4 An Ontology of Requirements Constructions 164

Definition: Requirements. *A condition or capability needed by a user to solve a problem or achieve an objective [134].*
Definition: Machine. *By the machine we understand the hardware[331] plus software[685] that implements some requirements[605], i.e., a computing system[151].* 165
Definition: Requirements Unit. *By a requirements unit[618] we mean a single sentence which expresses an "isolated" requirements. (We omit charaterising "single sentence" and "isolated".)*
Definition: Requirements Prescription. *By a requirements[605] prescription[540] we mean just that: the prescription of some requirements. Sometimes, by requirements prescription, we mean a relatively complete and consistent specification of all requirements, and sometimes just a requirements unit[618].* 166
Definition: Requirements Engineering. *The engineering of the development of a requirements prescription[615], from identification of requirements[605] stake-holders, via requirements acquisition[606], requirements analysis[607], and requirements prescription[615] to requirements validation[800] and requirements verification[807].*
We shall just focus on *requirements prescription*[615], that is, the modelling of *requirements*[605].

4.1 Business Process Re-engineering 167

Definition: Business Process. *By a business process we shall understand a behaviour[79] of an enterprise, a business, an institution, a factory. A business process reflects the ways in which a business conducts its affairs, and is a facet[285] of the domain[239]. Other facets of an enterprise are those of its intrinsics[399], support technology[725], rules and regulations[640], management and organisation[445] (a facet closely related to business processes), and human behaviour[345].* 168
Definition: Business Process Engineering. *By business process engineering[100] we shall understand the design[221], the determination, of business process[99]es. In doing business process engineering one is basically designing, i.e., prescribing entirely new business processes.*

169

Definition: Business Process Re-engineering. *By business process reengineering[101] we shall understand the re-design[221], the change, of business process[99]es. In doing business process re-engineering one is basically carrying out change management[109].*

4.1.1 The Kinds of Requirements 170

We distinguish between three kinds of requirements: (Sect. 4.2) the **domain requirements** are those requirements which can be expressed solely using terms of the domain; (Sect. 4.4) the **machine requirements** are those requirements which can be expressed solely using terms of the machine, and (Sect. 4.3) the **interface requirements** are those requirements which must use terms from both the domain and the machine in order to be expressed.

4.1.2 Goals Versus Requirements 171

Whereas a domain description presents a domain **as it is**, a requirements prescription presents a domain **as it would be** if some required **machine** was implemented (from these requirements). The **machine** is the **hardware** plus **software** to be designed from the
172 requirements. That is, the *machine* is what the requirements are about.

We make a distinction between **goals** and **requirements.** Goals are what we expect satisfied by the software implemented from the requirements. But goals could also be of the system for which the software is required. First we exemplify the latter, then the former.

Goals of a Toll Road System 173

- A goal for a toll road system may be
  - to decrease the travel time between certain hubs and
  - to lower the number of traffic accidents between certain hubs,

Goals of Toll Road System Software 174

- The goal of the toll road system software is to help automate
  - the recording of vehicles entering, passing and leaving the toll road system
  - and collecting the fees for doing so.

Goals are usually expressed in terms of properties. Requirements can then be proved to satisfy the $\mathcal{G}$oals: $\mathcal{D}, \mathcal{R} \models \mathcal{G}$. [149, Lamsweerde] focus on goals.

Arguing Goal-satisfaction of a Toll Road System 175

- By endowing links and hubs with average traversal times for both ordinary road and for toll road links and hubs
  - one can calculate traversal times between hubs
  - and thus argue that the toll road system satisfies [significantly] "quicker" traversal times.
- By endowing links and hubs with traffic accident statistics (real, respectively estimated)
  - for both ordinary road and for toll road links and hubs
  - one can calculate estimated traffic accident statistics between all hubs
  - and thus argue that the combined ordinary road plus toll road system satisfies [significantly] lower traffic fatalities.

Arguing Goal-satisfaction of Toll Road System Software 176

- By recording
  - tickets issued and collected at toll booths and
  - toll road hubs and links entered and left
  - as per the requirements specification brought in forthcoming examples (Sects. 4.2.1–4.2.4),
- we can eventually argue that
  - the requirements of the forthcoming examples (Sects. 4.2.1–4.2.4)
  - help satisfy the goal of the example **??** on page ??.

177

We shall assume that the (goal and) requirements engineer elicit both $\mathcal{G}$oals and $\mathcal{R}$equirements from requirements stake-holders.

$\mathcal{D}, \mathcal{R} \models \mathcal{G}$ The $\mathcal{G}$oals can be argued to hold by reasoning over the $\mathcal{R}$equirements and the $\mathcal{D}$omain.

But we shall focus only on domain and interface requirements such as "derived" from domain descriptions.

4.1.3 Re-engineered Nets 178

The nets defined in Sect. 3 could be of any topology. They could consist of two or more nets that were not linked to one another; they could consist of connected nets or nets that were acyclic; etc.; and the nets were not specifically road, rail, sea lane or air lane nets.
We shall now consider a special kind of road nets: basically the road nets we have in mind 179
are linear sequences of pairs of links of opposite direction link "states", where these links, let us call them toll road links, are connected to toll road hubs; where, in addition, these toll road hubs are linked, via toll plazas (i.e., "special" hubs) to toll road hubs by means of on/off links. 180

