Domains and Problem Frames
An Experiment in Triptych Software Development[*]

Dines Bjørner
Section on Computer Science and Engineering,
Department of Informatics and Mathematical Modelling,
Technical University of Denmark, DK–28000 Kgs.Lyngby, Denmark.

School of Information Science, JAIST,
1-1, Asahidai, Tatsunokuchi, Nomi, Ishikawa 923-1292, Japan.

db@imm.dtu.dk, bjorner@gmail.com

December 15, 2006

Abstract

In this report we investigate Michael Jackson's frame concept [4] in the context of the transition from a domain model of some broad application domain to a set of requirements models, one for each of a sufficiently distinct set of domain requirements — but for what is claimed to be "the same" broad application domain.

We shall thus follow the triptych dogma of [3].[1]

First we develop a domain model (for the application area of transportation nets). Then we sketch the development of a number of diverse domain requirements for the computerisation of transportation network management, monitoring and control. Finally we relate the diverse domain requirements to similarly different problem frames.

The claim of this report is that to better understand the underlying issues of Michael Jackson's problem frame one must see the concept of problem frames as a function of the relation between a domain model and a (domain) requirements model.

Contents

[*]Technical report for the JAIST Technical Memoranda series.

[1][3, 4] (together with references to the companion volumes [1, 2] of [3]) will be the only citations of this report.

1 Domains and Problem Frames

1.1 The Dogma

Before software can be designed we must understand its requirements. Before requirements can be prescribed we must understand the domain[2]. In this paper we exemplify **one** domain description and **four** related requirements prescriptions. The latter illustrate distinct frames.

1.2 Aims & Objectives

1.2.1 Aims

We aim to illustrate aspects of problem frame independent domain engineering, problem frame dependent requirements engineering, and the interplay between various requirements prescriptions.

1.2.2 Objectives

Our objective is to plead for more systematic software engineering work around domain engineering, before requirements engineering sets in.

1.3 Structure of Paper

We first bring a long and undoubtedly boring domain description, then four requirements presecriptions. In the conclusion we relate this quadruple development to the problem frame approach, and briefly discuss a rôle for the triptych cum problem frame approach in the Verified Software: Theories, Techniques and Experiments (VSTTE) and the Ubiquitous Computing grand challenges!

We need the "multiple masses of details" in ordee to properly substantiate our aims and objectives.

2 The Domain

Our domain is that of transportation nets. We abstract in such as way as to capture both road, rail, air and shipping transport nets. The basic concepts of street segments between street intersections, rail lines between train stations, air lanes between airports and shipping lanes between harbours are abstracted into segments and the street intersections, train stations, airports and harbours are abstracted into junctions.

[2]The term domain is here used instead of the — in problem frame contexts — perhaps more common term environment.

2.1 Net Topology

We "slowly" (read: carefully) unarrate and formalise a number of concepts related to segments and junctions.

2.1.1 Nets, Segments and Junctions

Nets consists of one or more segments and two or more junctions.

type
  N, S, J
value
  obs_Ss: N $\rightarrow$ S-**set**
  obs_Js: N $\rightarrow$ J-**set**
axiom
  $\forall$ n:N • **card** obs_Ss(n) $\geq 1 \wedge$ **card** obs_Js(n) ≥ 2

Annotations:

- N, S, J are considered abstract types, i.e., sorts. N, S and J are type names, i.e., names of types of values. Values of type N are nets, values of type S are segments and values of type J are junctions.
- One can observe from nets, n, their (one or more) segments (obs_Ss(n)) and their (two or more) junctions (obs_Js(n)); n is a value of type N.
- Functions have names, obs_Ss, and obs_Js, and functions, f, have signatures, f: A $\rightarrow$ B (not illustrated), where A and B are type names. A designates the definition set of f and B the range set.
- A-**set** is a type expression. It denotes the type whose values are finite, possibly empty set of A values.
- These observer functions are postulated.
- They cannot be formally defined.
- They are "defined" once a net has been pointed out[3]
- The axiom expresses that in any net there is at lest one segment and at least two junctions.

Applying the observer functions to the net of Fig. 1 yields:

  obs_Ss(n) = {sa,sb,sc,sd,se,sf,sg,sh,sj,sk}
  obs_Js(n) = {j1,j2,j3,j4,j5,j6,j7,j8}

Nets, segments and junctions are physically manifest, i.e., are phenomena.

[3]Take the transportation net Europe. By inspecting it, and by deciding which segments and which associated junctions to focus on (i.e., "the interesting ones") we know which are all the interesting roads, rail tracks, air lanes and shipping lanes, respectively the interesting (associated) street intersections, trains stations, airports and harbours.

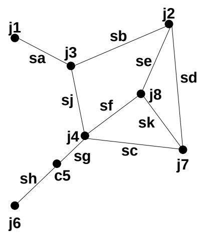


Figure 1: A simple net of segments and junctions

2.1.2 Segment and Junction Identifications

Segments and junctions have unique identifications.

type
  Si, Ji
value
  obs_Si: S → Si
  obs_Ji: J → Ji

Segment and junction identifications are abstract concepts. No two segments have the same segment identifier. And no two junctions have the same junction identifier.

axiom
  ∀ n:N • **card** obs_Ss(n) ≡ **card** {obs_Si(s)|s:S • s ∈ obs_Ss(n)}
  ∀ n:N • **card** obs_Js(n) ≡ **card** {obs_Ji(c)|j:J • j ∈ obs_Js(n)}

Annotations:

- **card** set expresses the cardinality of the set set, i.e., its number of distinct elements.
- {f(a)|a:A • p(a)} expresses the set of all those B elements f(a) where a is of type A and has property p(a) [where we do not further state f, A and B. p is a predicate, i.e., a function, here from A into truth values of type **Bool**, for Boolean].
- The axioms now express that the number of segments in n is the same as the number of segment identifiers of n — which is a circumscription for: No two segments have the same segment identifier.
- Similar for junctions.

The constraints that limit identification of segments and junctions can be physically motivated: Think of the geographic (x, y, z co-ordinate) point spaces "occupied" by a segment or by a junction. They must necessarily be distinct for otherwise physically distinct segments and junctions. Segments may thus cross each other without the crossing point (in x, y space) being a junction, but, for example, one segment may, at the crossing point be physically above the other segment (tunnels, bridges, etc.).

2.1.3 Segment and Junction Reference Identifications

Segments are delimited by two distinct junctions. From a segment one can also observe, obs_Jis, the identifications of the delimiting junctions.

```
type
    Jip = {|{ji,ji′}:Ji-set • ji≠ji′|}
value
    obs_Jis: S → Jip
```

Annotations:

- {|a:A • p(a)|} is a subtype expression. It expresses a subset of type A, namely those A values which enjoys property p(a) [p is a predicate, i.e., a function, here from A into truth values in the type **Bool**]. In the above p(a) is ji≠ji′.
- In this case Jip is the subtype of Ji-**set** whose values are exactly 2 element sets of Ji elements.

Any junction has a finite, but non-zero number of segments connected to it. From a junction one can also observe, obs_Sis, the identifications of the connected segments.

```
type
    Si1 = {|sis:Si-set•card sis ≥1|}
value
    obs_Sis: J → Si1
```

Annotations:

- Si1 is the type whose values are non-empty, but still finite sets of Si values.

One cannot from a segment alone observe the connected junctions. One can only refer to them. Similarly: one cannot from a junction along observe the connected segments. One can only refer to them. The identifications serve the role of being referents. In any net, if s is a segment connected to connectors identified by ji and ji′, respectively, then there must exist connectors j and j′ which have these identifications and such that the identification si of s is observable from both j and j′.

```
axiom
    ∀ n:N, s:S • s ∈ obs_Ss(n) ⇒
        let {ji,ji′} = obs_Jis(s) in
        ∃! j,j′:J • {j,j′}⊆obs_Js(n) ∧ j≠j′ ∧
            obs_Si(s) ∈ obs_Sis(c) ∩ obs_Sis(c′) end
```

Annotations:

- We read the above axiom:
    - for all nets n and for all segments s in n
    - let ji and ji′be the two distinct junction identifications observable from s, then
    - exists exactly two distinct junctions, j and j′ of the net, such that

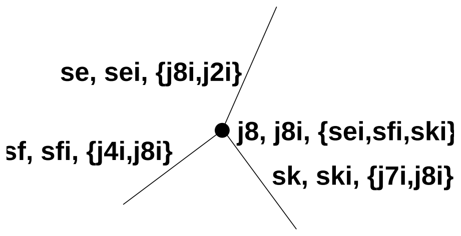


Figure 2: One junction and its connected segments

- the segment identification of s is in both the sets of segment identifications observable from j and j′.

Figure 2 illustrates the relation between observed identifications of segments and junctions.

