Topic 52
Requirements Facets

- The **prerequisite** for following this (part of the) lecture is that you, as a requirements engineer, need to know: *what are the constituents of a proper model of requirements?*
- The **aims** are
  - ⋆ to introduce the concept that a proper requirements prescription is made up from most of the following constituent prescriptions, i.e., facets:
    - ◇ (i) domain,
    - ◇ (ii) interface and
    - ◇ (iii) machine requirements, and,

within each of these three groups of facets, of

- ◇ (i) projections, determinations, instantiations, extensions and fittings, respectively of
- ◇ (ii) shared data intialisation and refreshment, computational data and control, man-machine dialogues, man-machine physiological, and machine-machine dialogues, and of
- ◇ (iii) performance, dependability, maintenance, platform, and documentation requirements respectively;

and

- ⋆ to present principles, techniques and tools for the prescription of these facets.

- The **objective** is
  - ⋆ to ensure that you will become a thoroughly professional requirements engineer.
- The **treatment** is from systematic to formal.

Throughout requirements engineering remember to adhere to:

The “Golden Rule” of Requirements Engineering Prescribe only those requirements that can be objectively shown to hold for the designed software.

- “Objectively shown” means that the designed software can
  - ⋆ either be proved (verified),
  - ⋆ or be model checked,
  - ⋆ or be tested,
- to satisfy the requirements.

Recall also:

An “Ideal Rule” of Requirements Engineering When prescribing (incl. formalising) requirements, also formulate tests (theorems, properties for model checking) whose actualisation should show adherence to the requirements.

- The rule is labelled ideal
  - ⋆ since such precautions will not be shown in this volume.
  - ⋆ It ought be shown,
  - ⋆ but either we would show one, or a few instances, and they would “drown” in the mass of material otherwise presented.
  - ⋆ Or they would, we claim, trivially take up too much space.
- The rule is clear. It is a question of proper management to see that it is adhered to.

Introduction

- As is the case with the lectures on “Domain Facets”, these next lectures constitutes a second “high point” of the present volume.
- It is in this chapter that we present
  - ⋆ principles and techniques of requirements engineering
  - ⋆ which are not, today, otherwise available in any other textbook on software engineering.
- So take your time to become thoroughly familiar with the contents of the present chapter.

- The next lectures are structured as follows:
  - ⋆ First we rough-sketch, with little or no consideration of the carefully worked out domain descriptions, an initial set of (eureka) requirements — such as they may emerge from a more or less undigested requirements acquisitions process.
  - ⋆ On the basis of the rough (eureka requirements) sketch
    - ◇ we create a requirements terminology and
    - ◇ install the first terms into that terminology.

- ⋆ Then we decompose the further requirements development into the four major facets, which are then covered in the next lectures:
  - ◇ “Business Process Reengineering”,
  - ◇ “Domain Requirements”,
  - ◇ “Interface Requirements” and
  - ◇ “Machine Requirements”.
- ⋆ As an ongoing effort, during the requirements facets development stages,
  - ◇ we use and maintain,
  - ◇ that is, revise and install, additional terms into

  the terminology.

Topic 53
Rough Sketching and Terminology

- The aim of this part of the lecture
- is to remind the student
- that in order to come up with a proper model of requirements
- we must first have
  - ⋆ performed proper identification of and
  - ⋆ requirements acquisition from stakeholders.

- After such a requirements acquisition stage
- we can analyse the acquired requirements prescription units.
- And after such an analysis we are ready to
  - ⋆ rough sketch, i.e., to make a first attempt at constructing, some requirements document,
  - ⋆ while, at the same time, establishing a terminology document.
- In this lecture we shall overview these two aspects of “Requirements Engineering”.

Initial Requirements Modelling

- In Example 17.166
- we illustrated examples of
  - ⋆ “one”, or “two”, or “three liner”
  - ⋆ requirements description units.
- Once,
  - ⋆ as a result of requirements acquisition
- you have gathered what you may think of as a sufficient number of such analysed requirements description units,
- you are ready to rough sketch a requirements prescription.

Rough-Sketch Requirements

- A rough-sketch requirements prescription
- is (thus) based on a number of partially “digested”, i.e., partially analysed and conceptualised, requirements description units.
- The requirements engineer is encouraged to try to formulate
- a reasonably complete and consistent rough-sketch requirements prescription,
- in order to do a more thorough requirements analysis and concept formation.

- The requirements description units, in a sense, only express the stakeholders’ views on requirements.
- These units may reflect a somewhat incoherent “total view”.
- After a reasonably proper requirements analysis and concept formation stage,
- the requirements engineer (i.e., the analyst),
- is able to formulate a more coherent total view.
- Rough-sketching these requirements thus affords a first opportunity for the requirements engineer to express the requirements.

Example 19.170 *A Rough-Sketch Container Terminal Domain:* To illustrate a rough-sketch requirements we need first be able to refer to a domain description. In this case we present a rough-sketch domain description.

Entities

We itemize list entities of container harbours, in no particular order, only as they come to mind:

- **Container terminal:** A container terminal is a composite entity. It consists of a harbour basin of water, of one or more quays, of one or more container pools, and of zero, one or more container freight stations. The harbour water basin connects on one side to the open sea, and on the other side to one or more quays. Attributes of a container terminal are: its name, its maritime location (latitude and longitude), its number of quays, number of pools, etc.

- **Quay:** A quay is a composite entity. A quay is like a straight road: The quays connect on one side to the harbour basin, and on the other side, possibly via a container terminal internal road net, to one or more container pools, and, possibly via these, to the possible container freight stations. The quay also consists of one or more cranes. Quays have attributes: length, width, number of cranes, position within the container terminal, possibly a name, etc.

- **Container:** A container is a composite entity. It consists of (i) the container box (which has length [say 20 or 40 feet], height, width, owner, etc., attributes), (ii) its contents (which may be empty, and which we choose to abstract from, i.e., to not consider (in other words: disregard)), and (iii) its bill of lading. The latter has attributes such as: contents listing, which agent (i.e., merchant) is sending this container, which agent (merchant) is to receive the container, from where, via where, and to where, etc.

- **Bill of Lading (BoL):** A document which evidences a contract of carriage by sea. The document has the following functions:
  1. A receipt for goods, signed by a duly authorised person on behalf of the carriers.
  2. A document of title to the goods described therein.
  3. Evidence of the terms and conditions of carriage agreed upon between the two parties.

At the moment three different models are used:

1. A document for either combined transport or port-to-port shipments, depending on whether the relevant spaces for place of receipt and/or place of delivery are indicated on the face of the document.
2. A classic marine BoL in which the carrier is also responsible for the part of the transport actually performed by himself.
3. Sea waybill: A nonnegotiable document, which can only be made out to a named consignee. No surrender of the document by the consignee is required.

- **Container ship:** A container ship is a composite entity. It consists of one or more locations which can each hold, or which actually hold a container. So the container ship also consists of these containers. Container locations are called cells, and cells are laid out in bays, rows and tiers (like an x, y, z coordinate system). Thus containers are stacked. The container ship is further so arranged as to have these columns (i.e., stacks) of containers be accessible from the top, through what is called a hatchway, an opening, that can be covered by what is called a hatch cover. This hatch cover is removed when unloading and loading containers to the appropriate stacks that it covers. Ship attributes have to do with the exact arrangement of bays, rows and tiers, and thus as to how many containers the ship can, and at any moment, actually carry. Ships can berth at a quay. They then occupy a certain length of that quay.

- **Ship/quay crane:** A crane, either aboard the ship, or positioned at quayside, can lift (unload) containers out of hatchways and onto (a truck on) the quay, or, the other way around (load containers). Cranes have attributes: operating area (along a quay), possibly a unique name (identifier), carrying (lifting) weight, handling speed (capacity), etc. For any ship there is a maximum number of such cranes that can serve the ship at any one time.
- **Container truck:** A truck is a composite entity. It consists of a chassis and usually zero or one container. The chassis may be considered either composite or atomic. Whatever is chosen, the chassis enables the container truck to move. Container trucks have attributes: carrying (load) capacity, service speed, etc.

- **(Un)Loading plan:** A load plan for a container ship is a document which specifies the sequences of stacking and unstacking of containers, as that container ship calls on a succession of container harbours. Since containers can only be removed from or added onto the top of the cell position stacks on a container ship, the order in which these stacks are loaded and unloaded determines is crucial.

- **Pool:** A pool is a composite entity. It consists of one or more areas where a stack of containers may be, as well as the containers actually positioned there. Some pools can receive and (can thus) handle refrigerated containers (reefers). Stacks within a pool are usually ordered by row and column. Pools have attributes: location (name and position) within the container terminal, capacity (number and height of stacks), whether reefer or ordinary containers, etc.
- **Pool crane:** A pool crane, like a quay/ship crane, can move containers, one at a time, between container trucks/chassis and pool stacks.

Functions

- **Calling:** A container ship contacts, i.e., calls, a container terminal to advise it of its intended arrival, giving its call sign. The 'calling may, or may not imply a request for permission to go to a previously scheduled quay position.
- **Unloading movement:** This is a simple function and could be regarded as an atomic function. Often it is called a movement. The function concerns the unloading of a single container from a cell position aboard the container ship by a designated crane onto a container truck or a container chassis.
- **Loading movement:** See the above, since it is basically the reverse movement.

These two movements reflect the fact that container truck and container chassis can only move one container at a time.

- **Chassis/truck movement:** We also consider this a simple, atomic function: Moving, by motor driven vehicle, one container from a crane at a quay to a crane at a pool, or vice versa.
- **Hatch cover removal (opening):** An atomic function which opens up for the hatchway so that containers can be loaded or unloaded.
- **Hatch cover replacement (closure):** An atomic function which closes the hatchway.

Events

We rough-sketch some possible events:

- The arrival of a container vessel to a quay position
- The departure of a container vessel from a quay position
- The failure to remove (to open) a hatch cover
- The failure to replace (to close) a hatch cover
- The failure of a crane to grip a container
- The failure of a crane to release a container
- The failure of a container truck/chassis to move
- The failure of a container vessel to move
- The outbreak of an epidemic disease

Behaviours

- **A ship visit:** A normal, "uneventful" ship visit behaviour starts with the ship calling (action) and proceeds to the arrival of the container vessel at a quay position (event). Some hatch covers may be opened. It then continues with one or more concurrent sequences of container unloadings and loadings (actions). It ends (possibly) with the closing of hatch covers (actions) and the departure of the container vessel from the quay position.

- **A merchant freight truck visit:** A freight truck usually carries just one (say a 40-foot) container, or, in cases, two (20-foot) containers. A merchant freight truck is a freight truck which carries one merchant's container(s), overland, to or from a container terminal. Its visit is for three purposes: to deliver one or two containers, to fetch one or two containers, or both. Its behaviour wrt. the container terminal is: arrival (an event) at the container terminal, registration (a function) at the container terminal gate (statement of purpose, showing of papers (waybills, bill of loadings), etc.), unloading and/or loading of containers (either at a special area, called the container yard (or, in certain cases, at the container freight station), or directly to a pool area, or, even, directly on the quay, for immediate ship loading or unloading).
- **A 24-hour crane behaviour:** We encourage the course student to try complete this item as an exercise.
- **A 24-hour container truck/chassis behaviour:** We encourage the course student to try complete this item as an exercise.

■

- Now, on the background of the above rough domain sketch,
- we are ready to express a rough requirements sketch.

Example 19.171 *A Rough-Sketch Requirements for Container Stowage:* After some discussion with stakeholders we arrive at the following base requirements for a *ship and pool areas container loading plan* computing system. (What we here name ship and pool area container loading plans are, more colloquially, called stowage plans.)

1. **Container:** Every *container c* (that is to be involved in the planning of loading plans, and hence subject to actual loading and unloading) *shall possess* the following *attributes:* (i) length and (ii) BoL, *b*.

2. **Bill of Lading:** The BoL states the route which the container, *c*, is to take, or is taking or has taken. It is a requirement that the system *shall* establish and maintain BoLs for all relevant containers.
3. [**Ship sailing**] **route:** A route is here considered a sequence of two or more container terminal visits. A container terminal visit is a pair: the name of a container terminal (T) and the name of a container ship (s) or, for the last in such a sequence, say **nil.** The ship S takes the container C from container terminal t. Let $r : < (t_1, s_1), \ldots, (t_i, s_i), (t_{i+1}, s_{i+1}), \ldots, (t_n, \mathbf{nil}) >$ designate a route for some container. It expresses that that container is transported from container terminal t_i to container terminal t_{i+1} by container ship s_i. It is a requirement that the system *shall* establish and maintain ship sailing routes, for all of a ship owner's relevant container ships.

4. **Ship container stack layout ('context'):** For every relevant container ship (say, in the ship owner's fleet of such), full **information shall be maintained** of how each ship is laid out wrt. container stacks (this is called contextual information).
5. **Ship container stack 'state':** For every container ship being considered, we further require that *a state shall be maintained.* The state is information about the location of all current containers: where, aboard, i.e., in which stack and cell position, they are stored. A well-formedness about this state expresses that each container has a BoL which states that it should indeed be aboard that ship at the moment the state is recorded.

6. **Pool area container stack layouts ('context'):** For every relevant container terminal, and for every container pool area (that is relevant to the ship owner for which these requirements are to be developed, and within these container terminals), *information* about the topological layout and pool area stacks, whether for ordinary containers, or for reefers, whether for 20-foot or for 40-foot (etc.) containers, *shall be kept and regularly updated* to reflect any changes in pool area layouts, etc.

7. **Pool area container stack 'states':** For every pool area container stack being (thus) considered, we further require that *a state shall be maintained.* The state is information about all current containers being stored in that pool area stack and their location, that is, BoLs and where (i.e., bay, row, cell position), etc.

8. **Shipping orders:** There *shall* at any moment *be a latest set of shipping orders*. By shipping orders we understand a current set of outstanding orders for the shipping of containers.

   (a) **Pragmatics: Outstanding (container shipping) order.** By an outstanding (container shipping) order we mean an order for a container transport, i.e., an order whose transport is being requested, but for which no acknowledgement of its precise shipping has yet been given.

   (b) **Syntax: Outstanding (container shipping) order.** The order document specifies (i.e., restates) the BoL of the container and a sequence of one or more container terminals.

   (c) **Semantics: Outstanding (container shipping) order.** The meaning of an outstanding (container shipping) order is, if it is accepted, that it enters the allocation and scheduling process of the relevant shipowner(s), and thus, eventually, is confirmed.

9. **Confirmed (container) shipping order:** By a confirmed (container) shipping order we mean a shipping order which is no longer outstanding: Its syntax has been understood, and its semantics has been implemented. That is, it has been used in the construction of one or more ship container loading plans (and possibly also in one or more container pool area loading plans). Whether the container in question is actually en route is here left open.

10. **Ship container loading plans:** Based on the above forms of information, i.e., items 1–9, the required computing system *shall generate two kinds* of reasonably optimal ship container loading plans (i.e., *documents*): static and dynamic.

11. **Static ship container loading plan:** A static ship container loading plan is a plan that prescribes which containers are loading and unloading at which container terminals, for a given ship, i.e., for a given route that this ship is to follow, and for a given set of outstanding shipping orders. The plan also states where each container is to be located aboard the ship.

12. **Dynamic ship container loading plan:** Given a static ship container loading plan, and given a container terminal (i.e., the name of a terminal at which the ship for that loading plan is berthed), the dynamic ship container loading plan specifies the sequences in which containers are to be unloaded and loaded.

