14. On A Theory of Transport Nets

- This section is under development.
  - The idea of this section is
    - not so much to present a **transport domain description**,
    - but rather to present fragments, “bits and pieces”, of a theory of such a domain.
- The purpose of having a theory
  - is to “draw” upon the ‘bits and pieces’
  - when expressing
    - properties of endurants and
    - definitions of
      - actions,
      - events and
      - behaviours.
- Again: this section is very much in embryo.

14.1. Some Pictures

- Nets can either be
  - rail nets,
  - road nets,
  - shipping lanes, or
  - air traffic nets.
- The following pictures illustrate some of these nets.

A rail net; a traffic light

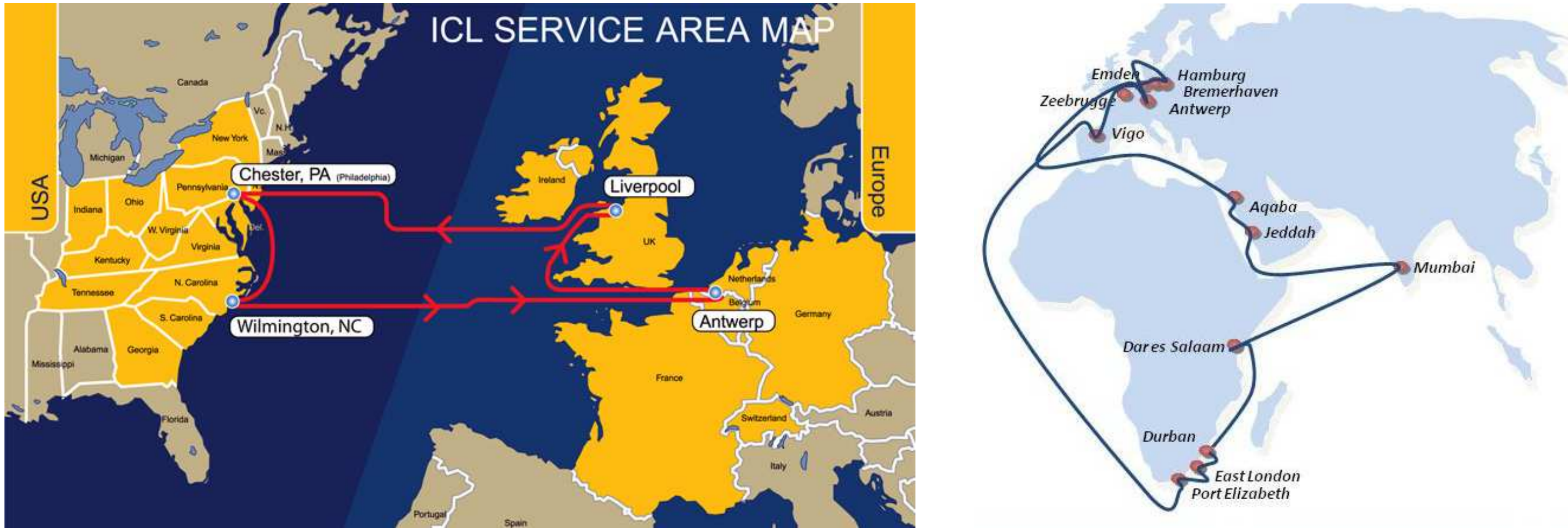
A freeway hub

Another freeway hub

- The left side of the road roundabout below is rather special.
  - Its traffic lights are also located in the inner circle of the roundabout.
  - One drives in,
    - at green light,
    - and may be guided by striping,
    - depending on where one is driving,
    - either directly to an outgoing link,
    - or is queued up against a red light
    - awaiting permission to continue.

A roundabout

- The map below left is for a container line serving one route between Liverpool (UK), Chester (PA, USA), Wilmington (NC, USA) and Antwerp (Belgium), an so forth, circularly.
- The map below right is an “around Africa” Mitsui O.S.K. Line.