The above constraints take on the mantle of being laws of nets: If segments and junctions otherwise have distinct identifications, then the above must follow as a law of man-made artifacts. Vice-versa: In any net, if j is a junction connecting segments identified by si, si′, …, si″ then there must exist segments s, s′, …, s″ which have these identifications and such that the identification ji of j is observable from all s, s′, …, s″.

```
axiom
    ∀ n:N, j:J • j ∈ obs_Js(n) ⇒
        let sis = obs_Sis(c), ji = obs_Ji(j) in
        ∃! ss:S-set • ss⊆obs_Ss(n) ∧ card ss=card sis ∧
        sis = {|obs_Si(s)|s:S•s ∈ ss|} end
```

Annotations:

- Let us read the above axiom:
  - for all nets, n, and all junctions, j, of that net
  - let sis be the set of segment identifications observed from j, and let ji be the junction identifier of j, then
  - there exists a unique set, ss, of segments of n with as many segments as there are segment identifications in sis, and such that
  - sis is exactly the set of segment identifications of segments in ss.

2.1.4 Paths and Routes

By a path we shall understand a triplet of a junction identification, a segment identification and a junction identification.

```
type
    P = Ji × Si × Ji
value
    paths: N → P-set
    paths(n) ≡
        {(ji,si,ji′)|s:S,ji,ji′:Ji,si:Si•
                        s ∈ obs_Ss(n)∧{ji,ji′} ∈ obs_Jis(s)∧si=obs_Si(s)}
```

Annotations:

- Paths are modelled as Cartesians.
- One can generate all the paths of a net.
- It is the set of path triplets, two for each segment of the net and such that the pair of junction identifications, ji and ji′, observable from a segment is at either "end" of the triplet, and such that the segment identification is common to the two triplets (and in the "middle").

Paths, and as we shall see next, routes are mental concepts. By a route of a net we shall understand a list, i.e., a sequence of paths as follows:

- A sequence of just one path of the net is a route.
- If r and r′ are routes of the net such that the last junction identification, ji, of the last path, (_,_,ji) of r and the first junction identification, ji′, of the first path (ji′,_,_) of r′ are the same, i.e., ji=ji′, then $r\frown r'$ is a route.
- Only routes that can be generated by uses of the first (the basis) and the second (the induction) clause above qualify as proper routes of a net.

```
type
  R = {|r:P*•wf_R(r)|}
value
  wf_R: P* → Bool
  wf_R(r) ≡
     ∀ i:Nat • {i,i+1}⊆inds(r) ⇒
        let (_,_,ji)=r(i), (ji′,_,_)=r(i+1) in ji = ji′ end

  routes: N → R-infset
  routes(n) ≡
     let rs = {⟨p⟩|p:P•p ∈ paths(n)}
              ∪ {r⁀r′|r,r′:R•{r,r′}⊆rs∧wf_R(r⁀r′)} in
     rs end
```

Annotations:

- Routes are well-formed sequences of paths.
- A sequence of paths is a well-formed route if adjacent path elements of the route share junction identification.
- Give a net we can compute all its routes as follows:
  - let rs be the set of routes to be computed. It consists first of all the single path routes of the net.
  - Then rs also contains the concatenation of all pairs of routes, r and r′, such that these are members of rs and such that their concatenation is a well-formed route.
  - If the net is circular then the set rs is an infinite set of routes. The least fix point of the recursive equation in rs is the solution to the "routes" computation.

2.1.5 Segment and Junction Identifications of Routes

For future purposes we need be able to identify various segment and junction identifications as well as various segments and junctions of a route.

value

```
    xtr_Jis: R → Ci-set, xtr_Sis: R → Si-set
    xtr_Jis(r) ≡ case r of ⟨⟩ → {}, ⟨(ji,_,ji′)⟩⁀r′ → {ji,ji′}∪ xtr_Jis(r′) end
    xtr_Sis(r) ≡ case r of ⟨⟩ → {}, ⟨(_,si,_)⟩⁀r′ → {si}∪ xtr_Sis(r′) end

    xtr_Ss: N × Ji → S-set
    xtr_Ss(n,ji) ≡ {s|s:S•s ∈ obs_Ss(n) ∧ ji ∈ obs_Jis(s)}

    xtr_C: N × Ji → C, xtr_S: N × Si → S
    xtr_C(n,ji) ≡ let j:J • j ∈ obs_Js(n) ∧ ji=obs_Ji(j) in j end
    xtr_S(n,si) ≡ let s:S • s ∈ obs_Ss(n) ∧ si=obs_Si(s) in s end

    first_Ji: R $\overset{\sim}{\to}$ Ji, last_Ji: R $\overset{\sim}{\to}$ Ji
    first_Ji(r) ≡ case r of ⟨⟩ → chaos, ⟨(ji,_,_)⟩⁀r′ → ji end
    last_Ji(r) ≡ case r of ⟨⟩ → chaos, r′⁀⟨(_,_,ji)⟩ → ji end

    first_Si: R $\overset{\sim}{\to}$ Si, last_Si: R $\overset{\sim}{\to}$ Si
    first_Si(r) ≡ case r of ⟨⟩ → chaos, ⟨(_,si,_)⟩⁀r′ → si end
    last_Si(r) ≡ case r of ⟨⟩ → chaos, r′⁀⟨(_,si,_)⟩ → si end

    first_J: R × N $\overset{\sim}{\to}$ J, last_J: R × N $\overset{\sim}{\to}$ J
    first_J(r,n) ≡ xtr_J(first_Ji(r),n)
    last_J(r,n) ≡ xtr_J(last_Ji(r),n)

    first_S: R × N $\overset{\sim}{\to}$ S, last_S: R × N $\overset{\sim}{\to}$ S
    first_S(r,n) ≡ xtr_S(first_Si(r),n)
    last_S(r,n) ≡ xtr_S(last_Si(r),n)
```

Annotations:

- Given a route one can extract the set of all its junction identifications.
  - If the route is empty, then the set is empty.
  - If the route is not empty than it consists of at least one path and the set of junction identifications is the pair of junction identifications of the path together with set of junction identifications of the remaining route.
  - Possible double "counting up" of route adjacent junction identifications "collapse", in the resulting set into one junction identification. (Similarely for cyclic routes.)
- Given a route one can similarly extract the set of all its segment identifications.
- Given a net and a junction identification one can extract all the segments connected to the identified junction.

- Given a net and a junction identification one can extract the identified junction.
- Given a net and a segment identification one can extract the identified segment.
- Given a route one can extract the first junction identification of the route.
  - This extraction should not be applied to empty routes.
  - A non-empty route can always be thought of as its first path and the remaining route. The first junction identification of the route is the first junction identification of that (first) path.
- Given a route one can similarly extract the last junction identification of the route.
- Given a route one can similarly extract the first segment identification of the route.
- Given a route one can similarly extract the last segment identification of the route.
- And similarly for extracting the first and last junctions, respectively first and last segments of a route.

2.1.6 Circular and Pendular Routes

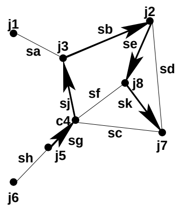
A route is circular if the same junction identification either occurs more than twice in the route, or if it occurs as both the first and the last junction identification of the route. Given a net we can compute the set of all non-circular routes by omitting from the above pairs of routes, r and r′, where the two paths share more than one junction identification.

```
non_circular_routes: N → R-set
non_circular_routes(n) ≡
    let rs = {⟨p⟩|p:P•p ∈ paths(n)}
              ∪ {r⁀r′|r,r′:R•{r,r′}⊆rs∧wf_R(r⁀r′)∧non_circular(r,r′)} in
    rs end
non_circular: R×R → Bool
non_circular(r,r′) ≡ card xtr_Jis(r) ∩ xtr_Jis(r′) =1
```

Annotations:

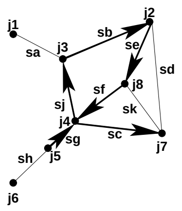
- To express the finite set of all non-circular routes
  - is to re-express the set of all routes
  - except constrained by the further predicate: non_circular.
- An otherwise well-formed route consisting of a first part r and a remaining part r′
  - is non-circular if the two parts share at most one junction identification.

Let a path be (ji_f, si, ji_t), then (ji_t, si, ji_f) is a *reverse path*. That is: the two junction identifications of a path are reversed in the reverse path. A route, rr, is the reverse route of a route r if the ith path of rr is the reverse path of the $n - i + 1$'st path of r where n is the length of the route r, i.e., its number of paths. A route is a *pendular* route if it is of an even length and the second half (which is a route) is the reverse of the first half route.



