17. From graphs one can observe an aggregate, i.e., a set, *ea:EA*, of edges –
18. From graphs one can observe an aggregate, i.e., a set, *na:NA*, of nodes –
19. From an aggregate of edges one can observe a set of edges.
20. From an aggregate of nodes one can observe a set of nodes.
21. Edges are considered atomic.
22. Nodes are considered atomic.
23. We can "lump" all endurants into a sort *parts*.

```
type
17.   EA
18.   NA
19.   ES = E-set
20.   NS = N-set
21.   E
22.   N
23.   P = G|EA|NA|ES|NA|N|E
value
17.   obs_EA: G → ES
18.   obs_NA: G → NS
19.   obs_ES: EA → ES
20.   obs_NS: NA → NS
```

A transport domain taxonomy is hinted at in Fig. 4.1 on the following page.

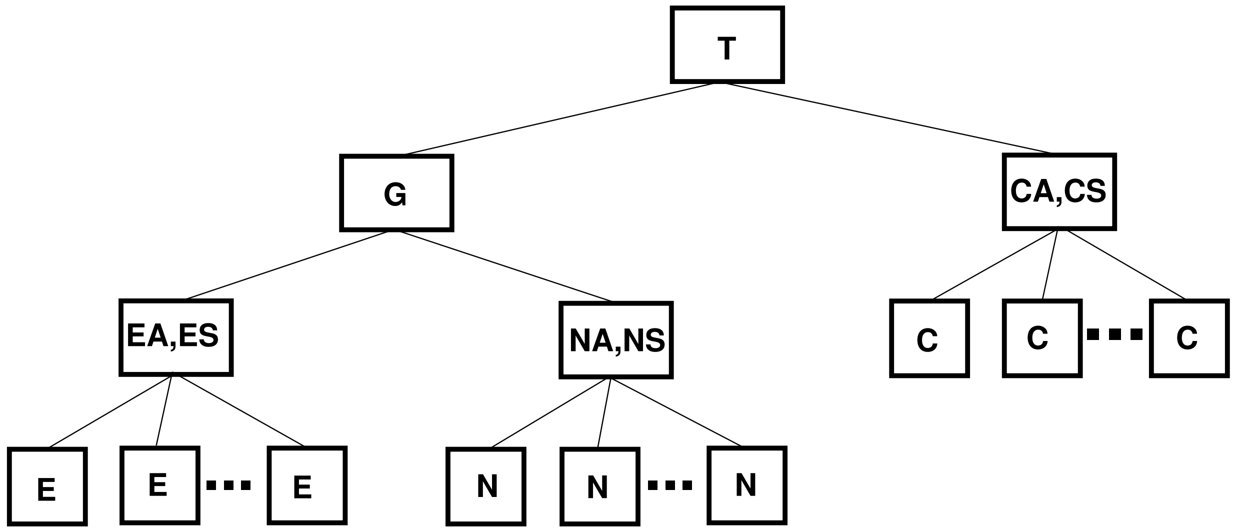


Figure 4.1: A Simplified Transport Domain Taxonomy: Transport Nets, **G**, and Conveyors, **C**

4.1.1 An Endurant State

24. Given the global graph value, there is therefore a "global" value of an edge aggregate.
25. Given the global graph value, there is therefore a "global" value of a node aggregate.
26. Given the global edge aggregate value, there is therefore a "global" node value of of the set of all edges.
27. Given the global graph value, there is therefore a "global" value of the set of all nodes.
28. The state of all graph endurants is therefore the set of all graph parts.

value
24. $ea = \textbf{obs}\_\texttt{EA}(g)$
25. $na = \textbf{obs}\_\texttt{NA}(g)$
26. $es = \textbf{obs}\_\texttt{ES}(g)$
27. $ns = \textbf{obs}\_\texttt{NS}(g)$
28. $\sigma_{ps}\texttt{:P-}\textbf{set} = \{g\}\cup\{ea\}\cup\{na\}\cup\texttt{es}\cup\texttt{ns}$

• • •

Internal qualities are fourfold: unique identification, mereology, attributes and intentional pull.

4.2 Unique Identifiers

Unique Identification has three facets: sort, observers and an axiom.

4.2.1 Unique Identifier Sorts and Observers

29. All parts have identification:
30. the graph,
31. the edge and node aggregates,
32. the sets of edges and nodes, and
33. each edge and node.
34. No two of these are the same, i.e., part identifiers are unique.

type
29. PI = GI|EAI|NAI|ESI|NSI|EI|NI
29. GI,EAI,NAI,ESI,NSI,EI,NI
value
30. **uid**_G: G→GI
31. **uid**_EA: EA→EAI, **uid**_NA: NA→NAI
32. **uid**_ES: ES→ESI, **uid**_NS: NS→NSI
33. **uid**_E: E→EI, **uid**_N: N→NI

4.2.2 A Unique Identifier State

35. There is a "global" unique graph identifier.
36. There are, correspondingly, "global" edge and node aggregate identifiers.
37. There are, correspondingly, "global" edge set and node set identifiers; and
38. set of edge identifiers and
39. set of node identifiers.
40. The unique identifier state is the union of all the unique identifiers.

value
35. gi = **uid**_G(g)
36. ea_{uis} = **uid**_EA(ea) , na_{uis} = **uid**_NA(na)
37. es_{uis} = **uid**_ES(ea) , ns_{uis} = **uid**_NS(na)
38. e_{uis} = {**uid**_E(e)|e:E•e∈es}
39. n_{uis} = {**uid**_N(n)|n:N•n∈ns}
40. σ_{uis}:PI-**set** = {**uid**_P(p)|p:P•p∈σ}
40. $\sigma_{uis} = \{gi\}\cup\{ea_{uis}\}\cup\{na_{uis}\}\cup\{es_{uis}\}\cup\{ns_{uis}\}\cup e_{uis}\cup n_{uis}$

4.2.3 Uniqueness

41. No two of these are the same, i.e., part identifiers are unique.

axiom [Uniqueness of Part Identification]
41. **card**σ=**card**σ_{uis}

4.3 Mereology

Mereology has three facets: types, observers and wellformedness.

4.3.1 Mereology Types and Wellformedness, I

42. The mereology of a node is a non-empty set of edge identifiers.
43. The mereology of an edge is a set of two node identifiers.

type
42. NM = EI-**set** **axiom** ∀ nm:NM • **card** nm>0
43. EM = NI-**set** **axiom** ∀ em:EM • **card** em=2

4.3.2 Mereology Observers

value
42. **mereo**_N: N → NM
43. **mereo**_E: E → EM

4.3.3 Mereology Wellformedness, II

44. The unique identifiers of a node must be those of the edges of the graph.
45. The unique identifiers of an edge must be those of the nodes of the graph.

axiom [Graph Mereology Wellformedness]
44. ∀ n:N•**mereo**_N(n)$\subseteq es_{uis}$
45. ∀ e:E•**mereo**_E(e)$\subseteq ns_{uis}$

4.4 Paths of a Graph

46. A path (of a graph) is a finite[17] sequence of one or more alternating node and edge identifiers such that
    (a) neighbouring edge identifiers are those of the mereology of the “in-between” node, and such that neighbouring node identifiers are/is those of the mereology of the “in-between” edge;
    (b) and node identifiers of a path are node identifiers of the graph,
    (c) and its neighbouring edge identifier(s) are in the mereology of the identified node;
    (d) and edge identifiers of a path are edge identifiers of the graph,
    (e) and its neighbouring node identifier(s) are/is in the mereology of the identified edge;
    (f) the kinds of the adjacent nodes and edges “fit”.
47. Given a node [an edge] identifier we can retrieve the identified node [edge].

```
type
46.  Path = (EI|NI)*
axiom [Wellformed Paths]
46.  ∀ path:Path •
46a.    ∀ {i,i+1}⊆inds path ⇒
46a.        ( (is_NI(path[i])∧is_EI(path[i+1])
46a.           ∨ is_EI(path[i])∧is_NI(path[i+1]))
46b.        ∧ (path[i]∈ns_uis⇒path[i+1]∈es_uis
46c.           ∧ uid_N(retr_node(path[i]))∈mereo_E(retr_node(path[i])))
46d.        ∧ (path[i]∈es_uis⇒path[i+1]∈ns_uis
46e.           ∧ uid_E(retr_edge(path[i])∈mereo_N(retr_edge(path[i]))))
46f.        ∧ kind(retr_unit(path[i]))∩kind(retr_unit(path[i+1]))≠{})
value
47.  retr_node: NI → N, retr_edge: EI → E, retr_unit: UI → U
47.  retr_node(ni) as n • n ∈ ns ∧ uid_(n)=ni
47.  retr_edge(ei) as e • e ∈ es ∧ uid_(e)=ei
47.  retr_unit(i) as u • ∈ ns∪es ∧ uid_U(u)=i
47.  uid_U(u) ≡ is_E(u)→uid_U(u),is_N(u)→uid_N(u)
```

The above **pre/post** condition allows for circular paths, i.e., possibly infinite paths that may contain the same node or edge identifier more than once.

We can define a function that given a graph calculates all its non-circular paths.

[17] We shall only consider finite paths. The `paths` function, Item 48 below, can easily be modified to yield also infinite length paths !

48. The paths[18] function takes a graph and yields a possibly infinite set of paths – satisfying the above wellformedness criterion.

    We define the paths function in two ways.

49. Either axiomatically

50. in terms of an **as** predicate, with the result being the "largest" such set all of whose paths satisfy the wellformedness criterion;

51. or inductively[19]:

    (a) **basis clause:** every singleton path of either node or edge identifiers of the graph form a path.
    (b) **inductive clause:** If `pi` and `pj` are finite, respectively possibly infinite paths of the "result", `ps`, such that
    (c) paths `pi`⌢⟨`ui`⟩ and ⟨uj⟩⌢pj are in `ps`, and
    (d) the resulting concatenated path is not circular, and
    (e) the mereology of the last element of `pi` identifies the first element of `pj`,
    (f) then their concatenation is a path in `ps`.
    (g) **extremal clause:** No path is an element of the desired set of paths unless it is obtained from the basis and the inductive clause by a finite number of uses.

```
value
48.  paths:  G → Path-infset
49.  paths(g) as ps
50.       such that:  ∀ p:ps satisfy the above wellformedness
51.  paths(g) ≡
51a.      let ps = {⟨ni⟩ | ni:NI ∈ ns_uis}∪{⟨ei⟩ | ei:EI ∈ es_uis}
51f.           ∪ { pi⌢⟨ui⟩⌢⟨uj⟩⌢pj | pi⌢⟨ui⟩:Path-set, ⟨uj⟩⌢pj:Path-infset •
51b.                    ∧ ({pi⌢⟨ui⟩,⟨uj⟩⌢pj}⊆ps
51c.                    ∧ (ui∼ ∈ elems pj ∧ uj∼ ∈ elems pi)
51e.                    ∧ (ui ∈ mereo_U(retr_unit(uj))
51e.                    ∧  uj ∈ mereo_U(retr_unit(ui))))} in
51g.      ps end
type
48.  U = E|N
```

Solution to the equation, lines 51a–51c, is "'obtained' by a smallest set fix-point reasoning.

52. Given a "global" graph, *g*, we can calculate a "similarly global" *paths* value:

```
value
52.  paths:Path-set = paths(g)
```

With the notion of paths of a graph one can now examine whether

- a graph is strongly connected, that is, whether any node or edge can be "reached" from any other node or edge; or
- a graph consists of two or more sub-graphs, i.e., there are no edges between nodes in two such sub-graphs;
- etc.

In the next section, i.e., Sect. 4.5.1, we shall now endow nodes and edges to reflect whether they are road intersections, railway stations, harbours, and road links, railway lines, or canal/river/sea- or air-routes, etc.

[18] **Alarm !** Check that this function indeed generates only finite length paths !
[19] https://www.cs.odu.edu/ toida/nerzic/content/recursive_def/more_ex_rec_def.html

53. We can formulate a *theorem:* for every graph we have that every path, p, in g, also contains its reverse path, rev(p) in g.

theorem: [All finite paths have finite reverse paths]

```
53.  ∀ g:G,p:Path•p ∈ paths(g) ⇒ rev_path(p) ∈ paths(g)
value
53.  rev_path:  P → P
53.  rev_path(p) ≡
53.      case p of
53.          ⟨⟩ → ⟨⟩,
53.          ⟨ui⟩ → ⟨ui⟩,
53.          ⟨ui⟩⌢p′⌢⟨uj⟩ → ⟨uj⟩⌢rev_path(p′)⌢⟨ui⟩
53.      end
```

We can define auxiliary functions, for example:

54. Given a kind we can select all the graph paths of that kind.

```
value
54.  path_kind:  Path → Kind → Path-set
54.  path_kind(p)(k) as pks
54.     • pks ⊆ paths ∧
54.         ∀ pk:Path•pk ∈ pks∧∀ elems pk•kind(retr_unit(pk))∩{k}≠{}
```

4.5 Attributes

With endurants now being endowed with, i.e., having attributes, graphs come to "look", more-and-more, as transport nets !

Attributes has three facets: types, observers and wellformedness.

4.5.1 Attribute Types & Observers

We introduce but just a few Graph Attributes.

55. From a node we can thus observe the "kind" of node: whether **"road crossing"**, train **"station"**, canal/river/sea boat/ship **"harbour"**, and/or **"airport"** – one or more ! [A static attribute]

Edge:

56. From an edge we can thus observe the "kind" of edge: whether it represents a street (segment between two neighbouring road crossings), or a rail track (between two neighbouring stations), or a sea route between two neighbouring (canal/river/sea) harbours or an aircraft route between two neighbouring airports.
57. From an edge we can we can observe its length[20]. [Static Attribute]
58. and the cost[21] of using the edge[22]. [Static Attribute]

```
type
55.   NodeKind = Kind-set  axiom ∀ nk:NodeKind • nk≠{}
56.   EdgeKind = Kind-set  axiom ∀ ek:EdgeKind • card ek=1
57.   LEN = Nat
58.   COST = Nat
value
55.   attr_NodeKind:  N → NodeKind
56.   attr_Edgekind:  E → EdgeKind
57.   attr_LEN: E → LEN
58.   attr_COST: E → COST
```

[20] LEN is here "formalized" in terms of **Nat**ural numbers. Whether such lengths stand for mm, cm, m, km, inches, feet, yard, mile or other we presently leave unspecified.

[21] COST is here "formalized" in terms of **Nat**ural numbers. Whether such costs stand for $, €, £, or other we presently leave unspecified.

[22] See [5]. The usual arithmetic operators apply: scaling between ... Check also [20].

59. Given a node or an edge we can observe its kinds.
60. Given a graph, and a “kind”, we can calculate all its paths of the same kind.
61. Given a finite route we can we can calculate its lengths
62. and costs.
63. We can also calculate the shortest route, possibly a set, of a graph,
64. and the least costly,[23]
65. etc.

```
value
59.  kind:  (E|N) → EdgeKind|NodeKind
59.  kind{en} ≡ is_E(en)→attr_Edgekind(en),is_N→attr_Edgekind(en)

60.  route_kind:  G → Kind → Path-set
60.  route_kind(g)(k) ≡
60.      { ⟨p[i]|i:Nat,p:P•p∈paths(p)∧1≤i≤len(p)∧k∈kind(p[i])⟩ }

61.  path_length:  P → LEN
61.  path_length(p) ≡
61.      case p of
61.          ⟨⟩ → 0
61.          ⟨ui⟩ → retr_path_length(ui),
61.          ⟨ui⟩^p′ → retr_length(ui)+path_path_length(p′)
61.      end
61.  retr_path_length:  UI → LEN
61.  retr_path_length(ui) ≡ (is_EI(ui)→attr_LEN(retr_edge(ui)),is_NI(ui)→0)

62.  path_cost:  P → LEN
62.  path_cost(p) ≡
62.      case p of
62.          ⟨⟩ → 0
62.          ⟨ui⟩ → retr_cost(ui),
62.          ⟨ui⟩^p′ → retr_path_cost(ui)+path_cost(p′)
62.      end

62.  retr_path_cost:  UI → COST
62.  retr_path_cost(ui) ≡ (is_EI(ui)→attr_COST(retr_edge(ui)),is_NI(ui)→0)

63.  shortest_route:  G → P-set
63.  shortest_route(g) ≡
63.      let ps = paths(g) in
63.      { p | p:P • retr_len(p) ∧ ∀ p′:P•p′∈ps ∧ retr_path_len(p)≤retr_path_len(p′) }
63.      end

65.  etc.
```

The “etc.” covers such auxiliary functions as shortest route of a given kind , least costly route of a given kind , etc. !

More Graph Attributes will be added [“later”].

[23] See William Cook’s Web page: https://www.math.uwaterloo.ca/tsp/index.html?mc_cid=a51d99f2aa&mc_eid=783b63461a and *Quanta Magazine*’s Fundamentals Computer Science Web page https://mail.google.com/mail/u/0/?ui=2#inbox/FMfcgz-QZTzdWzqtRWmVWkQrcNzzDrSnJ

4.5.2 Attribute Wellformedness

66. If a node is of some kind, then there must be at least one edge leading to/from it of the same kind.
67. If an edge is of some kind, then the nodes connected to it must also be of that [same] kind.
68. If a node is of kind other than `"car"`, then there there must be an edge "of" that node of kind `"car"`. [One must be able to drive to stations, harbours and airports by car, taxi, lorry (truck) or bus !]

axiom

```
66.
66.

67.
67.

68.
68.
```

Chapter 5

Conveyors, I

We remind the reader that conveyors are either for the **road:** cars, taxis, trucks, buses, etc.; or for the **rail:** trains, or for the **sea:** sailboats, barges, freighters, passenger liners, etc.; or for the **air:** helicopters and airplanes.

5.1 Conveyor Endurant Sorts & Observers

69. From a conveyor aggregate one can observe a finite set of conveyors.
70. A conveyor is either a
    - a road conveyor
      - car,
      - taxi,
      - bus,
      - truck, etc.,
    - or a rail conveyor
      - passenger train,
      - freight train, etc.,
    - or a water conveyor
      - sailboat,
      - barge,
      - fishing vessel,
      - freighter,
      - passenger liner, etc.,
    - or an airborne conveyor
      - civil aircraft,
      - freight plane, or
      - passenger aircraft, etc.
71. Conveyors are atomic parts.
72. Conveyors or "of kind".
73. Conveyor aggregates are uniquely identified.
74. Conveyors are uniquely identified.

```
type
69.  CS = C-set
70.  C = Road|Rail|Water|Air
70.  Road = ...
70.  Rail = ...
70.  Sea = ...
70.  Air = ...
73.  CAI
74.  CI
value
73.  uid_CA: CA → CAI
74.  uid_C: C → CI
```

5.2 Unique Identifiers

5.2.1 Unique Identifier State

75. The unique identifier of a conveyor aggregate contributes to the unique identifier state for the [entire] transport domain.

76. The unique identifiers of all conveyors contribute to the unique identifier state for the [entire] transport domain.

77. The overall unique identifier state, σ_{uis}, is therefore the union of all the unique identifiers of all parts of a transport domain.

value
75. cai:CAI = **uid**_CA(ca)
76. cis:CI-**set** = { **uid**_C(c) | c:C • c ∈ **obs**_CS(ca) }
77. $\sigma_{uis} = \sigma_p \cup \{cai\} \cup cis$

5.2.2 Uniqueness

78. All parts are uniquely identified.

axiom [All parts are uniquely identified]
78. **card** σ = **card** σ_{uis}

5.2.3 Conveyor Retrieval

79. From a conveyor identifier one can obtain, via cs, the conveyor of that identification.

value
79. retr_conveyor: CI → C
79. retr_conveyor(ci) ≡ ι c:C • c ∈ cs ∧ **uid**_C(c)=vi

5.3 Mereology

5.3.1 Mereology Types & Observers

80. The mereology of a conveyor is a finite set of edge and node identifiers that it may "visit".[24]

type
80. CM = UI-**set**
value
80. **mereo**_C: C → CM

[24] We shall extend this mereology in Sect. 13.1 on page 59.

5.3.2 Mereology Wellformedness

81. The identifiers of a conveyor mereology must be those of the edges and nodes of the transport graph, g.
82. The kind of conveyor must "fit" the kind of edges and nodes[25].

```
axiom [Conveyor Mereology of Right Kind]
81.  ∀ c:C•c∈cs⇒∀ui:UI•ui ∈mereo_C(c)
81.      ⇒ ui∈e_uis∪n_uis
82.         ∧ c_kind(c)∩kind(retr_unit(ui))≠{} ι82
```

5.4 Attributes

5.4.1 Conveyor Attribute Types & Observers

In this section we deal wit some attributes. Further conveyor attributes are brought forward in Sect. 12.3.3 page 56.

83. Conveyors are of kind. [Static Attribute]
84. These routes must be of the kind of the conveyors traveling them !
85. Conveyors either stand still or move. That is, they have position in the graph, an index on the service route. Either the position is at a node, or somewhere, a fraction, f, of a distance along an edge, from one node to an adjacent. [Programmable Attribute]
86. The service route index must be commensurate with the conveyor position.
87. We omit further possible attributes: `Speed, Acceleration, Weight, ....`

```
type
83.   Kind
85.   CPos = AtNode | OnEdge
85.   AtNode :: NI
85.   OnEdge :: NI × (F × EI) × NI
85.   F = Real axiom ∀ f:F•0<f<1
value
83.   attr_Kind: C → Kind
85.   attr_CPos: C → CPos
87.   ...
axiom [Routes of commensurate kind]
84.   ∀c:C•let ps=attr_Routes(c)in∀p:Path•p∈ps∧ps⊆path_kind(p)(kind(c)) end
```

[25] Cars, Taxis, Buses, Trucks move along edges and nodes of kind **road** [a literal value, like **true** and **false** are literal values], Passenger and Freight Trains move along edges and nodes of kind **rail** [a literal value], Sail Boats, Barges, Fishing Vessels, Ferries, Freighters, Ferries and Passenger Liners move along edges and nodes of kind **sea** [a literal value] and Private Aircraft, Helicopters, Freight Planes and Passenger Aircraft move along edges and nodes of kind **air"** [a literal value].

5.4.2 Routes

88. The following properties hold of any route:
    (a) the current route of a conveyor must always be in the routes of that conveyor.
    (b) The static attribute `Routes` must all start and end with a node identifier.
    (c) When initialized, a conveyor "starts" with a `CurrentRoute` chosen from the `Routes`.
    (d) At any moment a conveyor moves along a [programmable attribute] *current* route.
    (e) When moving from an edge to a node the current route is shortened by one.
    (f) When a route is thereby exhausted, i.e., $\langle\rangle$, the conveyor may decide to select a new route.
    (g) It does so from the static attribute `Routes`.
        i. The previous, exhausted route ended with a node identifier.
        ii. The next, to be current, route must start with that node identifier.

```
axiom [Commensurable Routes]
88.   ∀ c:C,r:Routes,cr:CurrRoute • r=attr_Routes(c)∧cr=attr_CurrRoute(c)
88a.      cr ∈ r
88b.  ∧ is_NI(hd r)∧is_NI(r[len r])
```

For *cars* the `Routes` attribute may exclude certain paths, for example such toll-roads for which they have no license. When, for example, *buses*, *trains*, *ferries* and *passenger aircraft*, the routes are such that for every pat there is at least one path that "connects" to the former: ends, respectively starts with identical node identifiers. Usually the set of routes contains just two paths: ode from node n_i to node n_j and the other from node n_j to node n_i. And so forth !

5.4.3 Conveyor Attribute Wellformedness

TO BE WRITTEN

Chapter 6

Intentional Pull, I

6.1 History Attributes

History attributes record when conveyors (cars, trains, boats and aircraft) were where and at which times. They are chronologically ordered, time-stamped sequences of event notices. History attributes are programmable.

History attributes "record" events. Conveyors, as controlled by, say humans, may not note down these **event**s, and edges and nodes, which we in some sense consider innate[26], "most likely" do not notice them.

But we, "us", humans, can speak about and recall [these, and "other"[27]] events – and they are therefore an essential aspect of modelling any manifest domain.