Two shipping line nets

14.2. Parts

14.2.1. Nets, Hubs and Links

149. From a transport net one can observe sets of hubs and links.

type
149. N, H, L
value
149. obs_Hs: N $\rightarrow$ H-**set**, obs_Ls: N $\rightarrow$ L-**set**

14.2.2. Mereology

150. From hubs and links one can observe their unique hub, respectively link identifiers and their respective mereologies.

151. The mereology of a link identifies exactly two distinct hubs.

152. The mereologies of hubs and links must identify actual links and hubs of the net.

```
type
150.   HI, LI
value
150.   uid_H: H → HI, uid_L: L → LI
150.   mereo_H: H → LI-set, mereo_L: L → HI-set
axiom
151.   ∀ l:L•card mereo_L(l)=2
152.   ∀ n:N,l:L•l ∈ obs_Ls(n) ⇒
152.      ∧ ∀ hi:HI•hi ∈ mereo_L(l)
152.            ⇒ ∃ h:h•h ∈ obs_Hs(n)∧uid_H(h)=hi
152.      ∧ ∀ h:H•h ∈ obs_Hs(n) ⇒
152.            ∀ li:LI•li ∈ mereo_H(h)
152.                ⇒ ∃ l:L•l ∈ obs_Ls(n)∧uid_L(l)=li
```

14.2.3. An Auxiliary Function

153. For every net we can define functions which

(a) extracts all its link identifiers,

(b) and all its hub identifiers.

value

153(a). xtr_HIs: N → HI-**set**

153(a). xtr_HIs(n) ≡ {uid_H(h)|h:H•h ∈ obs_Hs(n)}

153(b). xtr_LIs: N → LI-**set**

153(b). xtr_LIs(n) ≡ {uid_L(l)|l:L•l ∈ obs_Ls(n)}

14.2.4. Retrieving Hubs and Links

154. We can also define functions which
   (a) given a net and a hub identifier obtains the designated hub, respectively
   (b) given a net and a link identifier obtains the designated link.

value

154(a). get_H: N → HI $\stackrel{\sim}{\to}$ H
154(a). get_H(n)(hi) **as** h
154(a). **pre** hi ∈ xtr_HIs(n)
154(a). **post** h ∈ obs_Hs(n)∧hi=uid_H(h)
154(b). get_L: N → LI $\stackrel{\sim}{\to}$ L
154(b). **pre** li ∈ xtr_LIs(n)
154(b). **post** l ∈ obs_Ls(n)∧li=uid_L(l)

14.2.5. Invariants over Link and Hub States and State Spaces

155. Links include two attributes:
    (a) Link states. These are sets of pairs of the identifiers of the hubs to which the links are connected.
    (b) Link state spaces. These are the sets of link states that a link may attain.
156. The link states must mention only those hub identifiers of the two hubs to which the link is connected.
157. The link state spaces must likewise mention only such link states as are defined in Items 155(a) and 156.

type
155(a). $L\Sigma = (HI \times HI)$**-set axiom** $\forall\ l\sigma{:}L\Sigma \cdot$**card** $l\sigma \leq 2$
155(b). $L\Omega = L\Sigma$**-set**
value
155(a). $attr\_L\Sigma{:}\ L \rightarrow L\Sigma$
155(b). $attr\_L\Omega{:}\ L \rightarrow L\Omega$
axiom
156. $\forall\ l{:}L,\ l\sigma'{:}L\Sigma \cdot l\sigma' \in attr\_L\Omega(l)$
156. $\Rightarrow l\sigma' \subseteq \{(hi,hi')|hi,hi'{:}HI \cdot \{hi,hi'\} \subseteq mereo\_L(l)\}$
156. $\wedge\ attr\_L\Sigma(l) \in attr\_L\Omega(l)$

