Transport – a Domain Description

DINES BJØRNER, Technical University of Denmark – March 4, 2025, Denmark

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:* sequence of node and edge labels. Nodes (now called *hubs*) are then interpreted a street intersections, bus stops, railway stations, harbours and airports and edges as *links* between neighbouring nodes as street segments, bus links, simple rail lines, and simple air links. And from there it goes !

Dines Bjørner. 2025. Transport – a Domain Description. 1, 1 (March 2025), 33 pages.

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, [6, 8, 10–12], of domain studies of such infrastructure components as government, public utilities, banking, transport, insurance, health care, etc. The current, this 'Introduction', section is common to most of 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 of them already mentioned above: government structure: legislative, executive & judicicial units, transport: roads, navigable rivers and lakes, the open sea, banking, educational system, health care, utilities: water, electricity, telecommnications (e.g. the `Internet) gas, ` , etc.,[2]

Also: **[To be translated back into English:]** Winston Churchill citeres for i Underhuset, i 1946, at have sagt: *. . . Den unge Labour–taler, som vi netop har lyttet til, ønsker klart at imponere sin*

[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.

Author's address: Dines Bjørner, Technical University of Denmark – March 4, 2025, DTU Compute, Fredsvej 11, Holte, 2840, Denmark, bjorner@gmail.com.

valgkreds med det faktum at han har gået på Eton og Oxford siden han nu bruger sådanne moderne termer som 'infra–struktur' ...

1.2 Domain Models

1.2.1 Some Characterizations.

(1) 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*.
(2) 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].
(3) 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].
(4) 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 we shall show in Sect. 2 (pages 3–23).
(5) 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, [10, 11], we shall therefore present the result of infra structure studies. We have, over the years, developed many domain models: [1].

1.2.3 Domain Science & Engineering. In a series of publications: [2, 4, 5, 7, 9] I have developed 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[4]. 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[5]. The "laws" that these properties obey are not necessarily quite the same around the universe !

1.3.2 The Dichotomy. For our society to work, we are being educated (in primary, secondary, tertiary schools, colleagues 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,

[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.
[4]physical & chemical, botanical & zoological, geological & geographic, etc.
[5]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, ...),

electricity, chemistry, botanics, zoology, geology, geography, ...), but we are not taught about most of the infrastructure structures[6]. 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:

- **Banking**[7] [8]
- **Transport**[8] [12]
- **Insurance**[9] [11]
- **Health Care**[10] [10]
- etc.

A report on *double-entry bookkeeping* [6] relates strongly to most of these infra-structure component domains[11].

2 A FORMAL DOMAIN DESCRIPTION

Appendix A on page 26 outlines the textual structure of a domain description. It refers to the formal specification language RSL [13] for which Appendix B on page 26 gives an ultra-brief summary.

The formal domain description introduces over 180 identifiers, i.e., defines that many kinds of formal entities. These identifiers designate types, values, functions (auxiliary and otherwise), behaviours, actions, etc. They are [obviously] being used. To easen the domain describer and the domain description reader around these identifiers, an index is provided in Sect. D.3 on page 28.

2.1 Kind of Transport Graphs and Conveyors

2.1.1 ***Informal Outline.*** The transport we have in mind consists of a common transport net, in the following modelled as a graph of uniquely labelled, bi-directed edges and likewise labelled nodes. The transport net is ["intentional pull"] complemented, cf. Sect. 2.5 on page 16, by a set of conveyors.

Edges, nodes and conveyors are "of kind": **"road"**, **"rail"**, **"sea"**, and **"air"**. [**"road", "rail", "sea", "air"** are literal values of type **Text**] A conveyor is of one kind. Conveyors of kind **"road"** include cars, 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 private airplanes and 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 andair lanes. A node may be of one or more kinds. Nodes of kind **"road"** are called street point (street crossings,

[6] See footnote 5 on the facing page.
[7] https://www.imm.dtu.dk/ dibj/2025/infra/banking.pdf
[8] https://www.imm.dtu.dk/ dibj/2025/infra/main.pdf
[9] https://www.imm.dtu.dk/ dibj/2025/infra/insurance.pdf
[10] https://www.imm.dtu.dk/ dibj/2025/infra/healthcare.pdf
[11] `http://www.imm.dtu.dk/ dibj/2023/doubleentry/dblentrybook.pdf`

street ends, bus stops). Nodes of kind **"rail""** are called train stations. Nodes of kind **"sea"** are called harobours. Nodes of kind **"air"** are called airports.

2.1.2 ***Narrative & Formalization.***

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

```
type
6.  Kind = "road"|"rail"|"sea"|"air"
```

• • •

We divide the 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[12] and a conveyor[13] sub-domain.

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

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 intersection* s (hubs); *train station* s; *river* , *canal* and *sea harbour* s; and *airport* s. A node may be one or more of these. Edges are accordingly interpreted as either *street* (or *road*) *link* s, *rail track* s, *sail* ing or *air route* s. An edges can be only one of these. Hence there may be many edges between any two [neighbouring] nodes.

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

The presentation follows the ontology of Fig. 1 on the next page.

2.2 Overall Transport Endurants

2.2.1 ***External Qualities.***

2.2.1.1 Endurant Sorts & Observers. ‘

(7) There is the domain of transport.
(8) From transport endurants we can observe [transport] graphs.
(9) And from transport endurants we can observe [transport] a conveyor aggregate.

```
type
7.  T
8.  G
9.  CA
value
8.  obs_G: T→G
9.  obs_CA: T→CA
```

2.2.1.2 An Endurant State Notion. We can speak of a transport state.

(10) There is given a "global" transport value, t. It contributes to a transport state.
(11) From this transport value one can derive another transport state element: a global graph value, g.
(12) And from this transport value one can derive another transport state element: a global conveyor aggregate value, ga.
(13) We can postulate a transport state to consist of the three endurants: t, g, ca.

