Department of Computer Science and Engineering
Institute of Informatics and Mathematical Modelling
Technical University of Denmark
Richard Petersens Plads, DK-2800 Kgs.Lyngby, Denmark

Topic 19
Phenomena and Concepts

- The **prerequisite** for following this part of the lecture is that you are willing to think, and are able, or at least wish to think abstractly.

- The **aims** are to introduce basic principles and techniques for
  - ⋆ "discovering" *phenomena* that need *conceptualisation,*

  in particular, to introduce those phenomena whose conceptualisation is in terms of
  - ⋆ *entities* (information, data),
  - ⋆ *functions* (and relations),
  - ⋆ *events* (asynchronous and synchronous) and
  - ⋆ *behaviours,* and

  to introduce basic principles and techniques for the description of such phenomena and their underlying concepts.

- The **objective** is to intellectualise the reader, as necessary, but not yet sufficiently enough for the reader to become an effective professional software engineer.
- The **treatment** is from systematic to rigorous.

Introduction

- In the next lecture topics we shall cover a first set of facets of the concept of description: the problem of identification, that is, of being able to identify or delineate,
  - ⋆ *phenomena* and
  - ⋆ *concepts,*

  of interest; that is
  - ⋆ *physically manifest things*, and
  - ⋆ *mental constructions.*

- We shall try to wrestle with some abstract ideas.
- They impinge upon *what can be known, and what can be described.*
  - ⋆ As such these abstract ideas border on *philosophy,*
  - ⋆ in particular such philosophical disciplines as
    - ⋄ *epistemology*, and *ontology.*
- For this reason we cannot treat these ideas with the kind of certainty usual in a discourse of mathematics or the natural sciences,
- but must be prepared for a certain degree of uncertainty!

Phenomena and Concepts

- In these lectures we shall thread a careful course.
- We do not wish to establish a brand new theory of phenomena.
- We certainly do not wish to entertain such ideas as
  - ⋆ object-orientedness, conceptual schemas, or whatever.
- We simply wish to go as far as very simple mathematics can support us.
  - ⋆ By that we mean: types and values, functions, events and behaviours.
- No further!
- But first we discuss the ideas of phenomena and concepts.

Physically Manifest Phenomena

- In the world there are the physically manifest *phenomena.*
  - ⋆ We can sense them: touch, see, hear, feel, smell and taste them.
  - ⋆ Or we can measure them: mechanically, electrically/electronically, chemically, etc.
- Thus we can point to them and designate them, in one way or another.

Mentally Conceived Concepts

- We often abstract a phenomenon into a *concept.*

Example 5.43 *Automobile Phenomenon Versus Car Concept:*

- The specific phenomenon of "that automobile"
- is abstracted into the type, the class, the set of all cars,
- i.e., the concept *car.*

■

The above example illustrates

- the concrete *value concept* ("that automobile") versus
- the abstract *type concept* (*car*).

Categories of Phenomena and Concepts

- For pragmatic reasons
- we categorise phenomena and concepts into four categories:
  - ⋆ entities — their values, properties and types;
  - ⋆ functions over entities;
  - ⋆ events (related to behaviours); and
  - ⋆ behaviours (as traces of events and function applications, i.e., actions).
- These category names represent *metaconcepts*.

Characterisation 5.94 By a *phenomenon* we shall loosely understand

- some physical thing
- that humans can sense or
- which natural science based technology can measure,

and where a phenomenon is typically

- a natural thing or
- a human-made artifact

. ■

Characterisation 5.95 By a *concept* we shall loosely understand

- a mental construction
- — conceived by people —
- which, in an abstract manner
  - ⋆ captures an essence of usually a class of phenomena
  - ⋆ or a class of concepts

. ■

Concrete and Abstract Concepts

- Phenomena can be pointed to or measured. They are physical.
- Concepts are mental constructions.
- When we "lift" our consideration, for example, from *that specific car over there*, i.e., of a phenomenon, to a representative of the set of cars, then we have abstracted "away" from a specific phenomenon to the concrete concept of car.
- When we "lift" our considerations, for example, from those of dealing with cars, trains, airplanes or ships, that is from a set of concrete concepts, to consider specimens (i.e., instances) of these as vehicles (or conveyors), then we consider vehicles (conveyors) as abstract concepts.
- And so on.

Characterisation 5.96 By a *concrete concept* we mean an abstraction of a set of similar phenomena into one concept. ■

- Given a description of a concrete concept we can speak of its concretisation
- as an identification which establishes a relation between the concrete concept and at least one phenomenon which is intended covered by the concrete concept.

Characterisation 5.97 By an *abstract concept* we mean an abstraction of a set of similar concepts into one concept. ■

- Given a description of an abstract concept we can speak of its concretisation
- as an identification which establishes a relation between the abstract concept and at least one concrete concept which is intended covered by the abstract concept.

Discussion 5.17

- When we say "a car" we mean, not a specific one, that is, not a specific phenomenon, but a concrete concept, not a set of cars, but a single instance (the generic car).
- If we say "those cars there" we mean a set of phenomena.
- And if we say "consider a set of cars" then we mean a set of concrete concepts — for short: a concrete concept.
- We can describe a set, s, of values, a, all of which are of some type A.
- In that case the set s is a value of type A-**set**.

■

Categories of Descriptions

- Usually we shall be formulating our descriptions in terms of concepts, not in terms of phenomena.
- But occasionally it is required that a description in terms of phenomena is developed and presented.