158. Hubs include two attributes:
    (a) Hub states. These are sets of pairs of identifiers of the links to which the hubs are connected.
    (b) Hub state spaces. These are the sets of hub states that a hub may attain.
159. The hub states must mention only those link identifiers of the links to which the hub is connected.
160. The hub state spaces must likewise mention only such hub states as are defined in Items 158(a) and 159.

type

158(a). $\mathrm{H}\Sigma = (\mathrm{LI}\times\mathrm{LI})$**-set**

158(b). $\mathrm{H}\Omega = \mathrm{H}\Sigma$**-set**

value

158(a). $\mathrm{attr\_H}\Sigma\mathrm{: H} \rightarrow \mathrm{H}\Sigma$

158(b). $\mathrm{attr\_H}\Omega\mathrm{: H} \rightarrow \mathrm{H}\Omega$

axiom

159. $\forall\ \mathrm{h{:}H,\ h}\sigma'\mathrm{:H}\Sigma \cdot \mathrm{h}\sigma' \in \mathrm{attr\_H}\Omega\mathrm{(h)}$

159. $\quad\Rightarrow \mathrm{h}\sigma' \subseteq \{\mathrm{(li,li')|li,li'{:}LI}\cdot\{\mathrm{li,li'}\}\subseteq\mathrm{mereo\_H(h)}\}$

159. $\quad\quad\wedge\ \mathrm{attr\_H}\Sigma\mathrm{(h)} \in \mathrm{attr\_H}\Omega\mathrm{(h)}$

14.2.6. Maps

- A map is an abstraction of a net.
  - The map just shows the hub and link identifiers of the net, and hence its mereology.

type

$\text{Map}' = \text{HI} \xrightarrow[m]{} (\text{LI} \xrightarrow[m]{} \text{HI})$

$\text{Map} = \{|\text{m:Map}' \cdot \text{wf\_Map(m)}|\}$

value

$\text{wf\_Map: Map}' \to \textbf{Bool}$

$\text{wf\_Map(m)} \equiv \textbf{dom}\ \text{m} = \cup \{ \textbf{rng}\ \text{lhm} \mid \text{lhm:(LI} \xrightarrow[m]{} \text{HI)} \cdot \text{lhm} \in \textbf{rng}\ \text{m} \}$

- Let m be a map.
- The *definition set* of the map is domṁ.
- Let hi be in the definition set of map m.
- Then m(hi) is the *image* of hi in m.
- Let li be in the image of m(hi), that is, liISINdom (m(hi)), then hi'=(m(hi))(li) is the *target* of li in m(hi).

- Given a net which satisfies the axiom concerning mereology
- one can extract from that net a corresponding map.

value

```
xtr_Map: N → Map
xtr_Map(n) ≡
  [ hi ↦ [ li ↦ uid_H(retr_H(n)(hi)(li))
          | li:LI · li ∈ mere_H(get_H(n)(hi)) ]
      | h:H,hi:HI · h ∈ obs_Hs(n) ∧ hi = uid_H(h) ]
```

- The retrieve hub function
  - retrieve the "second" hub, i.e., "at the other end", of
  - a link wrt. a "first" hub.

$$
\begin{array}{l}
\text{retr\_H: N} \rightarrow \text{HI} \rightarrow \text{LI} \rightarrow \text{H} \\
\text{retr\_H(n)(hi)(li)} \equiv \\
\quad \textbf{let } \text{h} = \text{get\_H(n)(hi)} \textbf{ in} \\
\quad \textbf{let } \text{l} = \text{get\_L(n)(li)} \textbf{ in} \\
\quad \textbf{let } \{\text{hi}'''\} = \text{mereo\_L(l)} \backslash \{\text{hi}\} \textbf{ in} \\
\quad \text{get\_H(n)(hi}''') \textbf{ end end end} \\
\quad \textbf{pre}\text{: hi} \in \text{mereo\_L(get\_L(n)(li))}
\end{array}
$$

$$
\begin{array}{l}
\text{xtr\_LIs: Map} \rightarrow \text{LI}\textbf{-set} \\
\text{xtr\_LIs(m)} = \cup \, \{\textbf{dom}(\text{m(hi))} | \text{hi:HI} \cdot \text{hi} \in \textbf{dom} \text{ m}\}
\end{array}
$$