- As an example of the issues involved in loading and unloading, let us consider the following:
  - ⋆ Let container c_i be loaded on stack s in terminal t_i.
  - ⋆ Let container c_{i+1} be loaded on stack s in terminal t_i or t_{i+1} (i.e., immediately "on top of" c_i).
  - ⋆ Now container c_{i+1} can be unloaded from stack s in terminal t_{i+2}.
  - ⋆ Container c_i can be unloaded from stack s in terminal t_{i+2}, or some suitable later terminal.
- That is, a stack push and pop discipline must be adhered to.

13. **[Reasonably] optimal static ship container loading plan:** A static loading plan is said to be [reasonably] optimal if no other such plan can be found which "fills" all stacks of a ship to their (almost) fullest capacity while adhering to the stacking discipline.

14. **[Reasonably] optimal dynamic ship container loading plan:** A dynamic loading plan is said to be [reasonably] optimal if no other such plan can be found which generates the longest sequences of ship crane container movements with respect to the same ship stack.

15. **The generation of plans:** The intent of any dynamic ship container loading plan is that actual unloadings and loadings *shall* be commensurate with, i.e., "follow", that plan.

16. **Container pool area loading plan:** And so on; this plan will not be prescribed.

17. **Container ship loadings and unloadings:** By container ship loadings and unloadings we understand the sequences of ship crane positions, along the quay, next to, i.e., servicing, a given ship, as well as the movement, for each ship crane position, of containers to and from the ship (i.e., from and to the quay). Since translocating a ship crane (from one quay/ship position to another) takes time we wish to minimise the number of ship crane translocations.

Lest you have lost sight of what the rough-sketch requirements really were, we here summarise these:

2. Initialisation and refreshment of container BoLs

3. Initialisation and refreshment of ship sailing routes

4. Initialisation and refreshment of ship container stack layouts

5. Initialisation and refreshment of ship container stack states

6. Initialisation and refreshment of pool area container stack layouts

7. Initialisation and refreshment of pool area container stack states

8. Storage and reference to shipping orders, includes securing item 9

11. Generation of static ship container loading plan, securing item 13

12. Generation of dynamic ship container loading plan, securing item 14

16. Generation of container pool area loading plan (prescription omitted)

17. Minimise ship crane translations, securing item 15

We remind the course student that the above constitutes a set of rough-sketched requirements and that we likewise presented only rough-sketched descriptions of some aspects of the domain of container terminals in Example 19.170. ■

- So the above gave you some kind of rough-sketch example of what requirements may entail.
- The example was not that small.
- It had to be "semi-large".
- You have to see, with your "own eyes", that rough sketches are not small.
- In fact, they are much larger than the above example.
- Before we proceed to the main material of these current lectures on requirements facets,
- let us take a brief look at the interaction between
  - ⋆ rough-sketching and
  - ⋆ terminologisation.

Requirements Terminology

- We briefly covered, in a very early lecture, the topic of terminology.
- We do this to put that topic in a more proper context,
- that is, to hint at the size and complexity,
- of a realistic terminology.

Example 19.172 *An Incomplete Container Terminal Terminology:*

1. *Actual voyage number:* A code for identification purposes of the voyage and vessel which actually transports the container/cargo.
2. *Agency fee:* Fee payable by a ship owner or ship operator to a port agent.
3. *Agent:*

   (a) A person or organization authorised to act for or on behalf of another person or organization.

   (b) In P&O Nedlloyd, an Agent is a corporate body with which there is an agreement to perform particular functions on behalf of them for an agreed payment. An Agent is either a part of the P&O Nedlloyd organization or an independent body. The following functions and responsibilities may apply to the activities of an agent.

      i. *Sales:* Marketing, acquisition of cargo, issuing quotations, concluding contracts in coordination with P&O Nedlloyd. Basically the agent is the first point of entry into the P&O Nedlloyd organization for a shipper.
      ii. *Bookings:* Booking of cargo in accordance with allotments assigned to the agent for a certain voyage by P&O Nedlloyd.
      iii. *Customs:* Dealing with the national customs administration for cargo declarations, manifest alterations and cargo clearance on behalf of P&O Nedlloyd.
      iv. *Documentation:* Responsible for timeliness and correctness of all documentation required, regarding the carriage of cargo.
      v. *Handling:* Taking care of all procedures connected with physical handling of cargo.
      vi. *Equipment control:* Managing of all equipment stock in a particular area.
      vii. *Issuing:* Authorised to sign and issue Bills of Lading and other transport documents.
      viii. *Collecting:* Authorised to collect freight and charges on behalf of P&O Nedlloyd.
      ix. *Delivery:* The agent who releases the cargo and is responsible for its delivery to the consignee.
      x. *Handling of cargo claims:* Handling of cargo claims as per agency contract.
      xi. *Husbanding:* Handling non-cargo-related operations of a vessel as instructed by the master, owner or charterer.

4. *Area code:* A code for the area where a container is situated.
5. *Area off hire lease:* Geographical area where a leased container becomes off hire.
6. *Area off hire sublease:* Geographical area where a subleased container becomes off hire.
7. *Area on hire lease:* Geographical area where a leased container becomes on hire.
8. *Area on hire sublease:* Geographical area where a subleased container becomes on hire.
9. *Arrival date:* The date on which goods or a means of transport is due to arrive at the delivery site of the transport.
10. *Arrival notice:* A notice sent by a carrier to a nominated notify party advising of the arrival of a certain shipment or consignment.
11. *Auto container:* Container equipped for the transportation of vehicles.
12. *Automated guided vehicle system:* Unmanned vehicles equipped with automatic guidance equipment which follow a prescribed path, stopping at each necessary station for automatic or manual loading or unloading.
13. *Automatic identification:* A means of identifying an item, e.g., a product, parcel or transport unit, by a machine (device) entering the data automatically into a computer. The most widely used technology at present is barcode; others include radio frequency, magnetic strips and optical character recognition.

14. *BoL:* See Bill of Lading.
15. *Barcoding:* A method of encoding data for fast and accurate electronic readability. Barcodes are a series of alternating bars and spaces printed or stamped on products, labels, or other media, representing encoded information which can be read by electronic readers, used to facilitate timely and accurate input of data to a computer system. Barcodes represent letters and/or numbers and special characters like +, /, −, etc.
16. *Barge:* Flat-bottomed inland cargo vessel for canals and rivers with or without own propulsion for the purpose of transporting goods.
17. *Bay:* A vertical division of a vessel from stem to stern, used as a part of the indication of a stowage place for containers. The numbers run from stem to stern; odd numbers indicate a 20-foot position, even numbers indicate a 40-foot position.
18. *Bay plan:* A stowage plan which shows the locations of all the containers on the vessel.
19. *Berth:* A location in a port where a vessel can be moored, often indicated by a code or name.
20. *Bill of Lading:* Abbreviation: BoL. A document which evidences a contract of carriage by sea. The document has the following functions:

    (a) A receipt for goods, signed by a duly authorised person on behalf of the carriers.
    (b) A document of title to the goods described therein.
    (c) Evidence of the terms and conditions of carriage agreed upon between the two parties.

At the moment 3 different models are used:

(d) A document for either Combined Transport or Port-to-Port shipments depending on whether the relevant spaces for place of receipt and/or place of delivery are indicated on the face of the document.
(e) A classic marine Bill of Lading in which the carrier is also responsible for the part of the transport actually performed by himself.
(f) Sea Waybill: A non-negotiable document, which can only be made out to a named consignee. No surrender of the document by the consignee is required.

21. *Bill of Lading clause:* A particular article, stipulation or single proviso in a Bill of Lading. A clause can be standard and can be preprinted on the BoL.
22. *Bill of Material:* A list of all parts, subassemblies and raw materials that constitute a particular assembly, showing the quantity of each required item.
23. *Boat:* A small open-decked craft carried aboard ships for a specific purpose, e.g., lifeboat, workboat.
24. *Bonded:* The storage of certain goods under charge of customs viz. customs seal until the import duties are paid or until the goods are taken out of the country.

    (a) Bonded warehouse (place where goods can be placed under bond).
    (b) Bonded store (place on a vessel where goods are placed behind seal until the time that the vessel leaves the port or country again).
    (c) Bonded goods (dutiable goods upon which duties have not been paid, i.e., goods in transit or warehoused pending customs clearance).

25. *Box:* Colloquial name for container (e.g., Box-club).

26. *Bulk container:* A container designed for the carriage of free-flowing dry cargo, loaded through hatchways in the roof of the container and discharged through hatchways at one end of the container.
27. *Business process:* A business process is the action taken to respond to particular events, convert inputs into outputs, and produce particular results. Business processes are what the enterprise must do to conduct its business successfully.
28. *Business process model:* The business process model provides a breakdown (process decomposition) of all levels of business processes within the scope of a business area. It also shows process dynamics, lower-level process interrelationships. In summary it includes all diagrams related to a process definition, allowing for understanding what the business process is doing (and not how).
29. *Business process redesign (BPR):* The process of redesigning business practice models including the exchange of data and services amongst the stakeholders (i.e., finance, merchandising, production, distribution) involved in the life cycle of a client's product.
30. *Call:* The visit of a vessel to a port.
31. *Call sign:* A code published by the International Telecommunication Union in its annual List of Ships' Stations to be used for the information interchange between vessels, port authorities and other relevant participants in international trade. *Note:* The code structure is based on a three-digit designation series assigned by the ITU and one digit assigned by the country of registration. (PDHP = P&O Nedlloyd Rotterdam)

32. *Cargo:*
    (a) Goods transported or to be transported, all goods carried on a ship covered by a BoL.
    (b) Any goods, wares, merchandise, and articles of every kind whatsoever carried on a ship, other than mail, ship's stores, ship's spare parts, ship's equipment, stowage material, crew's effects and passengers' accompanied baggage.
    (c) Any property carried on an aircraft, other than mail, stores and accompanied or mishandled baggage. Also referred to as 'goods'.
33. *Carrier:* The party undertaking transport of goods from one point to another.
34. *Cell:* Location aboard a container vessel where one container can be stowed.
35. *Cell position:* The location of a cell aboard of a container vessel identified by a code for, successively, the bay, the row and the tier, indicating the position of a container on that vessel.
36. *Cellular vessel:* A vessel, specially designed and equipped for the carriage of containers.
37. *Consignee:* The party such as mentioned in the transport document by whom the goods, cargo or containers are to be received.
38. *Consignment:* A separate identifiable number of goods (available to be) transported from one consignor to one consignee via one or more than one modes of transport and specified in one single transport document.

39. *Consignment instructions:* Instructions from either the seller/consignor or the buyer/consignee to a freight forwarder, carrier or his agent, or other provider of a service, enabling the movement of goods and associated activities. The following functions can be covered:
    - Movement and handling of goods (shipping, forwarding and stowage).
    - Customs formalities.
    - Distribution of documents.
    - Allocation of documents (freight and charges for the connected operations).
    - Special instructions (insurance, dangerous goods, goods release, additional documents required).
40. *Container:* An item of equipment as defined by the International Organization for Standardization (ISO) for transport purposes. It must be:
    (a) a permanent character and accordingly strong enough to be suitable for repeated use;
    (b) specially designed to facilitate the carriage of goods, by one or more modes of transport, without intermediate reloading;
    (c) fitted with devices permitting its ready handling, particularly from one mode of transport to another;
    (d) so designed as to be easy to fill and empty;
    (e) having an internal volume of one cubic meter or more.
41. *Container chassis:* A vehicle specially built for the purpose of transporting a container so that, when container and chassis are assembled, the produced unit serves as a road trailer.
42. *CFS: Container freight station:* A facility at which (export) LCL (less than container load) cargo is received from merchants for loading (stuffing) into containers or at which (import) LCL cargo is unloaded (stripped) from containers and delivered to merchants.

43. *CLP: Container load plan:* A list of items loaded in a specific container and where appropriate, their sequence of loading.
44. *Container logistics:* The controlling and positioning of containers and other equipment.
45. *Container manifest:* The document specifying the contents of particular freight containers or other transport units, prepared by the party responsible for their loading into the container or unit.
46. *Container moves:* The number of actions performed by one container crane during a certain period.
47. *Container pool:* A certain stock of containers which is jointly used by several container carriers and/or leasing companies.
48. *Container ship:* A vessel, i.e., a floating structure designed for the transport of containers.
49. *Container stack:* Two or more containers, one placed above the other, forming a vertical column.
50. *Container terminal:* Place where loaded and/or empty containers are loaded or discharged into or from a means of transport.
51. *Container yard:* Abbreviation: CY. A facility at which FCL traffic and empty containers are received from or delivered to the Merchant by or on behalf of the Carrier.

52. *Fully cellular container ship:* Abbreviation: FCC. A vessel specially designed to carry containers, with cell-guides under deck and necessary fittings and equipment on deck.
53. *Full container load:* Abbreviation: FCL.
    (a) A container stuffed or stripped under risk and for account of the shipper and/or the consignee.
    (b) A general reference for identifying container loads of cargo loaded and/or discharged at merchants' premises.
54. *Grid number:* An indication of the position of a container in a bay plan by means of a combination of page number, column and line. The page number often represents the bay number.
55. *Hatch cover:* Watertight means of closing the hatchway of a vessel.
56. *Hatch way:* Opening in the deck of a vessel through which cargo is loaded into, or discharged from the hold and which is closed by means of a hatch cover.
57. *LCL:* Less than container load.
58. *Merchant:* For cargo carried under the terms and conditions of the Carrier's Bill of Lading and of a tariff, it means any trader or persons (e.g., Shipper, Consignee) and including anyone acting on their behalf, owning or entitled to possession of the goods.
59. *Reefer container:* A thermal container with refrigerating appliances (mechanical compressor unit, absorption unit, etc.) to control the temperature of cargo.
60. Etcetera!

■

- The "moral" of the above three examples is the composite of:
  - ⋆ a real domain description is long;
  - ⋆ a real requirements prescription is long; and
  - ⋆ a real terminology is long.
- In a textbook we can only hint at, but not illustrate, the real size of our descriptions, prescriptions and specifications.

Systematic Narration

- From the rough sketches of requirements to a
  - ⋆ properly expressed,
  - ⋆ consistent,
  - ⋆ relatively complete and
  - ⋆ well-structured

  requirements document,
- there is still a long way to go in order to
- cover all relevant aspects, here called facets,
- of the requirements.

- It is the purpose of the next lecture parts to overview
  - ⋆ proper structures,
  - ⋆ proper principles and
  - ⋆ proper prescription techniques,
- for attaining such well-designed requirements documents.

Topic 54
Business Process Reengineering Requirements

Characterisation 19.209 By *business process reengineering* we understand

- the reformulation of previously adopted business process descriptions,
- together with additional business process engineering work

. ■

- Business process reengineering (BPR) is about *change*,
- and hence BPR is also about *change management*.

Michael Hammer's Ideas on BPR

1. *Understand a method of reengineering before you do it for serious.*
2. *One can only reengineer processes.*
3. *Understanding the process is an essential first step in reengineering.*
4. *If you proceed to reengineer without the proper leadership, you are making a fatal mistake.*
5. *Reengineering requires radical, breakthrough ideas about process design.*

6. *Before implementing a process in the real world create a laboratory version in order to test whether your ideas work.*
7. *You must reengineer quickly.*
8. *You cannot reengineer a process in isolation. Everything must be on the table.*
9. *Reengineering needs its own style of implementation: fast, improvisational, and iterative.*
10. *Any successful reengineering effort must take into account the personal needs of the individuals it will affect.*

What Are *BPR Requirements?*

- Two "paths" lead to business process reengineering:
  - ⋆ A client wishes to improve enterprise operations by deploying new computing systems (i.e., new software).
    - ◇ In the course of formulating requirements for this new computing system
    - ◇ a need arises to also reengineer the human operations within and without the enterprise.