<(j5i,sgi,j4i),(j4i,sji,j3i),(j3i,sbi,j2i),(j2i,sei,j8i)>

Figure 3: A route, graphically and as an expression



<(j5i,sgi,j4i),(j4i,sji,j3i),(j3i,sbi,j2i),(j2i,sei,j8i),(j8i,sfi,j4i),(j4i,sci,j7i)>

Figure 4: A circular route, graphically and as an expression

```
value
    reverse: P → P
    reverse(jif,si,jit) ≡ (jit,si,jif)

    reverse: R → R
    reverse(r) ≡
        case r of
            ⟨⟩ → ⟨⟩,
            ⟨(jif,si,jit)⟩⌢r′ → reverse(r′)⌢⟨(jit,si,jif)⟩
        end

    reverse(r) ≡ ⟨reverse(r(i))|i in [n..1]⟩

    pendular: R → R
    pendular(r) ≡ r⌢reverse(r)

    is_pendular(r) ≡ ∃ r′,r″:R • r′⌢r″ = r ∧ r″=reverse(r′)
```

Annotations:

- The reverse of a path is a path with the same segment identification, but with reverse junction identifications.
- The reverse of a route, r, is
  - the empty route if r is empty, and otherwise
  - it is the reverse route of all of r except the first path of r concatenate (juxtaposed) with the singleton route of the reverse path of the first path of r.

- Given a route, r, we can construct a pendular route whose first half is the route r and whose last half is the reverse route of r.
- A (an even length) route is a pendular route if it can be expressed as the concatenation of two (equal length) routes, r′ and r″ such that r″ is the reverse of r′, that is, if its second half is the reverse of its first half.

2.1.7 Connected Nets

A net is connected if for any two junctions of the net there is a route between them.

```
value
  is_connected: N → Bool
  is_connected(n) ≡
     ∀ j,j′:J • {j,j′}⊆obs_Js(n) ∧ j≠j′ ⇒
        let (ji,ji′) = (obs_Ji(j),obs_Ji(j′)) in
        ∃ r:R • r ∈ routes(n) ∧
           first_Ji(r) = ji ∧ last_Ji(r) = ji′ end
```

Annotations:

- A net n is connected if
  - for all two distinct connectors of the net
  - where ji and ji′ are their junction identifications,
  - there exists a route, r, of the net,
  - whose first junction identification is ji and whose last junction identification is ji′.

2.1.8 Net Decomposition

One can decompose a net into all its connected subnets. If a net exhaustively consists of m disconnected nets, then for any pair of nets in different disconnected nets it is the case that they share no junctions and no segments. The set of disconnected nets is the smallest such set that together makes up all the segments and all the junctions of the ("original") net.

```
value
  decompose: N → N-set
  decompose(n) as ns
     obs_Ss(n) = ∪{obs_Ss(n′)|n′:N•n′ ∈ ns} ∧
     obs_Js(n) = ∪{obs_Js(n′)|n′:N•n′ ∈ ns} ∧
     {} = ∩{obs_Ss(n′)|n′:N•n′ ∈ ns} ∧
     {} = ∩{obs_Js(n′)|n′:N•n′ ∈ ns} ∧
     ∀ n′:N•n′ ∈ ns ⇒ connected(n′) ∧ ...
```

Annotations:

- A set ns of nets constitutes a decomposition of a net, n,
  1. if all the segments of n appear in some net of ns,

2. if all the junctions of n appear in some net of ns,
3. if no two or more distinct nets of ns share segments,
4. if no two or more distinct nets of ns share junctions, and
5. if all nets of ns are connected.

- **Comment:** It appears that items 3 and 4 are unnecessary, that is, are properties once items 1, 2 and 5 hold.

That is, we have the following:

Lemma:

```
  ∀ n:N •
      let ns = decompose (n) in
      ∀ n′,n″:N • {n′,n″}⊆ns ∧ n′≠n″ ⇒
          obs_Ss(n′) ∩ obs_Ss(n″) = {} ∧
          obs_Js(n′) ∩ obs_Js(n″) = {} end
```

The above items define a lot of what there is to know about transportation nets if we only operate with the sorts that have been introduced (N, S, Si, J, Ji) and the observer functions that have likewise been introduced (obs_Ss, obs_Js, obs_Si, obs_Ji, obs_Jis and obs_Sis). The relationships between sorts, i.e., net, segment, segment identification, junction and junction identification values are expressed by the axioms. The above is a so-called property-oriented model of the topology of transportation nets. That model is abstract in that it does not hint at a mathematical model or at a data structure representation of nets, segments and junctions, let alone their topology. By topology we shall here mean how segments and junctions are "wired up". The axioms above guarantee that no segment of a net is left "dangling": It is always connected to two distinct junctions; and no junctions of a net is left isolated: It is always connected to some segments of the net.

We have tacitly assumed that all segments are two way segments, that is, transport can take place i either direction. Hence a segment gives rise to two paths.

2.2 Multi-Modal Nets

Interesting transportation nets are multi-modal. That is, consists of segments of different transport modalities: roads, rails, air-lanes, shipping lanes, and, within these of different categories. Thus roads can be either freeways, motor-ways, ordinary highways, and so on.

2.2.1 General Issues

We introduce a concept, M, of transport mode. M is a small set of distinct, but otherwise further undefined tokens. An m in M designates a transport modality.

```
type
  M
```

2.2.2 Segment and Junction Modes

With each segment, s, we can associate a single mode, m, and with each junction we can associate the set of modes of its connected segments.

```
value
    obs_M: S → M
    obs_Ms: J → M-set
axiom
    ∀ n:N, j:J • j ∈ obs_Js(n) ⇒
        let ss = xtr_Ss(n,obs_Ji(j)) in
        obs_Ms(j) = {obs_M(s)|s:S • s ∈ ss} end
    ∀ n:N, s:S • s ∈ obs_Ss(n) ⇒
        let {ji,ji′} = obs_Jis(s) in
        let {j,j′} = {xtr_J(n,ji),xtr_J(n,ji′)} in
        obs_M(s) ∈ obs_Ms(j) ∩ obs_Ms(j′) end end
```

Annotations:

- From a segment one can observe its mode.
- From a junction one can observe its set of modes.
- Let us read the first axiom:
  - for all net, n, and all junctions, j, of that net
  - let ss be the set of segments connected to j,
  - now the set of modes of c is equal to the set of modes of the segments in ss.
- Let us read the second axiom:
  - for all net, n, and all segments, s, of that net
  - let ji and ji′ be the junction identifiers of the two junctions to which s is connected, and
  - let j and j′ be the two corresponding junctions,
  - then the segment mode is in both the set of modes of the two junctions.
- We can define a function, xtr_Ss, which from a net, n, and a junction identification, ji, extracts the set of segments, ss, connected to the junction identified by ji.
- xtr_Ss(n,ji) yields the set of segments, ss, in the net n for which ji is one of the observed junction identifications of s.
- And we can define a function, xtr_J, of signature N × Ji → J, which when applied to a net, n, and a junction identification, ji,
- extracts the junction in the net which has that junction identifier.

2.2.3 Single-Modal Nets and Net Projection

Given a multi-modal net one can project it onto a set of single modality nets, namely one for each modality registered in the multi-modal net.

```
type
   mmN = {|n:N • card xtr_Ms(n) > 1|}
   smN = {|n:N • card xtr_Ms(n) = 1|}
value
   xtr_Ms: N → M-set
   xtr_Ms(n) ≡ {obs_M(s) | s:S • s ∈ obs_Ss(n)}

   projs: N → smN-set
   projs(n) ≡ {proj(n,m) | m:M • m ∈ xtr_Ms(n)}
   proj: N × M → smN
   proj(n,m) as n′
      post
         let ss = obs_Ss(n), ss′ = obs_Ss(n′),
               js = obs_Js(n), js′ = obs_Js(n′) in
         ss′ = {s | s:S • s ∈ ss ∧ m=obs_M(s)} ∧
         js′ = {j | j:J • j ∈ js ∧ m ∈ obs_Ms(j)}
         end
```

Annotations:

- A multi-modal net is a net with more than one mode. mmN is thus the subtype of nets, n:N, which are multi-modal.
- A single-modal net is a net with exactly one mode. smN is thus the subtype of nets, n:N, which are multi-modal.
- The xtr_Ms function extracts the mode of every segment of a net.
- The projs function applies to any net, n:N, and yields the set of single-modal subnets of n, one for each mode of n. The projs function makes use of the proj function.
- The proj function applies to any n, n:N, and any mode of that net, and yields the single-modal subnet on n whose mode is the given mode.
  - The proj function is expressed by a post condition, i.e., a predicate that characterises the necessary and sufficient relation between the argument net, n, and the result net n′.
  - In a single-modal net, n′, projected from a multi-modal net, n, and of mode m, we keep exactly those segments, ss′, of n whose mode is m,
  - and we keep exactly those junctions, js′, of n whose mode contains m.
  - No more is needed in order to express the necessary and sufficient condition for a single-modal net to be a subnet of a proper net.
  - That is, some single-modal nets are not proper nets since in proper nets every junction have the set of modes of all the segments connected to the junction.