14.2.7. Routes

161. A route is an alternating sequence of hub and link identifiers.

161. $\mathrm{R}' = (\mathrm{HI}|\mathrm{LI})^\omega$, $\mathrm{R} = \{|\mathrm{r}:\mathrm{R}' \cdot \mathrm{wf\_R}(\mathrm{r})|\}$
value
161. wf_R: R′ → **Bool**
161. wf_R(r) ≡
161. ∀ i:**Nat** · {i,i+1}⊆**inds** r ⇒
161. is_HI(r(i))∧is_LI(r(i+1)) ∨ is_LI(r(i))∧is_HI(r(i+1))

162. A route of a map, m, is a route as follows:

(a) An empty sequence is a route.

(b) A sequence of just a single hub identifier or of hubs of the map is a route.

(c) A sequence of just a single link identifier of links of the map is a route.

(d) If $\mathsf{r}^\frown\langle\mathsf{hi}\rangle$ and $\langle\mathsf{li}\rangle^\frown\mathsf{r}'$ are routes of the map and li is in the definition set of $\mathsf{m(hi)}$ then $\mathsf{r}^\frown\langle\mathsf{hi,li}\rangle^\frown\mathsf{r}'$ is a route of the map.

(e) If $\mathsf{r}^\frown\langle\mathsf{li}\rangle$ and $\langle\mathsf{hi}\rangle^\frown\mathsf{r}'$ are routes of the map and hi is the target of $\mathsf{(m(hi'))(li)}$ then $\mathsf{r}^\frown\langle\mathsf{li,hi}\rangle^\frown\mathsf{r}'$ is a route of the map.

(f) Only such routes are routes of a net if they result from a finite [possibly infinite] set of uses of Items 162(a)-162(e).

type

type

162. $\text{MR}' = \text{R}$, $\text{MR} = \{\text{r:MR}' \cdot \exists\ \text{m:Map} \cdot \text{r} \in \text{routes(m)}|\}$

value

162. routes: N → MR-**infset**

162. routes(n) ≡ routes(xtr_Map(n))

162. routes: Map → MR-**infset**

162. routes(m) ≡

162(a). **let** rs = $\{\langle\rangle\}$

162(b). $\cup \cup \{\langle\text{hi}\rangle|\text{hi:HI} \cdot \text{hi} \in$ **dom** m$\}$

162(c). $\cup \cup \{\langle\text{li}\rangle|\text{li:LI,hi:HI} \cdot \text{li} \in \text{xtr\_LIs(m)}\}$

162(d). $\cup \cup \{\text{r}^\frown\langle\text{hi,li}\rangle^\frown\text{r}'|\text{r,r}':\text{MR,hi:HI,li:LI}\cdot\{\text{r,r}'\}\subseteq\text{rs}\wedge\text{li} \in$ **dom** m(hi)$\}$

162(e). $\cup \cup \{\text{r}^\frown\langle\text{li,hi}\rangle^\frown$**tl** $\text{r}'|\text{r,r}':\text{MR,li:LI,hi:HI}\cdot\{\text{r,r}'\}\subseteq\text{rs}\wedge\text{is\_target(m)(hi)(li)}\}$

162(f). **in** rs **end**

162(e). is_target: Map → HI × LI

162(e). is_target(m)(hi)(li) ≡

162(e). $\exists\ \text{h}'':\text{HI}\cdot\text{h}''\in$ **dom** $\text{m}\wedge\text{li} \in$ **dom** $\text{m(hi}'')\wedge\text{hi}=(\text{m(hi}''))(\text{li})$