89. We "lump" nodes and edges into single element ways [i.e., endurants].
90. The ordered, $\mathbb{TIME}$[28] -stamped, history attribute event notices record the vehicles, by their unique identifiers.
91. The ordered, $\mathbb{TIME}$-stamped, conveyor history attribute event notices record the ways, by their unique identifiers.

```
type
89.   W = N|E
89.   WI = NI|EI
90.   WHist = (s_t:𝕋𝕀𝕄𝔼×VI)*
91.   ConvHist = (s_t:𝕋𝕀𝕄𝔼×CI)*
value
89.   retr_W: WI → N|E
89.   retr_W(wi) ≡ !  w:W • w ∈ns∪es ∧ uid_W(w)=wi
90.   attr_WHist:  W → WHist
90.   attr_ConvHist:  C → ConvHist  ι90
axiom [Ordered Way and Conveyor Histories]
90.   ∀ wh:WHist • {i,i+1}⊆inds wh ⇒ s_t(rh[i])<s_t(wh[i+1])
91.   ∀ ch:ConvHist • {i,i+1}⊆inds ch ⇒ s_t(ch[i])<s_t(ch[i+1])
```

[26] An innate quality or ability is one that you were born with, not one you have learned. That is: we consider edges and nodes to be innate wrt. observing and recording the where-about events of conveyors – other than indirectly through the space they "occupy", the possible wear & tear of the road surface or rail track, or possible pollution of the sea and air, etc.

[27] By the seemingly cryptic "other" events, we may, in the context of transport, think of such events as *conveyor breakdown, edge collapse*, etc.

[28] $\mathbb{TIME}$ is a "global" phenomenon.
We say *15:23 June 12, 2025 CET*, and mean that it is now *23 minutes past 3pm, 25th of February 2025, Central European Time.*
$\mathbb{TI}$ stands for time-interval.
We say *3 hours and 23 minutes.*

6.2 An Intentional Pull

Nodes and edges are intended to “carry” traffic [only] in the form of vehicles, and vehicles are intended to move along [only] ways, i.e., nodes and edges.

92. for all conveyors (of a transport) if
    (a) a conveyor is said to be on a way, i.e, at a node [resp. on an edge], at time τ,
    (b) then that way must “carry” that conveyor
    (c) at exactly that same time;
93. and vice-versa, if-and-only-if, for all ways
    (a) a way is said to “carry” a conveyor at time τ,
    (b) then that conveyor must be on that way
    (c) at exactly that same time.

Intentional Pull:

```
92.    ∀ c:C • c ∈ cs •
92a.       let ch:CH = attr_CH(c) in
92a.       ∃!i:Nat • i ∈ inds ch •
92a.           let (τ,wi) = ch[i] in
92b.               let wh:WH = attr_WH(retr_way(wi)) in
92c.               ∃!j:Nat • j∈ inds WH • s_t(wh[j]) = τ
92.        end end end
93.    ≡
93.    ∀ w:W • w ∈ es∪ns •
93a.       let wh = attr_WH(w) in
93a.       ∃!k:Nat • k ∈ inds wh •
93a.           let (τ,ci) = wh[k] in
93b.               let ch:CH = attr_WH(retr_conveyor(ci)) in
93c.               ∃ℓ:Nat • ℓ∈ inds ch • s_t(ch[ℓ]) = τ
93.        end end end
```

Chapter 7

Single-mode Transport Behaviours

The previous sections, Sects. 3–6, studied, analyzed & described a transport domain syntactically, that is: its manifest forms and properties, but not its meaning, i.e., semantics. This sections is about that: the "meaning", so-to-speak, of endurants. This will be done by **transcendentally deducing** *behaviours* and *actions* from the description of endurants. Endurants are **transcendentally deduced** into behaviours, and described as s with arguments. Their internal properties are **transcendentally deduced** into arguments of these behaviours. We choose to only endow edges, nodes and conveyors with behaviours. Behaviours synchronize and communicate via "the ether" – here RSL/CSP-modeled as a **channel** array that allows conveyor, node and edge behaviours (u_i, u_j, u_k) to cooperate !

7.1 Communication

7.1.1 Communication Medium

94. There is a "global" communication, i.e., behaviour interaction medium, **comm**. It allows transport Behaviours to synchronize and exchange information of type M.

channel
94. **comm**[{i,j} | i,j:UI•{i,j}∈*uis*] MSG

7.1.2 Communication Causes

95. A conveyor, ci:CI, at a node decides to remain at that node.
96. A conveyor, ci:CI, at a node decides to change route.
97. A conveyor, ci:CI, at a node decides to leave the node, and
98. to enter an edge.
99. A conveyor, ci:CI, on an edge decides to move on.
100. A conveyor, ci:CI, on an edge decides to leave that edge, and
101. to enter the node.
102. And a conveyor, ci:CI, at a node or on an edge may decide, "surreptitiously" or otherwise, to just **stop**.

7.1.3 Communication Messages

103. The message is simple: a time stamp and the identity of a node, an edge or a conveyor.

type
103. MSG = $\mathbb{TIME}$ × (NI|EI|CI)

7.2 Behaviours

So we model conveyor, node and edge behaviours. Each of these behaviour functions has arguments of the following kind:

- a **unique identifier,** never changes, distinguishes between multiple instances of edges, or nodes, or conveyors;
- a *mereology*; and
- **attributes:**
  - **static attributes**, i.e., attributes whose value never changes;
  - **monitorable attributes**, i.e., attributes whose value changes "at their own volition": itself nor cooperating behaviours cannot influence their value – we shall not consider monitorable attributes in this study; and
  - **programmable values**, i.e., attributes whose value may be changed by the behaviour – i.e., acts like variables that can be read and updated !

Each of these behaviours are modelled as processes that may "go-on-and-on-forever" – modelled in terms of *tail-recursion* – modelled also in the specifying **Unit** as part, "the last", of the behaviour signature.

7.3 Behaviour Signatures

104. We present the `conveyor,` `edge` and `node` behaviour signatures.

```
value
104.   conveyor:  CI→CM→(Kind×Routes)→(CurrRoute×CPos×CH)→Unit
104.   edge:  EI→EM→(Kind×LEN×...)→NH→Unit
104.   node:  NI→NM→(Kind-set×...)→NH→Unit
```

7.4 Behaviour Definitions

7.4.1 Conveyor Behaviours

- A `conveyor` alternates between being at a node or on edge, so its behaviour is defined in terms of "either" and their "progress" onto "the other" !

• CONVEYOR **Behaviour** AT A NODE:

105. A `conveyor` at a node either
   (a) changes its current route, and choose another, the next current route, or
   (b) `remains` at that node, idling, or circling around, or
   (c) is `entering` an edge, or
   (d) **stop**s at that node, i.e., leaves the transport altogether.

```
value
105.   conveyor(ci)(cm)(k,routes)(cr,AtNode(ni),ch) ≡
105a.      conveyor_change_route(ci)(cm)(k,routes)(cr,AtNode(ni),ch)
105b.    ⌈⌉ conveyor_remains_at_node(ci)(cm)(k,routes)(cr,AtNode(ni),ch)
105c.    ⌈⌉ conveyor_enters_edge(ci)(cm)(k,routes)(cr,AtNode(ni),ch)
105d.    ⌈⌉ conveyor_stops_at_node(ci)(cm)(k,routes)(cr,AtNode(ni),ch)
```

• CONVEYOR **Actions** AT A NODE:

106. A conveyor may non-deterministically decide to **change** its current route at a node
    (a) at time τ,
    (b) selects of next, to be, current route from `routes` such that that the chosen route begins with the node being otherwise left,
    (c) so informing the node, and
    (d) updates its history,
    (e) whereupon it resumes being a conveyor with both updated current route and history.

```
106.   conveyor_change_route(ci)(cm)(k,routes)(cr,AtNode(ni),ch) ≡
106a.       let τ = record_TIME(),
106b.          ncr = select_next_route(ni,routes),
106d.          ch′ = ⟨(τ,ni)⟩⌢ch in
106c.       comm[{ci,ni}] ! (τ,ci) ;
106e.       conveyor_at_node(ci)(cm)(k,routes)(ncr,AtNode(ni),ch′) end

106b.  selects_next_route:NI × Routes → CurrRoute
106b.  selects_next_route(ni,routes) as ncr • ncr ∈ routes ∧ hd ncr = ni
```

107. A conveyor **remains** at a node
    (a) at some time, τ,
    (b) which is to be noted by the node behaviour ni
    (c) whereupon the conveyor resumes being a conveyor except now with an updated conveyor history, ch.

value

```
107.   conveyor_remains_at_node(ci)(cm)(k,routes)(cr,AtNode(ni),ch) ≡
107a.       let τ = record_TIME() in
107b.       comm[{ci,ni}] ! (τ,ci);
107c.       conveyor(ci)(cm)(k,routes)(cr,AtNode(ni),⟨(τ,ni)⟩⌢ch) end
```

108. A conveyor at a node may non-deterministically choose to leave a node and **enter** an edge
    (a) at some time, τ, and as determined by the current route's next element, enters that route, i.e., edge,
    (b) which is to be noted by the node and designated edge behaviours ni,
    (c) updates its position
    (d) and its history accordingly,, and
    (e) resumes being a conveyor on an edge .

value

```
108.   conveyor_enters_edge(ci)(cm)(k,routes)(cr,AtNode(ni),ch) ≡
108a.      let τ = record_TIME() in
108b.      ( comm[{ci,ni}] ! (τ,ni) ∥ comm[{ci,ni}] ! (τ,hd cr) ) ;
108c.      let ei = hd cr in let {ni,ni′} = mereo_E(retr_edge(ei)(es)) in
108c.      let cpos = onEdge(hd cr,(ei,(ni,f,ni),ni′)) in
108e.      conveyor(ci)(cm)(k,routes)(cr,cpos,⟨(τ,ni)⟩⌢ch) end end end end
```

109. And a conveyor may non-deterministically choose to abandon being a conveyor, i.e., leaving transport altogether – **stop**ping !

110. But first it notifies the node at which it stops.

```
value
109.   conveyor_stops_at_node(ci)(cm)(k,routes)(cr,AtNode(ni),ch) ≡
110.      let τ = record_TIME() in
110.      comm[{ci,ni}] ! (τ,ci) ;
109.      stop end
```

- A conveyor behaviour on an edge alternates.

• CONVEYOR **Behaviour** ON EDGE

111. An edge [behaviour] at an edge external non-deterministically either:
    (a) **move**s along the edge, a fraction "at a time",
    (b) **stop**s on the edge and thereby "leaves" transport; or
    (c) **enter**s a node.

```
111.   conveyor(ci)(cm)(k,routes)(cr,OnEdge(n_{ui_f},(f,e),n_{ui_t}),ch) ≡
111a.      conveyor_moves_on_edge(ci)(cm)(k,routes)(cr,OnEdge(n_{ui_f},(f,e),n_{ui_t}),ch)
111c.    ⌈⌉ conveyor_stops_on_edge(ci)(cm)(k,routes)(cr,OnEdge(n_{ui_f},(f,e),n_{ui_t}),ch)
111b.    ⌈⌉ conveyor_enters_node(ci)(cm)(k,routes)(cr,OnEdge(n_{ui_f},(f,e),n_{ui_t}),ch)
```

• CONVEYOR **Actions** ON AN EDGE:

112. A conveyor **moving** along an edge
    (a) at time τ is modelled by
    (b) incrementing the fraction of its position
    (c) (while updating its history)
    (d) notifying the edge [behaviour]
    (e) [technically speaking] adjusting its position, and, finally,
    (f) resuming being a thus updated conveyor [OnEdge]

```
value
112.   conveyor_moves_on_edge(ci)(cm)(k,routes)(cr,OnEdge(n_{ui_f},(f,e),n_{ui_t}),ch) ≡
112a.       let τ = record_TIME(),
112b.           ε:Real • 0 < ε ≪ 1 in
112b.       let f′ = f+ε,
112d.           cpos = OnEdge(n_{ui_{i_f}},(f′,e),n_{ui_t}) in
112c.       let ch′ = ⟨(τ,ci)⟩⁀ch in
112e.       comm[{ci,e_j}] ! (τ,ci) ;
112f.       conveyor(ci)(cm)(k,routes)(cr,cpos,ch′) end end end
112.   pre hd cr = n_{ui_f}
```

113. A conveyor **enters** a node
   (a) at time τ is modelled by altering its position
   (b) notifying both the edge and designated node behaviours
   (c) resumes being an updated `conveyor` behaviour.

value
```
113.   conveyor_enters_node(ci)(cm)(k,routes)(cr,OnEdge(n_{ui_f},(f,ei),n_{ui_t}),ch) ≡
113a.     let τ = record_TIME(), cpos = AtNode(hd cr) in
113b.     ( comm[{ci,ei}] ! (τ,ci) ∥ comm[{ci,n_{ui_t}}] ! (τ,ci) ) ;
113c.     conveyor(ci)(cm)(k,routes)(tl cr,cpos,⟨(τ,ci)⟩⌢ch) end
113.   pre hd cr = n_{ui_f}
```

114. A conveyor may non-deterministically choose to abandon being a conveyor, i.e., leaving transport altogether – **stop**ping !

115. But first it notifies the edge at which it stops.

value
```
114.   conveyor_stops_on_edge(ci)(cm)(k,routes)(cr,OnEdge(n_{ui_f},(f,e),n_{ui_t}),ch) ≡
115.     let τ = record_TIME() in
115.     comm[{ci,e_j}] ! (τ,ci) ;
114.     stop end
114.   pre hd cr = n_{ui_f}
```

7.4.2 Node Behaviour

116. **Node** [behaviours]
   (a) external non-deterministically accept conveyor, `ci`, actions
   (b) at times τ
   (c) augment their histories accordingly and
   (d) resumes being node behaviours.

value
```
116.   node: NI → NM → (NodeKind×...) → NH Unit
116a.  node(ni)(nm)(nk,...)(nh) ≡
116c.     let msg= ⨅ { comm[{ni,ci}] ? | ci:CI • ci ∈ nm } in
116d.     node(ni)(nm)(...)(⟨msg⟩⌢nh) end
```

7.4.3 Edge Behaviour

117. **Edge** [behaviours] – similarly,
   (a) external non-deterministically, accept conveyor, `ci`, actions
   (b) augment their histories accordingly and
   (c) resumes being edge behaviours.

value
```
117.   edge: EI → EM → (EdgeKind×LEN×COST×...) → EH Unit
117a.  edge(ei)(em)(len,cost,...)(eh) ≡
117b.     let msg= ⨅ { comm[{ei,ci}] ? | ci:CI • ci ∈ em } in
117c.     edge(ni)(em)(len,cost,...)(⟨msg⟩⌢eh) end
```

7.5 Domain Instantiation

By domain initialization we mean the *invocation*[29] of all behaviours.

118. The overall initialization expresses the parallel composition of the initialization of
119. all conveyors,
120. all nodes and
121. all edges.

```
118.  initialization:  Unit → Unit
118.  initialization() ≡   t
119.     ∥ { conveyor
119.             (uid_C(c))
119.                (mereo_C(c))
119.                   (attr_KindC(c),attr_RoutesC(c))       [Static Attrs.]
119.    [Programmable Attrs.]  (attr_CurrRouteC(c),attr_CPoC(c)s,attr_CHC(c))
119.              | c:C•c ∈ cs}
120.  ∥ ∥ { edge
120.             (uid_E(e))
120.                (mereo_E(e))
120.                   (attr_EdgeKind(e),...)              [Static Attrs.]
120.    [Programmable Attrs.]  (attr_(e),attr_EH(e))
120.              | e:E•e ∈ es }
121.  ∥ ∥ { node
121.             (]uidN(n))
121.                (mereo_N(n))
121.                   (attr_NodeKinds(n))                 [Static Attrs.]
121.    [Programmable Attrs.]  (attr_NH(n))
121.             | n:N•n ∈ ns}
```

But: the initializaton of conveyors is too simplified: To capture an essence of transport it seems reasonable to distinguish between the various kinds of conveyors.

Thus the initialization of conveyors "really" amounts to the initialization of all

- cars, trucks, taxis,
- buses,
- passenger & freight trains,
- sailboats, barges, vessels,
- passenger liners, ferries,
- civil aircraft,
- freight planes and
- passenger aircraft.

[29] Invocation – in the colloquial – "call"

Part II

A MULTI-MODE TRANSPORT: ENDURANTS

Chapter 8

Multi-mode Transport

The domain description of Chapters 4–7 was for single-mode transport: It focused on transport nets and conveyors. For a model of *multi-mode transport* we suggest to introduce:

- **Merchandise:** By merchandise we shall here understand a wider concept than usually thought of. To us *merchandise* is what customers wishes to and actually send and receive: *goods*, if You will, that have weight, volume and value. Could be a car, a book, 10.000 barrels of oil, etc. Merchandise is treated in Sect. 10.
- **Customers:** A [multi-mode transport] *customer* is either, if persons, wishes to travel from one place to another, or if otherwise wishes to send merchandise from one place, e.g., the customer's place, e.g. a node or an edge, to be received by a *recipient* at that another place. In the latter case customers are persons, businesses, organizations, or other, i.e., are *senders* or *receiver*, i.e., *recipients*. Customers are treated in Sect. 11.
- **Conveyor Companies:** A *conveyor company* is a business which manages a fleet of conveyors: trucks, freight trains freighters (i.e., vessels) and freight aircraft. Conveyor Companies are treated in Sect. 12.
- **Logistics Companies:** A transport logistics company handles requests from *sender*s of *passenger*s or *goods* (containers, oil, coal, gas, grain, salt, cars, machinery, etc.) to have these conveyed from one node to another, world-wide, by whatever means of combinations of conveyors and routes. A logistics company thus is a company which arranges for transport of merchandise. To do so logistics firms have access to the transport offerings of a number of, not necessarily all, conveyor companies: their routes, timetable and costs. Logistics Companies are treated in Sect. 14.
- **"Overall Top" Transport Endurants:** The *graph*, *conveyors*, *merchandise*, *customers*, *conveyor companies* and *logistics companies* form the transport domain. As a whole they are defined in Sect. 9.

After these sections we

- outline an **intentional pull** for multi-mode domains, Sect. 15,
- summarize the syntax of multi-mode transport **commands,** Sect. 16,
- and cover multi-mode transport **behaviours,** Sect. 18.

• • •

To obtain the services of merchandise transport comes at a **price**, the *cost*.

The notion of *cost* is related to the notion of **cash.** It costs to have merchandise transported. Customers shall pay costs. Say, in the form of cash[30]. Costs shall be modelled as integers. They are attributes of merchandise, customers, conveyor companies and logistics companies.

You may very well think of cash as manifest, i.e., as endurant parts. But in the context of transport we can abstract from that. If we were to model cash as endurants, then were we to model it as atomic or composite ? Now we avoid such questions !

[30]– or through withdrawal from bank accounts, or other. See [10].

Chapter 9

"Top" Transport Endurants

9.1 The Endurants – External Qualities

9.1.1 A Transport Taxonomy

We refer to Fig. 9.1 for a taxonomy of the transport domain.

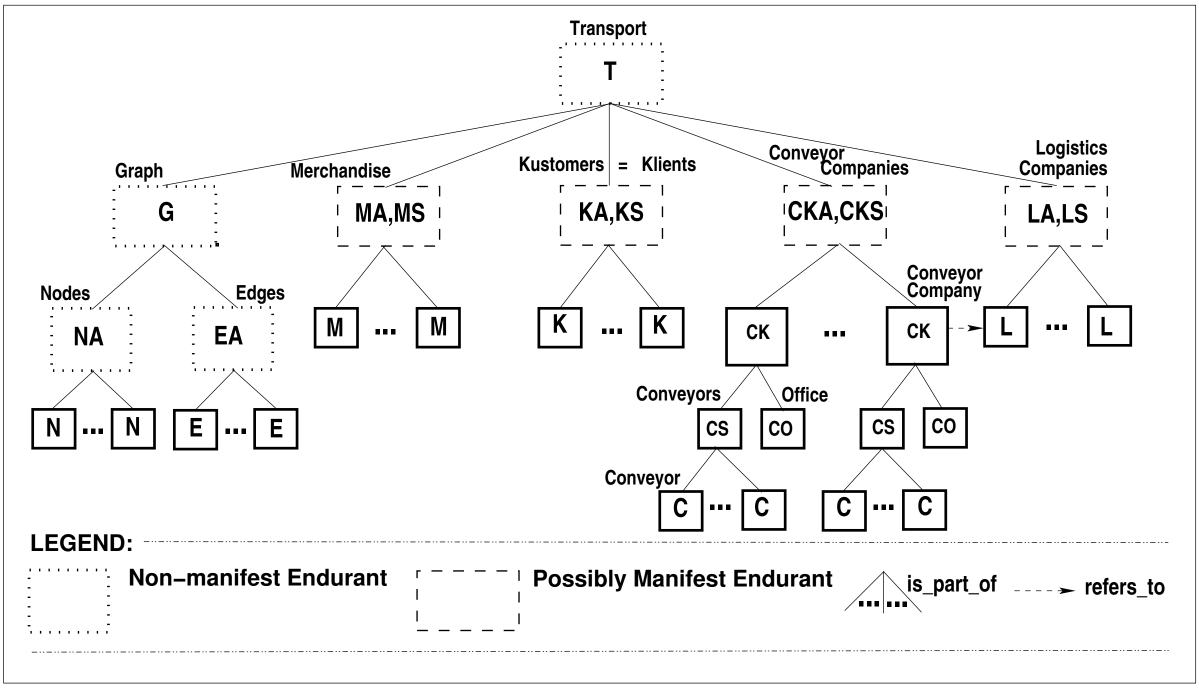


Figure 9.1: A Transport Domain Taxonomy

The "downwards" slanted lines express that the "lower" part is part of the "upper" part.

The "horizontal arrow" expresses that the source part embed to "arrow" part. [Only one is illustrated; more could !

9.1.2 An Overview of The Endurants

The Transport Domain

122. There is given the domain of interest, i.e., the universe of discourse, T.

```
type
122.  T
value
122.  t:T
```

Graphs

Graphs were treated in Sect. 4.

123. In a transport domain can observe the *transport net*, i.e., a *graph*, g:G.
124. From a graph we can observe a node aggregate,
125. and an edge aggregate.
126. From a node aggregate we can observe a set of nodes.
127. From an edge aggregate we can observe a set of edges.

```
type
123.  G
124.  NA
125.  EA
126.  NS = N-set
127.  ES = E-set
126.  N
127.  E
```

```
value
123.  obs_G: T → G
124.  obs_NA: G → NA
125.  obs_EA: G → EA
126.  obs_NS: NA → NA
127.  obs_ES: EA → ES
```

And likewise for the unique identification of the manifest of these endurants.

```
type
123.  GI
124.  NAI
125.  EAI
126.  NO
127.  EI
```

```
value
123.  uid_G: G → GI
124.  uid_NA: G → NAI
125.  uid_EA: G → EAI
126.  uid_N: N → NI
127.  uid_E: E → EI
```

Merchandise

Merchandise is treated in Sect. 10.

128. From a transport domain we can observe a *merchandise aggregate*, `ma:MA`;
129. and from a *merchandise aggregate* we can observe the set, `ms:MS` of merchandise.

And likewise for the unique identification of the manifest of these endurants.

```
type
128.  MA
129.  MS = M-set
value
128.  obs_MA: G → MA
129.  obs_MS: MA → MS
```

```
type
128.  MAI
129.  MI
value
128.  uid_MA: MA → MAI
129.  uid_M: M → MI
```

Customers

Customers are treated in Sect. 11.

130. From a transport domain we can observe a *"k"ustomers aggregate*, `ka:KA`;
131. and from a *customer aggregate* we can observe the set, `ks:KS` of customers.
132. We can speak of the set, *ks*, of all customers of a transport domain.

And likewise for the unique identification of the manifest of these endurants.

```
type
130.  KA
131.  KS = K-set
value
130.  obs_KA: T → KA
131.  obs_KS: KA → KS
```

```
type
130.  KAI
131.  KI
value
130.  uid_KA: KA → KAI
131.  uid_K: K → KI
132.  ks:K-set = obs_KS(obs_KA(t))
```

Conveyor Companies & Conveyors

Conveyors were treated in Sect. 5 and **Conveyor Companies** are treated in Sect. 12.