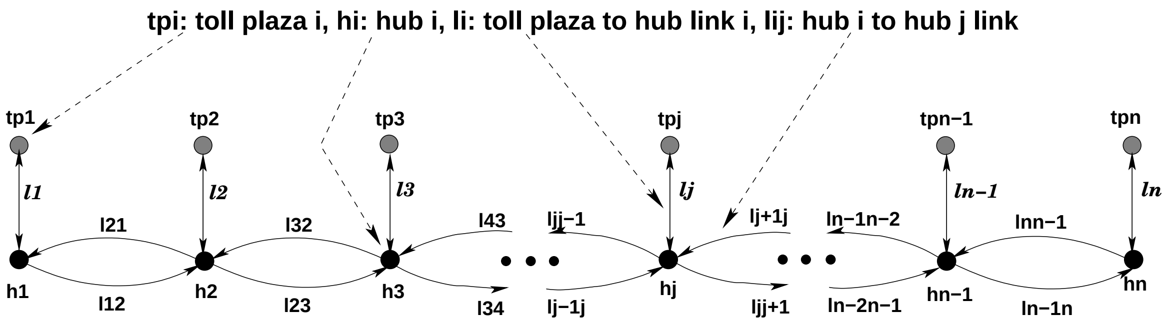

Figure 3: A Toll Road System

181

We do not consider the general nets that are (possibly) connected to the toll plazas. The pragmatics behind these nets is the following: Drivers enter and leave the toll road nets at toll road plazas; collect tickets from toll road plaza ticket-issuing booths when entering the toll road net and present these at toll road plaza ticket-collection booths and pay according to some function of the time and length (from entry to exit plaza) driven on the toll road net when leaving the net; drivers are otherwise free to “circle” the toll road net as they see
182 fit: multiple times “up and down” the net, circling toll road hubs, etc. Our sketch centers around a toll road net with toll booth plazas. The BPR focuses first on entities, actions,
183 events and behaviours (Sect. 2), then on the six domain facets (Sect. 3).

125. **Re-engineered Entities:** We shall focus on a linear sequence of toll road intersections (i.e., hubs) connected by pairs of one-way (opposite direction) toll roads (i.e., links). Each toll road intersection is connected by a two way road to a toll plaza. Each toll plaza contains a pair of sets of entry and exit toll booths. (Sect. 4.2.2 brings
184 more details.)
126. **Re-engineered Actions:** Cars enter and leave the toll road net through one of the toll plazas. Upon entering, car drivers receive, from the entry booth, a plastic/paper/electronic ticket which they place in a special holder in the front window. Cars arriving at intermediate toll road intersections choose, on their own, to turn either “up” the toll road or “down” the toll road — with that choice being registered by the electronic ticket. Cars arriving at a toll road intersection may choose to “circle” around that intersection one or more times — with that choice being registered by the electronic ticket. Upon leaving, car drivers “return” their electronic ticket to
185 the exit booth and pay the amount “asked” for.
127. **Re-engineered Events:** A car entering the toll road net at a toll both plaza entry booth constitutes an event. A car leaving the toll road net at a toll both plaza entry booth constitutes an event. A car entering a toll road hub constitutes an event. A
186 car entering a toll road link constitutes an event.
128. **Re-engineered Behaviours:** The journey of a car,from entering the toll road net at a toll booth plaza, via repeated visits to toll road intersections interleaved with repeated visits to toll road links to leaving the toll road net at a toll booth plaza, constitutes a behaviour — withreceipt of tickets, return of tickets and payment of fees being part of these behaviours. Notice that a toll road visitor is allowed to cruise “up” and “down” the linear toll road net – while (probably) paying for that pleasure
187 (through the recordings of “repeated” hub and link entries).
129. **Re-engineered Intrinsics:** Toll plazas and abstracted booths are added to domain intrinsics.
130. **Re-engineered Support Technologies:** There is a definite need for domain-describing the failure-prone toll plaza entry and exit booths.

131. **Re-engineered Rules and Regulations:** Rules for entering and leaving toll booth entry and exit booths must be described as must related regulations. Rules and regulations for driving around the toll road net must be likewise be described. 188

132. **Re-engineered Scripts:** No need.

133. **Re-engineered Management and Organisation:** There is a definite need for domain describing the management and possibly distributed organisation of toll booth plazas.

134. **Re-engineered Human Behaviour:** Humans, in this case car drivers, may not change their behaviour in the spectrum from diligent and accurate via sloppy and delinquent to outright traffic-law breaking – so we see no need for any "re-engineering".

4.2 Domain Requirements 189

Definition: Domain Requirements. *By domain requirements[605] we understand such requirements (save those of business process reengineering[101]) which can be expressed sôlely by using professional terms of the domain[239].* 190
Definition: Domain Requirements Facet. *By domain requirements[258] facets we understand such domain requirements that basically arise from either of the following operations on domain description[243]s (cum requirements prescription[615]s): domain projection[255], domain determination[245], domain extension[249], domain instantiation[253] and domain fitting[251].*

4.2.1 Projection 191

Definition: Projection. *By projection we shall here, in a somewhat narrow sense, mean a technique that applies to domain description[243]s and yields requirements prescription[615]s. Basically projection "reduces" a domain description by "removing" (or, but rarely, hiding[337]) entities[272], function[310]s, event[281]s and behaviour[79]s from the domain description. If the domain description is an informal one, say in English, it may have expressed that certain entities, functions, events and behaviours might be in (some instantiations of) the domain. If not "projected away" the similar, i.e., informal requirements prescription will express that these entities, functions, events and behaviours shall be in the domain and hence will be in the environment of the machine[436] being requirements prescribed.* 192
Keep the following parts (items) of the domain:

- from Item 1 on page 13 to and including Item 9 on page 14,
- from Item 47a on page 22 to and including Item 48c on page 22,
- from Item 52 on page 27 to and including Item 68 on page 30 and
- from Item 76 on page 35 to and including Item 87 on page 39.