14.2.8. Special Routes

14.2.8.1 Acyclic Routes

163. A route of a map is acyclic if no hub identifier appears twice or more.

value

163. is_Acyclic: MR → Map $\stackrel{\sim}{\to}$ **Bool**
163. is_Acyclic(mr)(m) ≡ ∼∃ hi:HI,i,j:**Nat**·{i,j}⊆**inds** mr ∧ i≠j ⇒ mr(i)=hi=mr(j)
163. **pre** mr ∈ routes(m)

14.2.8.2 Direct Routes

164. A route, r, of a map (from hub hi or linkli to hub hi′ or linkli′) is a direct route if r is acyclic.

164. direct_route: MR → Map $\stackrel{\sim}{\to}$ **Bool**
164. direct_route(mr) ≡ is_Acyclic(mr)
164. **pre** mr ∈ routes(m)

14.2.8.3 Routes Between Hubs

165. Let there be given two distinct hub identifiers of a route map. Find the set of acyclic routes between them, including zero if no routes.

value

165. find_MR: Map $\rightarrow$ (HI$\times$HI) $\stackrel{\sim}{\rightarrow}$ MR**-set**
165. find_MR(m)(hi,hi′) $\equiv$
165. **let** rs = routes(m) **in**
165. {mr | mr,mr′:MR $\cdot$ mr $\in$ rs
165. $\wedge$ mr $\in$ mr = $\langle$hi$\rangle$$\widehat{\ }$mr′$\langle$hi′$\rangle$ $\wedge$ is_Acyclic(mr)(m) }
165. **end**
165. **pre**: {hi,hi′}$\subseteq$**dom** m

14.2.9. Special Maps
14.2.9.1 Isolated Hubs

166. A net, n, consists of two or more isolated hubs
   (a) if there exists two hub identifiers, hi_1,hi_2, of the map of the net
   (b) such that there is no route from hi_1 to hi_2.

value
166. are_isolated_hubs: Map → **Bool**
166. are_isolated_hubs(m) ≡
166(a). $\exists$ hi_1,hi_2:HI · {hi_1,hi_2}⊆**dom** m ⇒
166(b). $\sim\exists$ mr,mr_i:MR · mr ∈ routes(m) ⇒ mr = $\langle hi_1\rangle \frown mr_i \frown \langle hi_2\rangle$

14.2.9.2 Isolated Maps

167. If there are isolated hubs in a net then the net can be seen as two or more isolated nets.

value
167. are_isolated_nets: Map → **Bool**
167. are_isolated_nets(m) ≡ are_isolated_hubs(m)

14.2.9.3 Sub_Maps

168. Given a map one can identify the set of all sub_maps which which contains a given hub identifier.

169. Given a map one can identify the sub_map which contains a given hub identifier.

value
168. sub_maps: Map → Map-**set**
168. sub_maps(m) **as** ms
168. { xtr_Map(m)(hi) | hi:HI • hi ∈ **dom** m }

169. sub_Map: Map → HI $\xrightarrow{\sim}$ Map
169. sub_Map(m)(hi) ≡
169. **let** his = { hi′ | hi′:HI ∧ hi′ ∈ **dom** m ∧ find_MRs(m)(hi,hi′)≠{} } **in**
169. [hi″ ↦ m(hi″) | hi″ ∈ his] **end**

theorem: are_isolated_nets(m) ⇒ sub_maps(m) ≠ { m }

14.3. Actions

14.3.1. Insert Hub

170. The insert action

(a) applies to a net and a hub and conditionally yields an updated net.

(b) The condition is that there must not be a hub in the initial net with the same unique hub identifier as that of the hub to be inserted and

(c) the hub to be inserted does not initially designate links with which it is to be connected.

(d) The updated net contains all the hubs of the initial net "plus" the new hub.