133. In a *transport domain*, `t:T`, we can observe the composite endurant of *conveyor companies aggregate*, `cca:CCA`.
134. From a *conveyor companies aggregate*, `cca:CCA`, we can observe a set,`cks:CKS`, of *conveyor companies*.
135. *Conveyor companies* are considered atomic.

From a *conveyor company*, `ck:CK`, we can observe

136. a *conveyor aggregate*, `ca:CA`,
137. and, from that, a *conveyor set*, `cs:CS`, which is a set of *conveyors*.

    From a *conveyor company*, `ck:CK`, we can also observe

138. we can observe an atomic *conveyor company office*, `co:CO`,
139. and an atomic, optional *logistics subsidiary*, `ol:oL`, i.e., the conveyor company may operate its own *logistics company*.

```
type
133.  CKA
134.  CKS = CK-set
135.  CK
136.  CA
137.  CS = C-set
138.  CO
139.  oL = LI | nil
```

```
value
133.  obs_CKA: T → CKA
134.  obs_CKS: CKA → CKS
136.  obs_CA: CK → CA
137.  obs_CS: CA → CS
138.  obs_CO: CK → CO
139.  obs_oL: CK → oL
```

And likewise for the unique identification of the manifest of these endurants.

```
type
133.  CKAI
135.  CKI
136.  CAI
138.  COI
139.  oLI = LI | nil
```

```
value
133.  uid_CKA: CKA → CKAI
135.  uid_CK: CK → CKI
136.  uid_CA: CK → CAI
138.  uid_CO: CK → COI
139.  uid_oL: CK → oLI
```

• • •

We shall, in the following, not treat the concepts of *conveyor [company] offices* and *the logistics company* parts of *conveyor companies*. We shall also not treat the concepts of *conveyor aggregates* and *conveyor sets*, but will treat the concept of *conveyors*.

Logistics Companies

Logistics Companies are treated in Sect. 14.

140. From a transport domain we can observe a *logistics companies aggregate*;

141. and from a *logistics companies aggregate* we can observe the set, `ls:LS` of *logistics companies.*

And likewise for the unique identification of the manifest of these endurants.

```
type
140.  LA
141.  LS = L-set
value
140.  obs_LA: T → LA
141.  obs_LS: LA → LS
```

```
type
140.  LAI
141.  LI
value
140.  uid_LA: LA → LAI
141.  uid_L: LA → LS
```

Node and Edges were first treated in Sect. 4. To this we now add a widened understanding of their mereologies and attributes.

142. The mereology of nodes is a pair of the set identifiers of edges imminent upon the nodes and the set of identifiers of the customers and conveyors that can deposit merchandises "on hold" at the nodes.

143. The mereology of nodes is a pair of the set identifiers of [the pair of] nodes "at ether end of the edge" and the set of identifiers of conveyors that may travel along the edge.

    Nodes and edges have the following attributes:

    (a) Nodes have merchandises "on hold" – by contract number,
    (b) and nodes have node histories: time-stamped events of which conveyors notified their presence at the node.
    (c) Edges have length,
    (d) cost of travel,
    (e) and event histories:: time-stamped events of which conveyors notified their presence at the edge.

```
type
142.   NM = EI-set × (KI|VI)-set
143.   EM = HI-set × VI-set
143a.  OnHold = ContractNu →m M-set
143b.  NHist = (TIME × CI)*
143c.  LEN
143d.  COST
143e.  EHist = (TIME × CI)*
value
142.   mereo_N: N → NM
143.   mereo_E: E → EM
143a.  attr_OnHold: N → OnHold
143b.  attr_NHist: N → NHist
143c.  attr_LEN: E → LEN
143d.  attr_COST: E → COST
143e.  attr_EHist: E → EHist
```

• • •

Atomic Parts:

144. Nodes, edges, merchandise, "k"ustomers, conveyors, conveyor company offices, and logistics firms are considered atomic.

```
type
144.  N, E, M, K, C, CO, L
```

We shall not [really] consider conveyor offices and logistics firms in this report.

9.2 On Internal Qualities.

We discuss which endurants may be considered manifest. That is, to which of the parts – as, for example, shown by the boxes of Fig. 9.1 on page 37 – one might associate internal qualities, say in preparation for their part behaviours.

- With the *transport* part, `t:T`, we might – here rather loosely – associate a *ministry of transport*, or ...; We shall omit such associations.
- With the *graph* part, `g:G`, we might associate various other public (or private) institutions: *ministry of roads*,*ministry of railways*, *ministry of shipping*, and *"ministry of air"* ! We shall omit such associations.
- With the *merchandise* part one might associate some institution of *consumer protection* or other. We shall omit such associations.
- With the *customer (client, consumer)* part one might associate some kind of institutions. We shall omit such associations.
- With the *conveyor company* part one might associate some *conveyor association*. We shall omit such associations.
- With the *logistics companies* part one might similarly associate some associations. We shall omit such associations.
- With nodes, edges, merchandise, customers [clients], conveyor sets and conveyor offices we have and shall associate internal qualities – in respective sections 4, 5 and 10, 11, 12 and 14.

So we shall not elaborate on any internal qualities of the "top-level" endurants, that is those of `T, G, NA, EA, MA, KA, CCA`, and `LA`. But we shall, later, in indicated sections, elaborate on internal qualities of the "next-level" endurants, i.e., those of `M, K, CK, CS, CO` and `L` [Sects. 10, 11, 12 and 14] – as we already have for `N, E` and `C` [Sects. 4 and 5].

Figure 9.1 on page 37 hints at manifest, possibly manifest and non-manifest parts.

9.3 Conveyor Companies versus Logistics Companies.

Is it really necessary to distinguish between the two: conveyor and logistics companies ? Examples of the two are:

- **conveyor companies:** Maersk[31], DSV[32] SAS, American Airlines, British Air, Deutsche Bahn, SNCF, Amtrack, Arriva, Greyhound, P&O, Dachser[33], etc.
- **logistics companies:** TUI, Expedia, etc.[34]

As You may have deduced from the examples: some of the conveyor companies also operate "own" logistics departments, i.e., companies. But their functions must be separated: Conveyor companies fundamentally operate conveyors, and, only as a necessity, embody logistics departments – which basically only handle only their "mother", i.e., the conveyor company's own conveyors. Logistic companies, in general, make use of several conveyor companies.

9.4 Financial Matters

Transport implies expenses. *Cost* and *payment* of conveyance, is implied, but we have chosen to omit modelling these facets. Both conveyor and logistics companies rely on *creating*, *writing/editing*, *reading*, *copying* and *destroying documents*. The implied *double bookkeeping* will also not be modelled. These financial facets are not an essence, so we have decided, of the core aspects of transport. We refer to [10, 11] and [8], respectively, for treatments of these three domains.

[31] Maersk, Danish, is one of the world's largest container shipping lines.
[32] DSV, Danish, is one of the world's largest trucking companies.
[33] https://www.dachser.dk/da/
[34] Yes, it has not gone unnoticed, that these "travel agencies" are, indeed, logistics companies – when seen from inside the daily operations of these. Also: I find it difficult to find conveyor companies that do not have a logistics [sub-]office !

Chapter 10

Merchandise

We shall use the term *merchandise* as a common denominator for "all that can be transported" ! living species: people[35], animals, plants, wheat, etc.; solid materials: iron ore, automobiles, timber, etc.; fluid materials: oil, gas, water, etc. Perhaps a better term would/should have been *goods*

10.1 Merchandise Endurants

10.1.1 External Qualities

145. There is the atomic endurant: merchandise.

type
145. `M`
value
145. m:`M`

[35] Please do not be confused: No, we do not refer to people as slaves !

10.1.2 Internal Qualities

We lump the presentation of identification, mereology and attributes of merchandises into one, the present, section.

Unique Identifiers:

146. Merchandises have unique identification. [That is: no two items of merchandise have the same identification, and these are distinct from the identification of all other parts of the transport domain.]

Mereology:

147. The mereology of any [item or piece of] merchandise is the set of customers and conveyors that may possess or transport that merchandise.

Attributes:

148. Merchandises have practical identification: names, manufacture, place of origin, etc. Two or more merchandise may have the same such identification.
149. Merchandises have current position – a programmable attributes
150. Merchandises have size, approximate height, width and depth.
151. Merchandises have weight.
152. Merchandises have cost.
153. Merchandises have flammability.
154. Merchandises may be insured.
155. Merchandises have a history: an chronologically descending, ordered sequence of event notes:
156. Events are either ...
157. Et cetera ...

```
type
  Unique Identifiers:
146.  MI
  Mereology:
147.  MM = KI-set × CI-set
  Attributes:
148.  MId = Name×Mfg×Origin×...
149.  Position = (NI × (F × EI) × NI) | NI | CI
150.  Size = Nat×Nat×Nat
151.  Weight = Real
152.  Cost = Nat
153.  Flammability = "flammable"|"inflammable"|"combustible"|...
154.  Insurance
155.  MHist = (TIME×Event)*
156.  Event = ... | ... | ... | ...
157.  ...
value
146.  uid_M: M → MI
147.  mereo_M: M → MM
148.  attr_MId:  M → MId
149.  attr_Position:  M → Position
150.  attr_Size:  M → Size
151.  attr_Weight:  M → Weight
152.  attr_Cost:  M → Cost
153.  attr_Flammability:  M → Flammability
154.  attr_Insurance:  M → Insurance
155.  attr_MHist:  M → MHist
```

Merchandises must satisfy some axiom[s]:

158. No one merchandise must be at exactly one position at any one time.

axiom
158. ...

10.2 Representation of Merchandises

Merchandises are inert: does not move by their own volition ! But merchandises are being moved – by conveyors. So how do we present merchandise ? In Sect. 5.4 on page 23, when we first described conveyor attributes, we did not endow them with merchandise. That will be remedied in Sect. 12.3.5 Page 53.

We shall then, in Sect. 12.3.5 Page 53, see that we choose to model merchandises on a conveyor as a set of merchandise unique identifiers !

159. Here we shall model the existence of a set of merchandises as a state value.

value
159. ms:M-**set** = **obs**_MS(**obs**_MA(t))

Given the unique identifier, `mi`, of a merchandise and given the "global" merchandises state we can "retrieve" the identified merchandise:

160. The `retrieve merchandise` function, `retr_merchandise`, takes a merchandise identifier and in the context of the "global" merchandises state *ms*,
161. yields the unique (ι) `m` with that identifier in *ms* that has that identifier.

value
160. retr_merchandise: MI×MS → M
161. retr_merchandise(mi)(ms) ≡ ι m:M • m ∈ ms ∧ **uid**_M(m)=mi

10.3 Humans

162. Humans can be merchandise.[36]

type
162. Human
value
162. **is**_Human: M → **Bool**

[36] Not in the sense of illegal immigrants, sadly, but in the sense of legally "ticketed" passengers of bus, train, ship and aircraft conveyors.

Chapter 11

Customer

We shall use the term **'customer'** for any person or institution that requests transportation of or receives transported merchandise. Other terms could be **'client'** or **'consumer'**. All have the advantage of beginning with a 'c'. Which we [quickly] convert into a 'k' – for same pronunciation !

11.1 Customer Endurants

11.1.1 Endurant Sort

163. There is the atomic endurant: customer.

type

```
163.  K
```

11.1.2 A State Notion

164. There is the "global" transport value, t:T.

165. From it we observe a likewise "global", the set of all customers, ks:KS.

value

164. t:T
165. ks:KS = **obs**_KS(**obs**_KA(t))

11.2 Customer Qualities

We lump the presentation of identification, mereology and attributes of customers into one, the present, section.

Unique Identifiers:

166. Customers have unique identification.

167. We can speak of the identities of all customers, as a "globally" known value.

Mereology:

168. The mereology of any customer is the triple of the set of merchandises and the logistics firms that such firms may be requested to arrange transport.

Attributes:

169. Customers have practical identification: name and address.

170. Customers posses merchandise.

171. Customers have outstanding requests: a time-stamped set of shipping notices: to be or being sent, or to request to or expecting to receive.

172. Customers accumulate, for every event, a Customer History: A time-stamped, chronologically ordered sequence of event records: most recent event first.

173. Events are either ...

174.

type
  Unique Identifiers:
166. KI
  Mereology:
168. KM = MI-**set** × (CKI|LI)-**set** × CI-**set**
  Attributes:
169. CustId = CustNam × CustAdd × ...
170. Possess = MI-**set**
171. OutReqs = ...
172. CustHist = ($\mathbb{TIME}$ × Event)*
173. Event = ...
174. ...
value
  Unique Identifiers:
166. **uid**_K: K → KI
value
167. *kis*:KI-**set** = { **uid**_K(k) | k:K•k ∈ *ks* [37]}
  Mereology:
168. **mereo**_K: K → KM
  Attributes:
169. **attr**_CustId: K → CustId
170. **attr**_Possess: K → Possess
171. **attr**_OutReqs: K → OutReqs
172. **attr**_CustHist: K → CustHist

[37] *ks* was defined in Item 165 on the preceding page.

11.3 Customer Retrieval

175. The `retrieve customer` function, `retr_customer`, takes a customer identifier and in the context of the "global" customers state, *ks*,

176. yields the unique, ι, `k` with that identifier in *ks* that has that identifier.

value

```
159. retr_customer: KI×KS → K
161. retr_customer(ki)(ks) ≡ ι k:K • k ∈ ks ∧ uid_K(k)=ki
```

11.4 Customer Commands

We refer to Sect. 16.6.1 on page 73.

Chapter 12

Conveyor Companies

We remind the reader of Sect. 9.3 on page 42.

The purpose of a conveyor company is to provide conveyors for the transport of merchandise. It does so in an interaction between customers and logistics companies.

Conveyor companies has basically two main functions wrt. transport provision: a conveyor office and an entity which manages the day-to-day movement of conveyors. A derivative, "in-house" function may be that of logistics: the more-or-less optimal allocation of conveyor resources, routes, etc.

12.1 Conveyor Authorities.

We shall not consider the various public government conveyor authorities that "oversee" specific kinds of conveyor traffic. In many countries there are, for example, several railway operators, but the underlying rail net is usually operated by a [semi-]public government authority.

12.2 Conveyor Company Endurants.

12.2.1 Conveyor Company External Qualities

12.2.1.1 Sorts and Observers

From page 40 we repeat:

```
type
133. CKA
134. CKS = CK-set
135. CK
136. CA
137. CS = C-set
138. CO
139. oL = LI | nil
value
133. obs_CKA: T → CKA
134. obs_CKS: CKA → CKS
136. obs_CA: CK → CA
137. obs_CS: CA → CS
138. obs_CO: CK → CO
139. obs_oL: CK → oL
```

12.2.1.2 A Conveyor Company Taxonomy

In preparation for our presentation of describing "the state" of the conveyor company segment we show a taxonomy for the full structure of conveyor company parts in Fig. 12.1. The rendition is just an edited segment of Fig. 9.1 on page 37.

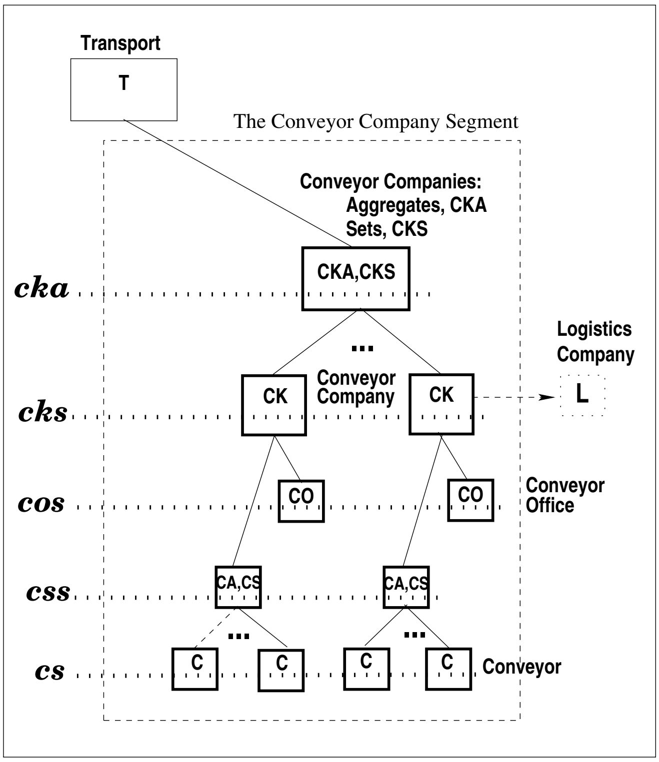


Figure 12.1: Conveyor Companies Taxonomy
We consider all parts to be manifest
Horizontal dotted lines indicate "state" components

12.2.2 A Conveyor Aggregate State Notion

There is the "global" transport domain value, t:T.

177. From t we can observe a likewise "global" *conveyor company aggregate* value, *cca*:CCA.

value
177. *cka*:CKA = **obs**_CKA(t)

178. From *cca* we can observe a likewise "global" *set of conveyor companies* value, *cks*:CKS.

value
178. *cks*:CKS = **obs**_CKS(*cka*)

179. From *cks* we can observe a likewise "global" *set of conveyors* value, *css*:CS-set.

value
179. *css*:CS-**set** = ∪{**obs**_CS(ck)|ck:CK•ck∈*css*}

180. From *ccs* we can observe a likewise "global" of all set of *conveyors* value, *cs*:`C-set`.

value
180. cs:C-**set** $= \cup\{$**obs**_CS(cs)|cs:CS•cs$\in cks\}$

181. From *cks* we can observe a likewise "global" set of *conveyor company offices* value, *cos*:`CO-set`.

value
181. cos:C-**set** $= \cup\{$**obs**_CO(cs)|cs:CS•cs$\in cks\}$

182. From *cks* we can observe a likewise "global" set of optional *logistics companies* value, *ols*:`oL-set`. They do not contribute to the conveyor company segment state.

value
182. ols:C-**set** $= \cup\{$**obs**_oL(ck)|ck:CK•ck$\in cks\} \setminus \{$**nil**$\}$

183. We can postulate an overall conveyor company state, σ_{CK}.

value
183. $\sigma_{CK} = \{cca\}\cup\{cks\}\cup css\cup cs\cup cos$

12.3 Conveyor Company Internal Qualities

12.3.1 Conveyor Company Identification

There are three issues here.

12.3.1.1 Conveyor Company Uniqueness of Identification.

The following conveyor companies parts have unique identifications:

184. the conveyor companies aggregate,
185. the conveyor companies set of conveyor companies
186. conveyor companies,
187. conveyors,
188. conveyor offices, and
189. optional logistics firms.

type
184. CCAI
185. CKSI
186. CAI
187. CI
188. COI
189. oLI

value
184. **uid**_CCA: CCA $\rightarrow$ CCAI
185. **uid**_CKS: CKS $\rightarrow$ CKSI
186. **uid**_CA: CK $\rightarrow$ CKI
187. **uid**_C: C $\rightarrow$ CI
188. **uid**_CO: CO $\rightarrow$ COI
189. **uid**_oL: oL $\rightarrow$ oLI

12.3.1.2 Conveyor Company Unique Identifier State.

190. We can postulate, cf. Item 183 on the previous page, an overall conveyor company unique identifiers state, $\sigma_{CK_{uid}}$.

value
190. $\sigma_{CK_{uid}}$ =
190. $\{$**uid**_CCA(cca)$\}$ [= cca_{ui}]
190. ∪ $\{$**uid**_CKS(cks)$\}$ [= $ccks_{uid}$]
190. ∪ $\{$**uid**_CK(ck)|ck:CK•ck∈$css$$\}$ [= cks_{uid}]
190. ∪ $\{$**uid**_C(cs)|c:C•cs∈$cs$$\}$ [= cs_{uid}]
190. ∪ $\{$**uid**_CO(co)|co:CO•co∈$cos$$\}$ [= cos_{uid}]

Where we use some non-RSL definitions of separate unique identifier sets – to be used in formulas 195–200 below.

12.3.1.3 Conveyor Company Uniqueness of Identification.

191. All conveyor company parts are uniquely identified.

axiom [Unique Conveyor Companies Parts]
191. **card**σ_{CK} = **card**$\sigma_{CK_{uid}}$

12.3.2 Conveyor Company Mereology

In the previous chapter Sect. 12.3.1, on unique identification, (pages 53-54), we treated all parts of the conveyor companies segment, as manifest. In the present chapter we shall only consider

- conveyor company set of conveyors, *cks*,
- conveyor company conveyors, *cs*, and
- conveyor company offices, *cos*,

as manifest.

192. The mereology of conveyor company sets of conveyors, are a pair of (i) the identities of the conveyors they "manage" and (ii) conveyor company, i.e., the conveyor company office they are"paired with".
193. The mereology of a conveyor is the identity conveyor company set of conveyors they "belong to".
194. The mereology of conveyor company office is a triplet: (i) the conveyor company sets of conveyors identity, (ii) a set of logistics company identities and (iii) a set of customers [who may handle their transport matters without the help of logistics firms].

type
192. CAM = CI-**set** × COI
193. CM = CAI
194. COM = CAI × LI-**set** × KI-**set**

value
192. **mereo**_CA: CA → CAM
193. **mereo**_C: C → CAI
194. **mereo**_CO: CO → COM

195. The `Well-formed Conveyor Company Mereologies` axiom has several clauses:
196. No two conveyor companies share [conveyor company sets of] conveyors.
197. The conveyor aggregate is correctly identified.
198. Conveyor, `c:C`, identities are those of actual conveyors,
199. **and** the identified logistics companies are actual
200. **and** the “k”ustomers are actual.

```
axiom [Well-formed Conveyor Company Mereologies]
196.  share_conveyors(cks)
195.  ∧ ∀ ck:CK • ck ∈ cks ⇒
              let (cai,lis,kis) = mereo_CO(ck),
                  cs = obs_CS(obs_CA(ck)) in
197.        cai=uid_CA(obs_CA(ck))
198.      ∧ {uid_C(c)|c:C•c ∈ cs} ∈ cs_uid
199.      ∧ lis ⊆ lis
200.      ∧ kis ⊆ kis
    end

196.   share_conveyors: CKS → Bool
196.   share_conveyors(cks) ≡
196.      ∀ ck,ck′:CK • ck≠ck′ ∧ {ck,ck′}⊆cks
196.            ⇒ obs_CS(obs_CA(ck))ck ∩ obs_CS(obs_CA(ck′)) ≠ {}
```

12.3.3 Conveyor Company Attributes

Conveyor Companies have a number of attributes. We mention a few:

201. General conveyor company information, which conveyors it manages, their timed routes, capacity, maximum load, etc.[38]

202. Resources: own and other conveyor companies' conveyors, their status, etc.

203. Contract history:

    (a) for every contract, once "on the move", which ways: from sending customer to node, from node to conveyor, from conveyor to node and from node to receiving customer[39].

204. Orders

    (a) by contract number

    (b) and an indexed set of offers,

    (c) each index being a choice number.

205. Current business: set of command messages.[40]

206. Past business: set of command messages.[41]

207. History: $\mathbb{TIME}$-stamped, chronologically ordered, descending sequence of `Events`: the messages received from customers and conveyors.

208. From choice and contract numbers one can observe the identity of the issuing conveyor company.

```
type
201.  ConvCompInfo = ...
202.  Resources = ...
203.  Contracts = ContractNu →m Move*
203a.     Move = (KI×NI)|(NI×CI)|(CI×NI)|(NI×KI)
204.  Orders = ContractNu →m Offers
204a.     ContractNu
204b.     Offers = ChoiceNu →m TR
204c.     ChoiceNu
205.  CurrBuss = MSG-set
206.  PastBuss = MSG-set
207.  CKHist = MSG*
value
201.  attr_ConvCompInfo: C → ConvCompInfo
203.  attr_Contracts: CK → Contracts
204.  attr_Orders: CK → Orders
205.  attr_CurrBuss: CK → CurrBuss
206.  attr_PastBuss: CK → PastBuss
207.  attr_CKHist: CK → CKHist

value
208.  xtr_CKI: (ChoiceNu|ContractNu) → CKI
```

[38] **Note:** The `conveyor company information` attribute contains "all" the information that is needed for the calculation of offers etc.
[39] **Note:** This conveyor company attribute is updated every time a conveyor [k12] and a customer [k15] acknowledges the transfer of merchandises
[40] **Note:** Received messages are "stashed" here for future handling – and removed once handled.
[41] **Note:** Handled [current business] messages here "stashed" here, transferred from the `current business` attributes.

