Dines Bjørner's MAP-i Lecture # 10

Interface Requirements

Thursday, 28 May 2015: 12:15–13:00

7.3. Interface Requirements

- By an **interface requirements** we mean
  - a requirements prescription
    which refines and extends the domain requirements
  - by considering those requirements
    of the domain requirements whose
    - endurants (parts, materials) and
    - perdurants (actions, events and behaviours)
  - are **"shared"**
  - between the domain and the machine
    (being requirements prescribed) 

7.3.1. Shared Phenomena

- By **sharing** we mean
  - that an **endurant** is represented both
    - in the domain and
    - "inside" the machine, and
    - that its machine representation
    - must at suitable times
    - reflect its state in the domain;

    and/or
  - that an **action**
    - requires a sequence of several "on-line" interactions
    - between the machine (being requirements prescribed) and
    - the domain, usually a person or another machine;

    and/or

- that an **event**
  - arises either in the domain,
    that is, in the environment of the machine,
  - or in the machine,
  - and need be communicated to the machine, respectively to the environment;

  and/or
- that a **behaviour** is manifested both
  - by actions and events of the domain and
  - by actions and events of the machine ■

- So a systematic reading of the domain requirements shall
  - result in an identification of all shared
    - endurants,
      - parts,
      - materials and
      - components;

      and
    - perdurants
      - actions,
      - events and
      - behaviours.

- Each such shared phenomenon shall then be individually dealt with:
  - **endurant sharing** shall lead to interface requirements for data initialisation and refreshment;
  - **action sharing** shall lead to interface requirements for interactive dialogues between the machine and its environment;
  - **event sharing** shall lead to interface requirements for how such event are communicated between the environment of the machine and the machine; and
  - **behaviour sharing** shall lead to interface requirements for action and event dialogues between the machine and its environment.

• • •

- We shall now illustrate these domain interface requirements
- development steps with respect to our ongoing example.

7.3.2. Shared Endurants

- We "split" our interface requirements development into two separate steps:
  - the development of $d_{r_{\text{net}}}$
    - (the common domain requirements for the shared hubs and links),
  - and the co-development of $d_{r_{\text{db:i/f}}}$
    - (the common domain requirements for the interface between $d_{r_{\text{net}}}$ and DB_{rel} —
- under the assumption of an available relational database system DB_{rel})

Example 92 . Interface Requirements. Shared Endurants:

- The main shared endurants are
  - the net (hubs, links) and
  - the vehicles.
- As domain endurants hubs and links undergo changes,
  - all the time,
  - with respect to the values of several attributes:
    - *length, cadestral information, names,*
    - *wear and tear* (where-ever applicable),
    - *last/next scheduled maintenance* (where-ever applicable),
    - *state* and *state space*,
    - and many others.

- Similarly for vehicles:
  - their position,
  - velocity and acceleration, and
  - many other attributes.
- When planning the common domain requirements for the net, i.e., the hubs and links,
  - we enlarge our scope of requirements concerns beyond the two so far treated ($d_r$toll, $d_r$maint.)
  - in order to make sure that the shared relational database of nets, their hubs and links, may be useful beyond those requirements.

- We then come up with something like
  - hubs and links are to be represented as tuples of relations;
  - each net will be represented by a pair of relations
    - a hubs relation and a links relation;
    - each hub and each link may or will be represented by several tuples;
  - etcetera.
- In this database modeling effort it must be secured that "standard" operations on nets, hubs and links can be supported by the chosen relational database system DB_{rel}

7.3.2.1. Data Initialisation

- As part of $d_{r_{\text{net}}}$ one must prescribe data initialisation, that is provision for
  - an interactive user interface dialogue with a set of proper display screens,
    - one for establishing net, hub or link attributes names and their types, and, for example,
    - two for the input of hub and link attribute values.
  - Interaction prompts may be prescribed:
    - next input,
    - on-line vetting and
    - display of evolving net, etc.
  - These and many other aspects may therefore need prescriptions.
- Essentially these prescriptions concretise the insert and remove link and hub actions.

