Start of Lecture 6: REQUIREMENTS – from Extension "out"

4.2.4. Extension

Definition: Extension.

- *Domain extension is a domain requirements facet.*
- *It is an operation performed on a domain description or a requirements prescription.*
- *It effectively extends a domain description 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 facing page abstracts some of the extensions to nets: the plaza entry and exit booths.

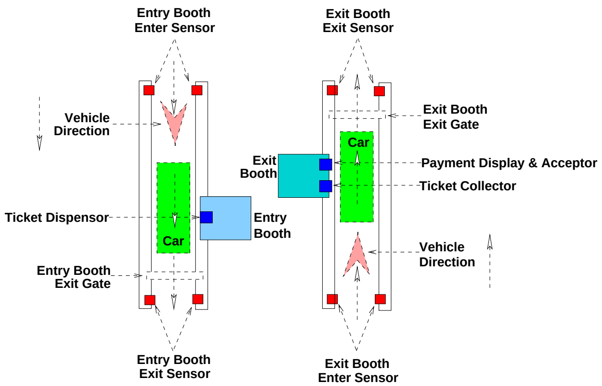

Figure 5: Entry and Exit Tool Booths

- The following is a prolonged example.
- It contains three kinds of formalisations:
  - a `RAISE/CSP` model,
  - a `Duration Calculus` model [zcc+mrh2002,olderogdirks2008] and
  - a `Timed Automata` model [AluDil:94,olderogdirks2008].
- The narrative for all three models are given when narrating the `RAISE/CSP` model.

4.2.4.1. Intuition

- 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.

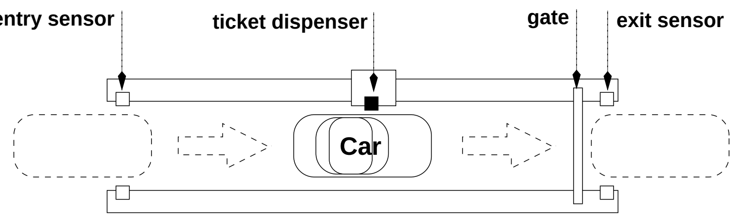

Figure 6: A toll plaza entry booth

4.2.4.2. Descriptions

4.2.4.2.1. • *A* RAISE/CSP *Model*•

We use the CSP property [TheSEBook123,CARH:Electronic] 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 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* ⊕

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* ⊕

- 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".

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* ⊕

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* ⊕

```
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
```



```
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
```



```
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
```



```
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 ~→ C
  pre add_Ticket(t,c): ∼has_Ticket(c)
  post: add_Ticket(t,c): has_Ticket(c)
```

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.

4.2.4.2.2. • *A* Duration Calculus *Model*•

- We use the Duration Calculus [zcc+mrh2002,olderogdirks2008] extension to RSL.
- We abstract the channels of the RAISE/CSP model
- to now be Boolean-valued variables.

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** ;

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).

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. $\Box(\lceil \text{gate=closing} \rceil \Rightarrow \lceil \sim \text{exit\_sensor} \rceil)$

4.2.4.2.3. • *A* `Timed Automata` *Model*•

- A timed automaton [AluDil:94,olderogdirks2008] for a configuration of an entry gate, its entry booth and a car is shown in Fig. 7 on the next page.
- Figure 8 on page 231 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 [[]Alur & Dill, 1994]AluDil:94 (Pages 42–45).
- The right half of the car timed automaton of Fig. 7 on the next page
  - is to be thought of as the same as the left half of the car timed automaton of Fig. 8 on page 231,
  - cf. the vertical dotted (⋮) line.

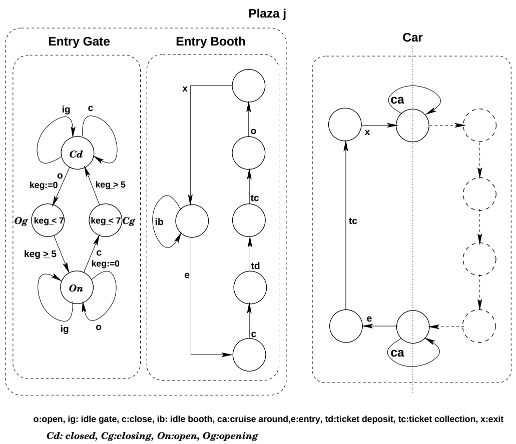

Figure 7: A `timed automata` model of gate, entry booth and car interactions

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)