12.3.3.1 Progress Updates

Conveyor companies are involved in many actions. Most of the actions [referred to by these commands] entail an update of conveyor companies' `Progress` attribute. Some directly by the conveyor companies. Others specifically initiated by [the] so-called `Acknowledgment` actions originating with customers and conveyors.

These explicit acknowledgments are of the form:

- **mk**_Acknowledgment($\mathbb{TIME}$,contract_number,(ui,uj))

  where:

- (ui,uj): (KI×CKI)|(CKI×KI)|(KI×NI)|(NI×CI)|(CI×NI)|(NI×KI)

The explicit acknowledgments entail updates to conveyor companies' `Progress` attribute:

209. The `upd_contracts` function takes a `contracts` attribute and an `acknowledgment` and yields an updated `contracts` attribute.

value

```
209. upd_contracts: Contracts → Acknowledgment → Contracts
209. upd_contracts(con)(mk_Acknowledgment(τ,cnu,(ui,uj))) ≡
209.     con † [cnu ↦ con(cnu)⌢⟨mk_Acknowledgment(τ,cnu,(ui,uj))⟩]
```

12.4 Conveyor Company Commands.

We refer to Sect. 16.8.2 on page 79.

Chapter 13

Conveyors, II

We have already dealt with conveyors: their external qualities, Sect. 5.1 on page 21, and two of their internal qualities, *unique identification*, Sect. 5.2 on page 22, and *mereology*, Sect. 5.3 on page 22. We shall, however, extend the mereology first sketched in Sect. 5.3 on page 22.

13.1 Conveyor Mereology

210. The mereology of a conveyor is a quadruple:

- the set of all identifiers of nodes and edges that the conveyor may travel;
- the set of all identifiers of conveyor companies that it may receive directives from and to which it shall have to acknowledge transfers of merchandises;
- the set of all identifiers of customers that it shall inform of pending collections and deliveries, and to which it shall deliver merchandises;

```
type
210.  CM = (NI|EI)set × CKI-set × KI-set
value
210.  mereo_C: C → CM
```

13.2 Conveyor Attributes

In Sect. 5.4 on page 23 we already touched upon some conveyor attributes.
We now extend these[42].

211. Conveyors are of kind [unchanged] [Static Attribute].
212. Conveyors convey, i.e., stores (holds), merchandises by contract number.
213. They follow a *service route*[43], `sr:SR` [programmable attribute] which is a path, of three or more node and edge identifiers – beginning with a node and ending with a node.
214. Conveyors "carry" and index attribute – `SRIndex` – – indicating as to where in the service route they, at present, are.
215. Conveyors also operate according to two "tables": for each node that it visits there are contracts to be unloaded, respectively loaded. This information is given to conveyors, at any time, by conveyor company directives.
216. Conveyors, having unloaded a contract at a final node informs the receiving customer of arrival. Note the difference between that attribute type name `Finals` (with a plural `'s'`) and the function argument identifier type `Final` (with no such plural).
217. Conveyors have, dynamically, a position – `CPos` – either they are at a node or are en route, i.e., on an edge between two adjacent nodes.
218. The `SR, SRIndex` and `CPos` must be commensurate: if index `i:SRIndex` designates a node `ni`, then `cpos:CPos` must be a `AtNode(ni)`, else, it designates and edge, `ej`, and `cpos:CPos` must be some `OnEdge(_,(_,ej),_)`.[44]
219. And conveyors have a history.
220. We omit further possible attributes: `Speed, Acceleration, Weight, ....`
221. These routes must be of the kind of the conveyors traveling them !

type
211. Kind
212. Stowage = ContractNu $\overrightarrow{m}$ M-**set**
215. TBU,TBL = NI $\overrightarrow{m}$ ContractNu-**set**
213. SR = Path
214. SRIndex = **Nat**
216. Finals = NI $\overrightarrow{m}$ (KI $\overrightarrow{m}$ ContractNu)
216. Final = NI × ContractNu × KI
217. CPos = [Item 85 on page 23]
219. CHist = MSG* [45]
220. ...

value
211. **attr**_Kind: Conveyor → Kind
211. **attr**_Stowage: Conveyor → Stowage
215. **attr**_TBU: Conveyor → TBU
215. **attr**_TBL: Conveyor → TBL
213. **attr**_SR: Conveyor → SR
214. **attr**_SRIndex: Conveyor → SRIndex
216. **attr**_Finals: Conveyor → Finals
217. **attr**_CPos Conveyor → Position
219. **attr**_CHist: Conveyor → CHist

axiom [Routes of commensurate kind]
221. [left to the reader !]
218. □ ... [left to the reader] ...

[42] Here we see a benefit from observing attributes, rather than explicitly defining the attributes of a part as a Cartesian of attributes.
[43] This service route concept reflects that the conveyor, at any time, may carry merchandise from many distinct contracts.
[44] The joint `i:SRIndex` and `cpos:CPos` may be a bit too much, but they come in conveniently for our subsequent formalizations.
[45] The messages are those directed at or emanating from conveyors

13.3 Conveyor Commands.

We refer to Sect. 16.8.2 on page 79.

Chapter 14

Logistics Companies

We remind the reader of Sect. 9.3 on page 42.

The purpose of a logistics company is to arrange of transportation. It does so in interaction between customers and conveyor companies.

The functions of logistics companies very much overlaps with some of the functions of conveyor companies.

An "extreme" example of a logistics company is that of a *travel agency* !

We shall, however, not pursue the logistics concept further – since its role is also played by conveyor companies.

Part III

A MULTI-MODE TRANSPORT: INTENTIONAL PULL

Chapter 15

Intentional Pull, II

TO BE WRITTEN

Part IV

A MULTI-MODE TRANSPORT: COMMANDS

Chapter 16

Multi-mode Transport Commands

16.1 Events and Commands

We distinguish events from commands:

Events are perdurants. The "occur instantaneously". At "their own" volition. In a state[46] and possibly cause a state change. Some events, the internal events, have their "root" in the [part] behaviour, hence "affect" the attributes of the underlying part. Other events, the external events, have their "root" "outside" the [part] behaviour, but may "affect" the attributes of the underlying part.

Commands are syntactic entities. Commands are "issued"[47] by part behaviours They "occur" as the result of actions taken by [receiving] part behaviours. They have a syntax. They constitute a script facet[48] related to the part [behaviour]. They have a semantics. The semantics of commands is expressed by behaviour actions. We distinguish between directive commands and response commands. Directive commands are issued by a part behaviour and is directed at another part behaviour. Response commands are acted upon by a part behaviour in response to a command issued by another part behaviour. For both kinds of commands there are thus at least two behaviours involved in expressing their semantics.

16.2 Command Traces

In order to describe the very many commands it has proven useful to sketch a possible diagram of command traces. Figure 16.1 on the next page[49] shows schematically a possible trace of commands. The ordering, "**i**" in **ki**, shall indicate some temporal ordering of the issue of these commands.

We shall elaborate on the transport behaviours – with reference to Fig. 16.1 on the next page.[50]

- **k1** After some preparatory work a sending customer inquires as to possible transport at a chosen conveyor or logistics company.
- **k2** After some preparatory work the conveyor or logistics company replies to the inquiry.
- **k3** After some preparatory work the customer places and order for transport.
- **k4** After some preparatory work the chosen conveyor or logistics company confirms the order,
- **k5** which the customer now [likewise firmly] accepts – with payments.
- **k6** At some point logistics companies hand over customer orders to [respective] conveyor companies.
- **k7** After some preparatory work these conveyor companies, one or more, select a the set of conveyors and inform them of the order, i.e., give them directives.
- **k8** The conveyor company, at some time after [k7] informs the customer that a designated node is ready to accept its merchandises for transport – "on hold", at a node.

[46] By 'state' we shall, in the context of perdurants, mean the value of all dynamic attributes of all behaviours.

[47] By "issued" we shall here mean that they are communicated, in the style of CSP communications by behaviours directed at other behaviours.

[48] For *facets* and *scripts* see [7, *Chap. 8*].

[49] In Fig. 16.1 on the following page we have "merged" the logistics company handling of commands with that of the conveyor company handling – as there is some "overlap" in their functionalities.

[50] That is: figures like Fig. 16.1 on the following page are not given a semantics. The "semantics" of Fig. 16.1 on the next page "transpires from the entire formal model of this report.

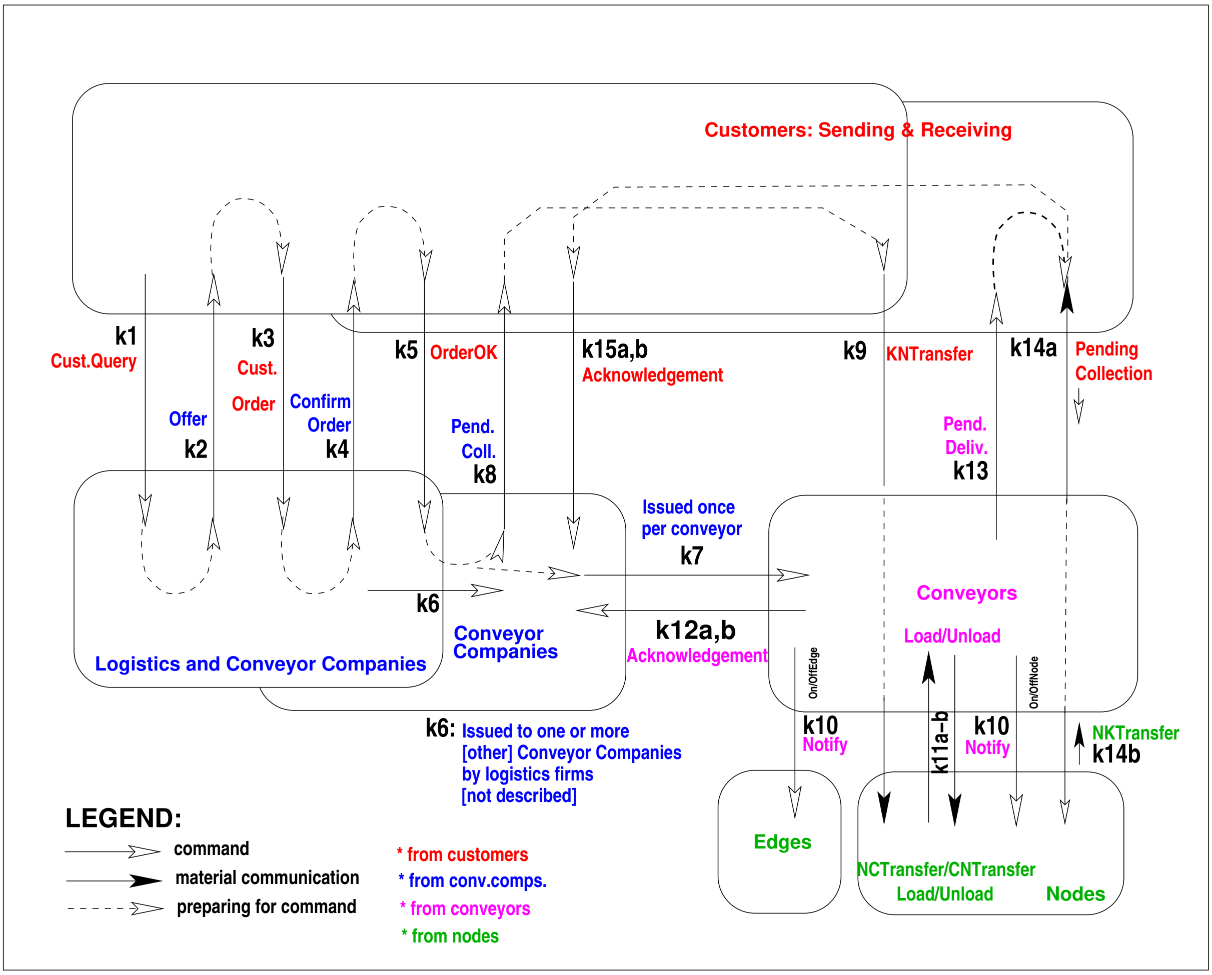


Figure 16.1: Command & Material Traces [→]

k9 Having been so notified by a conveyor the customer delivers the merchandises, to be transported, at a node, to be "on hold" for the conveyor.

k10 Conveyors, "on the move", notify edges and nodes of their presence.

k11 In synchronous communications conveyors exchange merchandises with nodes: either loading ([k11a]) or unloading ([k11b]).

k12 Those conveyors inform their companies of transfers.

k13 The "last" conveyor notifies the "end" customer receiver of pending arrival.

k14 Having been notified, by the conveyor, the "end" customer receives the transported merchandises.

k15 That customer informs the [final] conveyor company of the [final] transfer.

16.3 An Analysis

We now analyze Sect. 16.2 on the preceding page.

It seems tat there are four kinds of "commands": *ab initio*, *deferred*, *triggered* and *cascaded*.

- **Ab Initio:** There is only one command of this category: the customer query command, [k1].

  Customers, at their own instigation, that is, internal non-deterministically, decides to have some merchandises transported.

- **Deferred:** Most commands are of this category: they are implied by issue of other, that is, "previous" [k2] thus follows from [k1], [k3] from [k2], etc.

  There is no guarantee that [k2] will occur. The conveyor (or logistics) company may simply ignore that it has received [k1], respectively [k3] may not occur in response to [k2]. Etcetera.

- **Triggered:** "Commands" [k11a] and [k11b] are not "directly issued", external non-deterministically,"at some time" in response to [k7].

  [k7], such as we small model it, shall result in conveyors having an appropriate attribute, the *to be loaded* and *to be unloaded*, containing such information as when conveyors at nodes shall *load* and *unload* merchandises – and when conveyors are **At** such **Node**s, this attribute information is said to **trigger** these merchandise transfers.

- **Cascaded:** [k8] is issued either at the same time as [k7], or shortly thereafter. [k9] is issued when [k8] has been received – after which a first [k15] is issued.

16.4 Material and "Immaterial" Commands

Kommands **k1**-**k8, k10, k13, k15** and **k18** are "immaterial" in that they "just" communicate information. Commands **k9, k11** and **k14** are "material" in that they, besides information (data) also communicate, i.e., physically transfer material, i.e., merchandises.

16.5 Abstracting an Essence of Transport

By *"abstracting an essence of transport"* we mean that a number of transport "details" are omitted for "the benefit" of emphasizing "other details" ! For examples: (i) we omit details of the structure and contents of what is to be transported, (ii) keeping, somehow, details of who is sending, the address, by whom the merchandise is to be received, etc., (iii) omitting details of merchandise, identification, quantity, weight, value, etc., (iv) cost, payments, etc. In the description of commands, below, we therefore abstract "to the core" these commands – assuming that the various "actors": the customers, the logistics and conveyor companies and the conveyors can otherwise, i.e., somehow "find out" !

16.6 Commands – A First View

As You see, there are many commands. In this section we shall *"take an abstract view of these"* before, in Sect. 16.8 we go into the detailing of these commands This "abstract view" should then enable us to "design", as it were, a systematic form and set of less abstract commands.

16.6.1 Customer Commands, I

222. **k1** Customers inquire either logistics companies or conveyor companies about many things, for example time-tables, cost, etc., for the transport of merchandises from one customer to another, etc.

223. **k3** Customers place orders, with either logistics companies or conveyor companies for the transport – according to some offers, **k2**, made by these.

224. **k5** Customers "signs" the **k4** offer.

225. **k9** Customers deliver merchandise to nodes.

226. **k15** Customers acknowledge receipt of merchandises.

type

```
222. [k1]   CustQuery
223. [k3]   CustOrder
224. [k5]   OrderOK
225. [k9]   CustDel
226. [k15]  Acknowledgment
```

16.6.2 Conveyor Company Commands, I

227. **k2** Conveyor companies place an offer for transport in response to an inquiry, **k1**.
228. **k4** Conveyor companies OKs an order in response to an customer order, **k3**.
229. **k7** Conveyor companies inform conveyors of orders, **k4**, to be carried out.
230. **k8** Conveyor companies inform customers of pending collection of merchandises.

type
```
227. [k2] ConvCompOffer
228. [k4] ConvCompOrdOK
229. [k7] ConvCompConvDir
230. [k8] PendColl
```

16.6.3 Conveyor Commands, I

231. **k8** Conveyor notify customers of pending collection.
232. **k10** Conveyor notify edges and nodes of its presence.
233. **k11a-k11b** Conveyor transfers merchandises to and from node.
234. **k12** Conveyor acknowledges conveyor company of merchandise transfer.
235. **k13** Conveyor informs customer of pending delivery.

```
232. [k10]   Notify
233. [k11a]  CNTransfer
234. [k12]   Acknowledgement
235. [k13]   PendDel
```

• • •

Conveyors collect and deliver merchandise not only from and to nodes, but also from and to other conveyors. Therefore the **k10–k15a-b.** sequence of commands also takes place between distinct conveyors.

16.6.4 Logistics Company Commands

We shall skip this section,

16.7 TR: Transport Routes

We may have *"abstracted too much"* in Sect. 16.6. For example, where in the conveyor company and logistics company to customer order OK commands is the information "hidden" that outlines the course of actions: which route to take, with which conveyors, at which approximate times ? That information may be formalized:

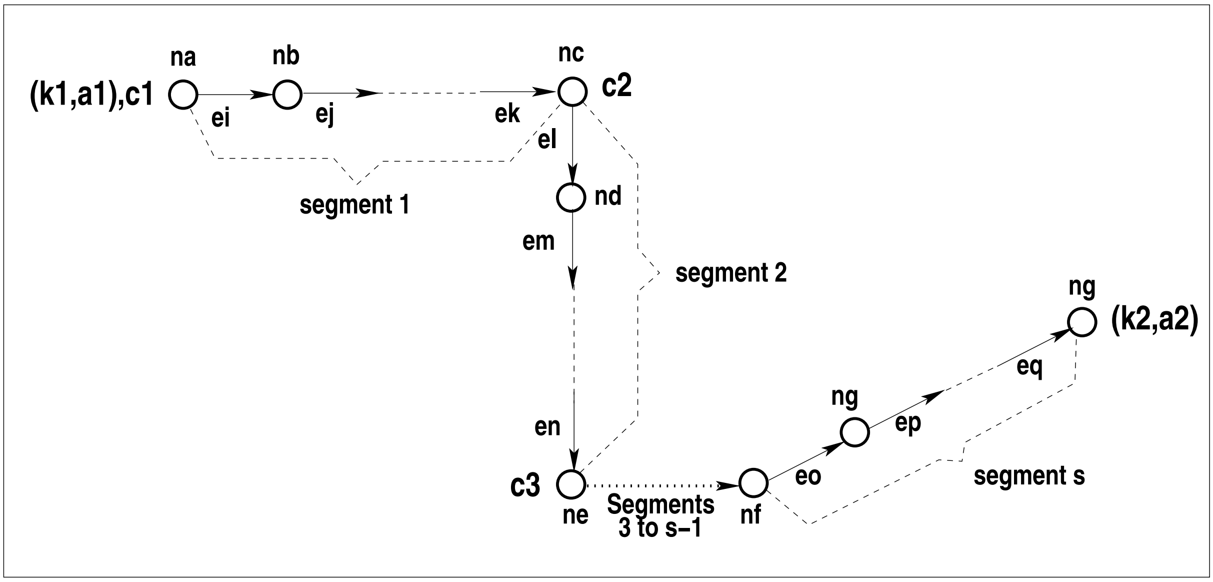


Figure 16.2: A Transport Route: k:kustomer, c:conveyor, a:address, n:node, e:edge

236. A *transport [route]* is a composite of
237. first a sending customer's identifier and place of pick-up (**(k1,a1)**);
238. then the storage: a non-empty set of unique identifiers of the merchandises transported – indexed by contract number;
239. followed by a sequence of one or more segments (**segment 1, segment 2, ..., segment n**)–
240. each segment beginning with a conveyor (**c1, c2, ..., c3**) identifier, then a node identifier (**na, nc, ..., ne**), and finally a non-empty edge-node-path –
241. an edge-node-path is sequence of alternating edge and node identifiers (⟨**ei, nb, ej, ..., ek, nc**⟩);
242. finally ending with a receiving customer's identifier and place of delivery (**k2,a2**)).
243. The two addresses must be different **a1**≠**a2**.
244. The paths formed by edge-node-paths headed by a, i.e., the, node identifier must be paths of the transport net[51],
245. and these paths must be of the same kind as the conveyor for those paths.
246. The time ordering is strictly ascending –
247. and the "end" node of one segment must match, i.e., be equal to the "beginning" node of the next segment.
248. The storage must be well-formed: no two contracts identify the same merchandises.
249. From a contract number one can observer, i.e., extract, the issuing conveyor company identifier.

```
type
236.  TR =  s_sndr:(KI × Addr)
238.     × s_cos:(ContractNu →m MI-set) axiom ∀ mis:MI-set • mis≠{}
239.     × s_sgl:Segment*  [axiom ∀ sl:Segment*•sl≠⟨⟩]
242.     × s_rcvr:(KI × Addr)
240.  Segment = 𝕋𝕀𝕄𝔼 × CI × NI × Edge_Node_Path
241.  Edge_Node_Path = (s_ei:EI×s_ni:NI)*  axiom ∀ enp:Edge_Node_Path•enp≠⟨⟩
238.     ContractNu
value
249.  xtr_CKI: ContractNu → CKI
axiom
238.  ∀ tr:TR • let cos=s_cos(tr) in ∀ cnu:dom cos•xtr_MIs(cnu)=cos(cnu) end
```

Wellformed Transports[52]

```
axiom [Wellformed Transports]
243.  ∀ ((__,a1),__,sl,(__,a2)):TR • a1≠a2 ∧
242.    ∀ seg:Segment•seg∈ elems sl ⇒
242.      ∀ (__,ci,ni,enp):Segment•(ci,enil,ei)∈ elems enp
244.        ∧ ⟨ni⟩⁀enp ∈ paths ∧ enil∈paths
245.        ∧ same_kind(enp,ci)
246.        ∧ ∀ i,i+1•{i,i+1}⊆inds sl ⇒
246.            let (τi,ci,ni,enp) = sl[i], (τj,cj,nj,enpj) = sl[i+1] in τi<τj
247.                ∧ s_ni(enpi[len enp]) = nj end
247.   ∀ storage:(ContractNu →m MI-set) •
247.      ∀ cni,cnj:ContractNu • {cni,cnj}⊆dom storage ∧ cni∼−cnj
247.          ⇒ storage(cni)∩storage(cnj)={}
```

[51] Cf. Item 52 on page 16

[52] Axiom 243–247 must be carefully checked

Auxiliary Functions

```
value
245.  same_kind:  Edge_Node_Path × CI → Bool
245.  same_kind(enpath,ci) ≡ ... [Left to the reader]
```

An aspect of the transport routes, `tr:TR`, when a transport route has more than one segment, is that the node between two adjacent segments, serve as a repository for merchandises. A conveyor unloading merchandises destined for other, one or more, conveyors may not arrive when either or all of these conveyors have arrived[53], so they deposit, put "on hold", those merchandises. For respective `kind`s of nodes these "deposit holds" are, for example, called *bus stops* for kind **road**, *train station waiting rooms* for kind **rail**, *airport passenger lounges* for kind **air**, and *container terminals* for kind **sea**.

`Segments` (Item 240 on the facing page) are static descriptions of where conveyors are to move. `Service Routes`, `SRs` (Item 213 on page 60), are static descriptions of when conveyors are to move.

16.8 A Closer Analysis of Commands

We refer back to the overview of all commands given in Sect. 16.6.