  - ⋆ An enterprise wishes to improve operations by redesigning the way staff operates within the enterprise and the way in which customers and staff operate across the enterprise-to-environment interface.
    - ◇ In the course of formulating reengineering directives
    - ◇ a need arises to also deploy new software, for which requirements therefore have to be enunciated.
- One way or the other, business process reengineering is an integral component in deploying new computing systems.

Overview of BPR Operations

- We suggest six domain-to-business process reengineering operations:

1. introduction of some new and removal of some old *intrinsics;*
2. introduction of some new and removal of some old *support technologies;*
3. introduction of some new and removal of some old *management and organisation substructures;*

4. introduction of some new and removal of some old *rules and regulations;*
5. introduction of some new and removal of some old work practices (relating to *human behaviours*); and
6. related *scripting.*

BPR and the Requirements Document

Requirements for New Business Processes

- The course student must be duly "warned":
  - ⋆ The BPR requirements are not for a computing system,
  - ⋆ but for the people who "surround" that (future) system.
  - ⋆ The BPR requirements state, unequivocally,
  - ⋆ how those people are to act,
  - ⋆ i.e., to use that system properly.
- Any implications, by the BPR requirements,
- as to concepts and facilities of the new computing system
- must be prescribed (also) in the domain and interface requirements.

Place in Narrative Document

- We shall thus, in the later part of this lecture, treat a number of BPR facets.
- Each of whatever you decide to focus on,
- in any one requirements development,
- must be prescribed.
- And the prescription must be put into the overall requirements prescription document.

- As the BPR requirements “rebuilds” the business process description part of the domain description[18],
- and as the BPR requirements are not directly requirements for the machine,
- we find that they (the BPR requirements texts) can be simply put in a separate section.

[18]— Even if that business process description part of the domain description is “empty” or nearly so!

- There are basically two ways of “rebuilding”
  - ⋆ the domain description’s business process’s description part (D_{BP})
  - ⋆ into the requirements prescription part’s BPR requirements (R_{BPR}).
  - ⋆ Either
    - ⋄ you keep all of D as a base part in R_{BPR},
    - ⋄ and then you follow that part (i.e., R_{BPR})
    - ⋄ with statements, R'_{BPR}, that express the new business process’s “differences”
    - ⋄ with respect to the “old” (D_{BP}).
    - ⋄ Call the result R_{BPR}.

- ⋆ Or
  - ⋄ you simply rewrite (in a sense, the whole of) D_{BP} directly into R_{BPR},
  - ⋄ copying all of D_{BP},
  - ⋄ and editing wherever necessary.

Place in Formalisation Document

- The above statements as how to express the “merging” of BPR requirements into the overall requirements document apply to the narrative as well as to the formalised prescriptions.
- We may assume that there is a formal domain description, $\mathcal{D}_{BP}$, (of business processes) from which we develop the formal prescription of the BPR requirements.

- We may then decide to
  - ⋆ either develop entirely new descriptions of the new business processes, i.e., actually prescriptions for the business reengineered processes, $\mathcal{R}_{BPR}$;
  - ⋆ or develop, from $\mathcal{D}_{BP}$, using a suitable schema calculus, such as the one in `RSL`, the requirements prescription $\mathcal{R}_{BPR}$, by suitable parameterisation, extension, hiding, etc., of the domain description $\mathcal{D}_{BP}$.

Intrinsics Review and Replacement

Characterisation 19.210 By *intrinsics review and replacement* we understand an evaluation

- as to whether current intrinsics stays or goes, and
- as to whether newer intrinsics need to be introduced

. ■

Example 19.173 *Intrinsics Replacement:* A railway net owner changes its business from owning, operating and maintaining railway nets (lines, stations and signals) to operating trains. Hence the more detailed state changing notions of rail units need no longer be part of that new company’s intrinsics while the notions of trains and passengers need be introduced as relevant intrinsics. ■

Replacement of intrinsics usually point to dramatic changes of the business and are usually not done in connection with subsequent and related software requirements development.

Support Technology Review and Replacement

Characterisation 19.211 By *support technology review and replacement* we understand an evaluation

- as to whether current support technology as used in the enterprise is adequate, and
- as to whether other (newer) support technology can better perform the desired services

. ■

Example 19.174 *Support Technology Review and Replacement:*

- Currently the main information flow of an enterprise is taken care of by printed paper, copying machines and physical distribution. All such documents, whether originals (masters), copies, or annotated versions of originals or copies, are subject to confidentiality.
- As part of a computerised system for handling the future information flow, it is specified, by some domain requirements, that document confidentiality is to be taken care of by encryption, public and private keys, and digital signatures.
- However, it is realised that there can be a need for taking physical, not just electronic, copies of documents.

- The following business process reengineering proposal is therefore considered:
  - ⋆ Specially made printing paper and printing and copying machines are to be procured, and so are printers and copiers whose use requires the insertion of special signature cards which, when used, check that the person printing or copying is the person identified on the card, and that that person may print the desired document.
  - ⋆ All copiers will refuse to copy such copied documents — hence the special paper.
  - ⋆ Such paper copies can thus be read at, but not carried outside the premises (of the printers and copiers).
  - ⋆ And such printers and copiers can register who printed, respectively who tried to copy, which documents.
  - ⋆ Thus people are now responsible for the security (whereabouts) of possible paper copies (not the required computing system).

- The above, somewhat construed example, shows the “division of labour” between the contemplated (required, desired) computing system (the “machine”) and the “business reengineered” persons authorised to print and possess confidential documents.
- It is implied in the above that the reengineered handling of documents would not be feasible without proper computing support.
- Thus there is a “spill-off” from the business reengineered world to the world of computing systems requirements.

■

Management and Organisation Reengineering

Characterisation 19.212 By *management and organisation reengineering* we understand an evaluation

- as to whether current management principles and organisation structures as used in the enterprise are adequate, and
- as to whether other management principles and organisation structures can better monitor and control the enterprise

. ■

Example 19.175 *Management and Organisation Reengineering:*

- A rather complete computerisation of the procurement practices of a company is being contemplated.
- Previously procurement was manifested in the following physically separate as well as designwise differently formatted paper documents: *requisition form, order form, purchase order, delivery inspection form, rejection and return form*, and *payment form*.
- The supplier had corresponding forms: *order acceptance and quotation form, delivery form, return acceptance form, invoice form, return verification form*, and *payment acceptance form*.
- The current concern is only the procurement forms, not the supplier forms.

- The proposed domain requirements are mandating
  - ⋆ that all procurer forms disappear in their paper version,
  - ⋆ that basically only one, the procurement document, represents all phases of procurement,
  - ⋆ and that order, rejection and return notification slips, and payment authorisation notes,
  - ⋆ be effected by electronically communicated and duly digitally signed messages that represent appropriate subparts of the one, now electronic procurement document.

- The business process reengineering part may now
  - ⋆ “short-circuit” previous staff’s review and
  - ⋆ acceptance/rejection of former forms,
- in favour of fewer staff interventions.
- The new business procedures, in this case, subsequently find their way into proper domain requirements: those that support, that is monitor and control all stages of the reengineered procurement process.

■

Rules and Regulations Reengineering

Characterisation 19.213 By *rules and regulations reengineering* we understand an evaluation

- as to whether current rules and regulations as used in the enterprise are adequate, and
- as to whether other rules and regulations can better guide and regulate the enterprise

. ■

- Here it should be remembered that rules and regulations principally stipulate business engineering processes.
- That is, they are — i.e., were — usually not computerised.

Example 19.176 *Rules and Regulations Reengineering:*

- Our example continues that of Example 11.144.
- Assume now, due to reengineered support technologies, that interlock signalling can be made magnitudes safer than before, without interlocking.
- Thence it makes sense to reengineer the rule of Example 11.144
  - ⋆ from: *In any three-minute interval at most one train may either arrive to or depart from a railway station*
  - ⋆ into: *In any 20-second interval at most two trains may either arrive to or depart from a railway station.*
- This reengineered rule is subsequently made into a domain requirements, namely that the software system for interlocking is bound by that rule.

■

Human Behaviour Reengineering

Characterisation 19.214 *Human Behaviour Reengineering:* By *human behaviour reengineering* we understand an evaluation

- as to whether current human behaviour as experienced in the enterprise is acceptable, and
- as to whether partially changed human behaviours are more suitable for the enterprise

. ■

Example 19.177 *Human Behaviour Reengineering:*

- A company has experienced certain lax attitudes among members of a certain category of staff.
- The progress of certain work procedures therefore is reengineered,
- implying that members of another category of staff are henceforth expected to follow up on the progress of “that” work.
- In a subsequent domain requirements stage the above reengineering
- leads to a number of requirements for computerised monitoring of the two groups of staff.

■

Script Reengineering

- On one hand, there is the engineering of the contents of rules and regulations,
- and, on another hand, there are
  - ⋆ the people (management, staff) who script these rules and regulations,
  - ⋆ and the way in which these rules and regulations are communicated to managers and staff concerned.

Characterisation 19.215 By *script reengineering* we understand evaluation

- as to whether the way in which rules and regulations are scripted and made known (i.e., posted) to stakeholders in and of the enterprise is adequate, and
- as to whether other ways of scripting and posting are more suitable for the enterprise

. ■

Example 19.178 *Script Reengineering:*

- We refer to Examples 11.147–11.149.
- In the lecture notes these two examples are carried through to their “business process reengineering end”, and thus
- they illustrated the description of a perceived bank script language. One that was used, for example, to explain to bank clients how demand/deposit and mortgage accounts, and hence loans, “worked”.

- With the given set of “schematised” and “user-friendly” script commands,
- such as they were identified in the referenced examples,
- only some banking transactions can be described.
- Some obvious ones cannot, for example,
  - ⋆ *merge two mortgage accounts,*
  - ⋆ *transfer money between accounts in two different banks,*
  - ⋆ *pay monthly and quarterly credit card bills,*
  - ⋆ *send and receive funds from stockbrokers,*
  - ⋆ etc.

- A reengineering is therefore called for, one that is really first to be done in the basic business processes of a bank offering these services to its customers.

■

Discussion: Business Process Reengineering

Who Should Do the Business Process Reengineering?

- It is not in our power, as software engineers,
- to make the kind of business process reengineering decisions implied above.
- Rather it is, perhaps, more the prerogative of appropriately educated, trained and skilled (i.e., gifted) other kinds of engineers or business people
- to make the kinds of decisions implied above.
- Once the BP reengineering has been made, it then behooves the client stakeholders to further decide whether the BP reengineering shall imply some requirements, or not.

- Once that last decision has been made in the affirmative, we, as software engineers, can then apply our abstraction and modelling skills, and,
- while collaborating with the former kinds of professionals,
- make the appropriate prescriptions for the BPR requirements.
- These will typically be in the form of domain requirements, which are covered extensively in later lectures.

General

- Business process reengineering is based on the premise
- that corporations must change their way of operating,
- and, hence, must "reinvent" themselves.
- Some corporations (enterprises, businesses, etc.) are
  - ⋆ "vertically" structured
    - ◇ along functions, products or geographical regions.
  - ⋆ Others are "horizontally" structured
    - ◇ along coherent business processes.
  - ⋆ In either case adjustments may need to be made as the business (i.e., products, sales, markets, etc.) changes.

Topic 55
Domain Requirements

Characterisation 19.216 By *domain requirements* we understand

- requirements which are expressed
- solely in terms of domain phenomena and concepts

. ■

- So in setting out, initially, acquiring (that is, eliciting or "extracting") requirements,
- the requirements engineer naturally starts "in" or "with" the domain.
- That is, the requirements engineer asks questions, of the stakeholders, that eventually should lead to the formulation of domain requirements.
- The structuring of these questions — it is strongly suggested — should follow
  - ⋆ the structuring and contents of the domain facets description of the domain model, and
  - ⋆ the five kinds of domain-to-requirements operations outlined next and treated in some depth in the following.

Domain-to-Requirements Operations

Characterisation 19.217 By a *domain-to-requirements operation* we shall understand

- a transformation of
  - ⋆ domain description documents into
  - ⋆ requirements description documents

. ■

- These document transformation operations are carried out by the requirements engineer.
- They follow as the result of the requirements engineer working closely with possibly alternating groups of stakeholders.

- We suggest the following five domain-to-requirements operations covered in depth in five parts of this lecture:

  1. domain projection
  2. domain determination
  3. domain instantiation
  4. domain extension
  5. domain fitting

Domain Requirements and the Requirements Document

- Some remarks need to be made
- before we go into details of domain requirements modelling techniques.

Requirements for Functionalities

- Domain requirements are about
  - ⋆ "operating" part of the domain "inside" the machine.
- Domain requirements engineering is about
  - ⋆ which parts to leave out,
  - ⋆ i.e., which parts to "emulate",
  - ⋆ and then in what "shape, forms and contents".

Place in Narrative Document

- In later in this lecture we shall treat a number of domain requirements facets.
  - ⋆ Each of whichever you decide to focus on,
  - ⋆ in any one requirements development,
  - ⋆ must be prescribed.

- The domain requirements all take their “departure point”,
- that is, are based upon, the entire domain description.
- That is, the domain requirements represent a kind of “rewrite”
- of the domain description.

- Whether this “rewrite” is done one way, or another way,
- for that we cannot really state any hard principles.
- It all depends, so much, on the subject domain and the subject requirements.
- There are basically two ways of doing the “rebuilding”
- of the domain description’s non-business process description part (D^{19})
- into the requirements prescription part’s domain requirements (R_{DR}),
- and that is as follows:

[19] Here D stands for the (i) intrinsics, the (ii) support technology, the (iii) management and organisation, the (iv) rules and regulations, the (v) script, and the (vi) human behaviour parts

- Either
  - ⋆ you keep all of D as a base part (R'_{DR}) in R_{DR},
  - ⋆ and then you follow that part (i.e., R'_{DR})
  - ⋆ with statements, R''_{DR}, that express the new business process’s “differences”
  - ⋆ with respect to the “old” (D).
  - ⋆ Call the result R_{DR}.

- Or
  - ⋆ you simply rewrite (in a sense, the whole of) D directly into R_{DR},
  - ⋆ copying all of D,
  - ⋆ and editing wherever necessary.

Place in Formalisation Document

- The above statements as how to express the “rewrite” of requirements into the overall requirements document applies, in particular, to narrative prescriptions.
- But as we shall see, it also applies to formal prescriptions.
- We may assume that there is a formal domain description, $\mathcal{D}$, from which we develop the formal prescription of the domain requirements.

- We may then decide to
  - ⋆ either develop entirely new descriptions of the new “domain”, i.e., actually prescriptions for the domain requirements, $\mathcal{R}_{DR}$;
  - ⋆ or develop, from $\mathcal{D}$, using a suitable schema calculus, such as the one in `RSL`, the requirements prescription, $\mathcal{R}_{DR}$, by suitable parameterisation, extension, hiding, etc., of the domain description $\mathcal{D}$.