Example 93 . Interface Requirements. Shared Endurant Initialisation:

- The domain is that of the road net, n:N, say of Chapter 6 —
  see also Example 92 on Slide 475
- By 'shared road net initialisation'
  we mean the "ab initio" establishment, "from scratch"
  of a data base recording the properties of all links, l:L, and hubs, h:H,
  - their unique identifications, **uid**_L(l) and **uid**_H(h),
  - their mereologies, **obs_mereo**_L(l) and **obs_mereo**_H(h) , and
  - the initial values of all their attributes, **attributes**(l) and **attributes**(h).

223 There are r_l and r_h "recorders" recording link, respectively hub properties with each recorder having a unique identity,

224 Each recorder is charged with a set of links or a set of hubs according to some partitioning of all such.

225 The recorders inform a central data base, net_db, of their recordings:

a. (ri,nol,(u_j,m_j,$attrs_j$)) where
b. ri is the identity of the recorder,
c. nol is either `link` or `hub`,
d. u_j = **uid**_L(l) or **uid**_H(h) for some link or hub,
e. m_j = **obs_mereo**_L(l) or **obs_mereo**_H(h) for that link or hub and
f. $attrs_j$ = **attributes**(l) or **attributes**(h) for that link or hub.

type
223. RI
value
223. rl,rh:NAT **axiom** rl>0 ∧ rh>0
type
225a.. M = RI×"link"×LNK | RI×"hub"×HUB
225a.. LNK = LI × HI-**set** × LATTRS
225a.. HUB = HI × LI-**set** × HATTRS

```
value
224.  partitioning: L-set → Nat → (L-set)*
224.                | H-set → Nat → (H-set)*
224.  partitioning(s)(r) as sl
224.    post: len sl = r
224.        ∧ ∪ elems sl = s
224.        ∧ ∀ si,sj:(L-set|H-set) ·
224.            si≠{}
224.          ∧ sj≠{}
224.          ∧ {si,sj}⊆elems ss ⇒ si ∩ sj = {}
```

226 The $r_l + r_h$ recorder behaviours interact with the one net_db behaviour

channel
226. r_db: RI×(LNK|HUB)
value
226. LNK_recorder: RI → L-**set** → **out** r_db **Unit**
226. HUB−recorder: RI → H-**set** → **out** r_db **Unit**
226. net_db: **Unit** → **in** r_db **Unit**

227 The data base behaviour, net_db, offers to receive messages from the link an hub recorders.

228 And the data base behaviour, net_db, deposits these messages in respective variables.

229 Initially there is a net, $n : N$,

230 from which is observed its links and hubs.

231 These sets are partitioned into r_l, respectively r_h length lists of non-empty links and hubs.

232 The ab-initio data initialisation behaviour, ab_initio_data, is then the parallel composition of link recorder, hub recorder and data base behaviours with link and hub recorder being allotted appropriate link, respectively hub sets.

233 We construct, for technical reasons, as the listener will soon see, disjoint lists of link, respectively hub recorder identities.

value
227. net_db:
variable
228. lnk_db: (RI×LNK)-**set**
228. hub_db: (RI×HUB)-**set**
value
229. n:N
230. ls:L-**set** = obs_Ls(obs_LS(n))
230. hs:H-**set** = obs_Hs(obs_HS(n))
231. lsl:(L-**set**)* = partition(ls)(rl)
231. lhl:(H-**set**)* = partition(hs)(rh)
233. rill:RI* **axiom len** rill = rl = **card elems** rill
233. rihl:RI* **axiom len** rihl = rh = **card elems** rihl

```
232. ab_initio_data: Unit → Unit
232. ab_initio_data() ≡
232.     ‖ {lnk_rec(rill[ i ])(lsl[ i ])|i:Nat·1≤i≤rl}
232.     ‖ {hub_rec(rihl[ i ])(lhl[ i ])|i:Nat·1≤i≤rh}
232.     ‖ net_db()
```