(e) and the same links.

value

170. insert_H: N → H $\stackrel{\sim}{\to}$ N
170(a). insert_H(n)(h) **as** n′
170(a). **pre**: pre_insert_H(n)(h)
170(a). **post**: post_insert_H(n)(h)(n′)

170(b). pre_insert_H(n)(h) ≡
170(b). ∼∃ h′:H • h′ ∈ obs_Hs(n) ∧ uid_H(h)=uid_H(h′)
170(c). ∧ mereo_H(h) = {}

170(d). post_insert_H(n)(h)(n′) ≡
170(d). obs_Hs(n) ∪ {h} = obs_Hs(n′)
170(e). ∧ obs_Ls(n) = obs_Ls(n′)

14.3.2. Insert Link

171. The **insert link** action

   (a) is given a “fresh” link,
   that is, one not in the **net** (before the action)

   (b) but where the two distinct **hub identifiers** of the **mereology** of the **inserted link** are of **hub**s in the **net**.

   (c) The **link** is inserted.

   (d) These two **hubs**

   (e) have their mereologies updated
   to reflect the new **link**

   (f) and nothing else;
   all other **links** and **hubs** of the **net** are unchanged.

value

171. insert_L: $N \to L \stackrel{\sim}{\to} N$

171. insert_L(n)(l) **as** n'

171. $\exists$ l:L $\cdot$ pre_insert_L(n)(l) $\Rightarrow$ pre_insert_L(n)(l) $\wedge$ post_insert_L(n,n')(l)

171. pre_insert_L: N → L → **Bool**
171. pre_insert_L(n)(l) ≡
171(a). uid_L(l) ∉ xtr_LIs(n)
171(b). ∧ mereo_L(l)⊆xtr_HIs(n)

171. post_insert_L: N × N → L → **Bool**
171. post_insert_L(n,n′)(l) ≡
171(c). obs_Ls(n) ∪ {l} = obs_Ls(n′)
171(d). ∧ **let** {hi1,hi2} = mereo_L(l) **in**
171(d). **let** (h1,h2) = (get_H(n)(hi1),get_H(n)(hi2)),
171(d). (h1′,h2′) = (get_H(n′)(hi1),get_H(n′)(hi2)) **in**
171(e). mereo_H(h)∪{uid_L(l)}=mereo_H(h′)
171(f). ∧ obs_Hs(n)\{h1,h2} = obs_Hs(n′)\{h1′,h2′}
171(f). ∧ [all other properties of h1 and h2 unchanged]
171(f). [that is, same as h1′ and h2′]
171. **end end**

- The **insert link** post-condition has too many lines.
- I will instead compose the post-condition
  - from the conjunction of a number of invocations
  - of predicates with “telling” names.
- For these action function definitions
  - such “small” predicates
  - amount to building a nicer theory.

14.3.3. Remove Hub

172. remove hub

(a) where a hub, known by its hub identifier, is given,

(b) where the [to be] **removed hub** is indeed in the net (before the action),

(c) where the **removed hub**'s **mereology** is empty (that is, the [to be] removed hub) is not connected to any links in the **net** (before the action)).

(d) All other links and hubs of the net are unchanged.

value

172. remove_H: $N \to HI \xrightarrow{\sim} N$

172(a). remove_H(n)(hi) **as** n′

172(b). $\exists$ h:H • uid_H(h)=hi $\wedge$ h $\in$ obs_Hs(n) $\Rightarrow$

172(c). pre_remove_H(n)(hi) $\wedge$ post_remove_H(n,n′)(hi)

- We leave the definitions of the pre/post conditions of this and the next action function to the listener.

14.3.4. Remove Link

173. remove link

(a) where a link, known by its link identifier, is given,

(b) where that link is indeed in the net (before the action),

(c) where hubs to which the link is connected after the action has the only change to their mereologies changed be that they do not list the [to be] removed link.

(d) All other links and hubs of the net are unchanged.

value

173. remove_L: N → LI $\stackrel{\sim}{\to}$ N
173(a). remove_L(n)(li) **as** n′
173(b). ∃ l:L • uid_L(l)=li ∧ l ∈ obs_Ls(n) ⇒
173(c). pre_remove_L(n)(li) ∧ post_remove_L(n,n′)(li)