A Domain Example

Example 19.179 *A Simple Domain Example: A Timetable System:*

- We choose a very simple domain:
- that of a traffic timetable, say flight timetable.
- In the domain you could, in “ye olde days”, hold such a timetable in your hand, you could browse it, you could look up a special flight, you could tear pages out of it, etc.
- There was no end as to what you could do to such a timetable.
- So we will just postulate a sort, `TT`, of timetables.

- Airline customers, clients, in general, only wish to inquire a timetable (so we will here omit treatment of more or less "malicious" or destructive acts).
- But you could still count the number of digits "7" in the timetable, and other such ridiculous things.
- So we postulate a broadest variety of inquiry functions, qu:QU, that apply to timetables, tt:TT, and yield values, val:VAL.

- Specifically designated airline staff may, however, in addition to what a client can do, update the timetable.
- But, recalling human behaviours, all we can ascertain for sure is that update functions, up:UP, apply to timetables and yield two things: another, replacement timetable, tt:TT, and a result, res:RES, such as: *"your update succeeded"*, or *"your update did not succeed"*, etc.
- In essence this is all we can say for sure about the domain of timetable creations and uses.

- We can view the domain of the
  - ⋆ timetable,
  - ⋆ clients and
  - ⋆ staff
- as a behaviour
  - ⋆ which nondeterministically alternates (⌈⌉) between
  - ⋆ the client querying the timetable client_0(tt),
  - ⋆ and the staff updating the same staff_0(tt).

```
scheme TI_TBL_0 =
 class
  type
    TT, VAL, RES
    QU = TT → VAL
    UP = TT → TT × RES
  value
    client_0: TT → VAL, client_0(tt) ≡ let q:QU in q(tt) end
    staff_0: TT → TT × RES, staff_0(tt) ≡ let u:UP in u(tt) end

    tim_tbl_0: TT → Unit
    tim_tbl_0(tt) ≡
         (let v = client_0(tt) in tim_tbl_0(tt) end)
       ⌈⌉ (let (tt′,r) = staff_0(tt) in tim_tbl_0(tt′) end)
 end
```

■

[19] The nondeterminism referred to is internal in the sense that no outside behaviour influences the choice.

Domain Projection

- Usually
  - ⋆ the *span* of the requirements is far "narrower"
  - ⋆ than the *scope* of the domain.
- That is,
  - ⋆ the conceived or actually described domain
  - ⋆ covers phenomena and concepts
  - ⋆ that will not be of concern
  - ⋆ when constructing requirements for some particular application.
- We shall therefore have to explicitly express a "projection".

Characterisation 19.218 By *domain projection* we understand an operation

- that applies to a domain description
- and yields a domain requirements prescription.
- The latter represents a projection of the former
- in which only those parts of the domain are present
- that shall be of interest in the ongoing requirements development

. ■

- In a sense, of course, the document resulting from a domain projection is still a domain description,
- but — for pragmatic reasons — we shall refer to it
- as a domain requirements prescription.

A Specific Example

Example 19.180 *Projection of Airline Timetable and Air Space:*

- We start out by formulating a *rough-sketch domain description* for the subdomain of airline timetables:
  - ⋆ There are airports, and one can fly between certain airports.
  - ⋆ There are airlines, and an airline offers flight services between such airports and at certain times.
  - ⋆ These services are recorded in an airline timetable. It lists
    - ⋄ for every flight offered its flight number and flight days,
    - ⋄ and a list of two or more airport visits: names of airports, and arrival and departure times.

- There is the air space. It consists
  - of airports,
  - of air corridors (zero, one or more between pairs of airports),
  - and of controlled areas around airports where the flight of aircraft is specially monitored (and partly controlled) by
  - air traffic control centres.

```
scheme AIR_TT_SPACE =
 extend TI_TBL_0 with
 class
  type
   AS, Airport, Air_Corridor, Controlled_Area, ATC
  ...
 end
```

- Now to a *rough-sketch domain projection prescription:*
  - From the above we leave out any description of the air space.
  - That is, we project “away”
    - air corridors,
    - controlled areas and
    - air traffic control centres.
  - We leave the details to the student.

```
scheme TI_TBL_1 = TI_TBL_0
```

We have taken the liberty, above, in AIR_TT_SPACE, not to model the details of timetables and the air space. ■

- You may rightfully claim that the above example was
- construed so as to fit the idea of projection.
- That may be so. But the idea has been demonstrated, has it not?

A General Example

- It is typical to have sorts in a domain description.
- Once these are projected onto the requirements they change
  - from being abstractions of phenomena
  - to being concepts of these.
  - The former are descriptions, informal or formal, of “things out there”, in the domain.
  - The latter are prescriptions, informal or formal, of “things in there”, in the software to be built!

- Whereas observer (and functions defined on the basis of observer) functions are just postulated,
- the projected observer (etc.) functions prescribe functions that must be implemented.
- To make that distinction clear we may choose to rename these functions.

Example 19.181 *From Domain Sorts to Requirements Sorts, I:*

- A transport net consists of segments and junctions
- such that every segment is connected to exactly two distinct junctions
- and such that to every junction there is connected one or more segments.
- Thus from a transport net one may observe its segments (e.g., street segments) and junctions (e.g., street intersections).
- To achieve a proper, consistent and complete net description we will, most likely, have introduced the concepts of segment and junction identifications — and related, via axioms, segments, junction and their identifiers.

```
type
 N, S, J, Si, Ji
value
 obs_Ss: N → S-set
 obs_Js: N → J-set
 obs_Si: S → Si
 obs_Ji: J → Ji
 obs_Jis: S → Ji-set
 obs_Sis: J → Si-set
```

```
axiom
  ∀ s:S • card obs_Jis(s)=2 ∧
  ∀ n:N, s,s′:S •
    {s,s′} ⊆ obs_Ss(n) ∧ s≠s′ ⇒ obs_Si(s)≠obs_Si(s′) ∧
    s ∈ obs_Ss(n) ⇒
      let {ji,ji′} = obs_Jis(s) in
      ∃ j,j′:J • {j,j′} ⊆ obs_Js(n) ∧ ji=obs_Ji(j) ∧ ji′=obs_Ji(j′) end ∧
  ∀ j:J • card obs_Sis(j)≥1 ∧
  ∀ n:N, j,j′:J •
    {j,j′} ⊆ obs_Js(n) ∧ j≠j′ ⇒ obs_Ji(j)≠obs_Ji(j′) ∧
    j ∈ obs_Js(n) ⇒
      let sis = obs_Sis(j) in
      ∀ si:Si • si ∈ sis ⇒ ∃ s:S • s ∈ obs_Ss(n) ∧ si=obs_Si(s) end
```

- We can annotate the above axioms, line by line:
  - ⋆ (1) Each segment is connected to exactly two distinct junctions.
  - ⋆ (3) Two segments of a net, if distinct, have distinct segment identifications.
  - ⋆ (4–6) For every segment of a net one can observe the identifications of two junctions — and these identifications must be those of junctions of the net.
  - ⋆ (7) Each junction is connected to one or more distinct segments.
  - ⋆ (9) Two junctions of a net, if distinct, have distinct junction identifications.
  - ⋆ (10–12) For every junction of a net one can observe the identifications of one or more segments — and these identifications must be those of segments of the net.

The annotation of the formalisation is really part also of the informal narrative description. ■

- Domain projection now considers which
  - ⋆ entities: sorts and values,
  - ⋆ axioms relating these,
  - ⋆ functions: observer functions, etc.,
  - ⋆ events and
  - ⋆ behaviours
- are to be represented, somehow, in the required software.

Example 19.182 *From Domain Sorts to Requirements Sorts, II:*

- We continue Example 19.181.
- In this example we may decide to project all that is described in Example 19.181.
- This means that nets, their segments and junctions shall be represented in the required software.
- This also means that segment and junction identifiers shall be represented in the required software.
- Whereas the nets, segments and junctions (i.e., their descriptions) were (models of) real phenomena in the domain,
- the net, segment and junction prescriptions are models of the required software.

- Observer functions become functions that must now be implemented.
- As such we may choose to rename them.
- Axioms are no longer axioms.
- They become invariants that must hold of any data structure representation of nets, segments and junctions.

```
type
  N, S, J, Si, Ji
value
  xtr_Ss: N → S-set
  xtr_Js: N → J-set
  xtr_Si: S → Si
  xtr_Ji: J → Ji
  xtr_Jis: S → Ji-set
  xtr_Sis: J → Si-set
```

```
wf_N: N → Bool
wf_N(n) ≡
  ∀ s:S•s ∈ xtr_Ss(n)⇒card xtr_Jis(s)=2 ∧
  ∀ s,s′:S •
    {s,s′}⊆xtr_Ss(n)∧s≠s′ ⇒ xtr_Si(s)≠xtr_Si(s′) ∧
    s ∈ xtr_Ss(n) ⇒
      let {ji,ji′}=xtr_Jis(s) in
      ∃ j,j′:J•{j,j′}⊆xtr_Js(n)∧ji=xtr_Ji(j)∧ji′=xtr_Ji(j′) end ∧
  ∀ j:J•j ∈ xtr_Js(n)⇒card xtr_Sis(j)≥1 ∧
  ∀ j,j′:J •
    {j,j′}⊆xtr_Js(n)∧j≠j′ ⇒ xtr_Ji(j)≠xtr_Ji(j′) ∧
    j ∈ xtr_Js(n) ⇒
    let sis=xtr_Sis(j) in
    ∀ si:Si•si ∈ sis⇒∃ s:S•s ∈ xtr_Ss(n)∧si=xtr_Si(s) end
```

■

From Concepts to Phenomena

- The projection of the domain description of Example 19.181
- onto the domain requirements prescription of Example 19.182
- reflects a subtlety:
  - ⋆ We may claim that the segment and junction identifications of Example 19.181 were mere concepts.
  - ⋆ There may not have been any physically recognisable phenomena amounting to these identifications
  - ⋆ *other than the — almost “law of nature” — fact that the mere manifestations of two distinct segments and two distinct junctions amount to the unique identifications of all such segments and junctions.*

- ⋆ There may thus not be any physically discoverable junction identifiers associated with segments (and segment identifiers associated with junctions).
- ⋆ But it is clear that from junctions one can identify connected segments, and from segments one can identify the "end" junctions.

- ⋆ Conceptual segment and junction identifiers of Example 19.182
- ⋆ now become eventually physically discoverable phenomena of the required software.
- ⋆ As such the segment and junction identifications of Example 19.182 are models of phenomena.

Domain Determination

- Often a domain exhibits *nondeterminism*, that is:
  - ⋆ A function result or a behaviour can either be such and such
  - ⋆ or it can be such and such (different from the first such and such),
  - ⋆ or it can be such and such (different from the first two such and suches!).
  - ⋆ Or a function result or a behaviour can be *loose* (i.e., loosely described):
    - ⋄ not all possible outcomes of a function application,
    - ⋄ or not all possible behaviours of a phenomenon

    may have been described, or even knowable.

- Sometimes, for a requirements, the stakeholders may wish to remove such seeming uncertainty — nondeterminism, or looseness — as to some function results or some behaviours.

Characterisation 19.219 By *domain determination* we understand an operation

- that applies to a (projected) domain description, i.e., a requirements prescription,
- and yields a domain requirements prescription,
- where the latter has made deterministic, or specific, some function results or some behaviours of the former

. ■

- Certainly the result of domain determination represents,
- not a domain description (any longer),
- but a requirements prescription.
  - ⋆ The point of requiring some software is to exactly
  - ⋆ make certain behaviours, certain function outcomes,
  - ⋆ determinate — predictable.

Example 19.183 *Determination of Airline Timetable Queries:* To exemplify this rough-sketch domain (to) requirements operation we first present a rough domain description, then the "more deterministic" domain requirements prescription.

- A rough-sketch timetable-querying domain description is:
  - ⋆ There is given a further undefined notion of timetables.
  - ⋆ There is also given a concept of querying a timetable.
  - ⋆ A timetable query, abstractly speaking, denotes (i.e., stands for) a function from timetables to results.
  - ⋆ Results are not further defined.

- A rough-sketch timetable querying domain requirements description is:
  - ⋆ There are given notions
    - ⋄ of departure and arrival times, and
    - ⋄ of airports, and
    - ⋄ of airline flight numbers.

```
scheme TI_TBL_2 =
  extend TI_TBL_1 with
    class
      type
        T, An, Fn
    end
```

- A timetable consists of a number of air flight journey entries.
  - ⋆ Each entry has a flight number,
  - ⋆ and a list of two or more airport visits.
    - ⋄ an airport visit consists of three parts: An airport name, and a pair of (gate) arrival and departure times.

```
scheme TI_TBL_3 =
  extend TI_TBL_2 with
    class
      type
        JR′ = (T × An × T)*
        JR = {| jr:JR′ · len jr ≥ 2 ∧ ... |}
        TT = Fn →m JR
    end
```

We illustrate just one, simple form of airline timetable queries.

- A simple airline timetable query
  - ⋆ either just browses all of an airline timetable,
  - ⋆ or inquires of the journey of a specific flight.

- The simple
  - ⋆ browse query thus need not provide specific argument data,
  - ⋆ whereas the flight journey query needs to provide a flight number.
- A simple
  - ⋆ update query inserts a new pairing of a flight number and a journey to the timetable,
  - ⋆ whereas a delete query need just provide the number of the flight to be deleted.

- The result of a query is a value:
  - ⋆ the specific journey inquired,
  - ⋆ or the entire timetable browsed.
- The result of an update is a possible timetable change
  - ⋆ and either an "OK" response if the update could be made,
  - ⋆ or a "Not OK" response if the update could not be made:
    - ⋄ Either the flight number of the journey to be inserted was already present in the timetable,
    - ⋄ or the flight number of the journey to be deleted was not present in the timetable.

First, we formalise the syntactic and the semantic types:

```
scheme TI_TBL_3Q =
  extend TI_TBL_3 with
    class
      type
        Query == mk_brow() | mk_jour(fn:Fn)
        Update == mk_inst(fn:Fn,jr:JR) | mk_delt(fn:Fn)
        VAL = TT
        RES == ok | not_ok
    end
```

Then we define the semantics of the query commands:

```
scheme TI_TBL_3U =
  extend TI_TBL_3 with
    class
      value
        𝓜_q: Query → QU
        𝓜_q(qu) ≡
          case qu of
            mk_brow() → λtt:TT·tt,
            mk_jour(fn)
              → λtt:TT · if fn ∈ dom tt
                          then [ fn↦tt(fn) ] else [ ] end
    end   end
```

```
scheme TI_TBL_3U =
  extend TI_TBL_3 with
    class
        𝓜_u: Update → UP
        𝓜_u(up) ≡
          case qu of
            mk_inst(fn,jr) → λtt:TT ·
              if fn ∈ dom tt
                then (tt,not_ok) else (tt ∪ [ fn↦jr ],ok) end,
            mk_delt(fn) → λtt:TT ·
              if fn ∈ dom tt
                then (tt \ {fn},ok) else (tt,not_ok) end
    end   end
```

- Before we had:

```
value
  tim_tbl_0: TT → Unit
  tim_tbl_0(tt) ≡
       (let v = client_0(tt) in tim_tbl_0(tt) end)
     ⌈⌉ (let (tt′,r) = staff_0(tt) in tim_tbl_0(tt′) end)
```

- Now we get:

```
value
  system: TT → Unit
  system() ≡
       (let q:Query in let v = 𝓜_q(q)(tt) in system(tt) end end)
     ⌈⌉ (let u:Update in let (r,tt′) = 𝓜_u(q)(tt) in system(tt′) end end)
```

Or, for use in Example 19.201:

system(tt) ≡ client(tt) $\sqcap$ staff(tt)

client: TT → **Unit**
client(tt) ≡
  let q:Query **in let** v = $\mathcal{M}_q$(q)(tt) **in** system(tt) **end end**

staff: TT → **Unit**
staff(tt) ≡
  let u:Update **in let** (r,tt′) = $\mathcal{M}_u$(q)(tt) **in** system(tt′) **end end**

■

Domain Instantiation

- Domain descriptions are usually “lifted” to cover several instances of domains:
  - ⋆ A railway system domain description may cover railways in several — or be claimed to cover them in “all” — countries!
  - ⋆ The similar situation holds true for a domain description of “the” financial service industry, “the” healthcare sector, etc.
- Usually software is being requested for specific instances of such application domains:
  - ⋆ the railway software of a specific region,
  - ⋆ the banking software for a specific bank,
  - ⋆ the hospital software for a specific region’s healthcare system,
  - ⋆ and so on.