234 The link and the hub recorders are near-identical behaviours.

a. They both revolve around an imperatively stated **for all ... do ... end**.
   The selected link (or hub) is inspected and the "data" for the data base is prepared from
b. the unique identifier,
c. the mereology, and
d. the attributes.
e. These "data" are sent, as a message, prefixed the senders identity, to the data base behaviour.
f. We presently leave the ... unexplained.

```
value
226.  link_rec: RI → L-set → Unit
234.  link_rec(ri,ls) ≡
234a..      for ∀ l:L·l ∈ ls do uid_L(l)
234b..          let lnk = (uid_L(l),
234c..                     obs_mereo_L(l),
234d..                     attributes(l)) in
234e..          rdb ! (ri,"link",lnk);
234f..          ... end
234a..      end
```

```
226.  hub_rec: RI × H-set → Unit
234.  hub_rec(ri,hs) ≡
234a..      for ∀ h:H·h ∈ hs do uid_H(h)
234b..          let hub = (uid_L(h),
234c..                     obs_mereo_H(h),
234d..                     attributes(h)) in
234e..          rdb ! (ri,"hub",hub);
234f..          ... end
234a..      end
```

235 The net_db data base behaviour revolves around a seemingly "never-ending" cyclic process.

236 Each cycle "starts" with acceptance of some,

237 either link or hub data.

238 If link data then it is deposited in the link data base,

239 if hub data then it is deposited in the hub data base.

```
value
235.  net_db() ≡
236.      let (ri,loh,data) = r_db ? in
237.      case loh of
238.          "link" → ... ; lnk_db := lnk_db ∪ (ri,data),
239.          "hub"  → ... ; hub_db := hub_db ∪ (ri,data)
237.      end end ;
235′.      ... ;
235.      net_db()
```

- The above model is an idealisation.
  - It assumes that the link and hub data represent a well-formed net.
  - Included in this well-formedness are the following issues:
    - (a) that all link or hub identifiers are communicated exactly once,
    - (b) that all mereologies refer to defined parts, and
    - (c) that all attribute values lie within an appropriate value range.
  - If we were to cope with possible recording errors then we could, for example, extend the model as follows:
    - (i) when a link or a hub recorder has completed its recording then it increments an initially zero counter (say at Item 234f., Slide 488);
    - (ii) before the net data base recycles it tests whether all recording sessions has ended and then proceeds to check the data base for well-formedness issues (a–b–c) (say at Item 235′, Slide 491) ■

- The above example illustrates the 'interface' phenomenon:
  - In the formulas, for example, we show both
    - manifest domain entities, viz., n, l, h etc., and
    - abstract (required) software objects, viz., $(ui, me, attrs)$.

7.3.2.2. Data Refreshment

- As part of $d_{r_{\text{net}}}$ one must also prescribe data refreshment:
  - an interactive user interface dialogue
    with a set of proper display screens
    - one for selecting the updating of net, of hub or of link attribute names and their types and, for example,
    - two for the respective update of hub and link attribute values.
  - Interaction-prompts may be prescribed:
    - next update,
    - on-line vetting and
    - display of revised net, etc.
  - These and many other aspects may therefore need prescriptions.
- These prescriptions also concretise insert and remove link and hub actions.

7.3.3. Shared Actions, Events and Behaviours

- We illustrate the ideas of
  - shared actions, events and behaviours
  - through the domain requirements extension
  - of Sect. 7.2.4,
  - more specifically Examples 87–89 Slides 442–449.

Example 94 . Interface Requirements. Shared Actions, Events and Behaviours:

This Example has yet to be written

Examples 88–90, Slides 445–453,
illustrate shared interactive actions, events and behaviours.