[12] https://en.wikipedia.org/wiki/Graph_(discrete_mathematics)
[13] Conveyor: anything that conveys, transports or delivers. Words are a conveyor of meaning [https://en.wiktionary.org/-wiki/conveyor]

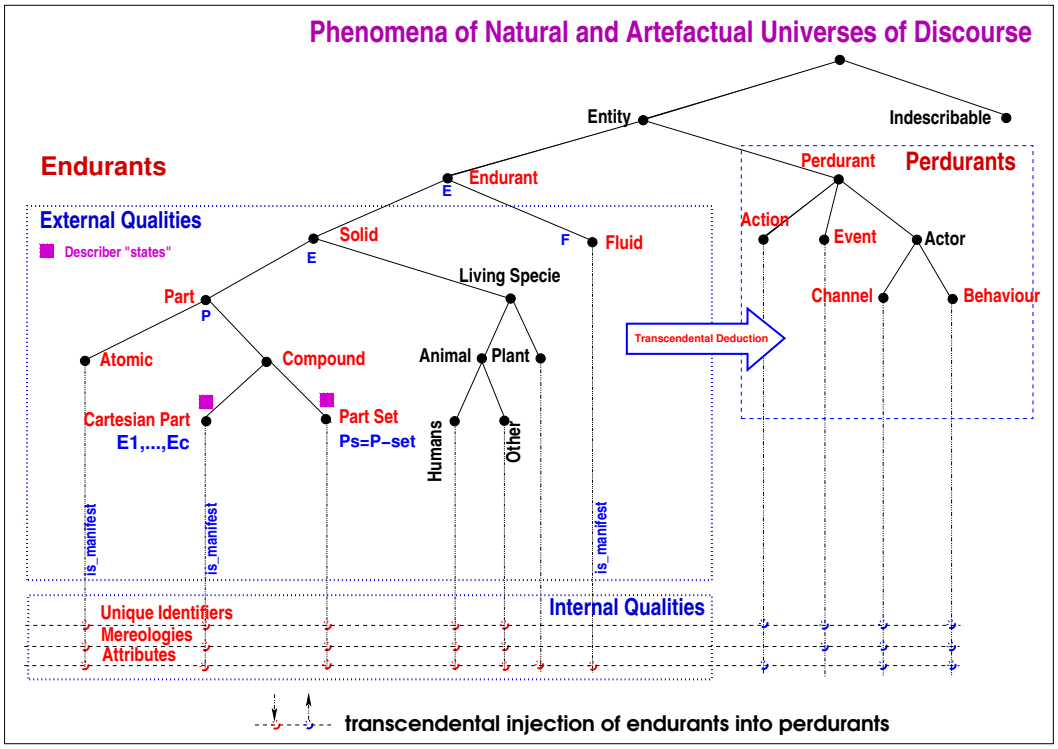

Fig. 1. Domain Analysis Ontology

value

10. t
11. g:G = **obs_**G(t)
12. ca:CA = **obs_**CA(t)
10. σ_t = $\{t\}\cup\{g\}\cup\{ca\}$

2.2.2 *Internal Qualities.*

2.2.2.1 Unique Identification.

2.2.2.1.1 Unique Identifier Sorts & Observers

(14) The transport endurant has a unique identifier.
(15) So has the graph, and
(16) the conveyor components.

type

14. TI
15. GI
16. CAI

value

14. **uid_**T: T → TI

```
15.  uid_G: T → GI
16.  uid_CA: T → CAI
```

2.2.2.1.2 A Unique Identifier State Notion

(17) We an postulate a "global" transport state value, t.
(18) Given t we can derive a "global" graph value g.
(19) And a "global" conveyor aggregate value ca.
(20) We can therefore postulate an "uppermost" endurant transport state to consist of the three endurants: t, g, ca.

```
value
17.  ti:TI =
18.  gi:GI = uid_G(g)
19.  cai:CAI = uid_CA(ca)
20.  σ_tuis = {ti}∪{gi}∪{cai}
```

2.2.2.1.3 Uniqueness

(21) The three ["uppermost"] transport endurants are distinct: have distinct unique identifiers.

```
axiom [Uniqueness of Transport Identifiers]
21.  card σ_t = card σ_tuis = 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.

2.3 Graph Endurants

Endurants have both external and internal qualities.

2.3.1 ***External Qualities.*** External qualities are the endurant sorts, their observers and endurant states.

2.3.1.1 The Endurant Sorts and Observers.

(22) From graphs one can observe an aggregate, i.e., a set, *ea:EA*, of edges –
(23) From graphs one can observe an aggregate, i.e., a set, *na:NA*, of nodes –
(24) From an aggregate of edges one can observe a set of edges.
(25) From an aggregate of nodes one can observe a set of nodes.
(26) Edges are atomic.
(27) Nodes are atomic.
(28) We can "lump" all endurants into a sort *parts*.