Characterisation 19.220 By *domain instantiation* we understand an operation

- that applies to a (projected and possibly determined) domain description, i.e., a requirements prescription,
- and yields a domain requirements prescription,
- where the latter has been made more specific, usually by constraining a domain description

. ■

Example 19.184 *Instantiation: Local Region Railway Nets:*

- The domain description to be (rough-sketch) requirements instantiated is provided by the rough sketch of Example 11.133-
- The constraints are:
  - ⋆ There are exactly n stations (where n is given).
  - ⋆ The n stations have the following names: $s_1, s_2, \ldots, s_n$.
  - ⋆ These stations can be linearly ordered ($< s_1, s_2, \ldots, s_n >$) such that

- ⋆ if two stations are connected by a line, as are s_i, s_{i+1} for $i \in \{1..n-1\}$, then they are connected by exactly two lines, $l_{f_{i,i+1}}, l_{f_{i+1,i}}$,
- ⋆ one permitting traffic in one direction ($l_{f_{i,i+1}}$ from s_i to s_{i+1}), the other in the other direction ($l_{f_{i+1,i}}$ from s_{i+1} to s_i).
- ⋆ Each station has exactly one platform, with tracks on either side.
- ⋆ Both tracks can be reached from any line incident upon the station.
- ⋆ Any line emanating from the station can be reached from both station tracks.

■

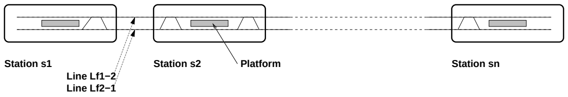

Figure 19.30: A schematic local region railway net

Domain Extension

- We make a distinction between
  - ⋆ genuine domain extensions
  - ⋆ and “domain extensions” due to “forgotten” domain facets.
- The distinction, as are the two kinds of extensions, are pragmatic notions.

Genuine Extensions

- Certain phenomena in a domain are conceivable “in theory”, but occur rarely in reality — like someone counting to a trillion!
- But with computing, computers can do your counting!
- So, although these phenomena,
  - ⋆ in a sense, “belong” to the domain,

  they are really only
  - ⋆ believably feasible when spoken of in connection with computing, hence requirements.

Characterisation 19.221 By *domain extension* we understand an operation

- that applies to a (projected and possibly determined and instantiated) domain description, i.e., a (domain) requirements prescription,
- and yields a (domain) requirements prescription.
- The latter prescribes that a software system is to support, partially or fully, an operation that is not only feasible but also computable in reasonable time

. ■

Example 19.185 *Extension: n-Transfer Travel Inquiry:* We assume a projected and instantiated timetable (see Example 19.183).

- A query of a timetable may, syntactically, specify an airport of origin, a_o, an airport of destination, a_d, and a maximum number, n, of intermediate stops.
- The query semantically designates the set of all those trips of one up to n direct air journeys between a_o and a_d, i.e., trips where the passenger may change flights (up to $n-1$ times) at intermediate airports.

```
scheme TI_TBL_3C =
  extend TI_TBL_3 with
    class
      type
        Query′ == Query | mk_conn(fa:An,ta:An,n:Nat)
        VAL′ = VAL | CNS
        CNS = (JR*)-set
      value
        $\mathcal{M}_q$(mk_conn(fa,ta,n)) ≡ ...
    end
```

■

- The point about this example is that for n being just 4 or above, a hand calculation is infeasible.
- But a `Prolog` program of less than a dozen lines, when the basis for executions, will start producing results after very few seconds on most PCs, for example for n=5.

"Forgotten" Domain Descriptions

- Sometimes one forgets to describe some domain facet.
- The discovery that one (might) have forgotten such a facet is usually made during domain requirements prescription.
- A stakeholder requirements is such that the domain requirements engineer lacks a "socket", some text and possibly formulas in the domain description which can serve as a basis for projection, instantiation, determination and extension.
- An example may serve to focus the idea.

Example 19.186 *A "Forgotten" Transport Net Domain Description:*

- We continue Examples 19.181–19.182.
  - ⋆ We have not equipped segments with attributes (such as lengths, geodetic (cadastral) coordinates, segment state of fitness (i.e., "need of repair"), or other).
  - ⋆ And we have therefore not described any functions that observe attributes, attribute values for given attributes, and, for example, those segments of a net which possess attributes (A) of specified values (VAL).

- We "discover" this general omission during the requirements gathering stage when stakeholders,
  - ⋆ for one set of requirements, express the requirement to offer travellers shortest routes in nets, or,
  - ⋆ for another set of requirements, express the requirement to maintain a high level of fitness of segments.

- So we extend the domain description of Example 19.181.
  - ⋆ With every segment we associate a finite, usually small number of attributes (that is, attribute names, $a : A$).
  - ⋆ And with every attribute we associate a set of attribute values $(v_1, v_2, \ldots : V)$.
  - ⋆ Thus we are able to observe which attributes are associated with a given segment,
  - ⋆ and, for that segment and an attribute of that segment, we are able to observe the associated attribute value.

- Now we can express the further extensions:
  - ⋆ Assume ordering relations, $\preceq_{a_i}$, one per attribute $a_i : A$, on attribute values.
  - ⋆ Now we shall require a function which, from a net, extracts all those segments which for a given attribute have attribute values within a given range.

```
type
  /∗ N, S, J, Si, and Ji as in Example 19.181 ∗/
  A, VAL
value
  obs_As: S → A-set
  obs_A_VAL: S × A ~→ VAL
    pre obs_A_VAL(s,a): a ∈ obs_As(s)

  ⪯_a: VAL × VAL → Bool

  is_in_range: S × (A × (VAL × VAL)) → Bool
  is_in_range(s,(a,(v,v′))) ≡
    v⪯_a obs_A_VAL(s,a)⪯_a v′
```

```
  extract_Ss: N × (A × (VAL × VAL)) → S-set
  extract_Ss(n,(a,(v,v′))) ≡
    {s|s:S•s ∈ obs_Ss(n)∧a ∈ obs_As(s)∧v⪯_a obs_A_VAL(s,a)⪯_a v′}
```

The reader can extend the above to also cover junctions. ■

- Once identified, "repairing" the description of a "forgotten" domain facet can either be thought of as a domain extension — and that is why we have placed the issue of "forgetfulness" in this section on domain extension — or it may prompt the requirements engineer to have the "original" domain description updated.
- To keep in line with our treatment of the omission, we decide to handle the "repair" in the extension part of our domain requirements engineering.
- Thus we have obviously decided to project the repaired domain facet onto the domain requirements prescription.
- This first part of the domain extension is then to be followed by possibly further domain to requirements operations.

Domain Requirements Fitting

- Often a domain being described
- "fits" onto, is "adjacent" to, "interacts" in some areas with,
- another domain:
  - ⋆ transportation with logistics,
  - ⋆ healthcare with insurance,
  - ⋆ banking with securities trading and/or insurance,
  - ⋆ and so on.

Characterisation 19.222 By *domain requirements fitting* we understand an operation

- that applies to two or more, say m, projected and possibly determined, instantiated and extended domain descriptions, i.e., to two or more, say m, original domain requirements prescriptions,
- and yields $m + n$ (resulting, revised original plus new, shared) domain requirements prescriptions.
- The m revised original domain requirements prescriptions resulting from the fitting prescribe most of the original (m) domain requirements.
- The n (new, shared) domain requirements prescriptions resulting from the fitting prescribe requirements that are shared between two or more of the m revised original domain requirements ■

Example 19.187 *Shared Domain Requirements:*

- Let the domain be that of multi-modal transportation nets:
  - ⋆ A multi-modal transportation net has segments and junctions.
  - ⋆ Segments and junctions are uniquely identified.
  - ⋆ Segments possess attributes: to which two junctions they are connected, length, standard traversal time, standard traversal cost, wear-and-tear (relevant for rail lines and roads), modality, and possibly other attributes.

  - ⋆ Junctions also possess attributes: to which one or more segments they are connected, standard traversal time, standard traversal cost (which is a function of the entry and exit segments: if of the same segment modality then maybe the cost is zero whereas if of different segment modalities then it reflects the cost of transfer (unloading and loading), and the set of one or more modalities of the connected segments.

- ⋆ One can speak of paths, from junction via a segment to a connected junction, and routes — as sequences of connected paths.
- ⋆ Hence one can speak of the longest route(s) and the shortest standard traversal time between two junctions.
- ⋆ One can also speak of best wear-and-tear quality route(s) also between two junctions.

- We outline two rough text original domain requirements.
  - ⋆ *A transportation net maintenance support system:*
    - ◇ The software package for this support system shall
    - ◇ help rail line maintenance planners to identify
    - ◇ segments (i.e., lines) in need of immediate repair (that is, corrective maintenance)
    - ◇ or scheduled preventive maintenance (that is inspection),
    - ◇ and, when such has been effected to record the (new) wear-and-tear status of maintained segments.
    - ◇ These requirements imply further determination of segment attributes.
    - ◇ Etcetera.

- ⋆ *A transportation net logistics support system:*
  - ◇ The software package for this support system shall
  - ◇ help combined road-rail travel planners
  - ◇ to identify combinations of one or more of
    - ○ shortest length route(s),
    - ○ shortest traversal time route(s),
    - ○ least costly route traversal(s), and/or
    - ○ route(s) with fewest transfers between transport modalities.
    - ○ Etcetera.

- The shared domain requirements are the following:
  - ⋆ Nets consisting of segments and junctions, thus also identification of segments and junctions;
  - ⋆ provision for segment attributes; and
  - ⋆ ability to select segments of a given modality.
- We leave it to the reader to formulate what is specific to the two revised original domain requirements.

■

Example 19.188 *Fitting of Passenger Transfers Between Busses and Trains:*

- We assume that there are two domain requirements prescriptions,
  - ⋆ one for metropolitan bus systems of bus lines, bus stops, etc., and
  - ⋆ one for railway systems of rail lines and stations.
- We further assume that
  - ⋆ one of the prescriptions has been in existence for some time —
  - ⋆ maybe even that an existing product is based on those requirements —
  - ⋆ and that the other prescription is currently being developed.

Rough sketches are as follows:

- The bus system consists of
  - ⋆ a set of bus lines, each being numbered and otherwise designated in a bus timetable,
  - ⋆ where this bus timetable, modulo “every” hour, for every bus line, specifies at which minutes (“past the hour”) the bus stops at each stop of the line.
- After this there follow a number of other entity, function and possibly behaviour descriptions.

```
scheme BUS =
  class
    type
      BSn, BLn, Min
      BTT′ = BLn →m (BSn × Min)*
      BTT = {| btt:BTT′ · wf_BTT(btt) |}
    value
      wf_BTT: BTT′ → Bool
      ...
  end
```

- The railway system consists of
  - ⋆ a set of train lines, each being numbered and otherwise designated in a train timetable,
  - ⋆ where this timetable, modulo “every” hour, for every train line specifies at which minutes (“past the hour”) the train stops at stations of the line.
- After this there follow a number of other entity, function and possibly behaviour descriptions.

```
**scheme** RAIL =
  **class**
    **type**
      Sn, RLn, Min
      RTT′ = RLn $\overrightarrow{m}$ (Sn × Min)*
      RTT = {| rtt:RTT′ • wf_RTT(rtt) |}
    **value**
      wf_RTT: RTT′ → **Bool**
      ...
  **end**
```

- Now the "fitting":
  - ⋆ Certain stations (bus stops) are to be designated as bus (train) transfer stations (bus stops).
  - ⋆ Passenger travel routes may include transfers at such stations (bus stops) between buses and trains.
- After this there follows a number of other entity, function and possibly behaviour prescriptions.

```
**scheme** BUS_RAIL =
  **extend** BUS **with extend** RAIL **with**
  **class**
    **type**
      Transfer′ = Bsn $\overrightarrow{m}$ Sn
      Transfer = {| tr:Transfer′ • **card dom** tr = **card rng** tr |}
    **value**
      ...
  **end**
```

End of Example 19.188 ■

Discussion: Domain Requirements

- We have outlined five reasonably distinguishable operations that the requirements engineer may need perform in order to construct a domain requirements prescription.
- There may be other such operations.
- The above five have been found useful in several development projects.
- Knowing about them, their underlying principles, and their techniques and tools should help the requirements engineer to more efficiently acquire domain requirements prescriptions, and to document them, i.e., to structure their documentation logically.

Topic 56

Interface Requirements

Characterisation 19.223 By *interface requirements* we understand

- those requirements that are expressed
- solely in terms of such phenomena and concepts
- that are *shared* between
  - ⋆ the domain and
  - ⋆ the machine.
    - ⋄ The machine is the hardware to be prescribed and
    - ⋄ the software to be developed

■

- The term 'shared' is crucial.
- For "something" to be *shared* between the domain and the machine,
- that "something" must be present in the domain.
  - ⋆ It must be en entity, a function, an event or a behaviour
  - ⋆ which has been projected, instantiated, possibly made more deterministic, possibly extended and possibly fitted.

- And that "something" must be present in the machine:
  - ⋆ Its attributes, including value, if an entity, must "somehow" be more or less regularly monitored by (read in from the domain, or set by, output from the) machine.
  - ⋆ Its functionality, if a function, must somehow replace that "present" in, or "co-opted", taken over from the domain,
  - ⋆ and its behaviour, if a behaviour, must somehow "simulate" the behaviour of the domain, or
  - ⋆ its occurrence, if an event, must somehow be replicated: If in the domain, then recorded by the machine, and if in the machine, then signaled to the domain.

- The "something" is said to be a *shared phenomenon cum concept*.
- We use the "somehow" hedge to indicate to the course student that the interface requirements shall stipulate, shall prescribe that 'somehow'!
- Shared phenomena cum concepts is what this section (Sect.) is all about!
- The shared "things" are usually phenomena in the domain, but always concepts in the machine. Domain concepts can also be shared.