That is, omit these parts:

- Sect. 2.1.5,
- Sects. 2.3–2.4,
- Sects. 2.5.2–2.5.3,
- Sect. 3.2.7 and
- Sects. 3.3–3.7.

and keep these:

- N, H, L,
- obs_Hs,
- obs_Ls,
- HI, LI,
- obs_HI,
- obs_LI,
- obs_LIs,
- obs_HIs,
- PLAN, LHIM,
- wf_PLAN,
- ND, wf_ND,
- LΣ, LΩ,
- obs_LΣ, obs_LΣ,
- HΣ, HΩ,
- obs_HΣ, obs_HΣ,
- V, VI, VP,
- obs_VI, obs_VP,
- TF, T and
- wf_TF.

4.2.2 Instantiation 193

Definition: Instantiation. *'To represent (an abstraction) by a concrete instance[384]' [214]. Domain instantiation is a domain requirements facet[259]. It is an operation performed on a domain description[243] (cum requirements prescription[615]). Where, in a domain description certain entities and function[310]s are left undefined, domain instantiation means that these entities or functions are now instantiated into constant value[802]s.*

Example 194 The following instantiation prescription only covers the static aspects of the toll road net, i.e., simple entities. That is, the states of hubs and links will first be dealt with in Sect. 4.2.3.

135. A toll road net (a subnet of a larger previously described net) consists of a pair: toll road links and toll road to plaza hubs and links.

    a) The toll road links component is a linear sequence of one or more pairs of toll road links.

    b) The toll road to plaza hubs and links component is a linear sequence of two or more triples of
195 a plaza, a (plaza to toll road hub) link and a toll road hub.

    c) The wellformedness of toll road nets are expressed next.

        i. The length of the toll road links sequence is one less than the length of the toll road to plaza hubs and links sequence. The idea is that the toll road links at position i connect the toll road hubs at positions i and $i+1$ of the toll road to plaza hubs and links sequence — i being the indexes of the toll road links sequence.
        ii. All links have distinct link identifiers.
        iii. All hubs and plazas have distinct hub identifiers.
        iv. From the links in the pairs of links, (l_i, l'_i), of position i in the toll road links component one observes exactly the same two element set of hub identifiers,
        v. and these are the identifiers of the hubs at positions i and $i+1$ of the toll road to plaza hubs and links sequence.
        vi. The plaza to toll road hub links are indeed connected to these plazas and hubs; and
196 vii. the plaza and toll road hubs are connected only to the links as mentioned above.

    d) A toll road plaza is like a hub, with an observable hub identifier (and equipped with ticket-issuing tool booths and ticket-collection and payment toll booths).

```
type
135.    TRN′ = TRLs × PHLs
135.    TRN = {|trn:TRN′•wf_TRN(trn)|}
135a.   TRLs = (L × L)*
135b.   PHLs = (PZ × L × H)*
```

197

```
value
135c.     wf_TRN: TRN′ → Bool
135c.     wf_TRN(trn:(trls,phls)) ≡
135(c)i.      len trls +1 = len phls ∧
135(c)ii.     card xtr_Hs(trn) = card xtr_HIs(trn) ∧
135(c)iii.    card xtr_Ls(trn) = card xtr_LIs(trn)
135(c)iv.     ∀ i:Nat•i ∈ inds trls ⇒
135(c)iv.        let (l,l′)=trsl(i),(p,l″,hi)=phls(i),(_,l‴,hj)=phls(i+1) in
135(c)iv.        obs_HIs(l) = obs_HIs(l′) =
135(c)v.         {obs_HI(hi),obs_HI(hj)}  ∧
135(c)vii.       case i of
135(c)vii.          1 → obs_LIs(hi) = xtr_LIs({l,l′,l″}),
135(c)vii.          len trsl − 1 → obs_LIs(hj) = xtr_LIs({l,l′,l‴}),
135(c)vii.          _ → let (l⁗,l′⁗)=trsl(i) in obs_LIs(hi)=xtr_LIs({l,l′,l″,l⁗,l′⁗}) end
135(c)vii.       end end ∧
135(c)vii.    ∀ i:Nat•i ∈ inds phls ⇒
135(c)vii.       let (p,l,h)=phls(i) in obs_HIs(l)=xtr_HIs({p,h}) ∧
135(c)vii.       obs_LIs(p) = {obs_LI(l)} end
```

198

```
type
135d.  PZ
value
135d.  obs_HI: PZ → HI

       xtr_Hs: TRN → H-set
       xtr_Hs(_,phls) ≡ {pz,h|(pz,l,h):(PZ×L×H)•(pz,l,h)∈ elems phls}
       xtr_Ls: TRN → L-set
       xtr_Ls(trls,phls) ≡
              {l,l′|l,l′:L•(l,l′)∈ elems trls} ∪ {l|(pz,l,h):(PZ×L×H)•(pz,l,h)∈ elems phls}

       xtr_HIs: TRN → HI-set,   xtr_HIs(trn) ≡ {obs_HI(h)|h:(H|PZ)•h ∈ xtr_Hs(trn)}
       xtr_LIs: TRN → LI-set,   xtr_LIs(trn) ≡ {obs_LI(l)|l:L•l ∈ xtr_Ls(trn)}
       xtr_HIs: H-set → HI-set, xtr_HIs(hs) = {obs_LI(h)|h:H•h ∈ hs}
       xtr_LIs: L-set → LI-set,  xtr_LIs(ls) = {obs_LI(l)|l:L•l ∈ ls}
```

Abstraction: From Concrete Toll Road Nets to Abstract Nets 199

136. From concrete toll road nets, trn:TRN, one can abstract the nets, n:N, of Items 1–9.
    a) the abstract net contains the hubs of the concrete net,
    b) and the links likewise.

value
136. abs_N: TRN → N
136. abs_N(trn) **as** n
136a. obs_Hs(n) = xtr_Hs(trn) ∧
136b. obs_Ls(n) = xtr_Ls(trn)

Theorem 200

137. One can prove the following theorem: If trn satisfies wf_TRN(trn) then abs_N(trn) satisfies Axioms 2–3 and 5–8 (Page 13).