```
type
22.  EA
23.  NA
24.  ES = E-set
25.  NS = N-set
26.  E
27.  N
28.  P = G|EA|NA|ES|NA|N|E
value
22.  obs_EA: G → ES
23.  obs_NA: G → NS
24.  obs_ES: EA → ES
25.  obs_NS: NA → NS
```

A transport domain taxonomy is hinted at in Fig. 2.

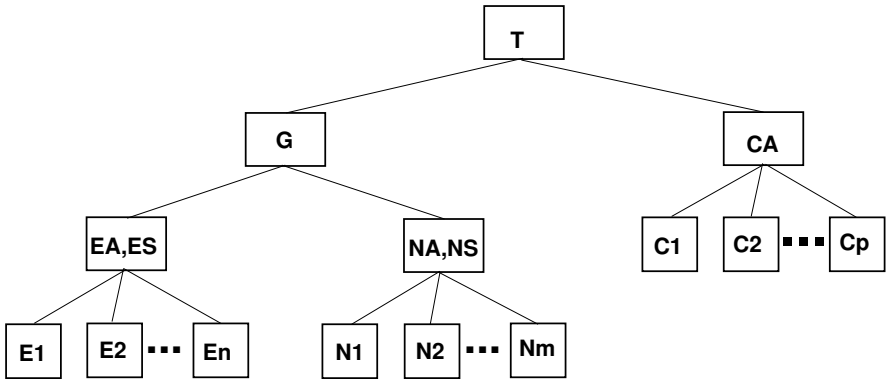

Fig. 2. A Transport Domain Taxonomy

2.3.1.2 An Endurant State.

(29) Given the global graph value, there is therefore a "global" value of an edge aggregate.
(30) Given the global graph value, there is therefore a "global" value of a node aggregate.
(31) Given the global edge aggregate value, there is therefore a "global" node value of of the set of all edges.
(32) Given the global graph value, there is therefore a "global" value of the set of all nodes.
(33) The state of all graph endurants is therefore the set of all graph parts.

```
value
29.  ea = obs_EA(g)
30.  na = obs_NA(g)
31.  es = obs_ES(g)
32.  ns = obs_NS(g)
33.  σps:P-set = {g}∪{ea}∪{na}∪es∪ns
```

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

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

2.3.2.1.1 Unique Identifier Sorts and Observers

(34) All parts have identification:
(35) the graph,
(36) the edge and node aggregates,
(37) the sets of edges and nodes, and
(38) each edge and node.
(39) No two of these are the same, i.e., part identifiers are unique.

```
type
34.  PI = GI|EAI|NAI|ESI|NSI|EI|NI
34.  GI,EAI,NAI,ESI,NSI,EI,NI
value
```

```
35. uid_G: G→GI
36. uid_EA: EA→EAI, uid_NA: NA→NAI
37. uid_ES: ES→ESI, uid_NS: NS→NSI
38. uid_E: E→EI, uid_N: N→NI
```

2.3.2.1.2 A Unique Identifier State

(40) There is a "global" unique graph identifier.
(41) There are, correspondingly, "global" edge and node aggregate identifiers.
(42) There are, correspondingly, "global" edge set and node set identifiers; and
(43) set of edge identifiers and
(44) set of node identifiers.
(45) The unique identifier state is the union of all the unique identifiers.

```
value
40. gi = uid_G(g)
41. ea_uis = uid_EA(ea) , na_uis = uid_NA(na)
42. es_uis = uid_ES(ea) , ns_uis = uid_NS(na)
43. e_uis = {uid_E(e)|e:E•e∈es}
44. n_uis = {uid_N(n)|n:N•n∈ns}
45. σ_uis:PI-set = {uid_P(p)|p:P•p∈σ}
axiom
45. σ_uis = {gi}∪{ea_uis}∪{na_uis}∪{es_uis}∪{ns_uis}∪e_uis∪n_uis
```

2.3.2.1.3 Uniqueness

(46) No two of these are the same, i.e., part identifiers are unique.

```
axiom [Uniqueness of Part Identification]
46. cardσ=cardσ_uis
```

2.3.2.2 Mereology. Mereology has three facets: types, observers and wellformedness.

2.3.2.2.1 Mereology Types and Wellformedness, I

(47) The mereology of a node is a non-empty set of edge identifiers.
(48) The mereology of an edge is a set of two node identifiers.

```
type
47. NM = EI-set axiom ∀ nm:NM • card nm>0
48. EM = NI-set axiom ∀ em:EM • card em=2
```

2.3.2.2.2 Mereology Observers

```
value
47. mereo_N: N → NM
48. mereo_E: E → EM
```

2.3.2.2.1 Mereology Wellformedness, II

(49) The unique identifiers of a node must be those of the edges of the graph.
(50) The unique identifiers of an edge must be those of the nodes of the graph.

```
axiom [Mereology Wellformedness]
49. ∀ n:N•mereo_N(n)⊆es_uis
50. ∀ e:E•mereo_E(e)⊆ns_uis
```

2.3.2.2 Paths of a Graph.

(51) A path (of a graph) is a finite[14] 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".
(52) Given a node [an edge] identifier we can retrieve the identified node [edge].

```
type
51.  Path = (EI|NI)*
axiom [Well-formed Paths]
51.  ∀ path:Path •
51a.    ∀ {i,i+1}⊆inds path ⇒
51a.        ( (is_NI(path[ i ])∧is_EI(path[ i+1 ])
51a.           ∨ is_EI(path[ i ])∧is_NI(path[ i+1 ]))
51b.        ∧ (path[ i ]∈ns_uis⇒path[ i+1 ]∈es_uis
51c.           ∧ uid_N(retr_node(path[ i ]))∈mereo_E(retr_node(path[ i ])))
51d.        ∧ (path[ i ]∈es_uis⇒path[ i+1 ]∈ns_uis
51e.           ∧ uid_E(retr_edge(path[ i ])∈mereo_N(retr_edge(path[ i ]))))
51f.        ∧ kind(retr_unit(path[i]))∩kind(retr_unit(path[i+1]))≠{})
value
52.  retr_node: NI → N, retr_edge: EI → E, retr_unit: UI → U
52.  retr_node(ni) as n • n ∈ ns ∧ uid_(n)=ni
52.  retr_edge(ei) as e • e ∈ es ∧ uid_(e)=ei
52.  retr_unit(i) as u • ∈ ns∪es ∧ uid_U(u)=i
52.  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.