Example 19.189 *Shared Phenomena:*

- We may think of a train traffic monitoring and control system being interface requirements developed.
- The following phenomena are identified as among those being shared:
  - ⋆ *rail units,*
  - ⋆ *signals,*
  - ⋆ *road level crossing gates,*
  - ⋆ *train sensors* (optical sensor sensing passing trains) and
  - ⋆ *trains.*

■

Example 19.190 *Shared Concepts:*

- We continue Example 19.189.
- The following train traffic concepts are among those being
- identified as being shared:
  - ⋆ *state of units,* including whether a unit is *open, closed, reserved, occupied,* etc.,
  - ⋆ *routes* (a route is, in general, not humanly visible (being often geographically widespread)), and hence *open routes.*

■

TWARE ENGINEERING: Domains, Requirements and Software Design Volume 3
1 Shared Phenomena and Concept Identification
e/db/volII/3ch19/3ch19-iii April 5, 2006, 13:05 Page 1713, Topic: 56, Foil: 7
Department of Computer Science and Engineering Institute of Informatics and Mathematical Modelling Technical University of Denmark Richard Petersens Plads, DK-2800 Kgs.Lyngby, Denmark DTU

Shared Phenomena and Concept Identification

- A crucial step of requirements development is therefore that of identifying,
- from among the many phenomena and concepts of the projected (etc.) domain
- which of these are shared.
- Examples 19.189 and 19.190 gave informal, rough-sketch examples.

OFTWARE ENGINEERING: Domains, Requirements and Software Design Volume 3
9.5.1 Shared Phenomena and Concept Identification
home/db/volII/3ch19/3ch19-iii April 5, 2006, 13:05 Page 1714, Topic: 56, Foil: 8
Department of Computer Science and Engineering Institute of Informatics and Mathematical Modelling Technical University of Denmark Richard Petersens Plads, DK-2800 Kgs.Lyngby, Denmark DTU

- Whether and how to categorise these shared phenomena and concepts
- is what the rest of this section on interface requirements is about.
- Suffice it to state that we here expect that the requirements engineers —
- in close collaboration with requirements stakeholders —
- list these shared “things”, and, along the road,
- while individually pursuing any one of the interface requirements facets, annotate this list with classifiers (whither one of the six interface requirements facets treated next, “where used”, etc.).

Interface Requirements Facets

- We shall consider six kinds of interface requirements:
  - ⋆ *shared data initialisation requirements,*
  - ⋆ *shared data refreshment requirements,*
  - ⋆ *computational data and control requirements,*
  - ⋆ *man-machine dialogue requirements,*
  - ⋆ *man-machine physiological interface requirements*, and
  - ⋆ *machine-machine dialogue requirements.*

- We foresee further identification of (i.e., other) interface requirements facets than the six so far listed.
- And we foresee an analysis, in the future, of some of the six listed facets into a more finely granulated set of (more or less) orthogonal interface requirements facets.
- Suffice it now, for the purposes of this part of these lectures, namely that of presenting basic principles and techniques of requirements engineering, to bring in just these six facets.

- The first three interface requirements facets motivate
- the need for the last three interface requirements facets.
  - ⋆ Shared data generally reside in the domain and in the machine.
  - ⋆ Computational data and control typically (but do not exclusively) reside in the human users who may interface with the machine during its computations, i.e., may interact with the machine.

- These first three interface requirements facets prescribe
  - ⋆ what information shall (need to) be shared,
  - ⋆ as well as some abstract principles according to which
    - ◇ the external domain information shall be communicated
    - ◇ into internal machine data
    - ◇ and vice versa.

- The dialogue requirements facets prescribe
  - ⋆ how that information concretely shall be communicated between
    - ◇ humans and/or
    - ◇ other machines (and equipment in general)
  - ⋆ and the machine being requirements prescribed.
- We now explain these six facets of interface requirements.
- But first we bring in a brief aside.

Interface Requirements and the Requirements Document

- Some remarks need to be made
- before we go into details of domain requirements modelling techniques.

Requirements for "Input/Output"

- Interface requirements are about:
  - ⋆ "putting" part of the domain "inside" the machine.
- Interface requirements engineering is about
  - ⋆ how to get parts of the domain into a machine (to become part of its state),
    - ◇ from the domain, or from other machines;
  - ⋆ and
    - ◇ how to reflect [new, computed] states back into the domain, or onto other machines.
- Thus interface requirements are about shared (usually entity) phenomena and concepts.

Place in Narrative Document

- Later in this lecture we shall treat a number of interface requirements facets.
  - ⋆ Each of whichever you decide to focus on,
  - ⋆ in any one requirements development,
  - ⋆ must be prescribed.

- The interface requirements all take their "departure point", that is are based upon,
  - ⋆ the entire domain description,
  - ⋆ as well as potentially available machine input/output technology.

- That is, the interface requirements represent a kind of
  - ⋆ "merging" of some form of the domain description,
  - ⋆ with descriptions of relevant, i.e., chosen, input/output technology.
- The two "merged" descriptions become a prescription, the interface requirements prescription.
- Since that "merge" was not present in the domain,
  - ⋆ the interface requirements prescription
  - ⋆ becomes an entirely new document part.

Place in Formalisation Document

- The above statements on how to express the interface requirements also apply to formal interface requirements prescriptions.
- We may assume that there is
  - ⋆ a formal domain description, $\mathcal{D}$ (from which we develop parts of the formal prescription of the interface requirements),
  - ⋆ and narrative descriptions of the input/output technologies.
  - ⋆ We further assume that there are formal descriptions, $\mathcal{D}_{IO}$, of these input/output technologies.

- We then develop an entirely new document,
- the interface requirements, $\mathcal{R}_{I/F}$.
- It somehow
  - ⋆ "merges" parts of $\mathcal{D}$
  - ⋆ with parts of $\mathcal{D}_{IO}$
- into the resulting $\mathcal{R}_{I/F}$.

This lecture on interface requirements is about the "merge" principles and techniques.

Shared Data Initialisation

- Information that is shared between the domain and the machine is often nontrivial in its structure and extent.
- Special care must be taken to introduce such information to the machine.

Characterisation 19.224 • By *shared data initialisation* we understand an operation that creates a *shared data structure* in the machine

. ■

- Thus a shared data initialisation requirements is an operation on requirements documents.
- It applies to a (projected and possibly determined, instantiated, extended and fitted) domain description, i.e., a domain requirements prescription,
- and yields an interface requirements prescription,
- where the latter prescribes that certain information of the domain
  - ⋆ is to be represented as a *shared data structure* in the machine, and
  - ⋆ generally how such data is initially to be set up by the machine.

Example 19.191 *Shared Data Initialisation of Railway Net:* We rough-sketch illustrate a case of shared data initialisation based on the rough sketch of Example 11.133 (Slide 1082).

- The software system shall start in an initial state
  - ⋆ which — rough-sketching —
  - ⋆ represents an empty rail net, and "ends" in a state which
  - ⋆ includes a representation of an "entire" rail net,
  - ⋆ i.e., a representation of all static and dynamic properties of each and every rail unit.

- In addition — as will be seen from other parts of these domain requirements[20] —
  - ⋆ it shall be possible to simply relate rail units to their physical surroundings:
    - ◇ whether the rail runs along a platform,
    - ◇ in a tunnel, up/down hill, is curved, etc.;
    - ◇ the pertinent electric train power line segment; etc.
  - ⋆ A special software subsystem shall handle the initial establishment of this start state as follows:
    - ◇ . . . , etc.

■

[20] This is not illustrated in these examples.

Shared Data Refreshment

- Shared data, once initialised, usually need be kept updated.
- The domain — usually — changes, irrespective of any computing system inserted into it.

Characterisation 19.225 By *shared data refreshment* we understand a machine operation

- which,
  - ⋆ at prescribed intervals,
  - ⋆ or in response to prescribed events,

  updates an (originally initialised) *shared data structure*

. ■

- Thus a shared data refreshment requirements is an operation on requirements documents.
- It applies to an interface requirements prescription,
- where the latter prescribes that certain information of the domain
  - ⋆ is to be represented as a *shared data structure* in the machine.
- The shared data refreshment requirements then prescribe how often, and by which means, that shared data structure is to be refreshed (i.e., updated).

Example 19.192 *Shared Data Refreshment of Railway Net:* We continue Example 19.191 by providing a rough sketch of a shared data refreshment requirements.

- Regular inspections of the wear and tear of the rail net units, signals, optical gates (and other sensors), road level crossings, etc., shall lead to similarly updating of that equipment's shared data structure,
- and such regular inspections shall be prompted by the machine and as prescribed by the required software.
- Inspections, with resulting updates, may take place before the usual expiry of inspection interval.
- And so on.

■

Computational Data and Control Interface Requirements

- For many applications it is the case that the flow of computations that may be desired by the users, i.e., the stakeholders,
- shall be influenced by interaction between the machine and these users.
- That is:
  - ⋆ It is often to be prescribed how such interaction shall take place,
    - ◇ whether by users interrupting the machine,
    - ◇ or the machine polling the users,
  - ⋆ and what it shall entail, i.e., which computational consequences the user interference shall have.
- It is this, perhaps "grey-zone" facet that we call the computational data and control interface.

Characterisation 19.226 By *computational data and control interface requirements* we understand requirements which prescribe

- that certain forms of input be provided over the user-machine interface,
- in order to help control the flow of computation:
  - ⋆ when to start or stop certain subcomputations, and/or
  - ⋆ with which argument data such subcomputations should be carried out,
  - ⋆ etc

. ■

Example 19.193 *Computational Data and Control Interface:* We continue Example 19.191.

- The railway net is represented, in the machine (database), by geographical area (i.e., area by area).
  - ⋆ Input of rail unit data is, in batches, by such areas.
  - ⋆ Hence a computational data input specifies that "until further notice" the next many future unit inputs are intended to "belong" to that area.
  - ⋆ Another computational data input (i.e., the "further notice") specifies "the end" of such a series of area-specific unit data.

- Occasionally, during unit data input, that and past input may need be checked ("vetted").
  - ⋆ Hence a computational data input may specify that such vetting is to be performed,[21]
  - ⋆ and other, immediately subsequent computational data input may be prompted as to the specific nature of the desired checks.
  - ⋆ Finally, prompts may inquire as to whether further checks need to be done, or the check series terminated.
  - ⋆ (We do not here specify the vetting procedures.)

■

Man-Machine Dialogue

Characterisation 19.227 By *man-machine dialogue requirements* we understand the prescription of the

- syntax (including sequential structure) and
- semantics
- of the communications (i.e., messages) transferred,
- in either direction, over the interface between man and machine,
- whether communicated textually through a keyboard (by the human) or on the screen (by machine), by a mouse or other tactile means (by human), or by voice (by human) or sound (by machine)

. ■

- It must be stressed that the man-machine dialogue referred to above subsumes the physiological interfaces mentioned next,
- but that it emphasises the sequencing of possibly alternative events and messages.
- Thus man-machine dialogue is "overall" wrt. the individual man-machine physiological events and messages.

Example 19.194 *Man-Machine Dialogue Requirements:* We continue Example 19.192.

- When, for any rail unit, its wear and tear information becomes older than six months,
- a message is to be displayed on the console (screen) of the railway net maintenance group responsible for that rail unit (this is an interface requirement).
- This group must respond within 72 hours with the requested update information (this is a business process reengineering requirement).

■

[21] We envisage that certain kinds of checks cannot be performed concurrently with the unit input.

Man-Machine Physiological Interface

- Humans can, “thanks” to a variety of technological “gadgets”, communicate with computers in various ways.
  - ⋆ Besides the conventional keyboard,
  - ⋆ they can also communicate by other tactile means:
    - ◇ the “mouse”;
    - ◇ “pointing with fingers at the screen”;
    - ◇ “pressing, with, for example, fingers”, fields of the screen;
    - ◇ etc.;
  - ⋆ possibly by voice, etc.
- These technological “gadgets” imply the man-machine physiological interface.
- Computers can likewise communicate with humans by means of graphics and sound.

Characterisation 19.228 By *man-machine physiological interface* we understand the possibly combined use of three forms of man-machine interfaces:

- *graphical (visual) user interface*,
- *audio (voice, sound) interface* and
- *tactile (keyboard, touch, “point”, button, etc.) interface*

. ■

Example 19.195 *Man-Machine Physiological Interface Requirements:* We continue Example 19.194.

- If no update of a rail unit’s wear and tear status has occurred within 72 hours of its visual display request,
- then a series of alarm bells shall sound (...) with one-hour intervals in designated offices of the railway groups responsible for recording this status, and,
- synchronised with this, bright red alarm lamps shall blink in line and station management offices.

■

- By a *graphical user interface* (GUI) we understand a
  - ⋆ visual display unit (VDU, e.g., a colour screen).
- Typically the VDU screen can be programmed to display various
  - ⋆ “windows”, icons, scroll-down “curtains”, etc.,
  - ⋆ with these being possibly labelled, and/or
  - ⋆ providing fields for text (keyboard) input.

Example 19.196 *Man-Machine Physiological Interface Requirements:*

- Assume that a database records the data
  - ⋆ which reflects the topology of some railway net,
  - ⋆ or that records the contents of a timetable.
- Also assume that some graphical user interface (GUI) windows
  - ⋆ represent the interface between man and machine
  - ⋆ such that items (fields) of the GUI are indeed “windows” into the underlying database.
- We prescribe and model, as an interface requirements, such GUIs and databases, the latter in terms of a relational, say a `SQL,` database:

type
Nm, Pos, Rn, An, Txt
$\mathrm{GUI} = \mathrm{Nm} \overrightarrow{m} (\mathrm{Item} \times \mathrm{Pos})$
$\mathrm{Item} = \mathrm{Txt} \times \mathrm{Imag}$
Imag = Icon | Curt | Tabl | Wind
Icon == mk_Icon(val:Val)
Curt == mk_Curt(vall:Val*)
Tabl == mk_Tabl(rn:Rn,tbl:TPL-**set**)
Wind == mk_Wind(gui:GUI)

Annotations:

- GUI Window Items
  - ⋆ A gui:GUI item, irrespective of the position, pos:Pos, of that item on the screen,
  - ⋆ maps distinct item names, Nm, into items, item:Item.
  - ⋆ An item has some “labeling” text, txt:Txt, and an image, imag:Imag.
  - ⋆ An image, imag:Imag, is either an icon, icon:Icon, a curtain, curt:Curt, a table, tabl:Tabl, or a window, wind:Wind.
  - ⋆ An icon has a value, mk_Icon(val:Val).
  - ⋆ A curtain consists of a list of values, mk_Curt(vall:Val*).
  - ⋆ A table, mk_Tabl(rn:Rn,tbl:TPL-**set**), names the relation, rn:Rn, from which the set tuples, tbl:TPL-**set**, of the table are queried.
  - ⋆ A window, mk_Wind(gui:GUI), is, hence recursively, a graphical user interface.

Val = VAL | REF | GUI
VAL = mk_Intg(i:Intg) | mk_Bool(b:**Bool**)
    | mk_Text(txt:**Text**) | mk_Char(c:**Char**)

Annotations:

- GUI Window Item Values
  - ⋆ A value (val:Val) is
    - ◇ either a proper value (in VAL),
    - ◇ or a reference (to a database entry),
    - ◇ or a graphical user interface (gui:GUI).
  - ⋆ A proper value (val:VAL) is
    - ◇ either an integer (mk_Intg(i:Intg)),
    - ◇ or a Boolean truth (mk_Bool(b:**Bool**)) value,
    - ◇ or a text string mk_Text(txt:**Text**) value,
    - ◇ or a character mk_Char(c:**Char**) value.

```
$\text{RDB} = \text{Rn} \xrightarrow[m]{} \text{TPL-}\mathbf{set}$
$\text{TPL} = \text{An} \xrightarrow[m]{} \text{VAL}$
$\text{REF} == \text{mk\_Ref(rn:Rn,an:An,sel:SEL)}$
$\text{SEL} = \text{An} \xrightarrow[m]{} \text{OptVal}$
$\text{OptVal} == \text{null} \mid \text{mk\_Val(val:VAL)}$
```

Annotations:

- The underlying Relational Database:
  - ⋆ A relational database (rdb:RDB) maps unique relation names (rn:Rn) into relations, and these are sets of tuples (tpls:TPL-**set**).
  - ⋆ A tuple (tpl:TPL) maps unique attribute names into proper values (val:VAL).
  - ⋆ A reference (is a proper value and) consists of a relation name, (rn:Rn), an attribute name (an:An) and a selection criterion (sel:SEL).
  - ⋆ A selection criterion ($\text{An} \xrightarrow[m]{} \text{OptVal}$) is a possibly empty map from attribute names into possibly optional, proper values.
  - ⋆ An optional value is either `nil`, or is a proper value (mk_Val(val:VAL)).