137. ∀ trn:TRN • wf_TRN(trn) ⊨
abs_N(trn) **satisfies axioms** 2.–3.
∧ **axioms** 5.–8.

4.2.3 Determination 201

Definition: Determination. *Domain determination is a domain requirements facet[259]. It is an operation performed on a domain description[243] cum requirements prescription[615]. Any nondeterminism[482] expressed by either of these specifications which is not desirable for some required software design must be made deterministic (by this requirements engineer[612] performed operation).*

Example 202 We shall focus on making more specific the rather generically defined nets, hubs and links. There are no traffic signals within the toll road net and pairs of toll road links are "one way, opposite direction" links.

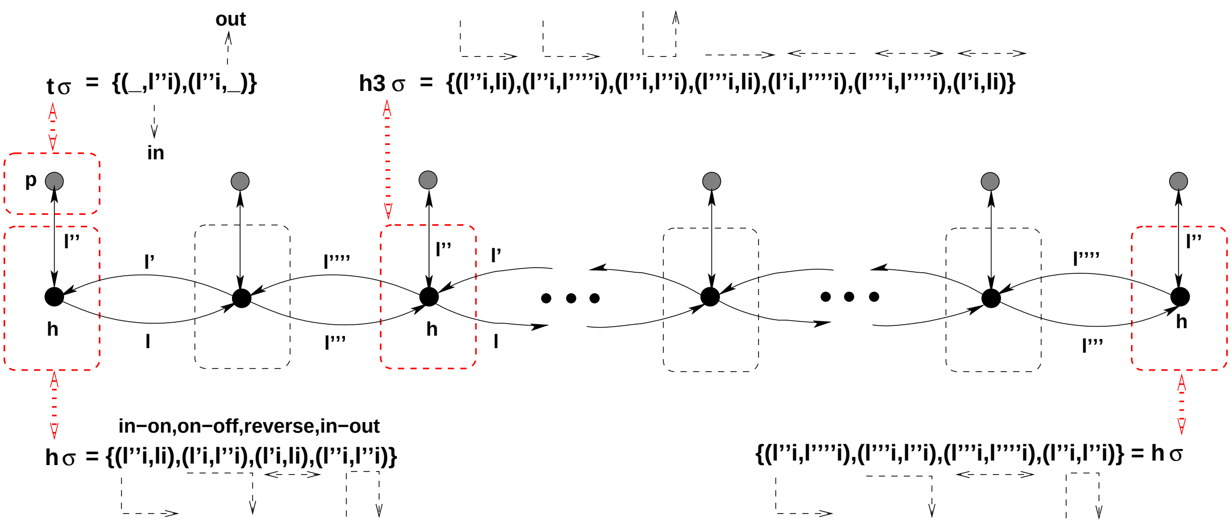

Figure 4: Four example hub states: plaza, end hubs, "middle" hub

203

138. Pairs of toll road links, l, l', connecting adjacent hubs hj, hk, of identifiers hj_i, hk_i, respectively, always and only allow traffic in opposite directions, that is, are always in respective states $\{(hj_i, hk_i)\}$ and $\{(hk_i, hj_i)\}$.

139. Hub, h, states, $h\sigma$, are constant and allow traffic onto connected links not closed for traffic in directions from hub h.
140. Plazas allow traffic only onto connected plaza to hub links of the toll road net. (Whatever other links, “outside” the toll road net, the plazas may be connected to is covered in the last line of the axiom below.) 204

```
axiom
  ∀ (trls,phls):TRN •
      ∀ i:Nat • i ∈ inds trls
         let (l,l′) = trls(i), (p,l′′,h) = phls(i) in
         case i of
              1 → obs_HΣ(h) = {(obs_LI(l′′),obs_LI(l)),
                                 (obs_LI(l′),obs_LI(l′′)),(obs_LI(l′),obs_LI(l)),
                                 (obs_LI(l′′),obs_LI(l′′))},
              _ → let (l′′′,l′′′′) = trls(i−1) in
                    obs_HΣ(h) = {(obs_LI(l′′),obs_LI(l)),
                                 (obs_LI(l′′),obs_LI(l′′′′)),(obs_LI(l′′),obs_LI(l′′)),
                                 (obs_LI(l′′),obs_LI(l′′)),(obs_LI(l′′′),obs_LI(l)),
                                 (obs_LI(l′),obs_LI(l′′′′)),(obs_LI(l′′′),obs_LI(l′′′′)),
                                 (obs_LI(l′),obs_LI(l))} end end end ∧
      let (l′′′,l′′′′) = trls(len trsl), (p,l′′,h) = phls(1 + len trsl) in
      obs_HΣ(h) = {(obs_LI(l′′),obs_LI(l′′′′)),
                    (obs_LI(l′′′),obs_LI(l′′)),(obs_LI(l′′′),obs_LI(l′′′′)),
                    (obs_LI(l′′),obs_LI(l′′))} end ∧
      ∀ (p,l′′,_):(PZ×L×H)•(p,l′′,_) ∈ elems phls ⇒
         let lis = obs_LIs(p) assert: obs_LI(l′′) ∈ lis in
         obs_HΣ(p) = {(li,obs_LI(l′′)),(obs_LI(l′′),li)|li:LI•li ∈ lis} end
```

205

In the last line of the wellformedness axiom above we express that the plaza maybe connected to many links not in the toll road net and that the plaza is open for all traffic from these into the net (via l''), from l'' to these and that traffic may even reverse at the plazas, that is, decide to not enter the toll road net after having just visited the plaza.