2.2.4 Sub-Junctions

Let ms:$\{m_1, m_2, ..., m_n\}$ be the set of modes of a junction j. To each such mode m_i we can associate a junction mj_{m_i}.[4] With any such junction mj_{m_i} we can associate a modal junction identification $mj_{m_{i_i}}$.

```
type
    MJ, MJi
value
    obs_MJs: J → MJ-set
    obs_MJ: J × M → MJ
    obs_MJi: MJ → MJi
    obs_M: MJ → M
axiom
    ∀ j:J,m:M•m ∈ obs_Ms(j) ⇒ let mj = obs_MJ(j,m) in obs_M(mj)=m end
```

2.3 Segment and Junction Attributes

2.3.1 Segment and Junction Attribute Observations

We now enrich our segments and junctions.

Segments have lengths. Junctions have modality-determined lengths between pairs of (same such modality) segments connected to the junction. Segments have standard transportation times, i.e., time durations that it takes to transport any number of units of freight from one end of the segment to the other. Junctions have standard transfer time per modality of transport between pairs of segments connected to the junction. Junctions have standard arrival time per modality of transport. Junctions have standard departure times per modality of transport. Segments have standard costs of transporting a unit of freight from one end of the segment to the other end. Junctions have standard costs of transporting a unit of freight from the end of one connecting segment to the beginning of another connecting segment.
We can now assess (i) length of a route, (ii) shortest routes between two junctions, (iii) duration time of standard transport along a route, including transfer, stopover and possible reloading times at junctions, and iv) shortest duration time route of standard transport between two junctions.

```
type
    L, TI
value
    ms:M-set,                      axiom ms≠{}
    obs_L: S → L
    obs_L: Si × J × M × Si → L
    obs_TI: S → TI
    obs_TI: Si × J × Si → TI
    obs_TI: J × M ⥲ TI,            pre obs_TI(j,m): m ∈ obs_Ms(j)
    obs_TI: J × M × M ⥲ TI,  pre obs_TI(j,m,m′): {m,m′}⊆obs_Ms(j)
    obs_arr_TI: J × M ⥲ TI,     pre obs_arr_TI(j,m): m ∈ obs_Ms(j)
```

[4] mj_{m_i} is not to be confused with the junction identification of j_i of j.

```
obs_dep_TI: J × M $\overset{\sim}{\rightarrow}$ TI,   **pre** obs_dep_TI(j,m): m ∈ obs_Ms(j)
+: L × L → L
+: TI × TI → TI
```

Annotations:

- L and Ti are sorts designating length and time values.
- ms denotes a non-empty set of modes.
- From a segment one can observe, obs_L, its length.
- From a segment one can observe, obs_TI, a time duration for a normal conveyour of the mode of the segment to travel the length of the segment.
- From a junction and a mode (of that junction) one can observe, obs_TI, a time duration for a normal conveyour of the mode to cross, i.e., to travel through the junction.
- From a junction and a pair of modes (m and m′ of that junction) one can observe, obs_TI, a time duration which represents the normal time it takes to transfer freight from a conveyour of mode m to a conveyour of mode m′. (The two modes may be the same.)
- From a junction and a mode (of that junction) one can observe, obs_arr_TI, a time duration for an item of freight destined for a normal conveyour of the mode to arrive and be "entry" processed (including loaded) at that junction.
- From a junction and a mode (of that junction) one can observe, obs_dep_TI, a time duration for an item of freight destined for a normal conveyour of the mode to arrive and be "exit" processed (including unloaded) at that junction.
- One can add lengths.
- One can add time durations.

2.3.2 Route Lengths

One can compute the length of a route of a net and one can find the shortest such route between two identified junctions.

```
value
  length: R → N $\overset{\sim}{\rightarrow}$ L
  length(r)(n) ≡
    case r of
      ⟨⟩ → 0,
      ⟨(jf,si,jt)⟩ → obs_L(xtr_S(si,n)),
      ⟨(ji1,sii,ji2),(jj1,sij,jj2)⟩^r′ →
        let si=xtr_S(sii,n),sj=xtr_S(sij,n) in
        obs_L(si) + obs_L(sii,xtr_J(ji2,n),sij) + length(⟨(jj1,sij,jj2)⟩^r′) end
    end
    pre: r ∈ routes(n) ∧ ji2=jj1
```

```
shortest_route: Ji × Ji → N $\xrightarrow{\sim}$ R
shortest_route(jf,jt)(n) ≡
    **let** rs = routes(n) **in**
    **let** crs = {r|r:R•r ∈ rs ∧ first_Ji(r)=jf ∧ last_Ji(r)=jt} **in**
    **let** sr:R • sr ∈ crs ∧ ∼∃ r:R • r ∈ crs ∧ length(r)(n)<length(sr)(n) **in**
    sr **end end end**
    **pre**: {jf,jt}⊆obs_Jis(n) ∧ jf≠jt
```

Annotations:

- The length of a single modality route of a net
  - is 0 if the route is empty,
  - otherwise it is the length of the first segment of the route plus the length of the rest of the route computed as follows:
    * If the route consists of just one segment, then 0,
    * else, the length of the junction from incident segment to emanating segment plus
    * the length of the rest of the route computed as otherwise specified above.
- The shortest route of a net between two of its identified junctions (the precondition) can be abstractly determined as follows:
  - First we find all the routes, rs, of the net.
  - Then we find those routes, crs, whose first and last junction identifications are the given ones, jf and jt.
  - Amongst those we find a shortest one, that is, one, in crs, for which there are no shorter routes, r, in crs.

2.3.3 Route Traversal Times

One can find the total time it takes to traverse a route, including the times it takes to pass through a junction, and one can find the quickest route between two identified junctions.

```
all_time: R → N → TI
all_time(r)(n) ≡
    obs_arr_TI(xtr_J(first_J(r),n),obs_M(first_S{r}))
    + time(r)(n)
    + obs_dep_TI(xtr_J(last_J{r},n),obs_M(last_S(r)))
time: R → N → TI
time(r)(n) ≡
    **case** r **of**
        ⟨⟩ → 0,
        ⟨(jf,si,jt)⟩ → obs_TI(xtr_S(si,n)),
        ⟨(ji1,sii,ji2),(jj1,sij,jj2)⟩^r′ →
            **let** si=xtr_S(sii,n),sj=xtr_S(sij,n) **in**
            obs_TI(si) + obs_TI(sii,xtr_J(ji2,n),sij) + time(⟨(jj1,sij,jj2)⟩^r′) **end**
    **end**
    **pre**: r ∈ routes(n) ∧ ji2=jj1
  quickest_route: Ji × Ji → N → R
  quickest_route(jf,jt)(n) ≡
    let rs = routes(n) in
    let crs = {r|r:R•r ∈ rs ∧ first_Ji(r)=jf ∧ last_Ji(r)=jt} in
    let qr:R • qr ∈ crs ∧ ∼∃ r:R • r ∈ crs ∧ all_time(r)(n)<all_time(qr)(n) in
    qr end end end
```

2.3.4 Function Lifting

Notice how the two functions shortest_route and quickest_route differ only by the length, respectively the time functions. Hence:

```
type
  Q
  FCT = R → N → Q
value
  less: Q × Q → Bool
  lowest: Ji × Ji → N → FCT → R
  lowest(jf,jt)(n)(fct) ≡
    let rs = routes(n) in
    let crs = {r|r:R•r ∈ rs ∧ first_Ji(r)=jf ∧ last_Ji(r)=jt} in
    let lr:R • lr ∈ crs ∧ ∼∃ r:R • r ∈ crs ∧ less(fct(r)(n),fct(qr)(n)) in
    lr end end end
```

Similarely one could also lift the ‘less’ predicate:

```
  Q
  PRE = Q × Q → Bool
  FCT = R → N → Q
value
  best: Ji × Ji → N → FCT → PRE → R
  best(cf,ct)(n)(fct)(pre) ≡
    let rs = routes(n) in
    let crs = {r|r:R•r ∈ rs ∧ first_Ji(r)=cf ∧ last_Ji(r)=ct} in
    let br:R • lr ∈ crs ∧ ∼∃ r:R • r ∈ crs ∧ pre(fct(r)(n),fct(qr)(n)) in
    br end end end
```

2.3.5 Transportation Costs

We can further assess (i) transport cost (tk:TK), (ii) lowest (per unit) freight cost (fk:FK) between two junctions, etc. We assume that if a freight item is transported into a junction and out of that junction by the same modality conveyour, then it is not reloaded, i.e., along segments of the same modality.[5]