- Further on database references:
  - ⋆ Wherever, in a GUI, there is a reference, it is the value designated by that reference which is displayed.
    - ⋄ The reference relation name designates a relation in the database.
    - ⋄ The reference attribute name, `an`, designates an attribute of any tuple in the designated relation.
    - ⋄ If there is a tuple in the relation whose values equal those expressed in the selector, attribute by attribute,
      - ∘ then that tuple's value at `an` is the value displayed;
      - ∘ otherwise the optional (i.e., the so-called surrogate) value `null` is displayed.
  - ⋆ That is, the reference is a hidden quantity.

```
value
  $\text{de\_ref: REF} \times \text{RDB} \rightarrow \text{OptVAL}$
  $\text{de\_ref(mk\_Ref(rn,an,sel))(rdb)} \equiv$
    if $\exists\ \text{tpl:TPL} \cdot \text{tpl} \in \text{rdb(rn)} \wedge \text{tpl}/\mathbf{dom}\ \text{sel} = \text{sel}$
      then
        let $\text{tpl:TPL} \cdot \text{tpl} \in \text{rdb(rn)} \wedge \text{tpl}/\mathbf{dom}\ \text{sel} = \text{sel}$ in
        $\text{tpl(an)}$ end
      else null
    end
    pre $\text{rn} \in \mathbf{dom}\ \text{rdb} \wedge$
      $\exists\ \text{tpl:TPL}\cdot\text{tpl} \in \text{rdb(rn)} \wedge \mathbf{dom}\ \text{sel} \cup \{\text{an}\} \subseteq \mathbf{dom}\ \text{tpl}$
```

Annotations:

- Further on database references:
  - ⋆ To de_reference a database reference
  - ⋆ consisting of a relation name, rn,
  - ⋆ an attribute name, an, and
  - ⋆ a selection criterion, sel,
  - ⋆ is to inquire whether there exists a tuple, tpl,
  - ⋆ in the name relation, rdb(rn),
  - ⋆ for which the selection criterion applies: tpl/**dom** sel = sel.
  - ⋆ If such a tuple is found, then it is the result of the dereferencing;
  - ⋆ if not, then the `null` value is yielded.

■

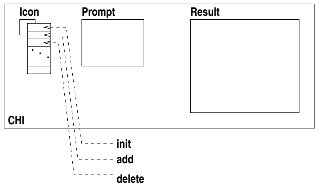

Figure 19.31: An example CHI: staff clicking icon

Example 19.197 *Man-Machine Physiological Interface Requirements:*

- We omit naming the only three items:
  - ⋆ the scroll-down curtain which displays (i.e., lists) the client and staff commands — as well as the no command (`nil`);
  - ⋆ a prompt field
    - ⋄ which initially is blank, i.e., nil,
    - ⋄ but which
    - ⋄ — depending on the clicked command name of the scroll-down curtain —
    - ⋄ lists the command field names for desired values,
    - ⋄ and for which the user (client or staff) is to provide appropriate text values;
  - ⋆ finally, a result field.

```
type
  $\text{GUI} = \text{Curt} \times \text{Prompt} \times \text{Result}$
  Curt == browse | display | connection | init | add | delete | nil
  Prompt = Query | Update | Conn | nil
  Result = RES
```

- On a Specific GUI:
  - ⋆ The graphical user interface, gui:GUI, consists of three items:
    - ⋄ a scroll-down curtain, curt:Curt,
    - ⋄ a prompt field, prompt:Prompt,
    - ⋄ and a result field, result:Result.
  - ⋆ A scroll-down curtain in the concrete lists exactly the available query and update commands possible on a timetable.
  - ⋆ These are designated by the keywords: browse, display, connection, init, add and delete.

- ⋆ At most one of these keywords can be selected, i.e., is therefore highlighted. Thus the above model defines a curtain to be just one of these, or, when none is selected, the nil option.
- ⋆ The prompt field, prompt:Prompt, is to contain an appropriate query/update command, as "selected" by the curtain highlight, or nil.
- ⋆ The result field, res:Result, will contain a result value.

- In Example 19.183 we defined the semantics of query and update commands.
- We now use these definitions to define the requirements, namely that these commands
  - ⋆ obtain their arguments, and, when subject to execution,
  - ⋆ deliver (deposit) their result into the user interface,
- that is, as part of the GUI.

```
value
  client: GUI → TT → GUI
  client(,,)(tt) ≡
    let icon = browse ⌈⌉ display ⌈⌉ connection in
    case icon of:
      browse → (browse, mk_Brws(),$\mathcal{M}_q$(mk_Brws())(tt)),
      display
        → let fn:Fn · fn ∈ dom tt ∨ ... in
          (display,mk_Disp(fn),$\mathcal{M}_q$(mk_Disp(fn))(tt)) end,
      connection
        → let ℓ:Nat,da,ta:An·{da,ta}⊆Ans(tt) ∧ ... in
          (connection,mk_Conn(ℓ,da,ta),$\mathcal{M}_q$(mk_Conn(ℓ,da,ta))(tt)) end
    end end
```

- Specific GUI: Timetable

  A client, by his own decision, either issues a browse, or a display, or a connection query.

  - ⋆ If browse then
    - ◇ it means that the curtain alternative browse has been "clicked", and is hence highlighted,
    - ◇ that the prompt field shows an obvious mk_Brws() command, requiring no arguments,
    - ◇ and the result field shows the result, $\mathcal{M}_q$(mk_Brws())(tt), of interpreting that command on the timetable.

- ⋆ If display then
  - ◇ it means that the curtain alternative display has been "clicked", and is hence highlighted,
  - ◇ that a flight number is provided by the client, here shown as nondeterministically selected,
  - ◇ that the prompt field shows the corresponding display command, mk_disp(fn),
  - ◇ and the result field shows the result, $\mathcal{M}_q$(mk_Disp(fn))(tt), of interpreting that command on the timetable.

- ⋆ If connection then
  - ◇ it means that the curtain alternative display has been "clicked", and is hence highlighted,
  - ◇ that the maximum number of flight changes, ℓ, and departure da and destination ta airports are provided by the client, here shown as nondeterministically selected,
  - ◇ that the prompt field shows the corresponding connection command mk_Conn(ℓ,da,ta),
  - ◇ and the result field shows the result of interpreting that command on the timetable, $\mathcal{M}_q$(mk_Conn(ℓ,da,ta))(tt).

```
value
  staff: GUI → TT → GUI × TT
  staff(,,)(tt) ≡
    let icon =  init ⌈⌉ add ⌈⌉ delete ⌈⌉ ... in
    case icon of:
      init → let (r,tt′) = $\mathcal{M}_u$(mk_init())(tt) in ((init,tt′,r),tt′) end,
      add → let fn:Fn,j:Journey · fn ∉ dom tt ∨ ... in
          let (r,tt′) = $\mathcal{M}_u$(mk_add(fn,j))(tt) in
          ((add,mk_add(fn,j),r),tt′) end end,
      delete → let fn:Fn · fn ∈ dom tt ∨ ... in
          let (r,tt′) = $\mathcal{M}_u$(mk_del(fn))(tt) in
          ((delete,mk_del(fn),r),tt′) end end
    end end
```

- The semantics functions illustrate the internal nondeterministic choices that the client, respectively the staff, makes — as seen from the point of view of the semantics — of the parameters that go into the specific query, respectively update commands.
  - ⋆ For the display query it is the choice of the flight number.
  - ⋆ For the connection query it is the choice of the maximum number of changes of flights, as well as the choice of the from (departure, or airport of origin) and to (destination) airports.
  - ⋆ For the add journey update it is the choice of the flight number and the journey (of that flight).
  - ⋆ For the delete flight update it is the choice of the flight number.
- We "reassemble" the above formula into the previously defined system function, cf. Example 19.183.

- Before we had:

```
value
  system: TT → Unit
      (let q:Query in let v = $\mathcal{M}_q$(q)(tt) in system(tt) end end)
    ⌈⌉ (let u:Update in let (r,tt′) = $\mathcal{M}_u$(q)(tt) in system(tt′) end end)
```

- Now we get:

```
value
  system: GUI → TT → Unit
      (let gui′ = client(gui)(tt) in system(gui′)(tt) end)
    ⌈⌉ (let (gui′,tt′) = staff(gui)(tt) in system(gui′)(tt′) end)
```

■

Machine-Machine Dialogue

- The desired machine is usually serving in a context in which it has been fitted to other machines or to supporting technologies.
- These may provide sensory data or accept actuation (i.e., control) data.
- Some fitted machines may provide for, or accept mass data transfers.
- Usually supporting technologies provide for, or accept rather "small", i.e., single (simple) data transfers.

Characterisation 19.229 By *machine-machine dialogue requirements* we understand

- syntax (incl. sequential structure) and
- semantics (i.e., meaning)
- of the communications (i.e., messages) transferred
- in either direction over the automated interface between machines (including supporting technologies)

. ■

Example 19.198 *Machine-Machine Dialogue Requirements:*

- Rail switches are assumed, upon request, to provide sensory signals, which report on their state: "straight" or "turn-off".
- And these rail switches will respond to control signals which, within an assumed response time of their being issued, set the switch to a desired state ("straight" or "turn-off").
- The cabin tower maintains a display which shows the states of all switches in its associated station.
- Associated with this cabin tower display are two buttons: Pressing either of these shall correspond to sending "straight" or "turn-off" control signals.

- Only one of these buttons can be pressed in any one-minute interval.
- At half-minute intervals each switch reports its status, and that status shall be reflected in the cabin tower display.
- When a "straight" or "turn-off" control button is depressed, then a signal shall be sent to the designated switch,
- and that switch shall react accordingly within a 15-second time lapse.
- The cabin tower switch display shall sound and flash appropriate alarms if the switch status, within half a minute, is not the desired (control signalled) one.

■

- The above example admittedly provides only a very rough sketch indication.
- It also "links" up to (that is, strongly depends on related) machine (including support technology) requirements, as covered next.

Example 19.199 *Machine-Machine Dialogue Requirements:*

- Suppose that an application calls for the massive transfer of data over noisy distances.
- That is, the probability that transferred data may be corrupted, i.e., change value during communication, is considerable.
- What is known as a suitable data communication protocol therefore has to be prescribed, one that helps ensure detection of corrupted data so as to enable retransmission until it has been decided that a correct, i.e., uncorrupted, transfer has been completed.
- These data communication protocols are of the kind that we would call machine-machine dialogues.
- Other than treating this as a metaexample we shall not go into detail in this book.

■

Discussion: Interface Requirements

Dialogue Prescription Techniques and Tools

- We have not, in this lecture on interface requirements, shown any examples of, or formalised the dialogue aspects of interface requirements.
- The term interface implies at least two interacting behaviours.
- Therefore techniques and tools (i.e., notations) for process modelling are used in such formalisations.
- We refer to Vol. 1, Chap. 21 (*Concurrent Specification Programming*) and Vol. 2, Chap. 13 (*Message and Live Sequence Charts*), where we cover formal tools and techniques for modelling such interaction.

General

- We have outlined six reasonably distinguishable facets that the requirements engineer may need perform in order to construct an interface requirements prescription.
- There may be other such facets.
- The above six have been found useful in several development projects.
- Knowing about them, their underlying principles, and their techniques and tools should help the requirements engineer to more efficiently acquire interface requirements prescriptions, and to document them, i.e., to structure their documentation logically.

Special Principles and Techniques

- Interface requirements, in most people's minds and expression, are concerned with so-called "user-friendliness".
- That is, interface requirements focus, very much, on the form of the dialogues and the layout of GUIs.
- Much can be said about this.
- We shall venture our definition of "user-friendliness".

Characterisation 19.230 By a *user-friendly man-machine interface* we understand one which somehow satisfies the following criteria:

- **Faithful:** The interface reflects only the shared phenomena and concepts, and reflects "absolutely" no machine (i.e., hardware + software) concepts (i.e., jargon). That is, the terminology used "across" the interface is that of the domain.
- **Didactic:** The sequence of presentation of shared phenomena and concepts reflects some clarified view on how these phenomena and concepts relate, which are the more important ones, and which reflect current or changing business processes, support technologies, managements and organisations, rules and regulations, etc.

- **Pedagogic:** The number of phenomena and concepts presented in any one step of interaction is small, say from one to at most five. The order of presentation is initially from core phenomena and concepts to increasingly derived phenomena and concepts. That order may initially be pedantic, but is accepted by novice users. For more experienced users means for clear, logical "shortcuts" should be made available.
- **Physiologic:** The number of current and alternative physiologic "gadgets"[22] needed to maintain interaction should be modest and be balanced against simplicity or complexity of interaction.

[22] Screen, keyboard, mouse, other tactile instruments ("pointing to", pressure-sensitive screens), audio (i.e., loudspeakers), microphone, etc.

- **Psychologic:** Interaction response, incl. prompt times and texts should not irritate[23] or shame the users, or make these users feel inadequate, or guilty (say, of "not knowing").
- **Artistic:** And then it is certainly user-friendly, this author believes, if the interface reflects some artistic ideas.

The above characterisation is only approximate. . ■

[23] The response to a user query, which took the user maybe a minute to prepare, should not follow the submission of that query in the order of microseconds, rather 1.5–3 seconds is more pleasing, psychologically. For short, "click"-type "queries", response times of 100 milliseconds seem OK.

- If referring to special textbooks [**?**, **?**] on the subject, we advise the student to pay strict attention to the issues we have raised:
  - ⋆ Make sure that interface requirements, when referring to phenomena and concepts,
  - ⋆ refer "strictly" to those that are well understood in the domain.

Topic 57
Machine Requirements

Characterisation 19.231 By *machine requirements* we understand

- those requirements that can be expressed
- solely in terms of (or with prime reference to) machine concepts

. ■

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.*

Machine Requirements and the Requirements Document

- Some remarks need to be made
- before we go into details of domain requirements modelling techniques.

Requirements for "the Machine Only"

- Machine requirements are about the machine only!
  - ⋆ They, the machine requirements, "in the extreme"
  - ⋆ contain no references to any specific aspect of the domain.

- But there may be general references, and they could be of the same nature for whichever domain was the base,
- such as,
  - ⋆ such and such function invocations shall terminate in less than m microseconds,
  - ⋆ whereas such and such function invocations shall terminate in less than n seconds.
- Or,
  - ⋆ such and such data shall be replicated for back-up reasons,
  - ⋆ or auxiliary storage for performing such and such functions shall be less than 500 KB.

- The machine requirements all take their "departure point",
- that is, are based upon,
- potentially available machine technology,
- whether
  - ⋆ central, or
  - ⋆ distributed, or
  - ⋆ input/output, or
  - ⋆ peripheral.