4.2.4 Extension 206

Definition: Extension. *Domain extension is a domain requirements facet*[259]*. It is an operation performed on a domain description*[243] *or a requirements prescription*[615]*. It effectively extends a domain description*[243] *by entities, functions, events and/or behaviours conceptually possible, but not necessarily humanly or technologically feasible in the domain (as it was).*

Figure 5 on the following page abstracts some of the extensions to nets: the plaza entry and exit booths. 207

The following is a prolonged example. It contains three kinds of formalisations: a `RAISE/CSP` model, 208 a `Duration Calculus` model [236, 182] and a `Timed Automata` model [5, 182]. The narrative for all three models are given when narrating the `RAISE/CSP` model.

Intuition 209 A toll road system is delimited by toll plazas with entry and exit booths with their gates. To get access, from outside, to the roads within the toll road system, a car must pass through an entry booth and its entry gate. To leave the roads within the toll road system a car must pass through an exit booth and its exit gate. Cars collect tickets upon entry and return these tickets upon exit and pay a fee for having driven on the toll roads. The gates help ensure that cars have collected tickets and have paid their dues. 210

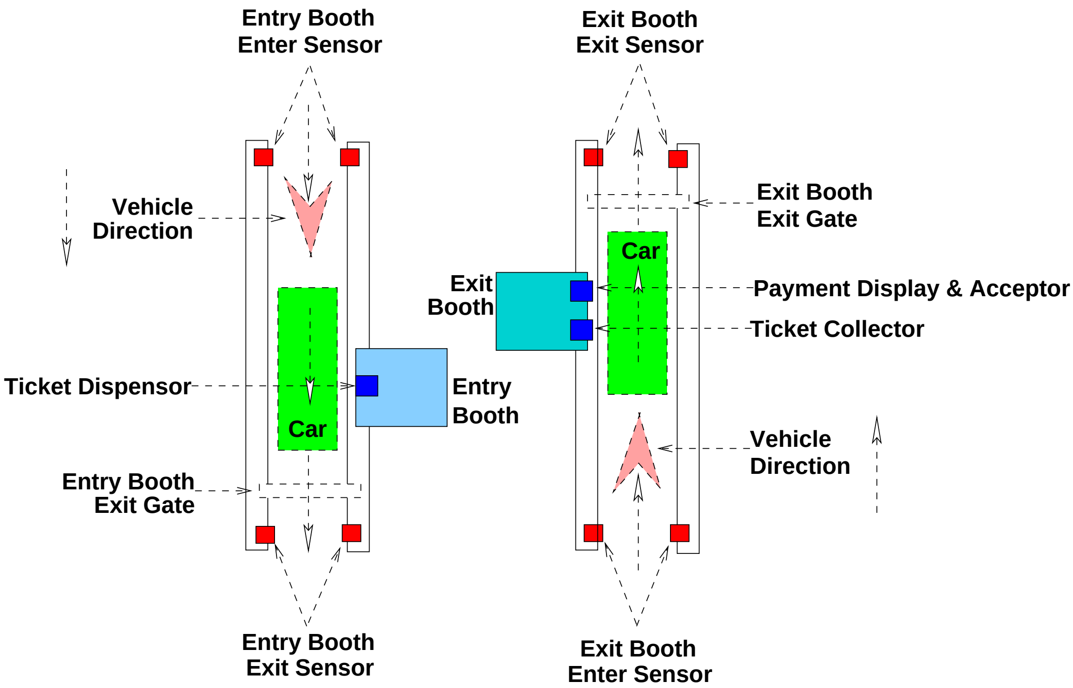

Figure 5: Entry and Exit Tool Booths

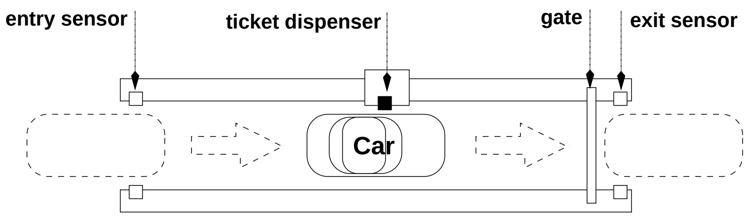

Figure 6: A toll plaza entry booth

Descriptions 211

- **A RAISE/CSP Model** We use the CSP property [32, 131] of RSL.

Toll Booth Plazas With respect to toll road systems we focus on just their plazas: that is, where cars enter and leave the systems. The below description is grossly simplified: instead of plazas having one or more entry and one or more exit booths (both with gates), we just assume one (pair: booth/gate) of
212 each.

141. A toll plaza consists of a one pair of an entry booth and and entry gate and one pair of an exit booth and an exit gate.
142. Entry booths consist of an entry sensor, a ticket dispenser and an exit sensor.
143. Exit booths consist of an entry sensor, a ticket collector, a payment display and a payment component.

type
141. PZ = (EB×G) × (XB×G)
142. EB = ...
143. XB = ...

Cars : 213

144. There are vehicles.

145. Vehicles have unique vehicle identifications.

type
144. V
145. VId
value
145. obs_VId: V → VId
axiom
145. $\forall$ v,v′:V • v≠v′ ⇒ obs_VId(v) ≠ obs_VId(v′)

Entry Booths : 214

The description now given is an idealisation. It assumes that everything works: that the vehicles behave as expected and that the electro-mechanics of booths and gates do likewise.