[5]This grossly simplifying assumption will be removed later. For the time being it allows us to operate with the simple notion of routes that was introduced above. For the reloading case we need to decorate the route notion, effectively making it into a bill of ladings notion: one that prescribes possible reloading at junctions.

```
type
   TK, FK, K = TK|FK
value
   obs_TK: (S|J) → TK
   obs_FK: (S|J) → FK

   +: K × K → K
   cost: R → N → K
   cost(r)(n) ≡
      case r of
         ⟨⟩ → 0,
         ⟨(jf,si,jt)⟩ →
            obs_K(xtr_J(jf,n))+obs_K(xtr_S(si,n))+obs_K(xtr_J(jt,n))
         ⟨(jf,si,jt),(jf′,si′,jt′)⟩⌢r′ → assert: jt=jf′
            obs_K(xtr_J(jf,n))+obs_K(xtr_S(si,n))+...+cost(r′)
      end

   cheapest: Ji×Ji → N → ((K×K)→K) → ((K×K)→Bool) → R
   cheapest(jf,jt)(n) ≡
      best(jf,jt)(n)(λ(k1,k2):(K×K)•k1+k2)(λ(k1,k2):(K×K)•k1<k2)
```

2.4 Road Nets

We wish to view road nets at different levels of abstraction. At a most detailed such level we make no distinction between the road kinds, whether community roads, provincial roads, motor roads or freeways. At another level of abstraction we wish to make exactly those distinctions. And at least detailed level of abstraction we consider certain road junctions to designate road nets of smaller or larger communities.

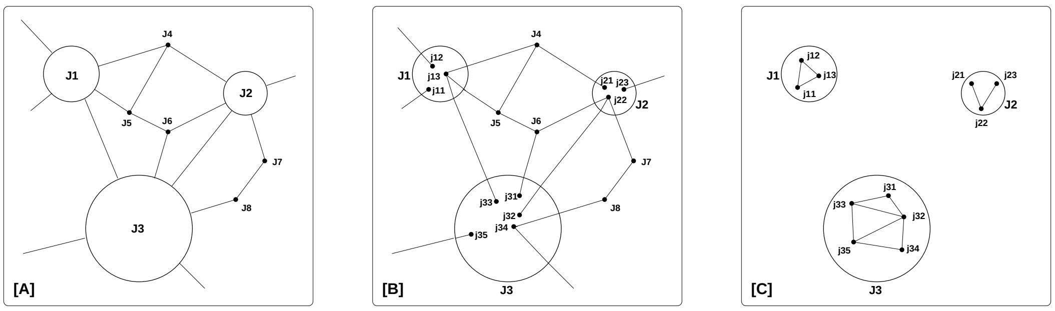


Figure 5: Gross [A] versus semi-detailed [B] road net — and community road nets [C]

Figure [A] 5 shows a road net. Instead of showing junctions J1, J2 and J3 as small black disks we show them as larger circles — for reasons that transpires from Fig. [B] 5.

Junctions J1, J2 and J3 are considered composite, that is, to represent communities.

We may consider the road net of Fig.[A] 5 to be an abstraction of the road net hinted at in Fig.[B] 5.

Junctions j11, j12, . . . , j35 are considered simple embedded junctions.

We decide to allow three kinds of junctions:

composite, simple embedded and simple.

They are as follows:

Composite junctions stand for road nets themselves. The junctions of those road nets are all simple embedded junctions. Simple embedded junctions are the junctions, hence, of composite junction road nets. Simple junctions are those junctions which are not composite (that is: are not standing for road nets) and are not simple embedded junctions (that is: simple, hence un-embedded junctions are those remaining junctions of a net which include modality road).

In Fig. [B] 5 on the preceding page we have left out the internal roads, that is, segments of junctions J1, J2 and J3, that is between the simple embedded junctions j11, j12 and j13, between j21, j22 and j23, and between j31, j32, j33, j34 ans j35.

The internal segments of junctions J1, J2 and J3 are shown in Fig. [C] 5 on the facing page. They are to be considered complete nets "in and by" themselves.

We may consider the implied junction identifications Ji1, Ji2 and Ji3 to be names of communities.

We may consider the implied junction identifications ji11, ji12 and ji13 to abstract to J1, ji21, ji22 and ji23 to abstract to J2, and ji31, ji32, ji33, ji34 and ji35 to abstract to J3.

We shall assume that from these junction identifications, say jikℓ, one can observe the more abstract junction identifications, i.e., Jik.

We shall, conversely, assume that from segment junction identifications one can observe whether they are identifications of composite, of simple embedded or of simple junctions, and, if of composite junctions, that one can further observe which simple embedded junction of the composite junction the segment is connected to.

In summary: When consider any multi-modality net and from it project, that is, consider only the net, n_r, of modality road, then we may find that some junctions are composite while are are simple. When then examining the road nets, r_n, contained in composite junctions then we will find that their junctions are simple embedded. The embedded road nets, r_n, otherwise satisfy all the properties (i.e., axioms) of nets in general. To link up the segments of n_r incident upon, that is, connected to composite junctions (in n_r) we provide their junction identifications with two levels of observability: the abstract one that made us see that they were connected to composite junctions (cf. Fig. [A] 5 on the preceding page), and a concrete one that enables us to decide which ones of the simple embedded junctions they are "finally" linked to (cf. Fig. [B] 5 on the facing page).

```
type
   M == road | ...
   Jc, Js, Jse
   Jic, Jis, Jise
   J = Jc | Js | Jse
   Cn
value
   is_composite_J: J → Bool
   is_simple_J: J → Bool
   is_simple_embedded_J: J → Bool
   obs_N: Jc → N
```

```
    obs_Jic: Jc → Jic, obs_Jis: Js → Jis, obs_Jise: Jse → Jise
    obs_Cn: Jic → Cn, obs_Cn: Jise → Cn
    obs_Jise: Jic → Jise
axiom
    ∀ j:Jc • is_composite_J(j) ∧ xtr_Ms(obs_N(j,road))={road},
    ∀ j:Js • is_simple_J(j),
    ∀ j:Jse • is_simple_embedded_J(j)

    ∀ n:N,j:J • j ∈ obs_Js(n) ∧ is_composite_J(j) ⇒
        let rn = obs_N(j) in

        end
```

2.5 Railway Nets

2.5.1 General

A transportation net of modality railway has segments be lines between stations and have junctions be stations.

We concretise the concept of modes. Mode m=railway will now designate railway nets:

```
type
    M == road | railway | ...
```

From a multi-modal transportation net we can project the railway net, rn:RN:

```
value
    proj: N × {railway} → RN
```

Junctions of a transportation net of modality railway have sub-junctions which are stations:

```
value
    proj: J × {railway} → ST
```

Segments of a transportation net of modality railway become lines:

```
value
    proj: S × {railway} → LI
```

2.5.2 Lines, Stations, Units and Connectors

Railway segments are thus called lines, and railway sub-junctions are thus called stations. A notion of connectors is introduced. It is not to be confused with the previous notion of junctions.

1. A railway net is a net of mode railway.

2. Its segments are lines of mode railway.
3. Its junctions are stations of mode railway.
4. A railway net consists of one or more lines and two or more stations.
5. A railway net consists of rail units.
6. A line is a linear sequence of one or more linear rail units.
7. The rail units of a line must be rail units of the railway net of the line.
8. A station is a set of one or more rail units.
9. The rail units of a station must be rail units of the railway net of the station.
10. No two distinct lines and/or stations of a railway net share rail units.
11. A station consists of one or more tracks.
12. A track is a linear sequence of one or more linear rail units.
13. No two distinct tracks share rail units.
14. The rail units of a track must be rail units of the station (of that track).
15. A rail unit is either a linear, or is a switch, or a is simple crossover, or is a switchable crossover, etc., rail unit.
16. A rail unit has one or more connectors.
17. A linear rail unit has two distinct connectors. A switch (a point) rail unit has three distinct connectors. Crossover rail units have four distinct connectors (whether simple or switchable), etc.
18. For every connector there are at most two rail units which have that connector in common.
19. Every line of a railway net is connected to exactly two distinct stations of that railway net.
20. A linear sequence of (linear) rail units is an acyclic sequence of linear units such that neighbouring units share connectors.