16.8.1 Customer Commands, II

222. For a customer to formulate a proper query about possible transports such a query must contain the following information:
    (a) a unique, customer-chosen inquiry identification and
    (b) a query compound.
250. The query compound, it seems, should contain such information as:
    (a) name, address, and other such data that "pin-points", "validates" the inquirer;
    (b) characterization of the merchandise to be transported: product information, quantity, total weight, total volume, total value [for insurance purposes], etc.;
    (c) time interval of transport;
    (d) from where to where;
    (e) expected cost frame; and, possibly, more !
    (f) Addresses are further unspecified.

```
type
222.  [k1]  CustQuery ::
249a.          QueryId
249b.          × QueryComp

250.  QueryComp =
250a.    Addr
250b.   × MInfo [ ... ]
250c.   × TI    [ = (TIME×TIME), axiom ∀ (ft,tt):TI•ft<tt ]
250d.   × FT    [ = NI×(NI×KI×AddrInfo), axiom ∀ (nf,(nt,_,_)):FT•nf≠nt ]
250e.   × ExpCost
250f.       Addr
```

[53] The conveyor or logistics company, when preparing the offers, are assumed to make sure that there is appropriate time intervals between unloading and loading conveyors for relevant merchandises.

223. For a customer to formulate a proper order for a specific transport such a query must be based on the conveyor or logistics company offer to a query like that outlined in Item 250, above, the order must contain the following information:

    (a) the customer inquiry identification, and
    (b) a reference to the logistics or conveyor company contract number given in query reply.
    (c) Then more-or-less the same information, formulated as a compound, as given in the original query – which is also expected to be contained in the reply offer;
    (d) name, address, etc.,
    (e) merchandise information,
    (f) precise times.
    (g) from-to transport details,
    (h) the offered cost,
    (i) etc.

251. From a query identifier one can extract the customer identity.

type
```
223.  [k3]  CustOrd ::
250a.          QueryId
250b.          × ContractNu
250c.          × OrdrComp

250c.  OrdrComp =
250d.       Addr
250e.     × MerchInfo
250f.     × TI
250g.     × FT
250h.     × Cost
250i.     × ...
```
value
```
251.  xtr_KI: QueryId → KI
```

224. For a customer to OK a proposed transport the the customer must provide

    (a) the contract number,
    (b) the choice number,
    (c) payment.

```
224.  [k5]  OrderOK ::
251a.            ContractNu
251b.           × ChoiceNo
251c.           × Payment
```

224. For a customer to deliver the merchandises according to the contracted order the customer must provide

    (a) a reference to to the contract number and
    (b) the therein indicated number of actual merchandises !

type
```
224.  [k9]  KNTransfer ::
251a.          ContractNu
251b.         × M-set
```

252. [k15] A customer having received merchandises (from another customer via conveyors) at a node acknowledges this receipt by so informing the conveyor company.

type

```
252.   [k15] Acknowledgment ::  TIME×ContractNu×(NI×KI)
```

Observe that the first two commands and the last command were strictly "informational", i.e., syntactic, whereas the Customer tDelivery command is "rather" physical:, i.e., semantic: the command, so-to-speak, "embodies" an action, the manifest movement of volumes of possibly heavy material !

There may be other customer commands – such as inquiring as to the progress of an actual transport, etc. We leave that to the reader.

16.8.2 Conveyor Commands, II

The conveyor commands, first outlined in Sect. 16.6.3 on page 74, are now summarized and detailed. First we list their treatment in Sect. 16.6.3 on page 74.

231. **k8** Conveyor informs customer of pending collection.
232. **k10** Conveyor notifies edges and nodes of conveyor presence.
233. **k11** Conveyor transfers (loads [k11a], unloads [k11b]) merchandises.
234. **k12** Conveyor acknowledges conveyor company of merchandise transfer.
235. **k13** Conveyor informs customer of pending delivery.

```
231.  [k8]      PendColl
232.  [k10]     Notify
233.  [k11a,b]  Transfer = CNTransfer | NCTransfer
234.  [k12]     Acknowledgment
235.  [k13]     PendDel
```

253. [k8] Conveyors inform either a customer of pending collection of merchandises.

     They do so by simply mentioning the contract number and the set of unique identifiers of the merchandise to be collected.

254. [k10] Conveyors notify edges and nodes of their presence.

Conveyors transfer:

255. [k11a] load from a node.
256. [k11b] or unload merchandises to a node.

They do so by stating the contract number and presenting the set of merchandise to be transferred.

257. [k12] Conveyors, time-stamped, acknowledges its company of, and at the completion of a transfer, collection or delivery of merchandise. They do so by mentioning the contract number and the two "parties" to the transfer:
258. either a customer and a node, or a of conveyor and a node.
259. [k13] Conveyors inform customers of pending delivery (at a node).

type

```
253.  [k8]    PendColl ::  (NI×(ContractNu> MI-set))
254.  [k10]   Notify ::  AtNode | OnEdge
255.  [k11a]  NCTransfer ::  (ContractNu×M-set)
256.  [k11b]  CNTransfer ::  (ContractNu×M-set)
257.  [k12]   Acknowledgment ::  TIME×ContractNu×FromTo
258.           FromTo = (NI×CI)|(CI×NI)
259.  [k13]   PendDel ::  (NI×(ContractNu×MI-set))
```

16.8.3 Conveyor Company Commands, II

Review:

```
type
ι227 π74.  [k2]  ConvCompOffer
ι228 π74.  [k4]  ConvCompOrdOK
ι229 π74.  [k7]  ConvCompConvDir
```

We now detail these.

260. An offer for transport must state
    (a) the conveyor company identity;
    (b) a contract[54] number;
    (c) refer to an inquiry, for example by stating its number or by repeated it; and
    (d) a set of zero, one or more choice number indexed offer-choices.
    An offer-choice
    (e) a timed route of transport, and
    (f) a cost.

261. An OK, binding acknowledgment of an order must state
    (a) the conveyor company identity,
    (b) a contract number,
    (c) refer to an offer and choice number,
    (d) "repeats" the contracted timed route of transport,
    (e) and the cost.

262. The conveyor company information to be given to conveyors of orders, **k4**, state
    (a) the conveyor company identity;
    (b) a contract number and
    (c) the contracted time route of transport.

263. From Offer numbers, contract numbers and choice numbers one can extract the offering and contracting company's identity

264. as well as the identity of the customer being offered and contracted.

```
type
260.   [k2] ConvCompOffer ::  CKI×ContractNu×QueryNu×(ChoiceNu →m OfferChoice)
260b.           ContractNu
260d.           ChoiceNu
260e.           OfferChoice = TR× ost

261.   [k4] ConvCompOrdOK ::  CKI×ContractNu×ChoiceNu×TR×Cost

262a.  [k7] ConvCompConvDir ::  CKI×ContractNu×Segment

value
263.   xtr_CKI: (OfferNu|ChoiceNu|ContractNu) → CKI
264.   xtr_KI:  (OfferNu|ChoiceNu|ContractNu) → KI
```

[54]– even though this may not result in a contract

16.8.4 Node Commands

Nodes, as behaviours, have now become reactive. They store contracted merchandises – "on hold between" conveyors. So they must react to conveyor commands requesting merchandises, unloaded, to be put "on hold" or fetched, to be loaded. They react by accepting and delivering merchandises from, respectively to conveyors and customers. To these requests node behaviours must react [immediately (?)]. These are the only transport commands that must be so synchronized[55]. All other transport commands are "buffered"[56]

265. [k14] Nodes transfer merchandises (from another customer via conveyors) from the 'on-hold' of a node to a customer.

type

```
265.  [k14] NKTransfer ::  NI×ContractNu
```

16.8.5 Edge Commands

Thee are no edge commands. Edge behaviours receive notifications from conveyors as to their presence on edges.

[55] **Alert:** Check that I actually describe so !

[56] **Alert:** Perhaps one should reconsider the customer to conveyor and conveyor to customer transfers of merchandises to also be synchronized.

Part V

IDENTITIES

Chapter 17

Identities

So far we have introduced a variety of identities:

- unique identities of endurants,
- query ‘numbers’,
- offer ‘numbers’,
- contract numbers,
- etc.

These are, of course, not identifiers nor numbers or numerals. They are abstract entities.
We can say a lot about these:

266. From the identity of a customer we can "extract" (i.e., "observe") such things as the name of the customer, the address (road name & number, district name, city name, county name, country name, telephone 'numbers', e-mail addresses, etc., etc.).

267. From the identity of a conveyor we can 'extract' the identity of its owner: a conveyor company.

268. From a query 'number' we can extract the identity of the querying customer.

269. From offer, order and contract 'numbers' we can extract the identities of conveyor (logistics) company and customer identities.

270. From a contract number we can extract the set of merchandise identifiers "involved" in the identified contract.

271. From a contract number we can extract a waybill[57],

272. From a contract number we can extract a a bill-of-lading[58].

273. From a contract number we can observe whether it (i.e.,the waybill/bill-of-lading) represents a ticket for human "merchandise" (cf. Sect. 10.3 on page 45).

274. Et cetera.

```
value
266.  xtr_Name:  KI→Name
266.  xtr_Addr:  KI→((RoadNam×Nat)×DisNam×CounNam×LandNAm×PhonNu×Email×...)
267.  xtr_CKI: CI→CKI
268.  xtr_CI: QueryNu→CI
269.  xtr_CKI: (OfferNu|OrderNu|ContractNu)→CKI
269.  xtr_CI: (OfferNu|OrderNu|ContractNu)→CI
270.  xtr_MIs:  ContractNu→MI-set
type
266.  RoadNam, DisNam, CounNam, LandNam, PhonNu, Email
271.  WayBill
272.  BoL
value
271.  xtr_WayBill:  CKI→WayBill
272.  xtr_BoL: CKI→BoL
273.  is_Ticket:  (WayBill|BoL)→Bool
```

MORE TO COME

[57] A waybill is a document issued by a carrier acknowledging the receipt of goods by the carrier and the contract for shipment of a consignment of that cargo. Typically it will show the names of the consignor and consignee, the point of origin of the consignment, its destination, and route [Wikipedia].

[58] 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. Although the term is historically related only to carriage by sea, a bill of lading may today be used for any type of carriage of goods. 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.[a] Whereas a bill of lading is negotiable, both a policy and an invoice are assignable [Wikipedia].

Part VI

A MULTI-MODE TRANSPORT: BEHAVIOURS

Chapter 18

Multi-mode Behaviours

Contents

18.1 Communication

275. There is a medium for synchronization of and communication between behaviours.

276. **comm**[{ui,uj}] ! value expresses an event [an action]: the "output" of `value`, from the behaviour identified by `ui` towards the behaviour identified by `uj`.

277. **comm**[{ui,uj}] ? expresses a value, i.e., the "input" of a value, from the behaviour identified by `ui` by the behaviour identified by `uj`.

channel

```
278.    { comm[{ui,uj}] | ui,uj:UI • {ui,uj}⊆σ_uis } :  MSG
```

278. The **comm** channel declaration above expresses that this medium is "two-dimensional" and communicates ("mediates") messages of type `M`.

279. Messages are timed commands

280. and the commands are those of `customers, conveyor companies, logistics companies` and `conveyors`.

type

```
279.       MSG = (UI×TIME×UI)^59 × Command
279.       UI =KI|CKI|CI
279.  [k1] Command = CustQuery         [Customer→Company]
279.  [k3]           | CustOrd          ...
279.  [k5]           | OrderOK          ...
279.  [k15]          | Acknowledgment   ...
279.  [k9]           | KNTransfer       [Customer→Node]
279.  [k14a]         | PendColl         ...
279.  [k2]           | ConvCompOffer    [Company→Customer]
279.  [k4]           | ConvCompOrdOK    ...
279.  [k8]           | PendColl          ...
279.  [k7]           | ConvCompConvDir  [Company→Conveyor]
279.  [k12]          | Acknowledgment   [Conveyor→Company]
```

```
279.  [k13]    | PendDeliv    [Conveyor→Customer]
279.  [k11a]   | CNTransfer   [Conveyor→Node]
279.  [k10]    | Notify       ...
279.  [k11b]   | NCTransfer   [Conveyor→Edge]
279.  [k10]    | Notify       ...
279.  [k11b]   | NCTransfer   [Node→Conveyor]
279.  [k14]    | NKTransfer   [Node→Customer]
```

• • •

A core property of CSP is that behaviours both **synchronize** their behacious and **exchange** messages, from one, **!**, to another, **?**.

18.2 Behaviour Signatures

We omit consideration of aggregate and merchandise behaviours. There are:

281. the customer behaviours,
282. the logistics company behaviours,
283. the conveyor company behaviours,
284. the conveyor behaviours,
285. the edge behaviours, and
286. the node behaviours.

In some other order their signatures are:

```
value
281.  customer: KI                                              [identifier]
281.  → KM[61]                                                  [mereology]
281.  → (CustId × AddrInfo × ...)                               [static attrs.]
281.  → (Possess × OutReqs × CustHist) Unit                     [progr. attrs.]

283.  conv_comp: CKI →                                          [identifier]
283.  → CKM                                                     [mereology]
283.  → (ConvCompInfo × ...)                                    [static attrs.]
283.  → (Resources×Contracts×Orders×CurrBuss×PastBuss×CKHist) Unit [progr. attrs.]

284.  conveyor: CI                                              [identifier]
284.  → CM                                                      [mereology]
284.  → (Kind × ...)                                            [static attrs.]
284.  → (Stowage×TBU×TBL×SR×SRIndex×Final×CPos×CHist) Unit      [progr. attrs.]

282.  logistics: LI →                                           [identifier]
282.  → LM                                                      [mereology]
282.  → (LogisticsCompInfo × ...)                               [static attrs.]
282.  → (PastBusiness × CurrBusiness × LHist) Unit              [progr. attrs.]

285.  edge: EI                                                  [identifier]
285.  → EM                                                      [mereology]
285.  → (EdgeKind × LEN × COST × ...)                           [static attrs.]
285.  → EHist Unit                                              [progr. attrs.]

286.  node: NI                                                  [identifier]
286.  → NM                                                      [mereology]
286.  → (NodeKind × ...)                                        [static attrs.]
286.  → (OnHold × NHist) Unit                                   [progr. attrs.]
```

[59]The triplet: (fui,t,tui) is subject to the following constraint, which we leave to the reader to formalize: if tui:KI then tui:CKI or tui:CI; if tui:CKI then tui:KI or tui:CI; if tui:CI then tui:KI or tui:CKI.

18.3 Which Behaviours to Describe ?

We treat the transcendentally deduced behaviours of some, but not all, the manifest parts: `customers`, `conveyor companies`, but **not** their `conveyor company offices` **nor** their `conveyor aggregates`, but their `conveyors`. We omit, also treatment of `Logistics companies` as their "function" is "very much like, i.e., "overlapping" with, that of `conveyor companies`.

• • •

The arrangement of the [narrative & formal] descriptions is by endurant, i.e., part, type; but the "reading" of these should be by pairs: each pair represents an arrow in Fig. 9.1 on page 37, one of the pair represents the source of the arrow, the "sending" behaviour, the second of the pair represents the target of the arrow, the "receiving" behaviour,

18.4 Multi-mode "Systems"

We can initialize a domain, and we can instatiate a domain.

18.4.1 Multi-mode Domain Initialization

287. An initialization of a transport domain means the parallel composition of the
288. parallel composition of the initialization of all customer behaviours with the
289. parallel composition of the initialization of all conveyor company behaviours with the
290. parallel composition of the initialization of all conveyor behaviours with the
291. parallel composition of the initialization of all logistics behaviours with the
292. parallel composition of the initialization of all edge behaviours with the
293. parallel composition of the initialization of all node behaviours.

```
287. instantiation: Unit → Unit
287. instantiation() ≡
288.   ∥ { customer(uid_K(k))
288.           (mereo_K(k))
288.           (attr_CustId(k),...)
288.           ([],{},⟨⟩)
288.      | k:K • k∈ks } [ks, see Item 132 on page 39]
288.   ∥
289.   ∥ { conv_comp(uid_CK(ck))
289.           (mereo_CK(ck))
289.           (attr_ConvCompInfo(ck),...)
289.           (attr_Resources(c),[],[],{},{},⟨⟩)
289.      | ck:CK • ck∈cks  } [cks, see Item 178 on page 52]
289.   ∥
290.   ∥ { conveyor(uid_C(c))
290.           (mereo_C(c))
290.           (attr_Kind(c),...)
290.           ([],[],[],attr_SR(c),1,[],attr_Position(c),⟨⟩)
290.      | c:C • c∈cs  } [cs, see Item 180 on page 53]
290.   ∥
291.   ∥ { logistics( ... ) | ... }  [see remark on page 111]
291.   ∥
292.   ∥ { edge(uid_E(e))
292.           (mereo_E(e))
292.           (attr_EdgeKind(e),attr_LEN(e),attr_COST(e),...)
292.           (⟨⟩)
292.      |  e:E • e∈es } [es, see Item 26 on page 12]
292.   ∥
293.   ∥ { node(uid_N(n))
293.           (mereo_N(n))
293.           (attr_NodeKind(n),...)
293.           ([],⟨⟩)
293.      |  n:N • n∈ns } [ns, see Item 27 on page 12]
```

18.4.2 Multi-mode Domain Instantiation

```
287. instantiation: Unit → Unit
287. instantiation() ≡
288.   ‖ { customer(uid_K(k))
288.           (mereo_K(k))
288.           (attr_CustId(k),...)
288.           (attr_Possess(k),attr_OutReqs(k),attr_CustHist(k))
288.       | k:K • k∈ks } [ks, see Item 132 on page 39]
288. ‖
289.   ‖ { conv_comp(uid_CK(ck))
289.           (mereo_CK(ck))
289.           (attr_ConvCompInfo(ck),...)
289.           (attr_Resources(c),attr_Contracts(ck),attr_Orders(ck),
289.               attr_CurrBuss(ck),attr_PastBuss(ck),attr_CKHist(ck))
289.       | ck:CK • ck∈cks  } [cks, see Item 178 on page 52]
289. ‖
290.   ‖ { conveyor(uid_C(c))
290.           (mereo_C(c))
290.           (attr_Kind(c),...)
290.           (attr_Stowage(c),attr_TBU(c),attr_TBL(c),attr_SR(c),
290.               attr_SRIndex(c),attr_Final(c),attr_Position(c),attr_CHist(c))
290.       | c:C • c∈cs  } [cs, see Item 180 on page 53]
290. ‖
291.   ‖ { logistics( ... ) | ... }  [see remark on page 111]
291. ‖
292.   ‖ { edge(uid_E(e))
292.           (mereo_E(e))
292.           (attr_EdgeKind(e),attr_LEN(e),attr_COST(e),...)
292.           (attr_EHist(e))
292.       |  e:E • e∈es } [es, see Item 26 on page 12]
292. ‖
293.   ‖ { node(uid_N(n))
293.           (mereo_N(n))
293.           (attr_NodeKind(n),...)
293.           (attr_OnHold(n),attr_NHist(n))
293.       |  n:N • n∈ns } [ns, see Item 27 on page 12]
```

We refer to Sect. 7.5 on page 32 for a first example of domain initialization.

Chapter 19

Customer Behaviours

Contents

19.1 Main Behaviour

19.1.1 Overall Behaviour

294. The customer internal non-deterministically alternates between being
   (a) a private entity, doing whatever,
   or possibly
   (b) [k1][62] querying conveyor or logistics companies about a possible transport;
   (c) [k3] examining a conveyor or logistics company offer;
   (d) [k5] accepting an offer from a conveyor or logistics company;
   (e) [k9] delivering merchandises to nodes;
   (f) [k14] requesting contracted `onhold` merchandises from nodes,and
   (g) external non-deterministically possibly receiving messages from conveyor companies or logistics companies, conveyors, or nodes ([k2,k4,k8,k13,k14]).

u The [k1] query is pivotal. It "sets everything else in motion". Responses from the `conveyor company` are "temporarily stored", cf. *customer receives messages*, i.e., `cust_receiv_messages`, Item 294g. "Storage" is in the form of an additional behaviour argument.

value
```
294.  customer(ki)(cm)(kid,kaddr)(po,or,ch) ≡
294a.       ...
294b.  [k1]  ⌈⌉ cust__issues_query(ki)(cid,...)(...)(po,or,r,ch)
294c.  [k3]  ⌈⌉ cust_issues_order(ki)(cid,...)(...)(po,or,r,ch)
294d.  [k5]  ⌈⌉ cust_order_OK(ki)(cid,...)(...)(po,or,r,ch)
294e.  [k9]  ⌈⌉ cust_delivers_merchandises(ki)(cid,...)(...)(po,or,r,ch)
294e.  [k14] ⌈⌉ cust_requests_merchandises(ki)(cid,...)(...)(po,or,r,ch)
294g.        ⌈⌉ cust_receives_messages(ki)(cid,...)(...)(po,or,r,ch)
```

[62]The bracketed numbers refer to those of Fig. 16.1 on page 72.

19.1.2 Overall Reactive Behaviour

295. The external non-deterministic reception of messages, **msg**:[63] MSG, proceed as follows:

    (a) Customer awaits messages[64] from either conveyor companies or conveyors.

    (b) Customers "remember" these messages as outstanding requests. They will be handled by [recursively] iterated invocations of the conveyor behaviour !

So we "handle" that "lastly" listed behaviour "first" !

```
value
295.   cust_receives_messages(ki)(cid,...)(...)(po,or,r,ch) ≡
295a.      let msg= ⫿ { comm[ki,ui]?  | ui ∈ ckis∪cis } in
295b.      customer(ki)(cid,...)(...)(po,or∪{((ki,\tida,ui),msg)},r,⟨msg⟩^ch) end
```

The "handling" of the orders, or "buffered" are defined in the 'Reactive Behaviours' subsections:

- Customer Issues Order [k3], Sect. 19.2.2.1, item 297 on the next page;
- Customer Accepts Offer [k5] (order OK), Sect. 19.2.2.2, item 297 on the facing page;
- Customer Delivers Mercandises [k9], Sect. 19.2.2.3, item 298 on page 96; and
- Customer Requests & Receives Merchandises [k14a-b,k15b], Sect. 19.2.2.4, item 299 on page 96.

19.2 Subsidiary Behaviours

19.2.1 Proactive Behaviours

19.2.1.1 [k1] Customer Issues Query

296. [k1] The customer decides

    (a) to inquire, with some conveyor or logistics company, with a selected query command[65],

    (b) which it then communicates to the conveyor company or logistics company, updates its outstanding requests and augments its history,

    (c) whereupon it resumes being a customer.

This query action [k1] is "matched" by the suggest offer action [k2] Sect. Suggest Offer]20.3.1 on page 99; cf. formula lines 296b and 302d on page 99.

```
296.   cust__issues_query(ki)(cid,...)(...)(po,or,ch) ≡
296a.     let (coli,mk_CustQuery(qi,qc)) = sel_q(ki,(cid,...),(...),(po,or,ch)) in
296a.     let msg = ((ki,record TIME(),coli),mk_CustQuery(qi,qc)) in
296b.     comm[{ki,coli}] ! msg;                                          [k1]
296c.     customer(ki)(cid,...)(...)(po,or∪{msg},⟨msg⟩^ch) end end

296a.    sel_q:  KI×(CustId×AddrInfo×...)× ..
296a.           ×(Posses×OutReqs×CustHist) → CustInq
296a.    sel_q(ki,(cid,ai,...),(...),(po,or,ch)) ≡ ... see footnote 65 pg 94
```

[63] We have emphasized the **message** arguments as these play a pivotal role in the behavior interaction.
[64] These messages are either [k4] ConvCompOffers, [k8] ConvCompOrderOK, [k9] PendColl, [k13] ConvCustPendDel, [k14] NKTransfer messages.
[65] – we leave unspecified how that query is formed from the basis of the customer attributes

19.2.2 Reactive Behaviours

19.2.2.1 [k3] Customer Issues Order

297. [k3] If there is an ongoing (or outstanding) conveyor company offer
    (a) then the customer selects a suitable one. If there is not such the choice number is forced to 0.
    (b) Time is recorded.
    (c) If the customer does not finds a suitable offer
    (d) it so informs the conveyor company.
    (e) Else it likewise informs the conveyor company of order and choice number.
    (f) Whereupon it resumes being a customer.[66]

This `issues order` action [k3] is "matched" by the `confirm order` action [k4] Sect. Confirm Order]20.3.2 on page 99.

value

```
297.   cust_issues_order(ki)(cid,ai,...)(...)
297.                       (po,{((cki,t,ki),mk_ConvCompOffer(on,t,choices))67}∪or,ch) ≡
296a.        let (cn,offer) = select_offer(choices) in
296c.        let msg = ((ki,record 𝕋𝕀𝕄𝔼(),cki),if cn=0
296c.             then mk_OrderOK(on,no)
296c.             else mk_OrderOK(on,cn,offer) end) in
296d.        comm[{cid,cki}] ! msg;                                          [k3]
296f.        customer(ki)(cid,ai,...)(...)(po,or,⟨msg⟩^ch) end end
```

19.2.2.2 [k5] Customer Accepts Offer

297. Customers
    (a) examine transport company offers: the `examine` analysis function is left to Your imagination; the `status` value is either a **no**, or is **OrderOK**.
    (b) A time-stamped message to that effect is communicated to the conveyor company.
    (c) And the customer resumes being so.