7.4. Machine Requirements
7.4.1. Delineation of Machine Requirements
7.4.1.1. On Machine Requirements

Definition 33 . **Machine Requirements:** *By* **machine requirements** *we shall understand*

- *such requirements*
- *which can be expressed "sôlely" using terms*
- *from, or of the machine* ■

Definition 34 . **The Machine:** *By the* **machine** *we shall understand*

- *the hardware*
- *and software*
- *to be built from the requirements* ■

- The expression
  - *which can be expressed*
  - *"sôlely" using terms*
  - *from, or of the machine*

  shall be understood with "a grain of salt".
  - Let us explain.
    - The **machine requirements** statements
    - may contain references to domain entities
    - but these are meant to be generic references,
    - that is, references to certain classes of entities in general.

  We shall illustrate this "genericitiy" in some of the examples below.

7.4.1.2. Machine Requirements Facets

- We shall, in particular, consider the following five kinds of machine requirements:
  - *performance requirements*,
  - *dependability requirements*,
  - *maintenance requirements*,
  - *platform requirements* and
  - *documentation requirements*.

7.4.2. Performance Requirements

Definition 35 . **Performance Requirements:** *By performance requirements we mean machine requirements that prescribe*

- *storage consumption,*
- *(execution, access, etc.) time consumption,*
- *as well as consumption of any other machine resource:*
  - *number of CPU units (incl. their quantitative characteristics such as cost, etc.),*
  - *number of printers, displays, etc., terminals (incl. their quantitative characteristics),*
  - *number of "other", ancillary software packages (incl. their quantitative characteristics),*
  - *of data communication bandwidth,*
  - *etcetera* ■

Example 95 . Machine Requirements. Road-pricing System Performance:

- Possible road pricing system performance requirements could evolve around:
  - maximum number of cars entering and leaving the sum total of all gates within a minimum period —
    for example 10.000 maximum within any interval of 10 seconds minimum;
  - maximum time between a car entering a gate and the raising of the gate barrier —
    for example 3 seconds;
  - etcetera,
- We cannot be more specific:
  - that would require more details about
  - gate sensors and
  - gate barriers.

7.4.3. Dependability Requirements

MORE TO COME

7.4.3.1. Failures, Errors and Faults

- To properly define the concept of *dependability* we need first introduce and define the concepts of
  - *failure*,
  - *error*, and
  - *fault*.

Definition 36 . Failure:

- *A machine failure occurs*
- *when the delivered service*
- *deviates from fulfilling the machine function,*
- *the latter being what the machine is aimed at* ■

Definition 37 . Error:

- *An error*
- *is that part of a machine state*
- *which is liable to lead to subsequent failure.*
- *An error affecting the service*
- *is an indication that a failure occurs or has occurred* ■

Definition 38 . Fault:

- *The adjudged (i.e., the 'so-judged') or hypothesised cause of an error*
- *is a fault* 
- The term hazard is here taken to mean the same as the term fault.
- One should read the phrase: "adjudged or hypothesised cause" carefully:
- In order to avoid an unending trace backward as to the cause,
- we stop at *the cause which is intended to be prevented or tolerated.*

Definition 39 . **Machine Service:** *The service delivered by a machine*

- *is its behaviour*
- *as it is perceptible by its user(s),*
- *where a user is a human, another machine or a(nother) system*
- *which interacts with it* ■

Definition 40 . Dependability: *Dependability is defined*

- *as the property of a machine*
- *such that* reliance can justifiably be placed on the service *it delivers* 
- We continue, less formally, by characterising the above defined concepts.
- "A given machine, operating in some particular environment (a wider system), may fail in the sense that some other machine (or system) makes, or could in principle have made, a *judgement* that the activity or inactivity of the given machine constitutes a *failure*".
- The concept of *dependability* can be simply defined as "the quality or the characteristic of being dependable", where the adjective 'dependable' is attributed to a machine whose failures are judged sufficiently rare or insignificant.

- *Impairments* to dependability are the unavoidably expectable circumstances causing or resulting from "undependability": faults, errors and failures.
- *Means* for dependability are the techniques enabling one
  - to provide the ability to deliver a service on which reliance can be placed,
  - and to reach confidence in this ability.
- *Attributes* of dependability enable
  - the properties which are expected from the system to be expressed,
  - and allow the machine quality resulting from the impairments and the means opposing them to be assessed.