Place in Narrative and Formalisation Document

- Later in this lecture we shall treat a number of machine requirements facets.
  - ⋆ Each of whichever you decide to focus on,
  - ⋆ in any one requirements development,
  - ⋆ must be prescribed.

- The machine requirements are really void of any (material) reference
- to domain phenomena and concepts.
- Hence the machine requirements prescriptions
- form a separate, "freestanding" document.
- That document must describe both the machine component
  - ⋆ (i.e., hardware,
  - ⋆ and software)
- interfaces and functionalities
- (the latter, say, in pre/postcondition form).

Performance Requirements

Characterisation 19.232 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

. ■

- Pragmatically speaking, performance requirements translate into financial resources spent, or to be spent.

Example 19.200 *Performance Requirements:* We continue Example 19.185.

- The machine shall serve 1000 users and 1 staff member.
- Average response time shall be at most 1.5 seconds,
- when the system is fully utilised.

■

- Till now we may have expressed certain (functions and) behaviours as generic (functions and) behaviours.
- From now on we may have to "split" a specified behaviour
  - ⋆ into an indexed family of behaviours,
  - ⋆ all "near identical" save for the unique index.
- And we may have to separate out, as a special behaviour, (those of) shared entities.

Example 19.201 *Performance Requirements:* We continue Example 19.183 and Example 19.200.

- In Example 19.183 the sharing of the timetable between users and staff was expressed parametrically.

$\mathrm{system(tt)} \equiv \mathrm{client(tt)} \sqcap \mathrm{staff(tt)}$

$\mathrm{client}\colon \mathrm{TT} \to \textbf{Unit}$
$\mathrm{client(tt)} \equiv \textbf{let}\ \mathrm{q{:}Query}\ \textbf{in}\ \textbf{let}\ \mathrm{v} = \mathcal{M}_q(\mathrm{q})(\mathrm{tt})\ \textbf{in}\ \mathrm{system(tt)}\ \textbf{end}\ \textbf{end}$

$\mathrm{staff}\colon \mathrm{TT} \to \textbf{Unit}$
$\mathrm{staff(tt)} \equiv$
$\quad \textbf{let}\ \mathrm{u{:}Update}\ \textbf{in}\ \textbf{let}\ (\mathrm{r,tt'}) = \mathcal{M}_u(\mathrm{u})(\mathrm{tt})\ \textbf{in}\ \mathrm{system(tt')}\ \textbf{end}\ \textbf{end}$

- We now factor the timetable entity out as a separate behaviour,
- accessible, via indexed communications, i.e., channels,
- by a family of client behaviours and the staff behaviour.

type
  CIdx /∗ Index set of, say 1000 terminals ∗/
channel
  { ct[i]:QU,tc[i]:VAL | i:CIdx }
  st:UP,ts:RES
value
  system: TT → **Unit**
  system(tt) ≡ time_table(tt) ∥ (∥ {client(i)|i:CIdx}) ∥ staff()

client: i:CIdx → **out** ct[i] **in** tc[i] **Unit**
client(i) ≡ **let** qc:Query **in** ct[i]!$\mathcal{M}_q$(qc) **end** tc[i]?;client(i)

staff: **Unit** → **out** st **in** ts **Unit**
staff() ≡ **let** uc:Update **in** st!$\mathcal{M}_u$(uc) **end let** res = ts? **in** staff() **end**

time_table: TT → **in** {ct[i]|i:CIdx},st **out** {tc[i]|i:CIdx},ts **Unit**
time_table(tt) ≡
  □ {**let** qf = ct[i]? **in** tc[i]!qf(tt) **end** | i:CIdx}
  □ **let** uf = st? **in let** (tt′,r)=uf(tt) **in** ts!r; time_table(tt′) **end end**

- Please observe the “shift”
  - ⋆ from using ⊓ in system earlier in this example
  - ⋆ to □ just above.
- The former expresses nondeterministic internal choice.
- The latter expresses nondeterministic external choice.
- The change can be justified as follows:
  - ⋆ The former, the nondeterministic internal choice, was “between” two expressions which express no external possibility of influencing the choice.
  - ⋆ The latter, the nondeterministic external choice, is “between” two expressions where both express the possibility of an external input, i.e., a choice.
- The latter is thus acceptable as an implementation of the former.

■

- The next example, Example 19.202, continues the performance requirements expressed just above.
- Those two requirements could have been put in one phrase, i.e., as one prescription unit.
- But we prefer to separate them, as they pertain to different kinds (types, categories) of resources: terminal + data communication equipment facilities versus time and space.

Example 19.202 *Performance Requirements:* We continue Example 19.185.

- When performing the *n-Transfer Travel Inquiry* (rough sketch) prescribed above,
  - ⋆ the first — of an expected many — result shall be communicated back to the inquirer in less than 5 seconds after the inquiry has been submitted,
  - ⋆ and, at no time during the calculation of the “next” results must the storage buffer needed to calculate these exceed around 100,000 bytes.

■

Dependability Requirements

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

Characterisation 19.233 • A machine *failure* occurs

- when the delivered service
- deviates from fulfilling the machine function,
- the latter being what the machine is aimed at

.

■

Characterisation 19.234 • 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

. ■

Characterisation 19.235 • 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.*

Characterisation 19.236 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

. ■

Characterisation 19.237 *Dependability* is defined

- as the property of a machine
- such that reliance can justifiably be placed on the service it delivers

. ■

- *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.*

Characterisation 19.238 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.).

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.

Characterisation 19.239 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 19.203 *Accessibility Requirements: Timetable Access:*

- The timetable (system) shall be inquirable by any number of users,
- and shall be updateable by a few, so authorised, airline staff.
- At any time it is expected that up towards a thousand users are directing queries at the timetable (system).
- And at regular times, say at midnights between Saturdays and Sundays, airline staff are making updates to the timetable (system).
- No matter how many users are "on line" with the timetable (system), each user shall be given the appearance that that user has exclusive access to the timetable (system).

■

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"!

Characterisation 19.240 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 19.204 *Availability Requirements: Timetable Availability:* We continue Examples 19.183, 19.185, and 19.203:

- No matter which query composition any number of (up to a thousand) users are directing at the timetable (system),
- each such user shall be given a reasonable amount of compute time per maximum of three seconds,
- so as to give the psychological appearance that each user — in principle — "possesses" the timetable (system).
- If the timetable system can predict that this will not be possible, then the system shall so advise all (relevant) users.

■

Integrity

Characterisation 19.241 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.

Reliability

Characterisation 19.242 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 19.205 *Timetable Reliability:*

- Mean time between failures shall be at least 30 days,
- and downtime due to failure (i.e., an availability requirements) shall, for 90% of such cases, be less than 2 hours.

■

Safety

Characterisation 19.243 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 19.206 *Timetable Safety:*

- Mean time between failures
- whose resulting downtime is more than 4 hours
- shall be at least 120 days.

■

Security

- Security 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.

Characterisation 19.244 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)

. ■

- The characterisation of security is rather abstract.
- As such it is really no good as an a priori design guide.
- That is, the characterisation gives no hints as how to implement a secure system.
- But, once a system is implemented, and claimed secure, the characterisation is useful as a guide on how to test for security!

Example 19.207 *Security Requirements: Timetable Security:* We continue Examples 19.183, 19.185, 19.203, and 19.204.

- Timetable users can be any airline client logging in as a user, and such (logged-in) users may inquire the timetable.
- The timetable machine shall be secure against timetable updates from any user.
- Airline staff shall be authorised to both update and inquire, in a same session.

■

Example 19.208 *Security Requirements: A Hospital Information System:*

- General access to (including copying rights of) patient's medical journals is granted only to designated hospital staff.
- In certain forms of emergency situations any hospital staff may , get access to a hospital patient's medical journal.
- Such incidents shall be duly and properly recorded and reported.

■

Robustness

Characterisation 19.245 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.

Fault Analysis:

- In pursuing the formulation of requirements for dependable systems
- it is often required that the requirements engineer perform
- what is called *fault analysis*.
- A particular approach is called *fault tree analysis*.
- Dependable systems development is worth a whole study in itself.
- So we cut short our mentioning of this very important subject
- by emphasising its importance and otherwise referring the course student to the relevant literature.

Maintenance Requirements

Characterisation 19.246 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.

We have, in an introductory lecture, given a proper definition of what we mean by software.

Adaptive Maintenance

Characterisation 19.247 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 19.209 *Adaptive Maintenance Requirements: Timetable System:*

- The timetable system is expected to be implemented in terms of a number of components that implement respective domain and interface requirements, as well as some (other) machine requirements.
- The overall timetable system shall have these components connected, i.e., interfaced with one another — where they need to be interfaced — in such a way that any component can later be replaced by another component ostensibly delivering the same service, i.e., functionalities and behaviour.

■

Corrective Maintenance

Characterisation 19.248 By *corrective maintenance* we understand such maintenance which

- corrects a software error

. ■

Example 19.210 *Corrective Maintenance Requirements:*

- Corrective maintenance shall be done remotely:
  - ⋆ from a developer site,
  - ⋆ via secure Internet connections.

■

Perfective Maintenance

Characterisation 19.249 By *perfective maintenance* we understand such maintenance which

- helps improve (i.e., lower) the need for
- hardware (storage, time, equipment),
- as well as software

. ■

Example 19.211 *Perfective Maintenance Requirements: Timetable System:*

- The system shall be designed in such a way as to
  - ⋆ clearly be able to monitor the use of
  - ⋆ "scratch" (i.e., buffer) storage and compute time

  for any instance of any query command.

■

Preventive Maintenance

Characterisation 19.250 By *preventive maintenance* we understand such maintenance which

- helps detect, i.e., forestall, future occurrence
- of software or hardware errors

. ■

Extensional Maintenance

Characterisation 19.251 By *extensional maintenance* we understand such maintenance which adds new functionalities to the software, i.e., which implements additional requirements. ■

Example 19.212 *Extensional Maintenance Requirements: Timetable System:*

- Assume a release of a timetable software system to implement a requirements that, for example, expresses
  - ⋆ that shortest routes
  - ⋆ but not that fastest routes be found
  - ⋆ in response to a travel query.

- If a subsequent release of that software
  - ⋆ is now expected to also calculate fastest routes
  - ⋆ in response to a travel query,
- then we say that the implementation of that last requirements
- constitutes extensional maintenance.

■

• • •

- Whenever a maintenance job has been concluded, the software system is to undergo an extensive acceptance test:
- a predetermined, large set of (typically thousands of) test programs has to be successfully executed.

Platform Requirements

Characterisation 19.252 By a [computing] *platform* is here understood a combination of hardware and systems software — so equipped as to be able to execute the software being requirements prescribed — and 'more'. ■

Characterisation 19.253 By *platform requirements* we mean a combination of the following:

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

■

Example 19.213 *Platform Requirements: Space Satellite Software:* Elsewhere prescribed software for some space satellite function is to satisfy the following platform requirements:

- shall be developed on a `Sun` workstation under `Sun UNIX,`
- shall execute on the military `MI1750` hardware computer running its proprietary `MI1750 Operating System,`
- shall be maintained at the NASA Houston, TX installation of `MI1750 Emulating Sun Sparc Stations,` and
- shall be demonstrated on ordinary `Sun` workstations under `Sun UNIX.`

■

Development Platform

Characterisation 19.254 *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

. ■

Execution Platform

Characterisation 19.255 *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

. ■

Maintenance Platform

Characterisation 19.256 *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

. ■

Demonstration Platform

Characterisation 19.257 *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

. ■

Discussion

- Example 19.213 is rather superficial.
- And we do not give examples for each of the specific four platforms.
- More realistic examples would go into rather extensive details,
- listing hardware and software product names, versions, releases, etc.

Documentation Requirements

Characterisation 19.258 By *documentation requirements* we mean requirements of any of the software documents that together make up software:

- 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*

. ■

- We do not attempt, in our characterisation, to detail what such documentation requirements could be.
- Such requirements could cover a spectrum
  - ⋆ from the simple presence, as a delivery, of specific ones,
  - ⋆ to detailed directions as to their contents, informal or formal.

Discussion: Machine Requirements

- We have — at long last — ended an extensive enumeration, explication and, in many, but not all cases, exemplification, of machine requirements.
- When examples were left out it was because the reader should, by now, be able to easily conjure up such examples.
- The enumeration is not claimed exhaustive.
- But, we think, it is rather representative.
- It is good enough to serve as a basis for professional software engineering.
- And it is better, by far, than what we have seen in “standard” software engineering textbooks.

Topic 58
Composition of Requirements Models
General

- In various “Domain Facet” parts of these past lectures
- we have briefly mentioned the topic of *“$\mathcal{X}$ and the Requirements Document”.*
- Here $\mathcal{X}$ stood for either
  - ⋆ BPR,
  - ⋆ Domain Requirements,
  - ⋆ Interface Requirements, or
  - ⋆ Machine Requirements.

- We shall remind the course student to review these four subsections.
- They tell you a lot about how to document the requirements,
- as basically a set of four more or less separate subdocuments,
- whether informally, as a narrative,
- or formally, as an annotated formal definition.

Topic 59

Discussion: Requirements Facets

General

- We have covered the three main facets of requirements models:
  - ⋆ domain requirements,
  - ⋆ interface requirements and
  - ⋆ machine requirements.
- The course student who studies this volume on the basis of emphasising the formal techniques
  - ⋆ will have noted that there were rather few, if any, formalised examples.
  - ⋆ This was especially true for the machine requirements.

- This does not mean that one could not furnish such examples.
- We have chosen not to show such examples for three reasons:
  - ⋆ First, the examples would be somewhat long.
  - ⋆ Second, such examples have already been shown in other lectures.
  - ⋆ But, more important, we still, as of 2006, lack appropriate formal techniques and tools.
- But we observe, today, steady and impressive progress in formal techniques and tools for expressing machine requirements.

Principles, Techniques and Tools

Principle 19.88 *Requirements Facets:* “Divide and Conquer”:

- Adopt a “separation of concerns” principle; hence
- model
  - ⋆ domain,
  - ⋆ interface and
  - ⋆ machine

  requirements separately, as near so as possible

. ■

OFTWARE ENGINEERING: Domains, Requirements and Software Design | Volume 3
9.8.2 Principles, Techniques and Tools
home/db/volII/3ch19/3ch19-v | April 5, 2006, 13:05 | Page 1850, Topic: 59, Foil: 4
Department of Computer Science and Engineering
Institute of Informatics and Mathematical Modelling
Technical University of Denmark
Richard Petersens Plads, DK-2800 Kgs.Lyngby, Denmark
DTU

Techniques 59 *Requirements Facets:* The techniques fall, as usual, into two classes:

- the informal techniques, which
  - ⋆ cover all the so-far-covered informal techniques of
  - ⋆ rough-sketching, terminologisation and narration;
- and the formal techniques,
  - ⋆ which. likewise, cover all the so-far-covered formal techniques of
  - ⋆ formal abstraction and modelling

. ■

Tools 19.23 *Requirements Facets:* The tools, like the techniques, fall, as usual, into two classes:

- the informal tools, which include
  - ⋆ ordinary text-processing tools
  - ⋆ with extensive cross-referencing
  - ⋆ and database storage facilities;
- and the formal tools, which include
  - ⋆ all the ones ordinarily used in connection with formal specification:
    - ◇ syntax editors, type checkers,
    - ◇ verification, model checking and test tools, and so on

. ■