This `customer order OK` action [k5] is "matched" by the `conveyor directives` action [k7] Sect. Conveyor Directives]20.3.3 on page 100. And also the `pending collection` action [k8] Sect. Pending Collection]20.3.4 on page 101.

```
297.   cust_order_OK(ki)(cid,...)(...)
297.        (po,{(ki,τ,cki),m:mk_ConvCompOffer(cki69,cnu,qno,offers)}∪or,ch) ≡
297a.      let okonok = examine(ki)(cid,...)(...)(po,{(ki,τ,cki),m}∪or,ch) in
297b.      let msg= ((ki,𝕋𝕀𝕄𝔼,cki),mk_OrderOK(oknok)) in
297b.      comm[{cid,cki}] ! msg;
297c.      customer(ki)(cid,...)(...)(po,or,⟨msg⟩^ch) end end
```

[66] We have used some informal notation, i.e., `[orderOK=]`

[67] Note the formal argument "trick": If the `ongoing requests` argument contains an element, `ConvCompOffer(on,t,choices)`, then the `cust_accept_offer` behaviour applies. If it does not, then **skip** !

[69] The two argument `ckis` are/must be [!] identical.

19.2.2.3 [k9] Customer Delivers Mercandises

298. [k9] Customer delivers merchandises:
    (a) collecting the identified merchandises;
    (b) composing messages to node and contracting conveyor company;
    (c) then transferring the merchandises to the identified node;
    (d) informing the contracting conveyor company; and
    (e) finally resuming being a customer.

This `delivery` action [k9] is "in consequence" of the `pending collection` action [k8] Sect. Pending Collection]20.3.4 on page 101.

value
```
298.   cust_delivers_merchandises(ki)(cid,ai,...)(...)
298.                  (po,{mk_PendColl(cki,on,mis,ni)}∪or,ch) ≡
298a.        let ms = {m|m:M•m∈po∧uid_M(m)∈mis}, τ = record TIME()in
298b.        let msg₁ = ((ki,τ,ni),mk_KNTransfer(on,ms)),
298b.            msg₂ = ((ki,,τ,cki),mk_Acknowledgment(τ,cnu,(ki,ni))) in
298c.   [k9]  (comm[{ki,ni}] ! msg₁
298d.   [k15a] ∥ comm[{ki,cki}] ! msg₁);
298e.        customer(ki)(cid,ai,...)(...)(po\ms,or,⟨{msg₁,msg₂}⟩⌢ch) end end
```

19.2.2.4 [k14a-b,k15b] Customer Requests & Receives Merchandises

299. [k14a] Customers are ready to receive merchandises once a message of pending delivery has been received from a conveyor.
    (a) [k14a] They can therefore accept such a delivery notice;
    (b) concocts an acknowledgment to the conveyor company,
    (c) [k15b] communicates this to the conveyor company,
    (d) whereupon it resumes being a customer.

This `cust_requests_merchandises` action [k14] is "matched" by the `node` action Sect. Main Behaviour]24.3 on page 116; cf. formula lines 299a and 323 on page 116.

value
```
299.   cust_requests_merchandises(ki)(cid,ai,...)(...)
299.             (po,{(ci,t,ki),mk_PendDeliv(ci,cnu,mis)}∪or,ch) ≡
299b.  [k14a] comm[{ki,ni}] ! mk_((ki,record TIME(),ni),PendColl(ni,(cnu,mis)))[70];
299a.  [k14b] let mk_NKTransfer(cms) = comm[{ki,ni}] ?[71] in
299c.  [k15b] comm[{ki,cki}] ! mk_Acknowledgment(record TIME(),cnu,(ci,ki)) ;
299d.         customer(ki)(cid,ai,...)(...)(po∪∪rng cms,or,⟨ms,msg⟩⌢ch) end
```

[70] Observe that the received message `ki` [in `(cki,t,ki)`] must match the formal argument `ki`. This informative communication is symbolized by the "open, white arrowhead" of the [k14] "double arrow" in Fig. 16.1 on page 72.

[71] This material communication is symbolized by the "black arrowhead" of the [k14] "double arrow" in Fig. 16.1 on page 72.

Chapter 20

Conveyor Company Behaviours

Contents

20.1 Main Behaviour

300. Conveyor companies non-deterministically alternates between

(a) being "themselves", sorting out daily, "internal" operations,

internal non-deterministically issuing

(b) [k2] (i.e., suggesting) offers,

(c) [k4] order confirmations,

(d) [k7] messages to conveyors about transports and

(e) [k8] pending collection;

external non-deterministically awaiting

(f) [k1] queries from customers, [k3] orders, [k5] sign-off on orders, or [k12,k15] acknowledgments of merchandise transfers.

```
300.  conveyor_company(cki)(me)(info)(res,co,ors,cb,pb,ckh) ≡
300a.              ...
300b.  [k2]       ⌈⌉ suggests_offer(cki)(me)(info)(res,co,ors,cb,pb,ckh)
300c.  [k4]       ⌈⌉ confirms_offer(cki)(me)(info)(res,co,ors,cb,pb,ckh)
300d.  [k7]       ⌈⌉ informs_conveyors(cki)(me)(info)(res,co,ors,cb,pb,ckh)
300e.  [k8]       ⌈⌉ pending_collection(cki)(me)(info)(res,co,ors,cb,pb,ckh)
300f.  [k12,k15]  ⌈⌉ awaits_msg(cki)(me)(info)(res,co,ors,cb,pb,ckh)
```

20.2 Main Reactive Behaviour

301. The conveyor company external non-deterministic reception of messages, i.e., responses, proceed as follows:

(a) The conveyor company awaits responses from either customers or conveyors. [72]

(b) If the message

(c) is an acknowledgment, [k12,k15], of merchandise transfers,

(d) then the `contracts` attribute is updated accordingly and

(e) the conveyor company resumes being so,

(f) else the conveyor company resumes being so, with updated `current business`,

```
301.   awaits_msg(cki)(me)(info)(res,co,ors,cb,pb,ckh) ≡
301a.      let msg :((koci,τ,cki),cmd)
301a.         = ⌈⌉ { comm[cci,koci]|koci:(KI|CI)•koci∈kis∪cis[73]} in
301b.      case msg of
301c.         (ui,τ,cki),mk_Acknowledgment(τ,cnu,(ui,uj))
301d.            → let co′ = upd_contracts(co,mk_Acknowledgment(τ,cnu,(ui,uj))) in
301e.               conveyor_company(cki)(me)(info)(res,co′,ors,cb,pb,⟨msg⟩⌢ckh) end
301f.           _ → conveyor_company(cki)(me)(info)(res,co,ors,cb∪msg,pb,⟨msg⟩⌢ckh)
301.       end end

301d.  upd_contracts:  Contracts×Acknowledgment → Contracts
301d.  upd_contracts(co,(τ,cnu,ft)) ≡  ⟨(τ,cnu,ft)⟩⌢con
```

[72]These responses are either [k1] customer queries, [k3] customer orders, [k5] customer order confirmation (and payment), or [k12,k15] conveyor and customer acknowledgment of merchandise transfers. Any other messages will be ignored

[73]*kis* and *cis* were defined in Items 167 on page 48 and 76 on page 22, respectively.

20.3 Subsidiary Behaviours

20.3.1 [k2] Suggest Offer

302. The conveyor company, with a customer query in its “in-basket”: current business, decides
   (a) to calculate an offer, commensurate with the query –
   (b) while updating the Offers and Orders attributes –
   (c) to form this offer into a commands, and to
   (d) communicate this offer to the inquiring customer,
   (e) updates its “past business” and history, and
   (f) resumes being a conveyor company.

This suggest offer action [k2] is “matched” by the query action [k1] Sect. Customer Issues Query]19.2.1.1 on page 94; cf. formula lines 296b on page 94 and 302d.

```
302.    suggests_offer(cki)(me)(info)
302.          (res,co,ors,msg:{((cki,τ,ki),mk_CustQuery(qi,qc))}∪cb74,pb,ckh) ≡
302a.      let offer:ConvCompOffer
302a.           = calc_offer(cki,res,co,ors,cb,pb,ckh)(mk_CustQuery(qi,ic)) in
302b.      let (res′,ors′) = update_res_and_ors(res,ors)(offer),
302c.           msg = ((cki,record TIME(),ki),offer) in
302d. [k2] comm[{cki,ki}] ! msg;
302e.      let pb′ = pb∪{msg}, ckh′=⟨msg⟩⌢ckh in
302f.      conveyor_company(cki)(me)(info)(res′,co,ors′,cb,pb′,ckh′) end end end
301.    post: commensurate_query_offers(mk_CustQuery(ki,iq,ic),offer)
```

```
302.    commensurate_query_offers: ...
302a.   calc_offer(...) ≡ ...
302b.   update_res[ources]_and_or[der]s ...
```

20.3.2 [k4] Confirm Order

(300c) The conveyor company with an OrderOK, decides to handle that:
   (a) If the order was not OK’ed then it does nothing,
   (b) else it cashes the payment[76] –
   (c) updates its current business and history,
   (d) and resumes being a conveyor company.

This confirm order action k4 is “matched” by the customer accepts offer action k5 Sect. Customer Accepts Offer]19.2.2.2 on page 95.

```
300c.   confirms_offer(cki)(me)(info)
300c.     (res,co,ors,{msg:((ki,t,cki),nok)}∪cb,pb,ckh) ≡
302a.    conveyor_company(cki)(me)(info)(res,co,ors,cb,{msg}∪pb,ckh)

300c.   confirms_offer(cki)(me)(info)
300c.     (res,co,ors,{msg:((ki,t,cki),mk_OrderOK(con,cn,pay))}∪cb,pb,ckh) ≡
302b.    [payment is registered ;]
302c.    let ors′ = update_orders(co,ors)(msg), ckh′ =⟨pay⟩⌢ckh in
302d.    conveyor_company(cki)(me)(info)(res,co,ors′,cb,pb∪{msg},ckh′) end
```

---

[74] See footnote 67 on page 95.

[76] The receipt and registration of payments, etc., etc., is a role for the conveyor company office.

303. The `update_orders` [auxiliary] function
   (a) examines the choice identified `offer`, and
   the identified choice, `tr`, and
   updates the contract to now only reflect that choice.

The "stashing" of `msg` in the "past business book" serves to remind the conveyor company to – sooner or later – issue [k7]. See next !

```
303.  updates_orders: Orders → MSG → Orders
303.  updates_orders(ors)((ki,t,cki),mk_OrderOK(cnu,cn,pay)) ≡
303a.     ors\{cn}∪[cn↦(ors(cn))(cnu)]
```

20.3.3 [k7] Conveyor Directives

(300d) "Sooner or later" the conveyor company reacts on the **orderOK** and

304. informs the one or more conveyors to be involved in the contracted transport.
   (a) If the **orderOK** was a **no** it does nothing, i.e., resumes being a conveyor company.
   (b) Else it decomposes the possibly multiple element segment list into separate conveyor company to conveyor directives,
   (c) communicates these to each involved conveyor, and
   (d) updates its history, and resumes being a conveyor company.

This `conveyor directives` action [k7] is "matched" by the `[k5]` action customer accepts offer Sect. Customer Accepts Offer]19.2.2.2 on page 95; cf. formula lines 297b on page 95 and 304c.

```
300d.  informs_conveyors(cki)(me)(info)
304.         (res,co,ors,cb,pb∪{((ki,t,cki),mk_OrderOK(cnu,chn,status))},ckh) ≡
304a.   if status = no axiom status ≠ orderOK
304a.      then conveyor_company(cki)(me)(info)(res,co,ors,cb,pb,ckh[77])
304b.      else let (status,tr) = (co(con))(chn) in
304b.           let dirl = elems construct_dirs(ki,record 𝕋𝕀𝕄𝔼(),cki,cnu,tr) in
304c.           {comm[{cki,ci}] ! dir|dir:ConvDir•dir∈ elems dirl∧dir=((cki,t,ci),_)} end end
304d.   conveyor_company(cki)(me)(info)(res,co,ors,cb,pb,⟨dir|dir∈dirs⟩⌢ckh) end
```

305. The `construct_dirs` function
   (a) from each segment from the contracted, `con` and chosen, [choice no.] `chn`, transport offer, it constructs a *convoy directive*,
   (b) and assembles into a `Conveyor Company` to `Conveyor Directive` command.
   (c) A *convoy directive* is a pair of *unload* and *load* directives.
   (d) An unload [load] directive is a quadruple of 𝕋𝕀𝕄𝔼, a node identifier, a contract number and a set of merchandise identifiers.

```
type
305c.  ConvDir = Unload×Load×[Final][78]
216.   Finals = NI →m (ContractNu →m KI)
216.   Final = (NI × (ContractNu × KI))|not_final
305d.  Load,Unload = 𝕋𝕀𝕄𝔼 × NI × ContractNu × MI-set
value
305.  construct_dirs: KI×𝕋𝕀𝕄𝔼 ×CKI×ContrNo×TR → ConvDir*
305.  construct_dirs(ki,t,cki,cnu,((fki,faddr),mis,sgl,(tki,taddr))) ≡
305a.     let dirl = ⟨ extract_dir(sgl[i],con,mis,i,len sgl,tki)|i:Nat•1≤i≤len sgl ⟩ in
305b.     ⟨ ((cki,t,ci),ConvDir(dirl[i],not_final))|i:Nat•1≤i<lens gl ⟩
305b.   ⌢ ⟨ ((cki,t,ci),ConvDir(dir[len sgl],(ni,(cnu,ki)))) ⟩ end
```

[77] **Alert:** I am not sure with what, if anything, to prefix the history with is OK. I was not ready to think about it when I wrote it, March 31, 2025, 16:01

306. The extract_directive function applies to a segment, contract number, a set of merchandise identifiers, the index of the segment list being examines, the length of that list, and the “end” customer identifier.
    (a) If the current index is less than the segment list, the no “final” is issued, just a pair of unload/loads.
    (b) Otherwise a final: nj,cnu,ki, the identifier of the last node where the contracted merchandises will be held for the customer ki.

```
type
306.   Segment = TIME × CI × NI × (EI|NI)*
value
306.   extract_dir:  Segment × ContractNu × MI-set × Nat × Nat × KI
306.                 → ((UnLoad×Load)×Final)
306.   extract_dir(sg:(t,ci,ni,enl⌢⟨nj⟩),cnu,mis,i,li,ki) ≡
306a.      if i<li then (((t,ni,con,mis),(t,nj,cnu,mis)),nil)
306b.                else (((t,ni,con,mis),(t,nj,cnu,mis)),(nj,cnu,ki)) end
306.   pre:  the edge-node identifier list is not empty, i.e., ≠ ⟨⟩
```

We apologize for the somewhat “tricky” functions: construct_dirs and extract_dir[79].

20.3.4 [k8] Pending Collection

307. At some time conveyor companies react to customers’ [k5] order OK (accepts offer) messages
    (a) by replying with a pending collection message –
    (b) whereupon the resume being conveyor companies,

This pending collection action [k8] is “in consequence” of the [k5] action order OK (accepts offer) Sect. Customer Accepts Offer]19.2.2.2 on page 95.

```
value
307.  pending_collection(cki)(me)(info)
307.      (res,co,ors,{((ki,τ,cki),mk_OrderOK(ni,cnu,chn,orderOK))}∪cb,pb,ckh) ≡
307a.     let msg= mk_((cki,record TIME(),ki),(PendColl(ni,(cnu,mis)))) in
307a.     comm[{cki,ki}] ! msg;
307b.     conveyor_company(cki)(me)(info)(res,co,ors,cb,pb,⟨msg⟩⌢ckh) end
```

[78] The type expression [T] stands for T|nil

[79] Most other function definitions are, in our opinion, straightforward

Chapter 21

Conveyor Behaviour

Contents

21.1 Earlier Treatment

In Sect. 7.4.1 on page 28 we first treated conveyor behaviours:

Signatures then:

```
value
ι104 π28.   conveyor:  CI→CM→(Kind×Routes)→(CurrRoute×CPos×CH) Unit
```

Behaviour, then at node:

```
value
ι105 π28.   conveyor(ci)(cm)(k,routes)(cr,AtNode(ni),ch) ≡
ι105a π28.      conveyor_change_route(ci)(cm)(k,routes)(cr,AtNode(ni),ch)
ι105b π28.   ⌈⌉ conveyor_remains_at−node(ci)(cm)(k,routes)(cr,AtNode(ni),ch)
ι105c π28.   ⌈⌉ conveyor_enters_edge(ci)(cm)(k,routes)(cr,AtNode(ni),ch)
ι105d π28.   ⌈⌉ conveyor_stops_at_node(ci)(cm)(k,routes)(cr,AtNode(ni),ch)
```

```
ι106 π29.   conveyor_change_route(ci)(cm)(k,routes)(cr,AtNode(ni),ch) ≡
ι106a π29.      let τ = record_TIME(),
ι106b π29.          ncr = select_next_route(ni,routes),
ι106d π29.          ch′ = ⟨(τ,ni)⟩⌢ch in
ι106c π29.      comm[{ci,ni}] ! (τ,ci) ;
ι106e π29.      conveyor_at_node(ci)(cm)(k,routes)(ncr,AtNode(ni),ch′) end

ι106b π29.   selects_next_route:NI × Routes → CurrRoute
ι106b π29.   selects_next_route(ni,routes) as ncr • ncr ∈ routes ∧ hd ncr = ni
```

Behaviour, then on edge:

```
ι111 π30.   conveyor(ci)(cm)(k,routes)
ι111 π30.            (cr,mk_OnEdge(n_uif,(f,e),n_uit),ch) ≡
ι111a π30.      conveyor_moves_on_edge(ci)(cm)(k,routes)
ι111a π30.           (cr,mk_OnEdge(n_uif,(f,e),n_uit),ch)
ι111c π30.    ⌈⌉ conveyor_stops_on_edge(ci)(cm)(k,routes)
ι111c π30.           (cr,mk_OnEdge(n_uif,(f,e),n_uit),ch)
ι111b π30.    ⌈⌉ conveyor_enters_node(ci)(cm)(k,routes)
ι111b π30.           (cr,mk_OnEdge(n_uif,(f,e),n_uit),ch)

ι107 π29.   conveyor_remains_at_node(ci)(cm)(k,routes)(cr,AtNode(ni),ch) ≡
ι107a π29.      let τ = record_TIME() in
ι107b π29.      comm[{ci,ni}] ! (τ,ci);
ι107c π29.      conveyor(ci)(cm)(k,routes)(cr,AtNode(ni),⟨(τ,ni)⟩⌢ch) end

ι108 π29.   conveyor_enters_edge(ci)(cm)(k,routes)(cr,AtNode(ni),ch) ≡
ι108a π29.     let τ = record_TIME() in
ι108b π29.     ( comm[{ci,ni}] ! (τ,ni) ∥ comm[{ci,ni}] ! (τ,hd cr) ) ;
ι108c π29.     let ei = hd cr in let {ni,ni′} = mereo_E(retr_edge(ei)(es)) in
ι108c π29.     let cpos = onEdge(hd cr,(ei,(ni,f,ni),ni′)) in
ι108e π29.     conveyor(ci)(cm)(k,routes)(cr,cpos,⟨(τ,ni)⟩⌢ch) end end end end

ι109 π30.   conveyor_stops_at_node(ci)(cm)(k,routes)(cr,AtNode(ni),ch) ≡
ι110 π30.      let τ = record_TIME() in
ι110 π30.      comm[{ci,ni}] ! (τ,ci) ;
ι109 π30.      stop end
```

21.2 Main Behaviour

In the context of customers and logistics and conveyor companies, as illustrated by Fig. 16.1 on page 72, conveyors, i.e., their behaviour, are a bit more intricate !

308. Conveyors non-deterministically alternates between
    (a) being themselves,

    or external non-deterministically receiving

    (b) [k7] directives from conveyor companies – their own or other,
    (c) and then handling these messages,[80]

    and internal non-deterministically sending messages

    (d) [k10] notifying edges and nodes of their presence,
    (e) [k12] and acknowledgments of transfer of merchandises from and to customers and nodes.

    When not responding to and handling messages from other behaviours ([k7] conveyor companies, or [k9] customers),

    (f) a conveyor is either at a node, possibly unloading or loading merchandises, or
    (g) along, i.e., on, an edge.

```
308.   conveyor(ci)(cm:(uis,ckis,kis,cis))(k,...)
308.                         (stow,tbu,tbl,sr,idx,finals,pos,ch) ≡
308a.        ... conveyor(ci)(cm:(uis,ckis,kis,cis))(k,...)
308a.                        (stow,tbu,tbl,sr,idx,finals,pos,ch)
308b.  [k7]   ⌈⌉ let msg   ⌈⌉ { comm[{ci,cki}]?   | cki∈ckis } in
308c.           conv_dir_handling(ci)(uis,ckis,kis,cis)(k,...)
308c.                 (stow,tbu,tbl,sr,idx,finals,pos,⟨msg⟩^ch)(msg) end
308d.  [k10] ⌈⌉ conv_node_notification(ci)(uis,ckis,kis,cis)(k,...)
308d.                                (stow,tbu,tbl,sr,idx,finals,pos,ch)
308d.  [k10] ⌈⌉ conv_edge_notification(ci)(uis,ckis,kis,cis)(k,...)
308d.                                (stow,tbu,tbl,sr,idx,finals,pos,ch)
308e.  [k12] ⌈⌉ conv_comp_ack(ci)(uis,ckis,kis,cis)(k,...)
308e.                       (stow,tbu,tbl,sr,idx,finals,pos,ch)
308f.        ⌈⌉ conv_at_node(ci)(uis,ckis,kis,cis)(k,...)
308f.                      (stow,tbu,tbl,sr,idx,finals,pos,ch)
308g.        ⌈⌉ conv_on_edge(ci)(uis,ckis,kis,cis)(k,...)
308g.                      (stow,tbu,tbl,sr,idx,finals,pos,ch)
```

[80] **Note:** This is the only message received by conveyors from contracting conveyor companies in this, the present transport domain model. For more realistic transport domain models there will, of course, be other such messages – but they deal, not with the *intrinsic* facets of transport (logistics) but with technology support, management & organization, human, and other facets – cf. Chapter 8 of my book [7].

21.3 Subsidiary Behaviours

21.3.1 Proactive Behaviours

21.3.1.1 [k7] Directives

309. The conv_directive_handling behaviour for handling conveyor company to conveyor directives
    (a) updates the to-be-unloaded, the to-be-loaded and the finals attributes, and
    (b) resumes being a conveyor.