- Having already discussed the "threats" aspect,
- we shall therefore discuss the "means" aspect of the *dependability tree*.

- Attributes:
  - Accessibility
  - Availability
  - Integrity
  - Reliability
  - Safety
  - Security
- Means:
  - Procurement
    - Fault prevention
    - Fault tolerance
  - Validation
    - Fault removal
    - Fault forecasting
- Threats:
  - Faults
  - Errors
  - Failures

- Despite all the principles, techniques and tools aimed at *fault prevention*,
- *faults* are created.
- Hence the need for *fault removal*.
- *Fault removal* is itself imperfect.
- Hence the need for *fault forecasting*.
- Our increasing dependence on computing systems in the end brings in the need for *fault tolerance*.

Definition 41 . Dependability Attribute: *By a dependability attribute we shall mean either one of the following:*

- *accessibility,*
- *availability,*
- *integrity,*
- *reliability,*
- *robustness,*
- *safety and*
- *security.*

That is, a machine is dependable if it satisfies some degree of "mixture" of being accessible, available, having integrity, and being reliable, safe and secure

- The crucial term above is "satisfies".
- The issue is: To what "degree"?
- As we shall see — in a later later lecture — to cope properly
  - with dependability requirements and
  - their resolution

  requires that we deploy
  - mathematical formulation techniques,
  - including analysis and simulation,

  from statistics (stochastics, etc.).

7.4.3.2. Accessibility

- Usually a desired, i.e., the required, computing system, i.e., the machine, will be used by many users — over "near-identical" time intervals.
- Their being granted access to computing time is usually specified, at an abstract level, as being determined by some internal nondeterministic choice, that is: essentially by *"tossing a coin"!*
- If such internal nondeterminism was carried over, into an implementation, some *"coin tossers"* might never get access to the machine.

Definition 42 . Accessibility: *A system being accessible — in the context of a machine being dependable —*

- *means that some form of "fairness"*
- *is achieved in guaranteeing users "equal" access*
- *to machine resources, notably computing time (and what derives from that)*

Example 96 . Machine Requirements. Road-pricing System Accessibility:

- Fairness of the calculator behaviour, cf. formula Item 220 on Slide 450 ⫿⫿
  - shall mean that "earlier" (wrt. time-stamped) messages
  - from either vehicles
  - or from gates
  - shall be accepted by the calculator
  - before "later" such messages.
- This is guaranteed by the semantics of RSL.
  - And, hence, shall be guaranteed
  - by any implementation of the deterministic choice⫿⫿

7.4.3.3. Availability

- Usually a desired, i.e., the required, computing system, i.e., the machine, will be used by many users — over "near-identical" time intervals.
- Once a user has been granted access to machine resources, usually computing time, that user's computation may effectively make the machine unavailable to other users —
- by "going on and on and on"!

Definition 43 . Availability: *By availability — in the context of a machine being dependable — we mean*

- *its readiness for usage.*
- *That is, that some form of "guaranteed percentage of computing time" per time interval (or percentage of some other computing resource consumption)*
- *is achieved — hence some form of "time slicing" is to be effected*

Example 97 . Machine Requirements. Road-pricing System Availability:

- Formula Item 216b. (Slide 445) specify that
  - vehicles "continuously" inform
  - the calculator (cf. formula Items 220 on Slide 450)
  - of their time-stamped local position.
- This may lead you to think that these messages
  - may effectively "block out"
  - "concurrent" messages from toll-road gates.
- In an implementation we may choose
  - to discretize vehicle-to-calculator messages.
  - That is, to "space them apart",
  - some time interval —
  - so long as an "intentional semantics is maintained"

7.4.3.4. Integrity

Definition 44 . Integrity: *A system has integrity — in the context of a machine being dependable — if*

- *it is and remains unimpaired,*
- *i.e., has no faults, errors and failures,*
- *and remains so, without these,*
- *even in the situations where the environment of the machine has faults, errors and failures* ■
- Integrity seems to be a highest form of dependability,
- i.e., a machine having integrity is 100% dependable!
- The machine is sound and is incorruptible.