(53) The paths[15] function takes a graph and yields a possibly infinite set of paths – satisfying the above wellformedness criterion.

[14] We shall only consider finite paths. The paths function, Item 53 below, can easily be modified to yield also infinite length paths !

[15] **Alarm !** Check that this function indeed generates only finite length paths !

We define the paths function in two ways.

(54) Either axiomatically

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

(56) or inductively[16]:

(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.

```
value
53.  paths: G → Path-infset
54.  paths(g) as ps
55.      such that: ∀ p:ps satisfy the above wellformedness
56.  paths(g) ≡
56a.     let ps = {⟨ni⟩ | ni:NI ∈ nsuis}∪{⟨ei⟩ | ei:EI ∈ esuis}
56f.           ∪ { pi⌢⟨ui⟩⌢⟨uj⟩⌢pj | pi⌢⟨ui⟩:Path-set, ⟨uj⟩⌢pj:Path-infset •
56b.                   ∧ ({pi⌢⟨ui⟩,⟨uj⟩⌢pj}⊆ps
56c.                   ∧ (ui~ ∈ elems pj ∧ uj~ ∈ elems pi)
56e.                   ∧ (ui ∈ mereo_U(retr_unit(uj))
56e.                   ∧  uj ∈ mereo_U(retr_unit(ui)))} in
56g.     ps end
type
53.  U = E|N
```

Solution to the equation, lines 56a–56c, is "'obtained' by a smallest set fix-point reasoning.

(57) Given a "global" graph, *g*, we can calculate a "similarly global" *paths* value:

```
value
57.  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. 2.3.2.3, 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.

[16] https://www.cs.odu.edu/ toida/nerzic/content/recursive_def/more_ex_rec_def.html

(58) 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 their reversed path]
58.  ∀ g:G, p:Path•p ∈ paths(g) ⇒ rev(p) ∈ paths(g)
value
58.  rev: P → P
58.  rev(p) ≡
58.      case p of
58.          ⟨⟩ → ⟨⟩,
58.          ⟨ui⟩ → ⟨ui⟩,
58.          ⟨ui⟩⌢p′⌢⟨uj⟩ → ⟨uj⟩⌢rev(p′)⌢⟨ui⟩
58.      end
```

We can define auxiliary functions, for example:

(59) Given a kind we can select all the graph paths of that kind.

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

2.3.2.3 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.

2.3.2.3.1 Attribute Types & Observers We introduce but just a few Graph Attributes.

(60) 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:

(61) 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.

(62) From an edge we can we can observe its length[17]. [Static Attribute]

(63) and the cost[18] of using the edge[19]. [Static Attribute]

```
type
60.   NodeKind = Kind-set  axiom ∀ nk:NodeKind • nk≠{}
61.   EdgeKind = Kind-set  axiom ∀ ek:EdgeKind • card ek=1
62.   LEN = Nat
63.   COST = Nat
```

[17] 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.

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

[19] See [3]. The usual arithmetic operators apply: scaling between ... Check also [15].

```
value
60.   attr_NodeKind: N → NodeKind
61.   attr_Edgekind: E → EdgeKind
62.   attr_LEN: E → LEN
63.   attr_COST: E → COST
```

(64) Given a node or an edge we can observe its kinds.
(65) Given a graph, and a "kind", we can calculate all its paths of the same kind.
(66) Given a finite route we can we can calculate its lengths
(67) and costs.
(68) We can also calculate the shortest route, possibly a set, of a graph,
(69) and the least costly,
(70) etc.

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

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

66.  length: P → LEN
66.  length(p) ≡
66.      case p of
66.          ⟨⟩ → 0
66.          ⟨ui⟩ → retr_length(ui),
66.          ⟨ui⟩⌢p′ → retr_length(ui)+length(p′)
66.      end
66.   retr_length: UI → LEN
66.   retr_length(ui) ≡ (is_EI(ui)→attr_LEN(retr_edge(ui)),is_NI(ui)→0)

67.  cost: P → LEN
67.  cost(p) ≡
67.      case p of
67.          ⟨⟩ → 0
67.          ⟨ui⟩ → retr_cost(ui),
67.          ⟨ui⟩⌢p′ → retr_cost(ui)+cost(p′)
67.      end
67.   retr_cost: UI → COST
67.   retr_cost(ui) ≡ (is_EI(ui)→attr_COST(retr_edge(ui)),is_NI(ui)→0)

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

```
69.  least_costly_route: G → P-set
69.      let ps = paths(g) in
69.      { p | p:P • retr_cost(p) ∧ ∀ p':P•p'∈ps ∧ retr_cost(p)≤retr_cost(p') }
69.      end

70.  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"].

2.3.2.3.3 Attribute Wellformedness

(71) If a node is of some kind, then there must be at least one edge leading to/from it of the same kind.
(72) If an edge is of some kind, then the nodes connected to it must also be of that [same] kind.
(73) 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
71.
71.

72.
72.

73.
73.
```

2.4 Conveyor Endurants

2.4.1 External Conveyor Qualities.

2.4.1.1 Conveyor Endurant Sorts & Observers.

(74) From a conveyor aggregate one can observe a finite set of conveyors.
(75) A conveyor is either a

- a road conveyor
  - car,
  - taxi,
  - bus,
  - truck,
- or a rail conveyor
  - passenger train,
  - freight train,
- or a water conveyor
  - sailboat,
  - barge,
  - fishing vessel,
  - freighter,
  - passenger liner,
- or an airborne conveyor
  - civil aircraft,
  - freight plane, or
  - passenger aircraft !