This conveyor directives handling action k7 is "matched" by the informs conveyors action k7 Sect. Conveyor Directives]20.3.3 on page 100; cf. formula lines 309 and 304c on page 100.

```
309.   [k7] conv_dir_handling(ci)(me)(k,r)
309.              (stow,tbu,tbl,sr,idx,finals,pos,ch)
309.              ((cki,t,ci),ConvDir((t′,ni,cnu,mis),(t″,nj,cnu,mis)),final) ≡
309a.       let tbu′ = tbu ∪ [nj↦tbu∪{cnu}],      [we disregard t,t′,t″]
309a.           tbl′ = tbl ∪ [ni↦tbl∪{cnu}],      [we disregard t,t′,t″]
309a.           finals′ = upd_finals(finals,final) in
309b.       conveyor(ci)(me)(k,r)(stow,tbu′,tbl′,sr,idx,finals′,pos,⟨dirs⟩⌢ch) end

309a.     upd_finals(finals,(ni,cnu,ki)) ≡ finals∪[ni↦[ki↦cnu]]
```

21.3.1.2 [k10] Conveyor to Node and Edge Notifications

310. Conveyor notify the edges and nodes along which it is moving:
    (a) either at a node,
    (b) or on an edge.

This conv_node_notification action k10 is "matched" by the node action k10 Sect. Main Behaviour]24.3 on page 116; cf. formula lines 310a and 322b on page 116.

```
value
310.   conv_node_notification(ci)(uis,ckis,kis,cis)(k,...)
310.              (stow,tbu,tbl,sr,idx,finals,mk_AtNode(ni),ch) ≡
310a.      let msg = ((ci,record TIME(),ni),mk_AtNode(ni)) in
310a. [k10] comm[{ci,ni}] ! msg ;
310.       conveyor(ci)(uis,ckis,kis,cis)(k,...)
310.         (stow,tbu,tbl,sr,idx,finals,mk_AtNode(ni),⟨msg⟩⌢ch) end
```

This conv_edge_notification action k10 is "matched" by the edge action k10 Sect. Main Behaviour]24.3 on page 116; cf. formula lines 310b and 322d on page 116.

```
310.   conv_edge_notification(ci)(uis,ckis,kis,cis)(k,...)
310.              (stow,tbu,tbl,sr,idx,finals,pos:mk_OnEdge(_,(_,ei),_),ch) ≡
310b.      let msg = ((ci,record TIME(),ei),mk_OnEdge(ei)) in
310b. [k10] comm[{ci,ei}] ! msg ;
310.       conveyor(ci)(uis,ckis,kis,cis)(k,...)
310.         (stow,tbu,tbl,sr,idx,finals,pos,⟨msg⟩⌢ch) end
```

[80]The ci is that of the conveyor
[79]The two formal argument occurrences of ci, respectively cki, must be pairwise identical ! See also the next conv_msg_handling definitions.

21.3.1.3 Conveyor on Edge

Conveyor on Edge – Then:

```
ι111 π30.    conveyor(ci)(cm)(k,routes)(cr,mk_OnEdge(n_uif,(f,e),n_uit),ch) ≡
ι111a π30.     conveyor_moves_on_edge(ci)(cm)(k,routes)(cr,mk_OnEdge(n_uif,(f,e),n_uit),ch)
ι111c π30.   ⌈⌉ conveyor_stops_on_edge(ci)(cm)(k,routes)(cr,mk_OnEdge(n_uif,(f,e),n_uit),ch)
ι111b π30.   ⌈⌉ conveyor_enters_node(ci)(cm)(k,routes)(cr,mk_OnEdge(n_uif,(f,e),n_uit),ch)
```

We leave it to the reader, this time, to review the functions: conveyor_moves_on_edge Sect. 7.4.1 items 112 on page 30 etc., conveyor_stops_on_edge Sect. 7.4.1 items 114 on page 31 etc. and conveyor_enters_node Sect. 7.4.1 items 113 on page 31 etc.

• • •

Conveyor on Edge – Now:

311. An edge [behaviour] at an edge external non-deterministically either:
    (a) **move**s along the edge, a fraction "at a time", or
    (b) **stop**s on the edge and thereby "leaves" transport; or
    (c) **enter**s a node.

```
311.  conveyor_on_edge(ci)(me:(uis,ckis,kis,cis))(k,len,cost)
311.                  (stow,tbu,tbl,sr,idx,finals,mk_OnEdge((fni,(ej,f),tni)),ch) ≡
311a.   ⌈⌉ conveyor_moves_on_edge(ci)(me:(uis,ckis,kis,cis))(k,len,cost)
311a.                  (stow,tbu,tbl,sr,idx,finals,mk_OnEdge((fni,(ej,f),tni)),ch)
311b.   ⌈⌉ conveyor_stops_on_edge(ci)(me:(uis,ckis,kis,cis))(k,len,cost)
311b.                  (stow,tbu,tbl,sr,idx,finals,mk_OnEdge((fni,(ej,f),tni)),ch)
311c.   ⌈⌉ conveyor_enters_node(ci)(me:(uis,ckis,kis,cis))(k,len,cost)
311c.                  (stow,tbu,tbl,sr,idx,finals,mk_OnEdge((fni,(ej,f),tni)),ch)
```

The next behaviour is "patterned" over Items 112a– 112e on page 30.

312. A conveyor which is moving along an edge, some fraction down the edge/road/track/route, but not "yet" near "the end":
    (a) at time τ,
    (b) increments the fraction of its position
    (c) (while updating its history)
    (d) notifying the edge [behaviour]
    (e) [technically speaking] adjusting its position], and, finally,
    (f) resuming being a thus updated conveyor [OnEdge.

```
312.  conveyor_moves_along_edge(ci)(me)(_,_,_)
312.                  (stow,tbu,tbl,sr,idx,finals,mk_OnEdge((fni,(ej,f),tni)),ch) ≡
312a.      let τ = record_TIME(), ε:Real • 0 < ε ≪ 1 in
312b.      let f′ = f+ε, cpos = mk_OnEdge(n_uiif,(f′,e),n_uit) in
312c.      let ch′ = ⟨(τ,ci)⟩⌢ch in
312d.      comm[{ci,ej}] ! (τ,ci) ;
312e.      conveyor(ci)(me)(_,_,_)
312f.            (stow,tbu,tbl,sr,idx,finals,mk_AtNode(tni),ch) end end end
312.   pre: f ≃ 1 ∧ sr(idx)=tni
```

313. A conveyor may, "surreptitiously" as it were, "decide" to stop being a conveyor altogether !

```
313.  conveyor_stops_on_edge(ci)(me:(uis,ckis,kis,cis))(k,len,cost)
313.         (stow,tbu,tbl,sr,idx,finals,mk_OnEdge((fni,(ej,f),tni)),ch) ≡ stop
```

314. A conveyor enters a node
   (a) at time τ, by altering its position,
   (b) notifying both edge and node behaviours,
   (c) and resumes being a conveyor.

```
314.  conveyor_enters_node(ci)(me)(_,_,_)
314.                 (stow,tbu,tbl,sr,idx,finals,mk_OnEdge(fni,(ej,1),tni),ch) ≡
314.     let τ = record TIME() in
314a.    (comm[{ci,ej}] ! (τ,ci)‖comm[{'tau,tni}] ! (τ,ci)) ;
314b.    conveyor(ci)(me)(_,_,_)
314b.       (stow,tbu,tbl,sr,idx,finals,mk_atNode(tni),⟨(τ,mk_atNode(tni))⟩ch) end
```

21.3.1.4 Conveyor at Node

Conveyor at Node – Then:

value

```
ι105 π28.   conveyor(ci)(cm)(k,routes)(cr,mk_AtNode(ni),ch) ≡
ι105a π28.     conveyor_change_route(ci)(cm)(k,routes)(cr,mk_AtNode(ni),ch)
ι105b π28.   ⌈⌉ conveyor_remains_at_node(ci)(cm)(k,routes)(cr,mk_AtNode(ni),ch)
ι105c π28.   ⌈⌉ conveyor_enters_edge(ci)(cm)(k,routes)(cr,mk_AtNode(ni),ch)
ι105d π28.   ⌈⌉ conveyor_stops_at_node(ci)(cm)(k,routes)(cr,mk_AtNode(ni),ch)
```

• • •

Conveyor at Node – Now:

A primary "business" of a conveyor at a node is to unload and load merchandises.

315. In general, a conveyor at a node internal non-deterministically "alternates" between
   (a) **unload**ing merchandises,
   (b) **load**ing merchandises,
   (c) **stop**ping altogether, and
   (d) **enter**ing a next edge – if not the end of the conveyor route –
   – an in these cases resuming being a conveyor.

```
315.  conveyor_at_node(ci)(uis,ckis,kis,cis)(k,...)
315.                 (stow,tbu,tbl,sr,idx,finals,mk_AtNode(ni),ch) ≡
315a.    conveyor_unloads_merch(ci)(uis,ckis,kis,cis)(k,...)
315a.                 (stow,tbu,tbl,sr,idx,finals,mk_AtNode(ni),ch)
315b.  ⌈⌉ conveyor_loads_merch(ci)(uis,ckis,kis,cis)(k,...)
315b.                 (stow,tbu,tbl,sr,idx,finals,mk_AtNode(ni),ch)
315c.  ⌈⌉ conveyor_stops_at_node(ci)(uis,ckis,kis,cis)(k,...)
315c.                 (stow,tbu,tbl,sr,idx,finals,mk_AtNode(ni),ch)
315d.  ⌈⌉ conveyor_enters_edge(ci)(uis,ckis,kis,cis)(k,...)
315d.                 (stow,tbu,tbl,sr,idx,finals,mk_AtNode(ni),ch)
```

316. Conveyors unload (deliver), onto the node they are at,
    (a) from their stowage, the one-or-more contracted merchandises, for that node,
    (b) [k11a] and communicates these to that node,
    (c) [k12a] and acknowledges that to the contracting conveyor companies.
    (d) For final 'unloads', if any, receiving customers
    (e) are informed of pending delivery.
    (f) Whereupon the conveyor resumes being a conveyor at that node.

```
value
316.         conveyor_unloads_merch(ci)(uis,ckis,kis,cis)(k,...)
316.                  (stow,tbu,tbl,sr,idx,finals,mk_AtNode(ni),ch) ≡
316a.               let unls = tbu(ni), stow′ = stow\{ni} in
316b.  [k11a]  comm[{ci,ni}] ! mk_CNTransfer(stow/unls)[80]
316c.  [k12a]  ‖ {comm[{ci,xtr_CKI(ci)}] ! mk_Acknowledgment(record TIME(),cnu,(ci,ni))
316c.               | cnu:ContractNu•cnu∈ unls } ;
316d.               if ni∉ dom finals
316d.                  then skip
316e.                  else { let cnu=(finals(ni))(ki), mis=(tbu(nu))(cnu) in
316e.  [k13]                   comm[{ci,ki}] ! mk_PendDeliv(ni,(cnu,mis)) ;
316e.                       | ki:KI•ki∈ dom finals(ni) end }
316d.               end
316f.               conveyor_unloads_merch(ci)(uis,ckis,kis,cis)(k,...)
316f.                     (stow′,tbu\{ni},tbl,sr,idx,finals\{ni},mk_AtNode(ni),⟨v⟩⌢ch) end
```

Alert: Fix **v**: CNTransfer(unls) ?

317. Conveyors load (fetch)

    [from the node they are at, onto their stowage]

    contracted merchandises:
    (a) if there are merchandises to
    (b) load these
    (c) communicate them to the node
    (d) and the contracting conveyor company notified.
    (e) otherwise nothing is done;
    (f) and the conveyor resumes being a conveyor at that node.

```
value
317.         conveyor_loads_merch(ci)(uis,ckis,kis,cis)(k,...)
317.                        (stow,tbu,tbl,sr,idx,finals,mk_AtNode(ni),ch) ≡
317a.           if ni∈ dom tbl
317b.              then let lds = tbl(ni), cki = xtr_CKI(cnu) in
317c.                    comm[{ci,ni}] ! mk_NCTransfer(lds) ;
317d.                    comm[{ci,cki}] ! mk_Acknowledgment(cnu,(ci,ni)) end
317a.              else skip end
317f.           conveyor_loads_merch(ci)(uis,ckis,kis,cis)(k,...)
317f.                        (stow,tbu,tbl\{ni},sr,idx,finals,mk_AtNode(ni),⟨load⟩⌢ch)
```

Alert: Check for proper **load** onto `ch`

[80] The value of `stow/unls` is that of `stow` [domain-]restricted to `unls`.

The next behaviour:

value

```
    conveyor_stops_at_node(ci)(uis,ckis,kis,cis)(k,...)
                    (stow,tbu,tbl,sr,idx,finals,mk_AtNode(ni),ch) ≡ stop
```

is a "mere" transcription" of the similarly named behaviour of Sect. 7.4.1 on page 28, items 114 on page 31-... .

318. Finally, the conveyor may [be ready to] leave the node for possibly continuing its journey.
    (a) If the conveyor is at the end of its current service route, sr,
    (b) then
    (c) it reverts sr, into rs,
    (d) which defines the next **mk**_onEdge(fni,(0,ei),tni) elements,
    (e) and the conveyor continues being a conveyor, on that edge.
    (f) Otherwise
    (g) the next **mk**_onEdge(fni,(0,ei),tni) elements, are defined by the current service route, sr,
    (h) and the conveyor continues being a conveyor, on that edge.

```
318.   conveyor_enters_edge(ci)(me)(k,...)
318.     (stow,tbu,tbl,sr,idx,finals,mk_AtNode(ni),ch) ≡
318a.    if idx = len sr
318b.      then
318c.        let rs = revert(sr) in
318d.        let fni = rs[1], ei = rs[2], tni = rs[3] in
318d.        let e = mk_onEdge(fni,(0,ei),tni) in
318e.        conveyor(ci)(me)(k,...)
318e.           (stow,tbu,tbl,rs,1,finals,e,⟨e⟩^ch) end end end
318f.      else
318g.        let fni = sr[idx], ei = sr[idx+1], tni = sr[idx+3] in
318h.        let e = mk_onEdge(fni,(0,ei),tni) in
318h.        conveyor(ci)(me)(k,...)
318h.           (stow,tbu,tbl,sr,idx+1,finals,e,⟨e⟩^ch) end end
318f.    end

318c.  revert: Path → Path
318c.  revert(p) ≡
318c.     case p of
318c.        ⟨⟩ → ⟨⟩,
318c.        r^⟨u⟩ → ⟨u⟩^revert(q)
318c.     end
```

The above reflects but one choice for continuing a conveyor once it has "exhausted" its current service route. Others can be thought of.

Chapter 22

Logistics Company Behaviour

We skip this chapter: the conveyor company behaviour “says it all !”.

Chapter 23

Edge Behaviour

Contents

23.1 Earlier Treatment

```
value
104.    edge:  EI→EM→(Kind×LEN×Cost)→NH→Unit
```

```
value
117.    edge:  EI → EM → (EdgeKind×LEN×Cost) … → EH
117a.   edge(ei)(em)(ekind,len,cost)(eh) ≡
117b.       let msg= ⌷ { comm[{ei,ci}] ?  | ci:CI • ci ∈ em } in
117c.       edge(ni)(em)(eki...)(⟨msg⟩⌢eh) end
```

23.2 Main Behaviour

319. An edge behaviour revolves around:
    (a) conveyors moving along, being so notified by messages which it remembers by “adding” them to their histories,
    (b) before resuming being adge behaviours.

```
319.   edge(ei)(em)(ekind,len,cost)(eh) ≡
319a.      let msg= ⌷ { comm[{ei,ci}]?  | ci:CI•ci ∈ em } in
319b.      edge(ei)(em)(ekind,len,cost)(⟨msg⟩⌢eh) end
```

That is, no change !

Chapter 24

Node Behaviour

Contents

24.1 Earlier Treatment

value

```
ι116 π31.   node: NI → NM → NodeKind → NH
ι116a π31.  node(ni)(nm)(nkind)(nh) ≡
ι116c π31.      let msg= ⌷ { comm[{ni,ci}] ? | ci:CI • ci ∈ nm } in
ι116d π31.      node(ni)(nm)(nkind)(⟨msg⟩⌢nh) end
```

24.2 Revised Node Attributes

320. Each node may potentially provide [also] as a temporary "on-hold" storage for customer merchandises.

type

```
320.  OnHold = ContractNu →m M-set
```

value

```
320.  attr_OnHold: N → OnHold
```

24.3 [k10,k11,k14] Main Behaviour

321. Node behaviours revolves around:

322. nodes external non-deterministically accepting messages from conveyors where these messages are

    (a) [k10] either notifications of the presence of (moving) conveyors – duly recorded in the node history attribute;

    (b) [k11a] or from conveyors unloading at nodes duly updated in the node `onhold` and `history` attributes;

    (c) [k11b] or from conveyors loading at nodes

    (d) [k12] and informing the "originating" conveyor company,

    – in which latter case

    (e) the merchandises identified in the load are communicated ("back") to the conveyor.

or non-deterministically externally receiving requests from customers to

323. to deliver contracted `onhold` merchandises,

```
321.   node(ni)(nm:(eis,kis,cis))(nkind)(onhold,nh) ≡
322.         let msg = ⌈⌉ {comm[{ni,ci}]?|ci:CI•ci∈cis} in
322.         case msg of

322a.  [k10]      (_,mk_AtNode(ni))
322a.              → node(ni)(nm)(nkind)(onhold,⟨msg⟩⌢nh),

322b.  [k11a]     ((ci,τ,ni),mk_CNTransfer(cnu,lds)) [cf. 316b on page 109]
322b.              → node(ni)(nm)(nkind)(onhold∪lds^81,⟨msg⟩⌢nh),

322c.             ((ci,τ,ni),mk_NCTransfer(cnu,mis)) [cf. 317a on page 109]
322d.  [k12]        → let ms = {m:M|m∈ onhold(cnu)∧uid_(m)∈mis} in
322e.  [k11b]          comm[{ni,ci}]!mk_NCTransfer([cnu↦ms]) ;
322d.                  node(ni)(nm)(nkind)(onhold\cnu,⟨msg⟩⌢nh) end

322.         end end

323.      ⌈⌉ let msg:mk_PendColl(ni,(cnu,mis)) = ⌈⌉ {comm[{ni,ki}]?|ki:KI•ki∈kis} in
323.         let ms = {m|m:M•m∈onhold(cnu)∧uid_M(m)∈mis} in
323.         let τ = record TIME()in
323.         msg = ((ni,τ,ki),mk_NKTransfer(ms)) in
323.    [k14] comm[{ni,ki}]!  msg ;
323.          node(ni)(nm)(nkind)(onhold\cnu,⟨((ni,τ,ki),ms_to_mis(ms))⟩⌢nh) end end
116.          end
```

[80] **dom** lds ∩ **dom** onhold = {}

[81] **Alert:** Fic unls; one or more !?

Part VII

CLOSING

Chapter 25

Discussion

Contents

25.1 Wither Logistics Companies

It was a mistake, it seems, to distinguish between conveyor and logistics companies. A conveyor company with no conveyors is a logistics company. Examples are travel agencies. A revised taxonomy for conveyor companies is as shown in Figs. 25.1 and 25.2 on the following page. They are revisions of Figs. 12.1 on page 52 and 9.1 on page 37.

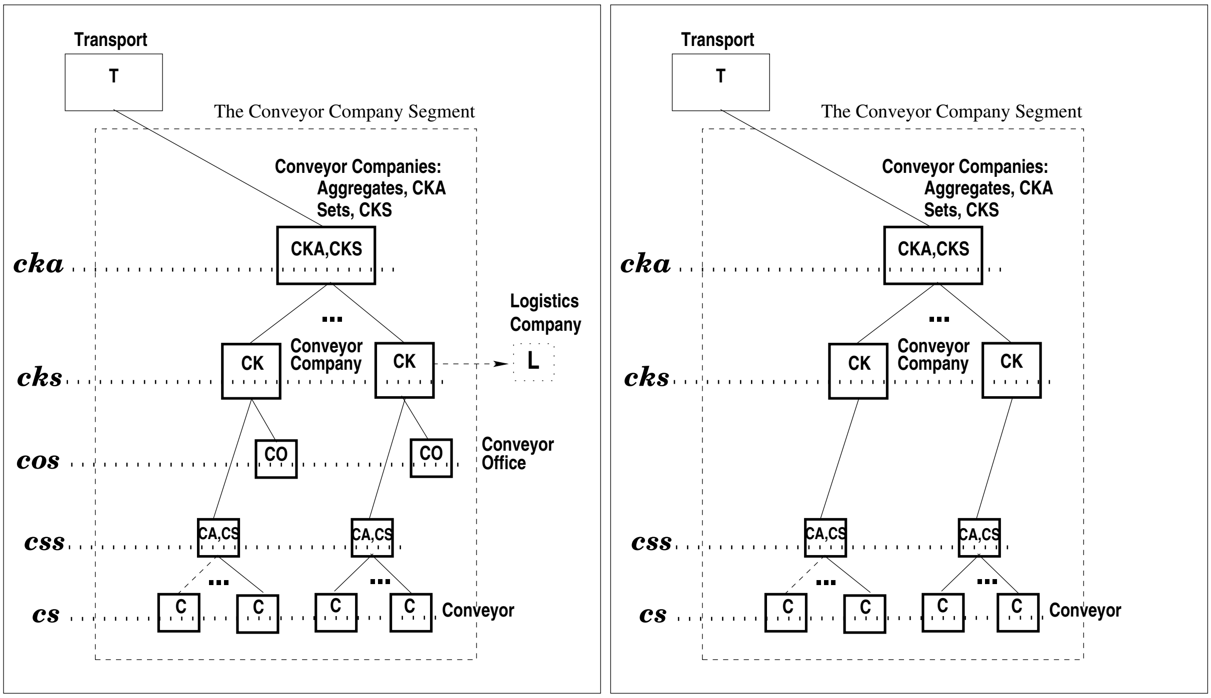


Figure 25.1: Old and Revised Conveyor Company Taxonomies

The corresponding `Command & Material Traces` figures is Fig. 25.3 on the next page:

MORE TO COME

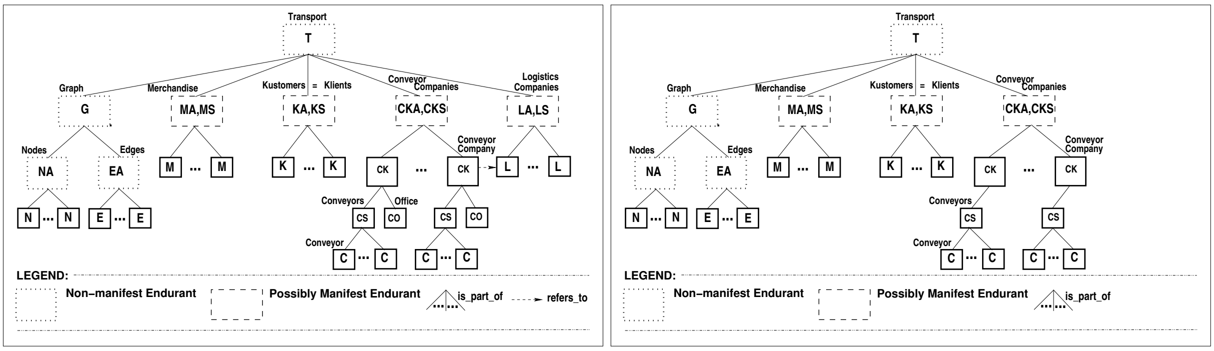


Figure 25.2: Old and Revised Transport Taxonomies

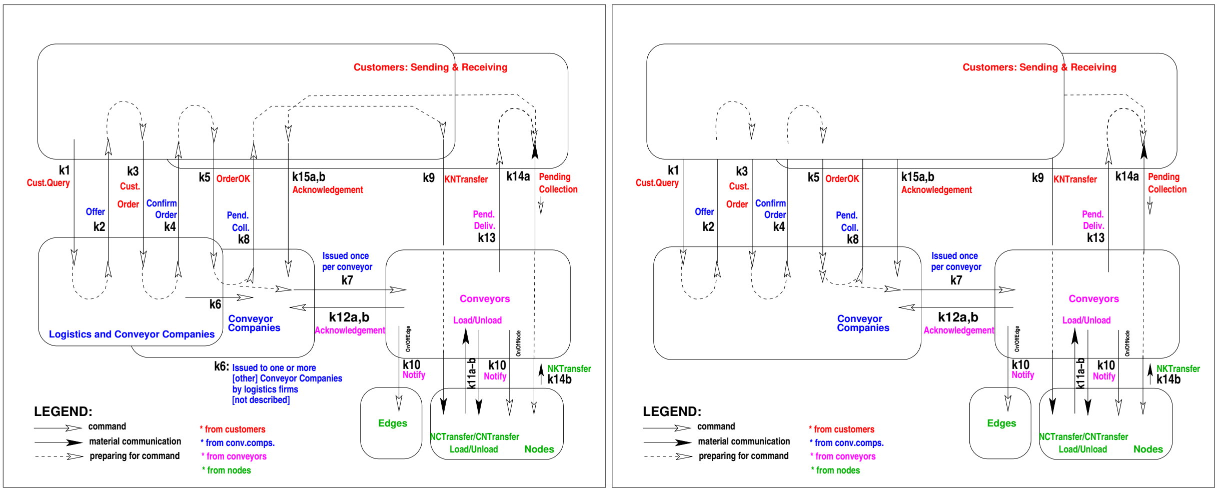


Figure 25.3: Old and Revised Command & Material Traces [→]

25.2 Some Parts Modelled, Others Not ! ?

The reader will have observed that we model only some of the internal qualities of composite parts ! Why ? Well the answer is this: We have chosen to emphasize the modelling of essential aspects of transport. The "omitted" full modelling of some, well most, composite parts [endurants], and hence their behaviours [perdurants], is therefor motivated as follows:

- **Graphs:** With `G`, `EA` and `NA` we do not associate any manifest "authority". But we could ! ? With `G` we could associate such more-or-less public authorities as the road authorities of Your city or country, rail net authorities, coastal and sea authorities, air traffic command & control, incl. *ICAO* [82],etc.
- **Merchandise Aggregate:** With `MA` we also do not associate any manifest "authority". But we could ! ? There are an abundance of private/public association which monitor and control publically available merchandise categories: food, toy, automobile, etc., agencies.
- **Customer Aggregate:** With `KA` we do not associate any manifest "authority". But we could ! ? We leave it to the reader to identify possibly relevant such candidates !
- **Conveyor Companies Aggregate:** With `CKA` we do not associate any manifest "authorities". But we could ! ? There are public/private associations which handle concerns of the conveyor industry, one or more for each *kind*. We omit their modelling.
- **Logistics Companies Aggregate:** With `LA` we do not associate any manifest "authorities". We could ! ? But we do not.