146. An entry_sensor registers whether a car is entering the entry booth or not,
    a) that is, for the duration of the car passing the entry_sensor that sensor senses the car identification cid
    b) otherwise it senses "nothing". 215

147. A ticket_dispenser
    a) either holds a ticket or does not hold a ticket, i.e., no_ticket;
    b) normally it does not hold a ticket;
    c) the ticket_dispenser holds a ticket soon after a car has passed the entry_sensor;
    d) the passing car collects the ticket –
    e) after which the ticket_dispenser no longer holds a ticket.

148. An exit_sensor
    a) registers the identification of a car leaving the toll booth
    b) otherwise it senses "nothing".

Gates : 216

149. A gate
    a) is either closed or open;
    b) it is normally closed;
    c) if a car is entering it is secured set to close (as a security measure);
    d) once a car has collected a ticket it is set to open;
    e) and once a car has passed the exit_sensor it is again set to close.

The Entry Plaza System : 217

```
type
    C, CI
    G = open | close
    TK == Ticket | no_ticket
value
    obs_CI: (C|Ticket) → CI
channel
    entry_sensor:CI
    ticket_dispenser:Ticket
    exit_sensor:CI
    gate_ch:G
value
    vs:V-set
    eb:EB,xb:XB,eg,xg:G
```

218

```
    system: G × EB × V-set × XB × G
    system(eg,eb,vs,xb,xg) ≡
        ‖{car(obs_CI(c),c)|c:C•c ∈ cs} ‖ entry_booth(eb) ‖ entry_gate(eg) ‖ ...

    car: CI × C → out entry_sensor,exit_sensor
                  in ticket_dispenser  Unit
    car(ci,c) ≡
        entry_sensor ! ci ;
        let ticket = ticket_dispenser ? assert: ticket ≠ no_ticket in
        ticket_dispenser ! no_ticket ;
        exit_sensor ! ci ;
        car(add(ticket,c)) end
```

219

```
    entry_booth: Unit → in entry_sensor, exit_sensor
                        out ticket_dispenser
                        out gate_ch  Unit
    entry_booth(b) ≡
        gate_ch ! close ;
        let ci = entry_sensor ? in
        ticket_dispenser ! make_ticket(cid) ;
        let res = ticket_dispenser ? in assert: res = no_ticket ;
        gate_ch ! open ;
        let ci′ = exit_sensor ? in assert: ci′ = ci ;
        gate_ch ! close ;
        entry_booth(add_Ticket(ticket,b)) end end end
```

220

```
    entry_gate: G → in gate  Unit
    entry_gate(g) ≡
        case gate_ch ? of
            close → exit_gate(close) assert: g = open,
            open → exit_gate(open) assert: g = close
```

end

add_Ticket: Ticket × C $\stackrel{\sim}{\to}$ C
pre add_Ticket(t,c): ∼has_Ticket(c)
post: add_Ticket(t,c): has_Ticket(c)

221

has_Ticket: (C|B) → **Bool**

obs_Ticket: (C|B) $\stackrel{\sim}{\to}$ Ticket
pre obs_Ticket(cb): has_Ticket(cb)

rem_Ticket: (C $\stackrel{\sim}{\to}$ C) | (B $\stackrel{\sim}{\to}$ B)
pre rem_Ticket(cb): has_Ticket(cb)
post rem_Ticket(cb): ∼has_Ticket(cb)

In the next section, "A `Duration Calculus` Model", we shall start refining the descriptions given above. We do so in order to handle failures of vehicles to behave as expected and of the electro-mechanics of booths and gates.

222

• A `Duration Calculus` Model We use the `Duration Calculus` [236, 182] extension to `RSL`. We abstract the channels of the `RAISE/CSP` model to now be Boolean-valued variables. 223

type
ES = **Bool** [**true**=passing, **false**=not_passing]
TD = **Bool** [**true**=ticket, **false**=no_ticket]
G = **Bool** [**true**=open, **false**=closing⌈⌉closed⌈⌉opening]
XS = **Bool** [**true**=car_has_just_passed, **false**=car_passing⌈⌉no-one_passing]
variable
entry_sensor:ES := **false** ;
ticket_dispenser:TD := **false** ;
gate:G := **false** ;
exit_sensor:XS := **false** ;

224

150. No matter its position, the gate must be closed within no more than δ_{eg} time units after the entry_sensor has registered that a car is entering the toll booth.
151. A ticket must be in the ticket_dispenser within δ_{et} time units after the entry_sensor has registered that a car is entering the toll booth.
152. The ticket is in the ticket_dispenser at most δ_{tdc} time units
153. The gate must be open within δ_{go} time units after a ticket has been collected.
154. The exit sensor is registering (i.e., is on) the identification of exiting cars and is not registering anything when no car is passing (i.e., is off).

225

150. $\sim(\lceil \text{entry\_sensor} \rceil ; (\ell = \delta_{eg} \wedge \lceil \text{gate} \rceil))$
151. $\sim(\lceil \text{entry\_sensor} \rceil ; (\ell = \delta_{et} \wedge \lceil \sim\text{ticket\_dispenser} \rceil))$
152. $\Box(\lceil \sim\text{ticket\_dispenser} \rceil \Rightarrow \ell < \delta_{tdc})$
153. $\sim(\lceil \text{ticket\_dispenser} \rceil ; (\lceil \sim\text{ticket\_dispenser} \wedge \sim\text{gate} \rceil \wedge \ell \geq \delta_{go}))$
154. $\Box(\lceil \text{gate=closing} \rceil \Rightarrow \lceil \sim \text{exit\_sensor} \rceil)$

• **A** `Timed Automata` **Model** A timed automaton [5, 182] for a configuration of an entry gate, its entry booth and a car is shown in Fig. 7. Figure 8 on the facing page shows the a car, an exit booth and its exit gate interactions. They are more-or-less "derived" from the example of Sect. 7.5 of [5, Alur & Dill, 1994] (Pages 42–45). The right half of the car timed automaton of Fig. 7 is to be thought of as the same 226
227 as the left half of the car timed automaton of Fig. 8 on the facing page, cf. the vertical dotted (⁝) line.