(76) Conveyors are atomic parts.
(77) Conveyors or "of kind".

```
type
74.  CS = C-set
75.  C = Road|Rail|Water|Air
```

```
75.  Road = Car|Taxi|Bus|Truck
75.  Rail = PassTrain|FreightTrain
75.  Water = SailBoat|Barge|FishVessel|Freighter|Ferry|PassLiner
75.  Air = PrivAir|Helicop|FreightPlane|PassAir
value
74.  obs_CS: CA → CS

77.  c_kind: C → Kind
77.  c_kind(c) ≡
77.     is_Road(c) → "road", is_Rail(c) → "rail",
77.     is_Water(c) → "sea", is_Air(c) → "air"
```

2.4.1.3 A Conveyor Endurant State.

(12) Given a "global" transport value, t, we can postulate a conveyor aggregate state, ca – as we already did in Sect. 2.2.1.2 Page 4.
(78) And, given ca, we can postulate a state of the set, cs, of all conveyors.
(79) An overall state of endurants of a transport domain is therefore the union of all its parts.

```
value
12. ca:CA = obs_CA(t)
78. cs:CS = obs_CS(ca)
79. σ = σ_ps ∪{ t }∪ cs
```

2.4.2 Internal Conveyor Qualities.

2.4.2.1 Unique Identification.

2.4.2.1.1 Unique Identifier Sorts & Observers

(80) Conveyor aggregates have unique identification.
(81) So have each of the conveyors in their set of conveyors.

```
type
80.  CAI
81.  CI
value
80.  uid_CA: CA → CAI
81.  uid_C: C → CI
```

2.4.2.1.3 Unique Identifier State

(82) The unique identifier of a conveyor aggregate contributes to the unique identifier state for the [entire] transport domain.
(83) The unique identifiers of all conveyors contribute to the unique identifier state for the [entire] transport domain.
(84) The overall unique identifier state, σ_{uis}, is therefore the union of all the unique identifiers of all parts of a transport domain.

```
value
82.  cai:CAI = uid_CA(ca)
83.  cis:CI-set = { uid_C(c) | c:C • c ∈ obs_CS(ca) }
```

84. $\sigma_{uis} = \sigma_p \cup \{cai\} \cup cis$

2.4.2.1.2 Uniqueness

(85) All parts are uniquely identified.

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

2.4.2.1.3 Conveyor Retrieval

(86) From a conveyor identifier one can obtain, via *cs*, the conveyor of that identification.

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

2.4.2.2 Mereology.

2.4.2.2.1 Mereology Types & Observers

(87) The mereology of a conveyor is a finite set of edge and node identifiers that it may "visit".

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

2.4.2.2.2 Mereology Wellformedness

(88) The identifiers of a conveyor mereology must be those of the edges and nodes of the transport graph, *g*.
(89) The kind of conveyor must "fit" the kind of edges and nodes[20].

axiom [Conveyor Mereology of Right Kind]
88. ∀ c:C•c∈*cs*⇒∀ ui:UI•ui ∈**mereo**_C(c)
88. ⇒ ui∈$e_{uis} \cup n_{uis}$
89. ∧ c_kind(c)∩kind(retr_unit(ui))≠{}

2.4.2.3 Attributes.

2.4.2.3.1 Conveyor Attribute Types & Observers

(90) Conveyours are of kind. [Static Attribute]
(91) Conveyors have paths – of their kind – that they [may] travel. [Let us consider that a Static Attribute.]
(92) At any one time a conveyor follows one of these, a current, path. [Programmable Attribute]
(93) These routes must be of the kind of the conveyors traveling them!

[20] 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].

(94) Conveyors either stand still or move. That is, they have position in the graph. Either they are at a node, or somewhere, a fraction, f, of a distance along an edge, from one node to an adjacent. [Programmable Attribute]
(95) We omit further possible attributes: `Speed, Acceleration, Weight, ...`.

```
type
91.  Routes = Path-set
91.  CurrRoute = Path-set
94.  CPos = AtNode | OnEdge
94.  AtNode :: NI
94.  OnEdge :: NI × (F × EI) × NI
94.  F = Real axiom ∀ f:F•0<f<1
value
90.  attr_Kind: C → Kind
91.  attr_Routes: C → Kind
92.  attr_CurrRoute: C → Kind
94.  attr_CPos: C → CPos
95.  ...
axiom [Routes of commensurate. kind]
93.  ∀c:C•let ps=attr_Routes(c)in ∀p:Path•p∈ps∧ps⊆path_kind(p)(kind(c)) end
```

On Routes:

(96) 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., ⟨⟩, 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 [Route Commensurability]
96.   ∀ c:C,r:Routes,cr:CurrRoute • r=attr_Routes(c)∧cr=attr_CurrRoute(c)
96a.      cr ∈ r
96b.   ∧ 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 !

2.5 Intentional Pull

2.5.1 ***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[21], "most likely" do not notice them.