Example 98 . Machine Requirements. Road-pricing System Integrity:

- We divide the integrity concerns for the road-pricing computing and communications system into two "spheres":
  - the integrity of the sensor and actuator equipment attached to
    - vehicles (i.e., their GNSS attributes), and to
    - toll-road gates:
      - in/out sensors,
      - vehicle identifiers and
      - gates;

    and
  - the software of the road-pricing computing and communications system,
    - that is, the software which interfaces with
      - vehicles,
      - toll-gates and
      - the calculator.

- As for the integrity of the the sensor and actuator equipment we do not require
  - that the road-pricing computing and communications system
  - is 100% dependable,
  - It is satisfactory if it retains its
    - accessibility,
    - availability,
    - reliability,
    - safety and
    - security

  in the presence of maintenance.

- As for the integrity of the software we require that it
  - is **proven correct**
    with respect to domain and requirements specifications
    under the assumption that
    sensor and actuator equipment functions
    with 100%'s integrity;
  - and where correctness proofs
    may not be feasible or possible,
    that the software is appropriately **model-checked**;
  - and where "complete" model-checks
    may not be feasible or possible,
    that the software is **formally tested**

Definition 45 . Reliability: *A system being reliable — in the context of a machine being dependable — means*

- *some measure of continuous correct service,*
- *that is, measure of time to failure*

Example 99 . Machine Requirements. Road-pricing System Reliability:

- *Mean-time between failures*, MTBF,
  - (i) of any vehicle's GNSS correct recording of local position must be at least 30.000 hours;
  - (ii) of any toll-gate complex, that is,
    - it's ability to correctly identify a passing vehicle, or
    - it's ability to correctly close and open gates

    must be at least 20.000 hours

7.4.3.5. Safety

Definition 46 . Safety: *By safety — in the context of a machine being dependable — we mean*

- *some measure of continuous delivery of service of*
  - ⟐ *either correct service, or incorrect service after benign failure,*
- *that is: Measure of time to catastrophic failure* ■

Example 100 . Machine Requirements. Road-pricing System Safety:

- *Mean time to catastrophic failure*, MTCF,
  - (i) for a vehicle's GNSS to function properly shall be 60.000 hours; and
  - (ii) of any toll-gate complex, that is,
    - it's ability to correctly identify a passing vehicle, or
    - it's ability to correctly close and open gates

    must be at least 40.000 hours

7.4.3.6. Security

We shall take a rather limited view of security. We are not including any consideration of security against brute-force terrorist attacks. We consider that an issue properly outside the realm of software engineering.

- Security, then, in our limited view, requires a notion of *authorised user*,
- with authorised users being fine-grained authorised to access only a well-defined subset of system resources (data, functions, etc.).
- An *unauthorised user* (for a resource) is anyone who is not authorised access to that resource.

Definition 47 . Security: *A system being secure — in the context of a machine being dependable —*

- *means that an unauthorised user, after believing that he or she has had access to a requested system resource:*
  - *cannot find out what the system resource is doing,*
  - *cannot find out how the system resource is working*
  - *and does not know that he/she does not know!*
- *That is, prevention of unauthorised access to computing and/or handling of information (i.e., data)* ■

Example 101 . Machine Requirements. Road-pricing System Security:

- Vehicles are authorised
  - ◈ to receive GNSS timed global positions,
    but not to tamper with, e.g. misrepresent them,

  are authorised
  - ◈ to, and shall correctly compute
    their local positions
    based on the received global positions,

  and are finally authorised
  - ◈ to, and shall correctly
    inform the calculator of their timed local positions

7.4.3.7. Robustness

Definition 48 . Robustness: *A system is robust — in the context of dependability —*

- *if it retains its attributes*
  - *after failure, and*
  - *after maintenance*

- Thus a robust system is "stable"
  - across failures
  - and "across" possibly intervening "repairs"
  - and "across" other forms of maintenance.