type
1. RN = {| n:smN • obs_M(n)=railway |}
2. LI = {| s:S • obs_M(s)=railway |}
3. ST = {| c:C • obs_M(c)=railway |}
   Tr, U, K

value
4. obs_LIs: RN → LI-**set**
4. obs_STs: RN → ST-**set**

```
5.  obs_Us: RN → U-set
6.  obs_Us: LI → U-set
8.  obs_Us: ST → U-set
11.  obs_Trs: ST → Tr-set
15.  is_Linear: U → Bool
15.  is_Switch: U → Bool
15.  is_Simple_Crossover: U → Bool
15.  is_Switchable_Crossover: U → Bool
16.  obs_Ks: U → K-set

20.  lin_seq: U-set → Bool
       lin_seq(us) ≡
         ∀ u:U • u ∈ us ⇒ is_Linear(u) ∧
         ∃ q:U* • len q = card us ∧ elems q = us ∧
           ∀ i:Nat • {i,i+1} ⊆ inds q ⇒ ∃ k:K •
             obs_Ks(q(i)) ∩ obs_Ks(q(i+1)) = {k} ∧
           len q > 1 ⇒ obs_Ks(q(i)) ∩ obs_Ks(q(len q)) = {}
```

axiom

4. ∀ n:RN • **card** obs_LIs(n) ≥ 1 ∧ **card** obs_STs(n) ≥ 2

6. ∀ n:RN, l:LI • l ∈ obs_LIs(n) ⇒ lin_seq(l)

7. ∀ n:RN, l:LI • l ∈ obs_LIs(n) ⇒ obs_Us(l) ⊆ obs_Us(n)

8. ∀ n:RN, s:ST • s ∈ obs_STs(n) ⇒ **card** obs_Us(s) ≥ 1

9. ∀ n:RN, s:ST • s ∈ obs_LIs(n) ⇒ obs_Us(s) ⊆ obs_Us(n)

10. ∀ n:RN,l,l′:LI•{l,l′}⊆obs_LIs(n)∧l≠l′⇒obs_Us(l)∩ obs_Us(l′)={}

10. ∀ n:RN,l:LI,s:ST•l ∈ obs_LIs(n)∧s ∈ obs_STs(n)⇒obs_Us(l)∩ obs_Us(s)={}

10. ∀ n:RN,s,s′:ST•{s,s′}⊆obs_STs(n)∧s≠s′⇒obs_Us(s)∩ obs_Us(s′)={}

11. ∀ s:ST•**card** obs_Trs(s)≥1

12. ∀ n:RN,s:ST,t:Tr•s ∈ obs_STs(n)∧t ∈ obs_Trs(s)⇒lin_seq(t)

```
13.  ∀ n:RN,s:ST,t,t′:Tr•s ∈ obs_STs(n)∧{t,t′}⊆obs_Trs(s)∧t≠t′
          ⇒ obs_Us(t) ∩ obs_Us(t′) = {}

18. ∀ n:RN • ∀ k:K •
          k ∈ ∪{obs_Ks(u)|u:U•u ∈ obs_Us(n)}
             ⇒card{u|u:U•u ∈ obs_Us(n)∧k ∈ obs_Ks(u)}≤2

19. ∀ n:RN,l:LI •  l ∈ obs_LIs(n) ⇒
```

```
∃ s,s′:ST • {s,s′} ⊆ obs_STs(n) ∧ s≠s′ ⇒
    let sus=obs_Us(s),sus′=obs_Us(s′),lus=obs_Us(l) in
    ∃ u,u′,u″,u‴:U • u ∈ sus ∧
        u′ ∈ sus′ ∧ {u″,u‴} ⊆ lus ⇒
        let sks = obs_Ks(u), sks′ = obs_Ks(u′),
            lks = obs_Ks(u″), lks′ = obs_Ks(u‴) in
        ∃!k,k′:K•k≠k′∧sks ∩ lks={k}∧sks′ ∩ lks′={k′}
    end end
```

2.6 Net Dynamics

By net dynamics we shall mean the changing possibilities of flow of conveyors (cars, trains, aircraft, ships, etc.) along segments and through junctions. We speak of direction of flow along segments in terms of *"from the junction at one end of the segment to the junction at the other end"*. And we speak of flow through a junction as *"proceeding from one segment incident upon the junction into a (udually different) segment emanating from that junction"*. Segments connected to a junction are both incident upon that junction and emanates from that junction.

2.6.1 Segment and Junction States

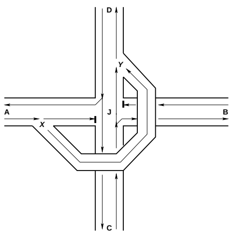


Figure 6: A Special "Carrefour" Junction

Segments may be open for traffic in either or both directions (between the segments' two junctions [identified by ji_x and ji_y]) or may be closed. We model the state, $s\sigma : S\Sigma$, of a segment, $s : S$, as a set of pairs of junction identifications, namely of the two identifications of the junctions that the segment connects. This state, $s\sigma : S\Sigma$, is either empty, i.e., the segment is closed ($\{\}$), or has one pair, $\{(ji_x, ji_y)\}$, that is, the segment is open in direction from junction ji_x to junction ji_y, or another pair $\{(ji_y, ji_x)\}$, or both pairs $\{(ji_x, ji_y), (ji_y, ji_x)\}$, that is, is open in both directions. Junctions may direct traffic from any subset of incident segments to any subset of emanating segments. We model the state, $j\sigma : J\Sigma$, of a junction, $j : J$, as a set of pairs of segment identifications, namely of identifications of segments connected to the junction. Let the set of identifications of segments connected to junction j be $\{si_1, si_2, ..., si_m)\}$. If, in some state, $j\sigma$ of the junction, it is possible (allowed) to pass through the junction from the segment identified by si_j to the segment identified by si_k, then the pair (si_j, si_k) is in $j\sigma$. The junction state may be empty, i.e., closed: no traffic is allowed through the junction. Or the junction state may be "anarchic full", that is, it contains all combinations of the pairs of identifiers of segments incident upon the junction.

```
type
    SΣ = (Ji×Ji)-set
    JΣ = (Si×Si)-set
value
    obs_SΣ: S → SΣ
    obs_JΣ: J → JΣ

    xtr_Jis: SΣ → Ji-set, xtr_Jis(sσ) ≡ {ji|ji:Ji • (ji,_) ∈ obs_sσ ∨ (_,ji) ∈ obs_sσ}
    xtr_Sis: JΣ → Si-set, xtr_Sis(jσ) ≡ {si|si:Si • (si,_) ∈ obs_jσ ∨ (_,si) ∈ obs_jσ}
axiom
    ∀ s:S • xtr_Jis(obs_SΣ(s)) ⊆ xtr_Jip(s),
    ∀ j:J • xtr_Sis(obs_JΣ(j)) ⊆ xtr_Sis(j)
```

Observations:

- A junction, $j : J$, of just one segment, $s : S$, that is, s is a cul de sac, may either be closed, and vehicles trying to enter j will be queued up, or it is open, and vehicles entering j will be lead back to s.
- As a consequence segment s, in order for this latter routing to happen, must be open in both directions when j is "open".
- In general, if the state of a junction j (identified by ji) contains a pair (si_x, si_y) then the state of the designated segments, sx and sy, must respectively contain pairs (ji', ji), respectively (ji, ji''), where $\{ji, ji'\}$ and $(ji, ji''\}$ are the pairs of junction identifications associated with si_x and si_y respectively.
- And this must hold for all states of junctions and adjacent segments.
- This is captured in the axioms below.

```
axiom
    ...
```

The junction of Fig. 6 shows four segments, identified by A, B, C and D. The figure also suggests a state in which traffic lights prohibit movements from A into J, from B into J, from C via J into A, and from D via J into B. The "bypass" from A/X into Y/D appears to be such that traffic can always pass from A into D. The current state alluded to in Fig. 6 on the previous page appears to be:

$$j\sigma_J : \{(A, D), (C, B), (C, D), (D, A), (D, C)\}$$

(A, D) is potentially a member of every state that the junction can possibly be in — see next section.

2.6.2 Segment and Junction State Spaces

A state space is a set of states. A segment can be in one of several segments states. A junction can be in one of several junction states. Hence we introduce segment and junction state spaces.

```
type
    SΩ = SΣ-set
    JΩ = JΣ-set
value
    obs_SΩ: S → SΩ
    obs_JΩ: J → JΩ
axiom
    ∀ s:S • obs_SΣ(s) ⊆ obs_SΩ(s),
    ∀ j:J • obs_JΣ(j) ⊆ obs_JΩ(j)
```

2.7 More on Net Dynamics: Traffic

2.7.1 Vehicles and Positions

There is a further undefined notion of vehicles, V. And there is a notion of the position, P, of a vehicle. Either a vehicle is positioned in a junction, and then its position is designated by the junction identifier. Or a vehicle is positioned along a segment, and then its position is designated by a triplet: the identifier of the junction it is moving away from, the identifier of the junction it is moving towards, and the fraction of the distances from the position to the two junctions: If the fraction is 0, then the vehicle has just entered the segment, if the fraction is 1, then the vehicle is just about to leave the segment, and, hence, if the fraction is a proper real between o and 1, but neither 0 nor 1, then the vehicle is properly within the segment.

```
type
    F = {|f:Real•0≤f≤1|}
    P == mkP_at_J(ji:Ji) | mkP_along_S(fji:Ji,f:F,tji:Ji)
```

2.7.2 Traffic

Traffic is now a function from time to a pair of a net, and the positions of vehicles within the net.

```
type
    V
    T
    TF = T $\overrightarrow{m}$ (N × (V $\overrightarrow{m}$ P))
```

Proper Vehicle Positions The positions of a traffic must designate proper junctions of the net.

```
axiom
    ∀ tf:TF •
        ∀ t ∈ dom tf •
            let (n,vps) = tf(t) in
```

```
∀ p:P • p ∈ **rng** vps ⇒
**case** p **of**
    mkP_at_J(ji) → ji ∈ obs_Jis(n),
    mkP_along_S(jf,_,jt) → {jf,jt}⊆obs_Jis(n)
**end end**
```

Other Traffic Constraints Traffic must be smooth: Positions of vehicles do not "jump around", i.e., movement are monotonic. No "ghost vehicles": If at times t and t' considered close to one another a vehicle is in the traffic then it is also in the traffic at all times in between t and t'. We omit the formalisations of these constraints.