But we, "us", humans, can speak about and recall [these, and "other"[22]] events – and they are therefore an essential aspect of modelling any manifest domain.

(97) We "lump" nodes and edges into single element ways [i.e., endurants].
(98) The ordered, 𝕋𝕀𝕄𝔼[23] -stamped, history attribute event notices record the vehicles, by their unique identifiers.
(99) The ordered, 𝕋𝕀𝕄𝔼-stamped, conveyor history attribute event notices record the ways, by their unique identifiers.

```
type
97.   W = N|E
97.   WI = NI|EI
98.   WH = (s_t:𝕋𝕀𝕄𝔼×VI)*
99.   CH = (s_t:𝕋𝕀𝕄𝔼×CI)*
value
97.   retr_W: WI → N|E
97.   retr_W(wi) ≡ ! w:W • w ∈ns∪es ∧ uid_W(w)=wi
98.   attr_WH: W → WH
98.   attr_CH: C → CH
axiom [Ordered Way and Conveyor Histories]
98.   ∀ wh:WH • {i,i+1}⊆inds wh ⇒ s_t(rh[i])<s_t(wh[i+1])
99.   ∀ ch:CH • {i,i+1}⊆inds ch ⇒ s_t(ch[i])<s_t(ch[i+1])
```

2.5.2 ***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.

(100) 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;
(101) 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:
100.   ∀ c:C • c ∈ cs •
100a.        let ch:CH = attr_CH(c) in
```

[21] 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.
[22] By the seemingly cryptic "other" events, we may, in the context of transport, think of such events as *conveyor breakdown, edge collapse,* etc.
[23] 𝕋𝕀𝕄𝔼 is a "global" phenomenon.

We say *15:23 March 4, 2025 CET*, and mean that it is now *23 minutes past 3pm, 25th of February 2025, Central European Time.*

𝕋𝕀 stands for time-interval.

We say *3 hours and 23 minutes.*

```
100a.       ∃! i:Nat • i ∈ inds ch •
100a.           let (τ,wi) = ch[ i ] in
100b.               let wh:WH = attr_WH(retr_way(wi)) in
100c.               ∃! j:Nat • j∈ inds WH • s_t(wh[ j ]) = τ
100.        end end end
101.   ≡
101.   ∀ w:W • w ∈ es∪ns •
101a.       let wh = attr_WH(w) in
101a.       ∃! k:Nat • k ∈ inds wh •
101a.           let (τ,ci) = wh[ k ] in
101b.               let ch:CH = attr_WH(retr_conveyor(ci)) in
101c.               ∃ℓ:Nat • ℓ∈ inds ch • s_t(ch[ ℓ ]) = τ
101.        end end end
```

2.6 Perdurants

The previous sections, Sects. 2.2–2.5, 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 !

2.6.1 ***Communication***.

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

```
channel
102.    comm[ {i,j} | i,j:UI•{i,j}∈uis ] M
```

(104) A conveyor, ci:CI, at a node decides to remain at that node.
(105) A conveyor, ci:CI, at a node decides to change route.
(106) A conveyor, ci:CI, at a node decides to leave the node, and
(107) to enter an edge.
(108) A conveyor, ci:CI, on an edge decides to move on.
(109) A conveyor, ci:CI, on an edge decides to leave that edge, and
(110) to enter the node.
(111) And a conveyor, ci:CI, at a node or on an edge may decide, "surreptitiously" or otherwide, to just **stop**.

```
type
103.  M =
104.      mkRemain(TIME,CI)
105.    | mkChangeRoute(TIME,CI)
106.    | mkLeaveNode(TIME,CI)
107.    | mkEnterEdge(TIME,CI)
```

```
108.     | mkMove(TIME,CI,OnEdge)
109.     | mkLeaveEdge(TIME,CI)
110.     | mkEnterNode(TIME,CI)
111.     | mkStop(TIME,CI)
```

2.6.2 ***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.

2.6.2.1 Behaviour Signatures.

(112) Conveyor behaviour signatures
(113) Node behaviours
(114) Edge behaviours

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

2.6.2.2 Behaviour Definitions.

2.6.2.2.1 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:

(115) 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
115.   conveyor(ci)(cm)(k,routes)(cr,AtNode(ni),ch) ≡
115a.       conveyor_change_route(ci)(cm)(k,routes)(cr,AtNode(ni),ch)
115b.    ⌈⌉ conveyor_remains_at–node(ci)(cm)(k,routes)(cr,AtNode(ni),ch)
115c.    ⌈⌉ conveyor_entering_edge(ci)(cm)(k,routes)(cr,AtNode(ni),ch)
```

```
115d.   ⫿ conveyor_stops_at_node(ci)(cm)(k,routes)(cr,AtNode(ni),ch)
```

• Conveyor **Actions** at a Node:

(116) 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.

```
116.  conveyor_change_route(ci)(cm)(k,routes)(cr,AtNode(ni),ch) ≡
116a.      let τ = record_TIME()
116b.          ncr = select_next_route(ni,routes),
116d.          ch′ = <mkChange(τ,ni,ncr)>⌢ch in
116c.      ( comm[{ci,ni}] ! mkChangeRoute(τ,ci,ncr)
116c.      ‖ comm[{ci,ni}] ! mkChangeRoute(τ,ci,ncr) ) ;
116e.      conveyor_at_node(ci)(cm)(k,routes)(ncr,AtNode(ni),ch′)  end

116b.  selects_next_route:NI × Routes → CurrRoute
116b.  selects_next_route(ni,routes) as ncr • ncr ∈ routes ∧ hd ncr = ni
```