Example 102 . Machine Requirements. Road-pricing System Robustness:

- The road-pricing computing and communications system shall retain its
  - performance and
  - dependability, that is,
    - accessibility,
    - availability,
    - reliability, and
    - safety

    requirements
- in the presence of maintenance.

7.4.4. Maintenance Requirements

TO BE TYPED

7.4.4.1. Delineation and Facets of Maintenance Requirements

Definition 49 . **Maintenance Requirements:** *By maintenance requirements we understand a combination of requirements with respect to:*

- *adaptive maintenance,*
- *corrective maintenance,*
- *perfective maintenance,*
- *preventive maintenance and*
- *extensional maintenance* ■

- Maintenance of building, mechanical, electrotechnical and electronic artifacts — i.e., of artifacts based on the natural sciences — is based both on documents and on the presence of the physical artifacts.
- Maintenance of software is based just on software, that is, on all the documents (including tests) entailed by software — see Definition 61 on Slide 553.

7.4.4.2. Adaptive Maintenance

Definition 50 . **Adaptive Maintenance:** *By adaptive maintenance we understand such maintenance*

- *that changes a part of that software so as to also, or instead, fit to*
  - *some other software, or*
  - *some other hardware equipment*

  (i.e., other software or hardware which provides new, respectively replacement, functions) ■

Example 103 . Machine Requirements. Road-pricing System Adaptive Maintenance:

- Two forms of adaptive maintenance occur in connection with the road-pricing computing and communication system:
  - adaptive maintenance of vehicle and toll-gate sensors and actuators, and
  - adaptive maintenance of the "interfacing" software, that is,
    - the vehicle software as prescribed by Item 216 on Slide 445,
    - the toll-gate software as prescribed by Item 219 on Slide 448, and
    - the calculator software as prescribed by Item 220 on Slide 450.

- Adaptive maintenance of vehicle and toll-gate sensors and actuators occurs when
  - existing sensors or actuators
  - are replaced due to failure.
- Adaptive maintenance of interfacing software is required when
  - existing sensors or actuators have been replaced and their characteristics are different from those of the replaced equipment,
  - hence requires modifications of interfacing software

7.4.4.3. Corrective Maintenance

Definition 51 . **Corrective Maintenance:** *By corrective maintenance we understand such maintenance which*

- *corrects a software error* ■

Example 104 . Machine Requirements. Road-pricing System Corrective Maintenance:

- Corrective maintenance of the road-pricing computing and communications system is required in two "spheres":
  - when system, that is, toll-gate and vehicles sensors or actuators fail, and
  - when, despite all verification efforts, the interfacing, that is,
    - the vehicle,
    - the gate, or
    - the calculator

    software fails.

- In the former case (equipment failure)
  - the failing sensor or actuator is replaced
  - possibly implying adaptive maintenance.
- In the latter case (software failure)
  - the failing software is analysed
  - in order to locate the erroneous code,
  - whereupon that code is replaced by such code
  - that can lead to a verification of the full system

7.4.4.4. Perfective Maintenance

Definition 52 . **Perfective Maintenance:** *By perfective maintenance we understand such maintenance which*

- *helps improve (i.e., lower) the need for*
- *hardware storage, time and (hard) equipment*

Example 105 . Machine Requirements. Road-pricing System Perfective Maintenance:

- We focus on perfective maintenance of
  - vehicle,
  - toll-gate and
  - calculator

  software.

- We focus, in particular, on
  - the reaction time in connection with response to external stimuli for the gate software
    - the timed local position, Item 216a. on Slide 445, of vehicles;
    - the attr_enter_ch[gi] event from a toll-gate's in coming sensor, Item 219a. on Slide 448;
    - the timed vehicle identity for a attr_TIVI_ch[gi] event form a toll-gate sensor, Item 219b. on Slide 448; and
    - the attr_leave_ch[gi] event from a toll-gate's out going sensor, Item 219d. on Slide 448;

- the reaction time, of the calculator, Item 220 on Slide 450, to incoming, alternating, communications from
  - either vehicles, Item 220a. on Slide 450,
  - or gates, Item 220b. on Slide 450.
- and the calculation time of the calculator
  - for billing, cf. Item 222e. on Slide 452.