2.8 Time Tables and Traffic

2.8.1 Time Tables

By a time table we understand an entity which to named transport vehicles associate journey descriptions. By a journey description we understand a sequence of junction visits. By a junction visit we understand a triple: Arrival time, junction identifier and departure time.

```
type
  TT = Vn  $\overrightarrow{m}$  Journey
  Journey = (at:T × ji:Ji × dt:T)$^*$
```

2.8.2 Scheduling

By scheduling we shall here, in a narrow sense, understand a function from nets and time tables to a possibly infinite set of traffics such that each traffic satisfies the time table.

```
value
  sched: TT → N → TF-infset
  sched(tt)(n) as tfs
    pre: wf_TT_and_N(tt,n)
    post: ∀ tf:TF • tf ∈ tfs ⇒ wf_TF(tf) ∧ sat(tf,tt)

  wf_TT_and_N: TT × N → Bool, ...
  sat: TF × TT → Bool, ...
```

2.9 And so on!

We have shown fragments of a description of a domain of transportation nets. There is, of course, much more. "Years of work still to be done!" But, for the time being we have enough to illustrate some reasonably interesting requirements.

3 A Set of Requirements

We shall consider the following three sets of requirements: requirements for software to monitor net maintenance, requirements for software to monitor & control net traffic, requirements for software to simulate net traffic, and requirements for software to support transport logistics: optimal routes etc.

3.1 Plan of Development of Requirements

The plan is now to first give a brief, rough sketch narrative of the four sets of requirements. We do so, here, in this paper, in an unusual way. First we 'extend' the domain description given earlier. Then we 'project', 'instantiate', and make less non-deterministic ('determination') the extended domain description, that is: We transform the description description into domain requirements prescriptions. But first we present the domain extensions. After that the plan is to analyse these four domain extension sketches wrt. such common "features" that may be shared by the four (or triples of three or pairs of two) software implementations; to present the requirements for each of the four specific software "packages"; and finally to present the requirements for such a shared "core" of software. That is, we are 'fitting'. 'Extension', 'projection', 'instantiation', 'determination' and 'fitting' are three major domain description-to-reguirements prescription operations. (See Chap. 19 of [3] for details.)

3.2 Brief Narrative of the Four Sets of Requirements

Domain Requirements: By domain requirements we understand requirements that can be expressed by sôlely using terms of the domain (and ordinary, non-technical language).[6] In this paper we shall only consider domain requirements. Of course, many, if not most of the interesting problems of software development in relation also to 'problem frames' may be those due to interface and machine requirements.

3.2.1 'Net Maintenance' Software

We propose a (parameterised) software package to be developed for monitoring and supporting the management of the maintenance of both road and rail nets. An instantiation parameter (road,rail) shall determine whether the package works for road or for rail nets.

'Net Maintenance' Domain Description – An Extension: Segments and junctions need be maintained, that is, we may associate a set of quality attributes related to the upkeep of segments and junctions, as well as of any traffic signals associated with these, we may further associate actual and estimated date(s), cost(s), and duration(s) of previous and next maintenance services, etc., and we may keep "such" records of all segments, junctions and signals of the net. To monitor the net quality attributes, in the domain, some need perform work that help advise maintenance staff to evaluate and report quality attributes of segments, junctions and signals, follow-up on missing such reports, and help update the attributes of the records kept when reported. To support the management of net maintenance some need

[6]By machine requirements we understand requirements that can be sôlely expressed using terms of the machine (and ordinary, non-technical language). By interface requirements we understand requirements that can be expressed only by using terms of both the domain and the machine (and ordinary, non-technical language).

perform, in the domain, work that help management schedule and allocate resources for the monitoring of net quality and corresponding update of records, for the actual maintenance work, and for handling "unforeseen" reports on segment, junction and signal malfunctioning (i.e., in need of repair).

'Net Maintenance' Domain Requirements

Entities: Segments and junctions need be maintained, that is, we must associate a set of quality attributes related to the upkeep of segments and junctions, as well as of any traffic signals associated with these, we must further associate actual and estimated date(s), cost(s), and duration(s) of previous and next maintenance services, etc., and we must keep "such" records of all segments, junctions and signals of the net.

Monitoring Functions: To monitor the net quality attributes, in the domain, the software must have functions that help advise maintenance staff to evaluate and report quality attributes of segments, junctions and signals, follow-up on missing such reports, and help update the attributes of the records kept when reported.

Management Functions: To support the management of net maintenance the software must have functions that help management schedule and allocate resources for the monitoring of net quality and corresponding update of records, for the actual maintenance work, and for handling "unforeseen" reports on segment, junction and signal malfunctioning (i.e., in need of repair). ... Here follows precise requirements details (omitted) ...

Domain to Requirements Operations: We give a terse summary. Projection: Most of the net attributes have been kept. Many of the concepts (routes, ..) and evaluation functions (time, length, ...) have been "projected away". Instantiation: Usually the software, when delivered to a client, is instantiated to the specific net characteristics of the client. Determination: No example as looseness and non-determinism is basically absent from this part of the domain. *$\mathcal{E}$tcetera!*

3.2.2 'Traffic Control' Software

We propose a software package to be developed for monitoring and controlling road net traffic not just at local junctions but along segments, and providing for "green" flow along certain route directions.

Domain Description – A Very Rough Sketch Extension: Traffic control in the conventional, non-technological net domain is done by traffic police controlling junction flows or by local sensors and actuators positioned near junctions sensors monitor only local traffic and actuators control only local junction semaphores. An assessment is made (by police or sensors) of local traffic density only, and appropriate arm signals or semaphore ligting (red, yellow, green) acts as controls.

Domain Requirements

Net Representation "In the Machine": The road net must be represented: segments, junctions and signals. Signals must be controlled. Segment, junction and signal states must be represented. Segment lengths and segment and junction (e.g., average) "traversal" times must be represented. Vehicle positions in segments and junctions must be represented. Vehicle positions must be monitored. We assume sensors to record and inform of "density" of vehicles at segment lanes in vicinity of junctions and leading into these. ... here follows precise requirements details ...

Traffic Monitoring Functions: Functions shall regularly sample traffic density. There must be functions for inquiring about and reporting on unusual traffic situations (accidents, fog, road conditions in general). It is assumed that there are functions which otherwise report on the statues of the road net. (That is, functions which relate to the net maintenance software.) ... here follows precise requirements dertails ...

Traffic Control Functions: The objective of the use of these functions is to ensure smooth traffic. Individual functions shall determine the setting of signals at junctions. Composite functions shall determine the setting of signals, say in "green waves" along routes — hence the road net representation must be augmented with information about major and minor routes, time of day preferred directions: am "into town", pm "out of town", and the like. ... Here follows precise requirements details (omitted) ...

Domain to Requirements Operations. Projection: Only the junction and segment state attributes need be kept. Instantiation: The net is instantiated to a particular road net of a particular city, i.e., that of the client. Determination: Some segments are designated as priority segments, with determined directions being "favoured" for "green traffic flow" at determined time intervals of the day. Accordingly some junction state transitions are "favoured" over others. *$\mathcal{E}$tcetera!*

3.2.3 'Traffic Simulation' Software

We propose a software package to be developed for simulating road net traffic. In the domain there is, we assume, as yet no such simulation software. So we cannot domain describe what we mean by simulation — or rather: any such domain description becomes the domain requirements.

Net Representation: Net representation " in the machine": The road net must be represented: segments, junctions and signals. Segment, junction and signal states must be represented. Segment lengths and segment and junction (e.g., average) "traversal" times must be represented. Vehicle positions in segments and junctions must be represented. Assumptions: Vehicles, when moving, move at average speed plus/minus some minor deviations.