(117) 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
117.   conveyor_remains_at_node(ci)(cm)(k,routes)(cr,AtNode(ni),ch) ≡
117a.      let τ = record_TIME() in
117b.      comm[{ci,ni}] ! mkRemain(τ,ci);
117c.      conveyor(ci)(cm)(k,routes)(cr,AtNode(ni),⟨(τ,ni)⟩⌢ch) end
```

(118) 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
118.   conveyor_enters_edge(ci)(cm)(k,routes)(cr,AtNode(ni),ch) ≡
118a.     let τ = record_TIME() in
118b.     ( comm[{ci,ni}] ! mkLeaveNode(τ,ni,tl cr)
118b.     ‖ comm[{ci,ni}] ! mkEnterEdge(τ,hd cr,tl cr) ) ;
118c.      let cpos = onEdge(hd r,(,ei),ni′),
118d.          ch′ = <mkChg(τ,ni,ncr)>⌢ch in
```

```
118e.     conveyor(ci)(cm)(k,routes)(cr,cpos,⟨(τ,ni)⟩⌢ch) end end
```

(119) And a conveyor may non-deterministically choose to abandon being a conveyor, i.e., leaving transport altogether – **stop**ping !
(120) But first it notifies the node at which it stops.

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

- A conveyor behaviour on an edge alternates.

• Conveyor **Behaviour** on Edge

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

```
121.  conveyor(ci)(cm)(k,routes)(cr,OnEdge(n_uif,(f,e),n_uit),ch) ≡
121a.     conveyor_moves_on_edge(ci)(cm)(k,routes)(cr,OnEdge(n_uif,(f,e),n_uit),ch)
121c.  ⌈⌉ conveyor_stops_on_edge(ci)(cm)(k,routes)(cr,OnEdge(n_uif,(f,e),n_uit),ch)
121b.  ⌈⌉ conveyor_enters_node(ci)(cm)(k,routes)(cr,OnEdge(n_uif,(f,e),n_uit),ch)
```

• Conveyor **Actions** on an Edge:

(122) 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
122.  conveyor_moves_on_edge(ci)(cm)(k,routes)(cr,OnEdge(n_uif,(f,e),n_uit),ch) ≡
122a.     let τ = record_TIME(),
122b.         ϵ:Real • 0 < ϵ ≪ in
122b.     let f′ = f+ϵ,
122c.         ch′ = ⟨()⟩⌢ch,
122d.         cpos = OnEdge(n_uif,(f′,e),n_uit) in
122e.     comm[{ci,e_j}] ! mkMove(τ,ci,cpos) ;
122f.     conveyor(ci)(cm)(k,routes)(cr,cpos,ch′) end end
122.  pre hd cr = nuif
```

(123) A conveyor **enters** a node
  (a) at time τ is modelled by
  (b) altering its position
  (c) notifying both the edge and designated node behaviours
  (d) updating its history and

(e) become an node behaviour.

```
value
123.   conveyor_enters_node(ci)(cm)(k,routes)(cr,OnEdge(nui_f,(f,e),nui_t),ch) ≡
123a.      let τ = record_TIME(),
123b.          cpos = AtEdge{hd cr} in
123.       comm[{ci,nui_t}] ! mkEnterNode(τ,ci) ;
123d.      let ch' = ⟨mkEnterNode(τ,ci)⟩⌢ch in
123e.      conveyor(ci)(cm)(k,routes)(tl cr,cpos,ch') end end
123.   pre hd cr = n_f
```

(124) A conveyor may non-deterministically choose to abandon being a conveyor, i.e., leaving transport altogether – **stop**ping !
(125) But first it notifies the edge at which it stops.

```
value
124.   conveyor_stops_on_edge(ci)(cm)(k,routes)(cr,OnEdge(nui_f,(f,e),n_ui_t),ch) ≡
125.      let τ = record_TIME() in
125.      comm[{ci,e_j}] ! mkStop(τ,ci) ;
124.      stop end
124.   pre hd cr = nui_f
```

2.6.2.2.2 Node Behaviour

(126) **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
126.   node: NI → NM → ... → NH
126a.  node(ni)(nm)(...)(nh) ≡
126b.      let τ = record_TIME() in
126c.      let msg = [] { comm[{ni,ci}] ? | ci:CI • ci ∈ nm } in
126d.      node(ni)(nm)(...)(⟨msg⟩⌢nh) end end
```

2.6.2.2.3 Edge Behaviour

(127) **Edge** [behaviours] – similarly:
  (a) external non-deterministically accept conveyor, ci, actions
  (b) at times τ
  (c) augment their histories accordingly and
  (d) resumes being edge behaviours.

```
value
127.   edge: EI → EM → ... → EH
127a.  edge(ei)(nm)(...)(nh) ≡
127b.      let τ = record_TIME() in
127c.      let msg = [] { comm[{ei,ci}] ? | ci:CI • ci ∈ em } in
127d.      edge(ni)(em)(...)(⟨msg⟩⌢nh) end end
```

2.6.3 ***Domain Initialization***. By domain initialization we mean the *invocation*[24] of all behaviours.