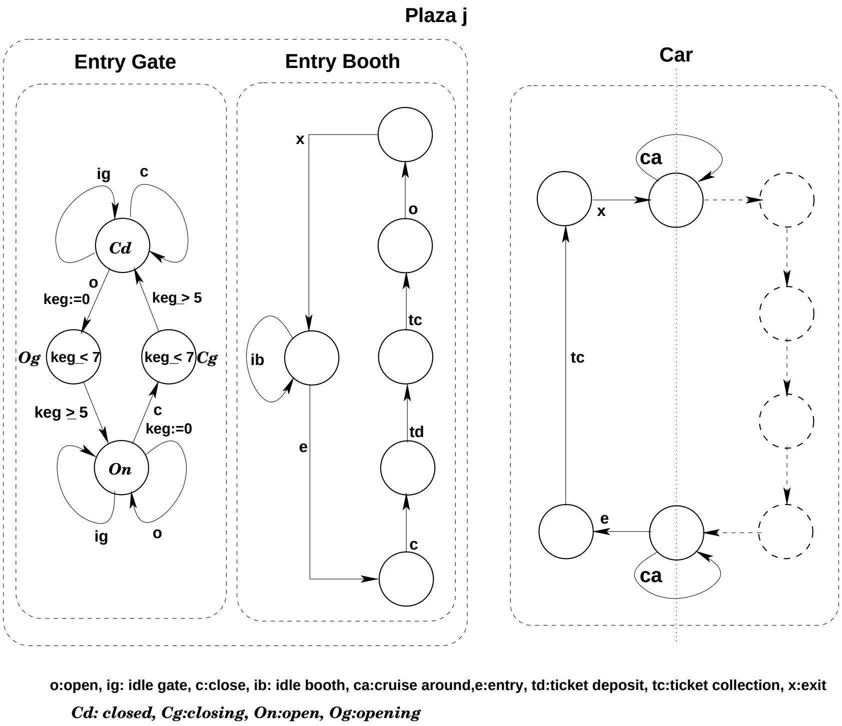

Figure 7: A `timed automata` model of gate, entry booth and car interactions

228

value
eg,xg:G, eb:EB, xb:XB, vs:V-**set**

System: G×EV×V-**set**×XB×G → **Unit**
System(eg,eb,vs,xb,xg) ≡
Entry_Gate(eg) ‖ Entry_Booth(eb) ‖
‖{Car(obs_CId(c),c)|ci:C,v:C•c ∈ cs} ‖
Exit_Booth(xb) ‖ Exit_Gate(xg)

229

4.2.5 Fitting 230

Definition: Fitting. *By domain requirements fitting we understand an operation which takes n domain requirements prescriptions, d_{r_i} $(i = \{1..n\})$, claimed to share m independent sets of tightly related*

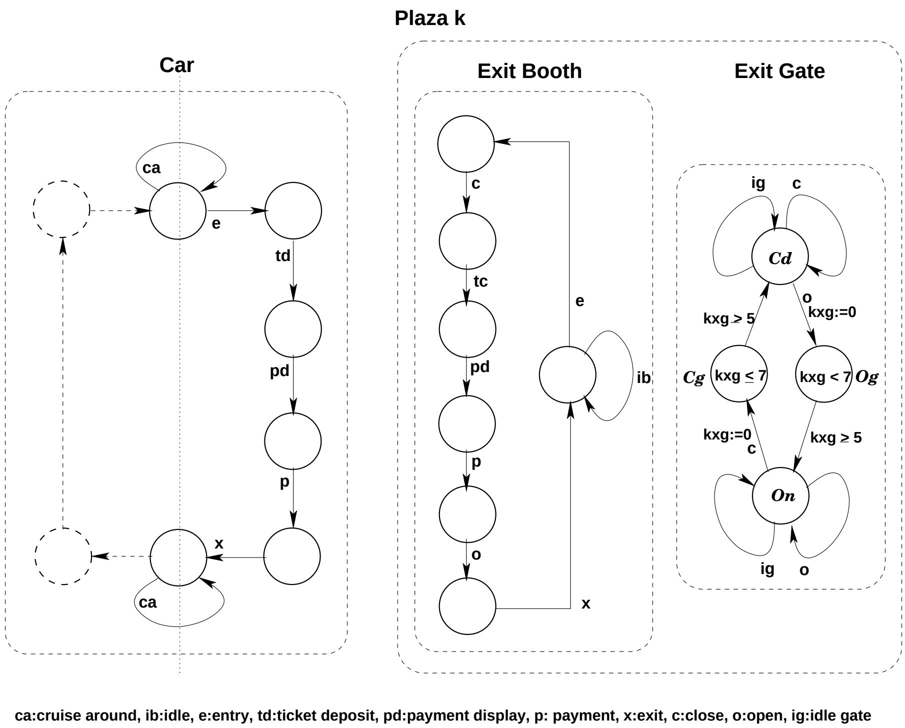

Figure 8: A `timed automata` model of car, exit booth and gate interactions

sets of simple entities, actions, events and/or behaviours and map these into $n+m$ domain requirements prescriptions, δ_{r_j} ($j = \{1..n+m\}$), where m of these, $\delta_{r_{n+k}}$ ($k = \{1..m\}$) capture the m shared phenomena and concepts and the other n prescriptions, δ_{r_ℓ} ($\ell = \{1..n\}$), are like the n "input" domain requirements prescriptions, d_{r_i} ($i = \{1..n\}$), except that they now,(instead of the "more-or-less" shared prescriptions, that are now consolidated in $\delta_{r_{n+k}}$)prescribe interfaces between δ_{r_i} and $\delta_{r_{n+k}}$ for $i : \{1..n\}$.