Simulation Concepts: We suggest, not as part of the requirements, but as a software implementation idea, the following two ideas:

Representation of segment geodetic profile: A segment is decomposed into geodetic blocks. The curvature of each block is represented by two 3D vectors, from which a Bezier curve for that block can be constructed.

Representation of segment velocity profile: A segment is decomposed into velocity blocks. The increase/decrease of speed for each block can be represented by two 2D vectors, from which a Bezier velocity curve for that block can be constructed: The computation of the curve will, depending on vector characteristics (long or short vectors), compute close, or less close, or "far away" points on the curve, and we shall take the varying density of these computed points to designate positions of a vehicle at any one time, one vehicle per computation of the velocity curve.

Traffic Simulation Functions: Initialise states of segments and junctions wrt. signals. Initialise states of segments and junctions wrt. vehicle positions. That is: allow vehicles to start their journey along segments and in junctions when the simukation begins, and/or at different times during the simulation (say according to some time table). Schedule simulation interval and resolution (granularity, i.e., one unit of simulation time = R units or real time.[7]). "Play, stop, recommence" simulation. Change granularity while "playing". Insert vehicles during simulation.

Domain to Requirements Operations. Projection: We project away almost all but the net and time tables. We adhere to definition of traffic (i.e., TF). Instantiation: We instantiate to a specific net. Determination: We may decide to constrain to segment-determined constant velocity traffic. *$\mathcal{E}$tcetera!*

3.2.4 'Transport Logistics' Software

We propose a software package to be developed for supporting freight (incl. container) transport logistics.

Domain Description – An Extension: In the domain planning a journey, for travelling (on a crucial trip) as a passenger on trains, by bus, airplane or by ship, usually requires the use of one or more time tables. Considerations of alternative routes, of multi modal travel, of cost: fast, perhaps expensive, hrried travel versus slower, perhaps less costly, and of overnight stays en route may be important. This applies to freight transport too: refrigeration of freight load, "first to market", etc.

Domain Requirements

Net Representation "In the Machine": The multi modal net must be represented: segments and junctions Segment lengths and average traversal times and traversal costs of segments and junctions[8] must be represented — usually the latter (times and costs) are provided by transport vehicle (truck, train, boat and aircraft) time tables. We may thus discover that we need to extend our domain description: Junction hubs, where freight is transferred from one modality transport to another, may need be further detailed, e.g., as to warehouse facilities (godowns), etc.

[7] R can be any real above 0. If R is less than 1 simulation is microscopic, if it is 1 simulation is "real", if it is larger than 1 simulation is macroscopic.

[8] The traversal time and cost of junctions could be differentiated wrt. modalities: freight being unload-/loaded when incoming and outgoing segment modalities are different, etc.

Logistics Functions: $\mathcal{E}$tcetera !

Domain to Requirements Operations. Projection: The net, its segments and junctions, their length, time, and cost attributes. Also time tables. Most functions related to these. Instantiation: Maybe we instantiate to only a shipping net, or only a rail net? Determination: As a representation of the segment and junction traversal times we may rely on the time tables. *$\mathcal{E}$tcetera!*

3.3 Requirements Prescription of Shared Software

All four rough sketch requirements prescriptions projected into their requirements a core of the net, its segments and junctions. We therefore conclude that a repository, i.e., a database, is needed, one in which segments and junctions are stored. A repository (software system) which allows flexible representation of segment and junction attributes and their initialisation, retrieval and update. So we decide on using some relational database management system.

3.3.1 Net Repository (i.e., Net Database)

Informal Rough Sketch: Segment representations are in the form of relation tuples. Segment attributes are attributes of relations. Junction representations are in the form of relation tuples. Junction attributes are attributes of relations.

Formalisation – a Sketch:

```
type
   SR = ST-set
   JR = JT-set
   ST :: si:Si ftj:Jip m:M le:L ti:TI k:K f:F sσ:SΣ sω:SΩ
   JT :: ji:Ji si:Si-set m:M ti:TI k:K f:F jσ:JΣ jω:JΩ
```

This is noit quite first normal form relational representation. A junction connected to n segments and with a state-space of m possible states — in (primitive) first normal form would require $m \times n$ tuples. Of course “smarter” ways of representing sets of segment identifiers and state space (ω) can be devised. That is not a requirements issue, but a software design issue.

3.3.2 Repository Functions

Rough Sketch Ideas: The observer functions of the domain description are now simple tuple projections. Query facilities offered by the relational DBMS[9] being deployed can be used in connection with many of the functions transformed from the domain description into the specific domain requirements prescriptions. They are the functions that make “heavy” use of observer functions. The various domain requirements prescriptions additionally prescribe repository initialisation and refreshment (i.e., update) functions — and again their design and implementation can be greatly facilitated by the update functions of the chosen relational DBMS. Of course, queries “against” an RDBMS really deposit results in a designated workspace and displays this on the GUI.

[9]DBMS: Database Management System, like Frontbase www.frontbase.com the best, or DB2 www.ibm.com/db2 or SQL www.oracle.com.

Specific Function Signatures:

value
  obs_Jip: S $\rightarrow$ Jip
  sql_project: RelNm$\times${$|$″`si=seg_name`″$|$}$\times${$|$″`ftj`″$|$}$\times$Wn $\rightarrow$ Jip$\times$GUI

The former function is the "further undefined" domain specification observer function. The latter function "approximates" an SQL query — where we do not show the functional arguments for the RDBMS and the workspace.

"General" Function Signatures: We intimate database retrieve (query, observer), initialise, and refresh (update), function signatures:

value
  query: retrive_function $\times$ RDBMS $\times$ Wn $\rightarrow$ GUI
  init: (S$|$J)-**set** $\times$ RDBMS $\rightarrow$ RDBMS $\times$ GUI
  refresh: (S$|$J)-**set** $\times$ RDBMS $\rightarrow$ RDBMS $\times$ GUI

4 And So On!

4.1 What Have we Covered

We have given a rather large fragment of a domain description. We have postulated and given small fragments of four domain requirement prescriptions. We have indicated how these domain requirements were "derived" from the domain description. We have formalised the domain description. We hardly formalised the domain requirements. But could (easily) do that! The four domain requirements reflect different problem frames.

4.2 Domains, Requirements and Problem Frames

We claim to have intimated the following problem frames (PF):

- Common Software: II: Information Intensive PF.
- Maintenance: Weak Reactive[10] $\oplus$ II PF
- Traffic Control: Strong Reactive[11] $\oplus$ II PF.
- Simulation: Computation $\oplus$ Virtual Real-time $\oplus$ II PF.
- Logistics: Computation $\oplus$ II PF.

[10]Weak reactive: Non real-time
[11]Strong reactive: "Critical" (i.e., hard) real-time

4.3 The Triptych and the Problem Frame Approaches

4.3.1 General Observations

The triptych approach advises that software development includes: domain engineering (DE), requirements engineering (RE), and software design (SD). The triptych approach does not replace the PF approach. To me the triptych approach augments, supplements the PF approach.

4.3.2 Specific Observations

The triptych approach does not mandate strict linear adherence to DE $\rightarrow$ RE $\rightarrow$ SD but assumes DE $\leftrightarrow$ RE $\leftrightarrow$ SD $\leftrightarrow$ DE iteration. In fact:It is impossible to "discover" all that is relevant about the domain before proceeding to understand the requirements, and all that is relevant about the requirements before proceeding to design the software, $\mathcal{E}$tcetera!

4.4 Grand Challenges of Computing Science

4.4.1 The Grand Challenge of VSTTE

The GC of VSTTE[12] to me appears to focus on "a million lines" of program code that to me appears to be verified with respect to program code annotations where it is not clear to what extent those annotations relate to properties of the code, to requirements, and to domain assumptions.

4.4.2 The Grand Challenge of Ubiquitous Computing

The grand challenge of ubiquitous computing appears to offer a very nice opportunity for a "foothill"[13] experimental project. Take the proposed Automated Highway project. As it could be conceived one is thinking of deploying computers and communication wherever feasible (sometime in future) in the safe and efficient driving of cars, in sorting out cross traffic, etc. So here a far more detailed domain description of transportation nets than intimated here is needed. Etcetera!

4.5 Conclusion

So I immodestly propose that research into and use of the PF approach be augmented by research into and use of the triptych approach, and to adjoin the ("otherwise") highly laudable VSTTE effort with some serious, viz., triptych-oriented program code development. I hope to be able to contribute to the grand challenge of ubiquitous computing.

4.6 Bibliographical Notes

We refer to [1, 2, 3] for a complete coverage of informal as well as formal abstraction and modelling principles and techniques [1], principles and techniques specification of systems and languages [2], and principles and techniques of domain engineering, requirements engineering

[12]VSTTE: Verified Software: Theories, Techniques and Experiments

[13]"Foothill" project: This is one of those terible "americanisms": apparently used to characterise a precursor like, or perhaps rather intial stage project.