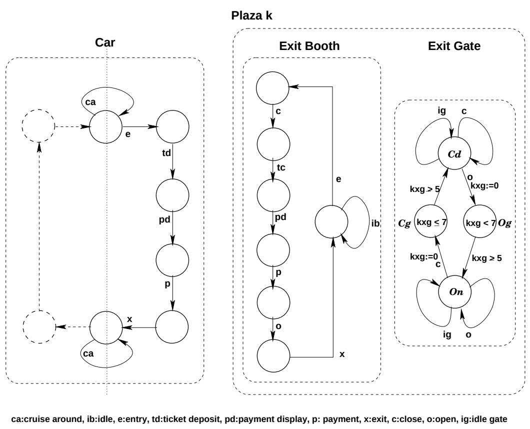

Figure 8: A `timed automata` model of car, exit booth and gate interactions

4.2.5. Fitting

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 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 $i : \{1..n\}$.*

4.2.5.1. Examples

TO BE WRITTEN

4.3. Interface Requirements

Definition: Interface Requirements.

- *Interface requirements are those* ***requirements***
- *which can on be expressed using professional terms*
- *from both the* ***domain*** *and the* ***machine****.*

Thus, by interface requirements we understand

- *the expression of expectations*
- *as to which software-software, or software-hardware* ***interface*** *places (i.e.,* ***channels****),*
- ***inputs*** *and* ***outputs*** *(including the* ***semiotics*** *of these input/outputs)*
- *there shall be in some contemplated* ***computing system****.*

Interface requirements can often, usefully, be classified in terms of

- *shared data initialisation requirements*,
- *shared data refreshment requirements*,
- *computational data+control requirements*,
- *man-machine dialogue requirements*,
- *man-machine physiological requirements* and
- *machine-machine dialogue requirements*.

Interface requirements constitute one requirements facet.

- *Other requirements facets are:*
  - *business process reengineering*,
  - *domain requirements* and
  - *machine requirements*.

4.3.1. But First: On Shared Phenomena and Concepts

Definition: Shared Phenomenon or Concept.

- *A shared phenomenon (or concept) is a phenomenon (respectively a concept)*
  - *which is present in some domain (say in the form of facts, knowledge or information)*
  - *and which is also represented in the machine (say in the form of some entity, 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

Definition: Shared Simple Entity.

- *By a shared simple entity we mean a simple entity*
  - *which both occurs*
  - *in the domain (as a phenomenon or a concept)*
  - *and in themachine.*
- *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 input*
  - *and all such machine entities*
  - *must have their attributes updated, when need arise.*
- *Requirements for shared simple entities*
  - *thus entail requirements for their representation*
  - *and for their human/machine and/or machine/machine transfer dialogue.*

4.3.2.1. Example

- 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

Definition: Shared Action.

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

4.3.3.1. Example

- 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 and
    - calculates 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

Definition: Shared Event.

- *By a shared event we mean*
  - *an event whose occurrence in the domain*
  - *need be communicated to the machine*

  and, vice-versa,
  - *an event whose occurrence in the machine*
  - *need be communicated to the domain.*

4.3.4.1. Examples

- 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

Definition: Shared Behaviour.

- *By a shared behaviour we mean a behaviour*
  - *many of whose actions and events occur both*
  - *in the* domain
  - *and in the* machine
  - *(in some encoded form, and in the same squence).*

4.3.5.1. Example

- 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.

4.4. Machine Requirements

Definition: Machine Requirements.

- *Machine requirements are those* requirements *which, in principle,*
  - *can be expressed without using professional domain terms*
  - *(for which these requirements are established).*
- Thus, by *machine requirements*,
  - we understand *requirements* put specifically to,
  - i.e., expected specifically from, the *machine*.
- We normally analyse machine requirements into
  - *performance requirements*,
  - *dependability requirements*,
  - *maintenance requirements*,
  - *platform requirements* and
  - *documentation requirements*.

4.4.1. An Enumeration of Classes of Machine Requirements

- 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

End of Lecture 6: REQUIREMENTS – from Extension “out”