7.4.4.5. Preventive Maintenance

Definition 53 . **Preventive Maintenance:** *By preventive maintenance we understand such maintenance which*

- *helps detect, i.e., forestall, future occurrence*
- *of software or hardware failures* ■

Example 106 . **Machine Requirements. Road-pricing System Preventive Maintenance**:

TO BE WRITTEN

7.4.4.6. Extensional Maintenance

Definition 54 . **Extensional Maintenance:** *By extensional maintenance we understand such maintenance which adds new functionalities to the software, i.e., which implements additional requirements* ■

Example 107 . **Machine Requirements. Road-pricing System Extensional Maintenance**:

TO BE WRITTEN

7.4.5. Platform Requirements

TO BE WRITTEN

7.4.5.1. Delineation and Facets of Platform Requirements

Definition 55 . **Platform:** *By a [computing] platform is here understood*

- *a combination of hardware and systems software*
- *so equipped as to be able to develop and execute software,*
- *in one form or another* ■

- What the "in one form or another" is
- transpires from the next characterisation.

Definition 56 . **Platform Requirements:** *By platform requirements we mean a combination of the following:*

- *development platform requirements,*
- *execution platform requirements,*
- *maintenance platform requirements and*
- *demonstration platform requirements* ■

7.4.5.2. Development Platform

Definition 57 . Development Platform Requirements: *By development platform requirements we shall understand such machine requirements which*

- *detail the specific software and hardware*
- *for the platform on which the software*
- *is to be developed* ■

7.4.5.3. Execution Platform

Definition 58 . **Execution Platform Requirements:** *By execution platform requirements we shall understand such machine requirements which*

- *detail the specific (other) software and hardware*
- *for the platform on which the software*
- *is to be executed* ■

7.4.5.4. Maintenance Platform

Definition 59 . **Maintenance Platform Requirements:** *By maintenance platform requirements we shall understand such machine requirements which*

- *detail the specific (other) software and hardware*
- *for the platform on which the software*
- *is to be maintained* ■

7.4.5.5. Demonstration Platform

Definition 60 . **Demonstration Platform Requirements:** *By demonstration platform requirements we shall understand such machine requirements which*

- *detail the specific (other) software and hardware*
- *for the platform on which the software*
- *is to be demonstrated to the customer — say for acceptance tests, or for management demos, or for user training* ■

Example 108 . Machine Requirements. Road-pricing System Platform Requirements:

- The platform requirements are the following:
  - the **development platform** to be typed
  - the **execution platform** to be typed
  - the **maintenance platform** to be typed

    and
  - the **demonstration platform** to be typed.

7.4.6. Documentation Requirements

Definition 61 . **Software:** *By* **software** *we shall understand*

- *not only* **code** *that may be the basis for executions by a computer,*
- *but also its full* **development documentation**:
  - *the stages and steps of* **application domain description,**
  - *the stages and steps of* **requirements prescription,** *and*
  - *the stages and steps of* **software design** *prior to code,*

  with all of the above including all validation and verification (incl., test) documents.

- *In addition, as part of our wider concept of software, we also include a comprehensive collection of supporting documents:*
  - **training manuals,**
  - **installation manuals,**
  - **user manuals,**
  - **maintenance manuals,** *and*
  - **development and maintenance logbooks.** ■

Definition 62 . Documentation Requirements: *By documentation requirements*

- *we mean requirements*
- *of any of the software documents*
- *that together make up*
  - *software and*
  - *hardware*[30]

Example 109 . Machine Requirements — Documentation:

TO BE WRITTEN

[30]— we omit a definition of what we mean by hardware such as the one we gave for software, cf. Definition 61 on Slide 553.

7.4.7. Discussion

TO BE TYPED

Dines Bjørner's MAP-i Lecture # 10

End of MAP-i Lecture # 10:
Interface Requirements

Thursday, 28 May 2015: 12:15–13:00