[82] https://www.icao.int/about-icao/Pages/default.aspx

25.3 Formal Structuring

By *formal structuring* we mean the way we have chosen some endurant parts to be composite, i.e., Cartesians an sets of parts. This structuring is most clearly reflected in Fig. 9.1. We now regret the “messy” handling of logistics, both as separate parts, and as an element of conveyor companies. A better “decomposition” must be found in a continuation project. There are other, in our mind, minor, such restructurings to be made.

25.4 Mnemonics

Mnemonics is the study and development of systems for improving and assisting the memory[83]. One such system is naming. We have strived some “logic” in choosing names. Endurant parts have been given very short one, two or three letter identifiers. Commands, functions and behaviours have been assigned longer identifiers, trying to compress their full names in the informal texts. A careful review, for any possible continuation project should carefully review these latter names.

25.5 Narratives

All (or almost all) **formulas** have been preceded by **narratives**. Pairwise their numbering “match” ! But these narratives are, in our mind, far from satisfactory. Much more care should be taken in formulating and “repetitively” express these narratives. Perhaps one should serve two narratives for each one presented here ? One, short, coupled with and receding the formulas; another, longer, perhaps appearing as footnotes, or as notes in a separate appendix ?

[83] https://languages.oup.com/google-dictionary-en/ and https://dictionary.cambridge.org/dictionary/english/mnemonic

Chapter 26

Conclusion

Contents

Chapters 2–24 (pages 7–116) sketched a "strict" narrative coupled to a formal description of an essence of transport domains. These were engineering descriptions. Your understanding of these rely on Your having understood [12, 9, 7, 6, 4].

26.1 Logistics & Operations Research

As for 'logistics companies': Yes, I have left them out.

26.1.1 Logistics

324. By *logistics* we shall mean *the detailed planning of the organization and implementation of a complex operation.*.

In this report logistics, in this sense of *planning* has been concentrated in the function `cal_offer`, cf. Item 302a on page 99.

26.1.2 Operations Research

That is: the often exciting and beautiful properties of optimization algorithms are to be "buried" here. They do not belong to the 'transport' aspects – but to the *strategic, tactical an operational* **facets** of the transport domain[84].

26.2 Interpretations

The domain description of Sects. 2–18 (pages 7–116) can be viewed in three ways:

(i) as a step in the general, say socio-economic study of a specific infra-structure [sub-]domain;

(ii) as a prerequisite for *business process re-engineering*;

[84] Cf. Sect.8.7, Example 107, pages 232–233 of my book [7].

(iii) as an, albeit, in this case, and this stage of unfolding study, basis document for preparing teachers material for subsequent development, i.e., writing, of secondary school course element for teaching such specific infra-structure [sub-]domains; and

(iv) as an initial feasibility study for possible subsequent development of software for multi-mode transport systems.

We shall now comment on each of these.

26.2.1 Socio-Economic Study

TO BE WRITTEN

26.2.2 Business Process Re-Engineering

TO BE WRITTEN

26.2.3 Primary and Secondary School Topic

We should like to see reports on the study, analysis and description of several societal infrastructure components:

- **the banking system**, from Your local, "brick and mortar" branch office via its head quarter, the national bank of Your country[85], the regional bank of your continent to The World Bank[86] and the IMF[87];
- **the insurance industry**;
- **the health care industry**, from Your family doctor, via local clinics, to hospitals – with pharmacies, home care and health insurance providers included;
- **the education system**, from primary and secondary schools, to high schools, colleges and universities;
- **et cetera !**

26.2.4 Algorithms & Data Structures

Many functions, like `get_offers`, imply, for their software realization, rather complex data structures and intricate algorithms. Since we are describing domains, and not designing software. we need, in a sense, not be concerned. But we have achieved, one might say, a clear identification, of where such clever software designs may be warranted.

26.2.5 Software System Development

This study and experimental report began with espousing **The Triptych Dogma**. But we have advocated that domain modelling be used for other purposes than "just" software development. Now we "*return to the fore*" *!* We now assume that there is, indeed, to be professionally & commercially, at least in a seriously funded effort, to be developed actual software for essential aspects of transport as they have been laid out in this study and experimental report. How would we go about doing that ?

Based on more than 40 years of experience[88] we would do as follows:

- First we would, as we have already started doing, perform the three phases of so-called `''SEA''` preparatory work.

  – **S**tudy, – **A**nalyze, and – **E**xperiment.

  We have just, more-or-less, completed these three phases.
- Now we are ready for a project committed to produce a "full-blown" domain model.

[85] https://www.nationalbanken.dk/en
[86] https://www.worldbank.org/ext/en/home
[87] https://www.imf.org/en/Home
[88] We refer to the Dansk Datamatik Center's [17] `CHILL` and `Ada` projects [18]

- After that, the similar development of a requirements prescription.
- And after that, the development of a software design, is coding, validation, etc.

How would we organize the "full-blown" domain modelling

- First we would assemble, in this case, six people, well-familiar with the domain modelling approach pursued in this report.
- They would be organized with the following responsibilities – being responsible for the development of:
  - the transport net, i.e., graph, model – 1 person;
  - the conveyor model – 2 persons;
  - the merchandises model – 1 person; and
  - the logistics and conveyor companies model – 2 persons.

  All under the leadership of an overall domain modelling "architect" !

They would each have "an own", private and "inviolable" office. After a very few days of domain modelling they would

- each morning review the previous day's work of a colleague, on a rotating shift basis, a "new colleague" on consecutive days;
- meet around a coffee/tea machine and a white board mid-morning for the possible discussion of common issues – across their modelling – while also handing back the possibly annotated work of their reviewed colleague;
- go back to correcting possible collegial remarks;
- and otherwise continue their main assigned work !

26.3 Formality and Verification

Jean-Raymond Abrial[89] passed away 26 May 2025. He was one of the greats of our science. His contributions, especially through *Z, B* and *The B Method*s [2, 1] to *construction by proof* are seminal.

So where, in our description, do we find "traces" of that ?

The answer is: nowhere !

Why ?

Well, usually proof of program correctness is usually [carried out] with respect to some property, some "prior" specification. For domains there is no prior "specification" ! There is the manifest reality of the subject domain.

Thus we must first specify, i.e., describe that domain.

A domain description, a domain model, cannot be said to be correct.

It is either a bad, or a not so bad, or not quite so "approximate" a description as to be accepted by domain stakeholders; or it is a reasonably good model.

Verification of a domain model is by its acceptance by domain stakeholders.

When, below, we refer to verification we mean that properties of the description can be expressed, in mathematical logic and then formally proved: verified, tested, checked !

• • •

But: But the above is not good enough ! Certainly J.R. Abrial's work must or ought apply here ! ? A study should be made, by professionals well-familiar with, for example, `Event B`[90]. Based on the description/modelling taxonomy, cf. Fig. 2.1, it might very well be possible to formulate the formal model along the principles set out by J.R. Abrial

• • •

The next remarks were written before the J.R. Abrial discourse above.

• • •

[89] https://en.wikipedia.org/wiki/Jean-Raymond_Abrial
[90] https://www.event-b.org/, https://www.southampton.ac.uk/~tsh2n14/publications/chapters/eventb-dbook13.pdf

The reader may well have observed two aspects of our "formal" model:

- (i) **"Formality" of the Specification:** I have been rather "lax", some would say, in my use for RSL. An example is "trick", referred to in footnote 67 on page 95, and used in several formal parameter of behaviours. Other examples is the use of discriminated union of ::-defined command types. These "lax" uses have been done, deliberately, in the interest of shortening the formulas. They can all be edited into "correct" RSL.
- (ii) **Lack of Verification:** Yes, indeed. I have not been as careful as I would wish, to highlight all the places where appropriate **theorem**s should be enunciated, let alone proved. Similarly for **axiom**s. I trust the reader can spot these places. And I trust that appropriate proofs be provided. Not necessarily formal proofs in the sense of there being a proof system for the RSL for all of these cases: there is not. But then I am "almost" sure that classical proofs, such as mathematicians "always" do, can suffice. And, for cases that that is not immediately possible ? Well, great, then this domain description provides rich possibilities for the able computer scientist to excel !

26.4 On the Development of This Model

I started on this document on Saturday February 22, 2025. I finished, "more-or-less" all the formalization and this concluding section on Monday March 3, 2025. Nine days, Nine days of great fun.

I am not really ashamed to confess that other than the RSL formula text editing system I have not had access to proper RSL tools, such as they indeed do exist. Thus I have not been able to more-or-less automatically check my RSL formulas. Et cetera - et cetera !

During the development many model-formulations changed. Figure 16.1 on page 72, for example, underwent numerous versions.

26.5 Acknowledgements

Chapter 27

Bibliography

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

[2] Jean-Raymond Abrial. From Z to B and then Event B: Assigning Proofs to Meaningful Programs. In *IFM 2013*, LNCS 7940, Åbo, Finland, June 2013. Springer.

[3] Dines Bjørner. Domain Case Studies:

- 2025: *Documents – a Domain Description*, Winter/Spring 2025, www.imm.dtu.dk/~dibj/2025/documents/main.pdf
- 2023: *Nuclear Power Plants, A Domain Sketch*, 21 July, 2023 www.imm.dtu.dk/~dibj/2023/nupopl/nupopl.pdf
- 2021: *Shipping*, April 2021. www.imm.dtu.dk/~dibj/2021/ral/ral.pdf
- 2021: *Rivers and Canals – Endurants*, March 2021. www.imm.dtu.dk/~dibj/2021/Graphs/Rivers-and-Canals.pdf
- 2021: *A Retailer Market*, January 2021. www.imm.dtu.dk/~dibj/2021/Retailer/BjornerHeraklit27January2021.pdf
- 2019: *Container Terminals*, ECNU, Shanghai, China www.imm.dtu.dk/~dibj/2018/yangshan/maersk-pa.pdf
- 2018: *Documents*, TongJi Univ., Shanghai, China www.imm.dtu.dk/~dibj/2017/docs/docs.pdf
- 2017: *Urban Planning*, TongJi Univ., Shanghai, China www.imm.dtu.dk/~dibj/2017/urban-planning.pdf
- 2017: *Swarms of Drones*, IS/CAS[91], Peking, China www.imm.dtu.dk/~dibj/2017/swarms/swarm-paper.pdf
- 2013: *Road Transport*, Techn. Univ. of Denmark www.imm.dtu.dk/~dibj/road-p.pdf
- 2012: *Credit Cards*, Uppsala, Sweden www.imm.dtu.dk/~dibj/2016/credit/accs.pdf
- 2012: *Weather Information*, Bergen, Norway www.imm.dtu.dk/~dibj/2016/wis/wis-p.pdf
- 2010: *Web-based Transaction Processing*, Techn. Univ. of Vienna, Austria, 186 pages www.imm.dtu.dk/~dibj/wfdftp.pdf
- 2010: *The Tokyo Stock Exchange*, Tokyo Univ., Japan www.imm.dtu.dk/~db/todai/tse-2.pdf
- 2009: *Pipelines*, Techn. Univ. of Graz, Austria www.imm.dtu.dk/~dibj/pipe-p.pdf
- 2007: *A Container Line Industry Domain*, Techn. Univ. of Denmark www.imm.dtu.dk/~dibj/container-paper.pdf
- 2002: *The Market*, Techn. Univ. of Denmark www.imm.dtu.dk/~dibj/themarket.pdf
- 1995–2004: *Railways*, Techn. Univ. of Denmark - a compendium www.imm.dtu.dk/~dibj/train-book.pdf

Experimental research carried out to "discover", try-out and refine method principles, techniques and tools, 1995–2025.

[91] Inst. of Softw., Chinese Acad. of Sci.

[4] Dines Bjørner. Manifest Domains: Analysis & Description www.imm.dtu.dk/~dibj/2015/faoc/faoc-bjorner.pdf. *Formal Aspects of Computing*, 29(2):175–225, March 2017. Online: 26 July 2016.

[5] Dines Bjørner. Domain analysis & description - the implicit and explicit semantics problem www.imm.dtu.dk/~dibj/2017/bjorner-impex.pdf. In Régine Laleau, Dominique Méry, Shin Nakajima, and Elena Troubitsyna, editors, Proceedings Joint Workshop on *Handling IMPlicit and EXplicit knowledge in formal system development (IMPEX)* and *Formal and Model-Driven Techniques for Developing Trustworthy Systems (FM&MDD),* Xi'An, China, 16th November 2017, volume 271 of *Electronic Proceedings in Theoretical Computer Science*, pages 1–23. Open Publishing Association, 2018.

[6] Dines Bjørner. Domain Analysis & Description. www.imm.dtu.dk/~dibj/2018/tosem/Bjorner-TOSEM.pdf. *ACM Trans. on Software Engineering and Methodology*, 28(2):66 pages, March 2019.

[7] Dines Bjørner. *Domain Science & Engineering – A Foundation for Software Development*. EATCS Monographs in Theoretical Computer Science. Springer, Heidelberg, Germany, 2021. A revised version of this book is [9].

[8] Dines Bjørner. Double-entry Bookkeeping. Research, Institute of Mathematics and Computer Science. Technical University of Denmark, DK-2800 Kgs.Lyngby, Denmark, August 2023. http://www.imm.dtu.dk/~dibj/2023/doubleentry/dblentrybook.pdf. One in a series of planned studies: [10, 16, 15, 14].

[9] Dines Bjørner. Domain Modelling – A Primer. A significantly revised version of [7]. xii+202 pages[92], Summer 2024.

[10] Dines Bjørner. Banking – A Domain Description. Sci. & techn. study, Technical University of Denmark, Fredsvej 11, DK 2840 Holte, Denmark, March 2025. One in a series of planned studies: [16, 15, 14, 8].

[11] Dines Bjørner. Documents – A Domain Description. Sci. & techn. study, Technical University of Denmark, Fredsvej 11, DK 2840 Holte, Denmark, March 2025. One in a series of planned studies: [10, 16, 15, 14, 8].

[12] Dines Bjørner. Domain Analysis & Description. *To be submitted*, page 33, March 2025. Institute of Mathematics and Computer Science. Technical University of Denmark.

[13] Dines Bjørner. Domain Modelling. *Submitted to ACM FAC*, page 18, February 2025. Institute of Mathematics and Computer Science. Technical University of Denmark.

[14] Dines Bjørner. Health Care – A Domain Description. Sci. & techn. study, Technical University of Denmark, Fredsvej 11, DK 2840 Holte, Denmark, March 2025. One in a series of planned studies: [10, 16, 15, 8].

[15] Dines Bjørner. Insurance – A Domain Description. Sci. & techn. study, Technical University of Denmark, Fredsvej 11, DK 2840 Holte, Denmark, March 2025. One in a series of planned studies: [10, 16, 14, 8].

[16] Dines Bjørner. Transport – A Domain Description. Sci. & techn. study, Technical University of Denmark, Fredsvej 11, DK 2840 Holte, Denmark, March 2025. One in a series of planned studies: [10, 15, 14, 8].

[17] Dines Bjørner, Chr. Gram, Ole N. Oest, and Leif Rystrøm. Dansk Datamatik Center. In Benkt Wangler and Per Lundin, editors, *History of Nordic Computing*, Stockholm, Sweden, 18-20 October 2010. Springer.

[18] Dines Bjørner and Ole N. Oest. The DDC Ada Compiler Development Project. In Dines Bjørner and Ole N. Oest, editors, *Towards a Formal Description of Ada, [19]*, volume 98 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 1980.

[19] Dines Bjørner and Ole N. Oest, editors. *Towards a Formal Description of Ada*, volume 98 of *Lecture Notes in Computer Science*. Springer, Heidelberg, Germany, 1980.

[20] Andrew Kennedy. *Programming languages and dimensions*. PhD thesis, University of Cambridge, Computer Laboratory, April 1996. 149 pages: cl.cam.ac.uk/techreports/UCAM-CL-TR-391.pdf. Technical report UCAM-CL-TR-391, ISSN 1476-298.

[92] This book is currently being translated into Chinese by Dr. Yang ShaoFa, IoS/CAS (Institute of Software, Chinese Academy of Sciences), Beijing and into Russian by Dr. Mikhail Chupilko and colleagues, ISP/RAS (Institute of Systems Programming, Russian Academy of Sciences), Moscow

Part VIII

APPENDIX

Appendix A

Indexes

A.1 Transport Domain Concepts

A.2 Domain Modelling Ontology

A.3 Formal Entities

The formal entries first lists formula entries by ontological category, then all:

Unique Identification

- * **Unique Identifiers: Sorts and Observers**
- * **A Unique Identifier State Concept**
- * **A Wellformedness Axiom**

Mereology

- * **Mereology: Sorts and Observers**
- * **A Wellformedness Axiom**

Attributes

- * **Attributes: Sorts and Observers**
- * **Wellformedness Axioms**

* **Intentional Pull**

* **Commands**

Perdurants

- * **Communication**
- * **Messages**
- * **Behaviour Signatures**
- * **Behaviour Definitions**
- * **Initialization**

* **Values**
* **Auxiliary Types**
* **Auxiliary Functions**
* **Theorems**

Only the *'ed entries are listed.

There are 483 formal RSL entities, and there are 504 RSL definitions – the former counted among the latter.

Appendix B

Summaries

B.1 Commands

ι222 π73. [k1] CustQuery ::<QueryId×QueryComp

ι260 π80. [k2] ConvCompOffer :: CKI×ContractNu×QueryNu×(ChoiceNu $\overrightarrow{m}$ OfferChoice)

ι260e π80. OfferChoice = TR×Cost

ι223 π73. [k3] CustOrd :: QueryId×ContractNu×OrdrComp

ι261 π80. [k4] ConvCompOrdOK :: CKI×ContractNu×ChoiceNu×TR×Cost

ι224 π73. [k5] OrderOK :: ContractNu×ChoiceNu×Payment

ι262a π80. [k7] ConvCompConvDir :: CKI×ContractNu×Segment

ι253 π79. [k8] PendColl :: (NI×ContractNu×MI-**set**) mayby not the MI-**set**

ι224 π73. [k9] KNTransfer :: ContractNu×M-**set**

ι254 π79. [k10] Notify :: AtNode | OnEdge

ι255 π79. [k11a] NCTransfer :: ContractNu $\overrightarrow{m}$ M-**set**

ι256 π79. [k11b] CNTransfer :: ContractNu $\overrightarrow{m}$ M-**set**

ι257 π79. [k12] Acknowledgment :: $\mathbb{TIME}$×ContractNu×((NI×CI)|(CI×NI))

ι259 π79. [k13] PendDel :: NI×ContractNu×MI-**set** mayby not the MI-**set**

ι265 π81. [k14a] NKTransfer :: NI×ContractNu×MI-**set** mayby not the MI-**set**

ι252 π79. [k15a] Acknowledgment :: $\mathbb{TIME}$×ContractNu×(NI×KI)

ι252 π79. [k15b] Acknowledgment :: $\mathbb{TIME}$×ContractNu×(KI×NI)

B.2 Mereologies and Attributes

B.2.1 Customers

Mereology:

ι168 π48. KM = MI-**set** × (CKI|LI)-**set** × CI-**set**

Attributes:

ι169 π48. CustId = CustNam × CustAdd × ...

ι170 π48. Possess = MI-**set**

ι171 π48. OutReqs = ...

ι172 π48. CustHist = ($\mathbb{TIME}$ × Event)*

ι173 π48. Event = ...

ι174 π48. ...

B.2.2 Conveyor Companies

Mereology:

ι192 π54. CAM = CI-**set** $\times$ COI

Attributes:

ι201 π56. ConvCompInfo = ...

ι203 π56. Contracts = ContractNu $\overrightarrow{m}$ Move*

ι203a π56. Move = (KI$\times$NI)|(NI$\times$CI)|(CI$\times$NI)|(NI$\times$KI)

ι204 π56. Orders = ContractNu $\overrightarrow{m}$ Offers

ι204a π56. ContractNu

ι204b π56. Offers = ChoiceNu $\overrightarrow{m}$ TR

ι204c π56. ChoiceNu

ι205 π56. CurrBuss = MSG-**set**

ι206 π56. PastBuss = MSG-**set**

ι207 π56. CKHist = MSG*

B.2.3 Conveyors

Mereology:

ι210 π59. CM = (NI|EI)set $\times$ CKI-**set** $\times$ KI-**set**

Attributes:

ι211 π60. Kind

ι212 π60. Stowage = ContractNu $\overrightarrow{m}$ M-**set**

ι215 π60. TBU,TBL = NI $\overrightarrow{m}$ ContractNu-**set**

ι213 π60. SR = Path

ι214 π60. SRIndex = Na

ι216 π60. Finals = NI $\overrightarrow{m}$ (KI $\overrightarrow{m}$ ContractNu)

ι216 π60. Final = NI $\times$ ContractNo $\times$ KI

ι217 π60. CPos = OnEdge $(= \text{NI}\times(\text{F}<>\text{EI})\times\text{NI})$

ι217 π60. CPos = AtNode (= NI)

ι219 π60. CHist = MSG*

B.2.4 Nodes and Edges

Mereology:

ι42 π14. NM = EI-**set** **axiom** $\forall$ nm:NM • **card** nm>0

ι43 π14. EM = NI-**set** **axiom** $\forall$ em:EM • **card** em=2

Attributes:

ι55 π18. NodeKind = Kind-**set** **axiom** $\forall$ nk:NodeKind • nk$\neq${}

ι56 π18. EdgeKind = Kind-**set** **axiom** $\forall$ ek:EdgeKind • **card** ek=1

ι57 π18. LEN = **Nat**

ι58 π18. COST = **Nat**

ι320 π115. OnHold = ContractNu $\overrightarrow{m}$ M-**set**