Examples 231

TO BE WRITTEN

4.3 Interface Requirements 232

Definition: Interface Requirements. *Interface requirements are those requirements[605] which can on be expressed using professional terms from both the domain[239] and the machine[436]. Thus, by interface requirements we understand the expression of expectations as to which software-software, or software-hardware interface[393] places (i.e., channel[110]s), input[382]s and output[502]s (including the semiotics[658] of these input/outputs) there shall be in some contemplated computing system[151]. Interface requirements can* 233 *often, usefully, be classified in terms of shared data initialisation requirements[671], shared data refreshment requirements[673], computational data+control requirements[146], man-machine dialogue requirements[447], man-machine physiological requirements[448] and machine-machine dialogue requirements[437]. Interface requirements constitute one requirements facet[285]. Other requirements facets are: business process reengineering[101], domain requirements[258] and machine requirements[438].*

4.3.1 But First: On Shared Phenomena and Concepts 234

Definition: Shared Phenomenon or Concept. *A shared phenomenon (or concept) is a phenomenon (respectively a concept) which is present in some domain[239] (say in the form of facts, knowledge[407] or information[373]) and which is also represented in the machine[436] (say in the form of some entity[272], simple, action, event or behaviour). A phenomenon of a domain, when shared, becomes a concept of the machine.* We shall give some examples – but they are just illustrative. Proper narration and formalisation is left to the reader !

4.3.2 Shared Simple Entities 235

Definition: Shared Simple Entity. *By a shared simple entity we mean a simple entitywhich both occurs in the domain[239] (as a phenomenon or a concept) and in themachine[436]. Simple entities that are shared between the domain and the machine must initially be input to the machine. Dynamically arising simple entities must likewise be inputand all such machine entities must have their attributes updated, when need arise. Requirements for shared simple entitiesthus entail requirements for their representation and for their human/machine and/or machine/machine transfer dialogue.*

Example 236 Main shared entities are those of hubs and links. Representations of hubs and links "within" the machine necessarily abstracts many of the properties of hubs and links; some (such) attributes may not be represented altogether.

As for human input, some man/machine dialogue based around a set of visual display unit screens with fields for the input of hub, respectively link attributes can then be devised. Etc.

4.3.3 Shared Actions 237

Definition: Shared Action. *By a shared action we mean an action that can only be partly computed by the machine[436]. That is, the machine[436], in order to complete an action, may have to inquire with the domain[239] (in order, say, to extract some measurable, time-varying simple entity attribute value) in order to proceed in its computation.*

Example 238 In order for a car **driver** to leave an **exit toll booth** the following component actions must take place: (a) the **driver** inserts the electronic pass into the exit toll booth; (b) the **exit toll booth** scans and accepts the ticket andcalculates the fee for the car journey from entry booth via the toll road net to the exit booth; (c) **exit toll booth** alerts the driver as to the cost and is requested to pay this amount; (d) once the **driver** has paid (e) the **exit booth toll** gate is raised. Actions (a,d) are **driver** actions, (b,c,e) are **machine** actions.

4.3.4 Shared Events 239

Definition: Shared Event. *By a shared event we mean an event whose occurrence in the domain[239] need be communicated to the machine[436] and, vice-versa, an event whose occurrence in the machine[436] need be communicated to the domain[239].*

Examples 240 The arrival of a car at a toll plaza entry booth is an event that must be communicated to the machine so that the entry booth may issue a proper pass (ticket). Similarly for the arrival of a car at a toll plaza exit booth is an event that must be communicated to the machine so that the machine may request the return of the pass and compute the fee. The end of that computation is an event that is communicated to the driver (in the domain) requesting that person to pay a certain fee after which the exit gate is opened.

4.3.5 Shared Behaviours 241

Definition: Shared Behaviour. *By a shared behaviour we mean a behaviour many of whose actions and events occur both in the domain[239] and in the machine[436] (in some encoded form, and in the same squence).*

Example 242 A typical toll road net use behaviour is as follows: Entry at some toll plaza: receipt of electronic ticket, placement of ticket in special ticket "pocket" in front window, the raising of the entry booth toll gate; drive up to [first] toll road hub (with electronic registration of time of occurrence), drive down a selected link (with electronic registration of time of occurrence of entry to and exit from link), then a repeated number of zero, one or more toll road hub and link visits – some of which may be "repeats" – ending with a drive down from a toll road hub to a toll plaza with the return of the electronic ticket, etc. – cf. Sect, 4.3.4.

4.4 Machine Requirements 243

Definition: Machine Requirements. *Machine requirements are those requirements[605] which, in principle, can be expressed without using professional domain terms (for which these requirements are established).*

Thus, by *machine*[436] *requirements*[605], we understand *requirements*[605] put specifically to, i.e., expected specifically from, the *machine*[436]. We normally analyse machine requirements into *performance requirements*[521], *dependability requirements*[218], *maintenance requirements*[443], *platform requirements*[527] and *documentation requirements*[238].

4.4.1 An Enumeration of Classes of Machine Requirements 244

We shall in these lecture notes not go into any detail about machine requirements. But we shall classify machine requirements into a long list of specific kinds of machine requirements.

- Performance
  - Storage
  - Time
  - Software Size
- Dependability
  - Accessability
  - Availability
  - Reliability
  - Robustness
  - Safety
  - Security
- Maintenance
  - Adaptive
  - Corrective
  - Perfective
  - Preventive
- Platforms
  - Development
  - Demonstration
  - Execution
  - Maintenance
- Documentation
- Other