(128) The overall initialization expresses the parallel composition of the initialization of
(129) all conveyors,
(130) all nodes and
(131) all edges.

```
128.  initialization: Unit → Unit
128.  initializatio() ≡
129.     ‖ { conveyor
129.              (uid_C(c))
129.                 (mereo_C(c))
129.                    (attr_KindC(c),attr_RoutesC(c))       [Static Attrs.]
129.    [Programmable Attrs.]  (attr_CurrRouteC(c),attr_CPoC(c)s,attr_CHC(c))
129.                | c:C•c ∈ cs}
130.   ‖ ‖ { edge
130.              (uid_E(e))
130.                 (mereo_E(e))
130.                    (attr_EdgeKind(e),...)               [Static Attrs.]
130.    [Programmable Attrs.]  (attr_(e),attr_EH(e))
130.                | e:E•e ∈ es }
131.   ‖ ‖ { node
131.              (]uidN(n))
131.                 (mereo_N(n))
131.                    (attr_NodeKinds(n))                  [Static Attrs.]
131.    [Programmable Attrs.]  (attr_NH(n))
131.              | 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 conceyors.

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.

Here we illustrate the initialization a few of these !

To be written

2.7 A Review

To be written

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

3 CONCLUSION

3.1 Multi-Mode Transport

The domain description of Sect. 2 was for single-mode transport: It focused on conveyours. For a model of *multi-mode transport* we suggest to introduce *transport 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, ay world-wide, by whatever means of combinations of conveyors and routes. We intend to extend the above study to include multi-mode transport.

MORE TO COME

3.2 Interpretations

The domain description of Sect. 2 can be viwed 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*;
(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 ofor teaching such specific infra-structure [sub-]domains; and
(iv) as an initial feasability study for possible subsequent development of software for multi-mode transport systems.

We shall now comment on each of these.

3.2.1 *Socio-Economic Study.*

TO BE WRITTEN

3.2.2 *Business Process Re-Engineering.*

TO BE WRITTEN

3.2.3 *Secondary School Topic.*

TO BE WRITTEN

3.2.4 *Software System Development.*

TO BE WRITTEN

3.3 On the Development of This Model

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

MORE TO COME

3.4 Acknowledgements

4 BIBLIOGRAPHY

REFERENCES

[1] 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[25], 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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 [7].

[6] 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/double-entry/dblentrybook.pdf`. One in a series of planned studies: [8, 10–12].

[7] Dines Bjørner. Domain Modelling – A Primer. A significantly revised version of [5]. xii+202 pages[26], Summer 2024.

[8] 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: [6, 10–12].

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

[10] 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: [6, 8, 11, 12].

[11] 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: [6, 8, 10, 12].

[12] 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: [6, 8, 10, 11].

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

[26] 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

[13] Chris W. George, Peter Haff, Klaus Havelund, Anne Elisabeth Haxthausen, Robert Milne, Claus Bendix Nielsen, Søren Prehn, and Kim Ritter Wagner. *The RAISE Specification Language.* The BCS Practitioner Series. Prentice-Hall, Hemel Hampstead, England, 1992.
[14] Charles Anthony Richard Hoare. Communicating Sequential Processes. Published electronically: usingcsp.com/-cspbook.pdf, 2004.
[15] 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.

A DOMAIN DESCRIPTION UNITS

Domain descriptions consists of a set of *description units.* . Usually these are presented in some sequential order, such as, for example, yielded by a breath-first traversal of the description ontology as illustrated by Fig. 1 on page 5, but usually also ordered such that formal domain concepts are defined before their use – except for this section whose RSL constructs are first described in Appendix B. These are the domain description units:

(132) A **type** description unit introduces a type, T, abstract, or "concrete", the latter by providing a type expression, $\mathcal{TE}$. Types are here considered simple sets of values.
(133) A **value** description unit introduces a value, v or type T, abstract, or "concrete", the latter by providing a value expression, $\mathcal{VE}$(f,a).
(134) A **function** description unit introduces a function, f of function type A→B, total, →, or partial $\xrightarrow{\sim}$, with function arguments a – where f may occur [recursively] in the definition of f.
(135) A **axiom** description unit introduces a [usually labeled] predicate, $\mathcal{P}$, over type, value, function, and variable identifiers – limiting the range of their values.
(136) A **variable** description unit introduces an assignable, global entity, x of type T, without or with an initial value as determined by some
(137) A **channel** description unit introduces a communication medium – in this paper referred to as **comm** – in the form of a "two-dimensional" CSP [14] array which communicate values of type T. The array indices range over behaviours, i.e., over their unique identifiers.

132. **type** T, **type** T = $\mathcal{TE}$
133. **value** v:T, **value** v:T = $\mathcal{VE}$
134. **value** f:A→B, f(a) ≡ $\mathcal{VE}$(f,a), f:A$\xrightarrow{\sim}$B
135. **axiom** [Label:] $\mathcal{P}$
136. **variable** x:T, **variable** x:T := $\mathcal{VE}$
137. **comm**[{i,j}|i:I,j:J• $\mathcal{P}$(i,j)]: T

B ULTRA BRIEF RSL PRIMER

To be written

C TRANSPORT GLOSSARY

(138) **Aircraft:**
(139) **Bill of Lading:**
(140) **Boat:**
(141) **Car:**
(142) **Container:**
(143) **Conveyor:**
(144) **Edge:**
(145) **Freight:**
(a) **Freight Cost:**
(146) **Graph:**
(147) **Node:**
(148) **Passenger:**

(a) **Passenger Cost:**
(149) **Path:**
(150) **Ship:**
(151) **Train:**
(152) **Ticket:**
(153) **Time Table:**

More to come

D INDEXES

D.1 Transport Domain Concepts

D.2 Domain Modelling Ontology

D.3 Formal Entities

Auxiliary Functions

There are 181 formal RSL entities.

Contents