Characterisation 5.98 A *description* is said to be *phenomenological* if all of its entities, functions, events and behaviours are of phenomena. ■

Characterisation 5.99 A *description* is said to be *conceptual* if all of its entities, functions, events and behaviours are of concepts. ■

Characterisation 5.100 A *description* is said to be *specific problem-oriented* if all of its entities, functions, events and behaviours are of some mixture of both phenomena (one or more) and concepts (one or more). ■

Department of Computer Science and Engineering
Institute of Informatics and Mathematical Modelling
Technical University of Denmark
Richard Petersens Plads, DK-2800 Kgs.Lyngby, Denmark

Example 5.44 *A Specific Problem-Oriented "Toy" Description:*

- The *South Coast Rail Line* regional railway net consists of
  - ⋆ a single linear route, r,
  - ⋆ made up from six stations, $s_1, s_2, s_3, s_4, s_5, s_6$, and
  - ⋆ five consecutive lines, $\ell_{12}, \ell_{23}, \ell_{34}, \ell_{45}, \ell_{56}$,
  - ⋆ such that route r can be thought of as a sequence of alternating stations and lines: $\langle s_1, \ell_{12}, s_2, \ell_{23}, s_3, \ell_{34}, s_4, \ell_{45}, s_5, \ell_{56}, s_6 \rangle$ or the other way around: $\langle s_6, \ell_{56}, s_5, \ell_{45}, s_4, \ell_{34}, s_3, \ell_{23}, s_2, \ell_{12}, s_1 \rangle$.
  - ⋆ More specifically line ℓ_{12} is 17 km long, line ℓ_{23} is 19 km long, ..., and line ℓ_{56} is 23 kms long.
  - ⋆ Topologically and geodetically line ℓ_{12} runs as follows: ...(etcetera).
  - ⋆ Stations s_1 is named Arlington, s_2 Burlington, ..., and station s_6 is named Georgetown.
  - ⋆ Topologically and geodetically station s_1 is organised as follows: ...(etcetera).
  - ⋆ Etcetera.

■

What Is a Description?

- Finally we are ready to address the crucial issue of what a description is.

Characterisation 5.101 By a *description* we mean

- some text which either designates
  - ⋆ a set of phenomena in such a way
  - ⋆ that the reader of the description
  - ⋆ can recognise these phenomena from the description

- or a description designates
  - ⋆ a set of concepts in such a way
  - ⋆ that the reader of the description
  - ⋆ can concretise these into recognisable phenomena,
- or a description is a text which designates both
  - ⋆ recognisable phenomena and
  - ⋆ recognisable concepts

. ■

- A later lecture covers
  - ⋆ (i) recognisability of descriptions,
  - ⋆ (ii) the issue of providing as few phenomena (or actually concrete concept) descriptions as possible (i.e., the so-called "narrow bridge"),
  - ⋆ (iii) the issue of instead relying on definitions and
  - ⋆ (iv) the issue of risking refutable descriptions.

Topic 20
Entities

- *One man's entity is another man's function!*
- We hedge the opening of this part of the lecture by a caveat, and a veiled warning.
- When we now shall try to characterise what an entity is,
- it must be understood as a choice that the developer has to make
  - ⋆ of whether to consider a phenomenon or a concept as an entity or as a function.
  - ⋆ Usually that choice is an easy one to make.
  - ⋆ Colloquially speaking: if you think of the phenomena or concepts as information typically computerisable as data, then the phenomena or concepts are entities.

- Similar remarks can be made for the possible relations between entities and behaviours.

Characterisation 5.102 By an *entity* we shall loosely understand

- something fixed, immobile or static.
- Although that thing may move,
- after it has moved it is essentially the same thing,
- an entity

. ■

- From a pragmatic point of view,
- entities are the "things" that, if implemented inside computers,
- could typically be represented as data.

Example 5.45 *Entities:* We give some examples:

- a (specific) pencil,
- a (specific) chocolate bar,
- a (specific) pail of paint,
- a (specific) person,
- a (specific) railway net,
- a (specific) train or
- a (specific) transport industry.

■

- We make the distinction between
  - ⋆ atomic entities and
  - ⋆ composed (i.e., composite) entities.

Atomic Entities

Characterisation 5.103 By an *atomic entity* we shall understand an entity

- which cannot be understood as composed from
- other entities

. ■

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

Example 5.46 *Atomic Entities:* We give a few examples:

- a (specific) pencil,
- a (specific) chocolate bar,
- a (specific) person or
- a (specific) timetable.

■

Lecturer:

- Discuss parts of pencil, parts of person:
- whether entities or not — depending on viewpoint.

Composite Entities

Characterisation 5.104 By a *composite entity* e we shall understand an entity

- which can best be understood as composed from
- other entities, called the subentities, $e_1, e_2, \ldots, e_n$, of entity e

. ■

Example 5.47 *Composed Entities:* We give a few examples:

- a (specific) railway net (as composed from lines and stations),
- a (specific, say passenger) train (as composed from passenger cars and engines (locomotives)) or
- a (specific) transport industry (as composed from the transport net, the transport vehicles, and so on).

■

Subentities

Characterisation 5.105 By a *subentity* we shall understand an entity

- which is a component of another entity

. ■

Example 5.48 *Subentities:* We give a few examples:

- the lines and stations of a (specific) railway net,
- the locomotive(s) and cars of a (specific) train,
- the railway net of a (specific) railway system.

■

Values, Mereology and Attributes

- Examples 5.46–5.48 designated certain entities.
- For each of them we can speak of a *value* of that entity.
- And for each of them we can speak of zero, one or more *attributes* of that entity.
- A concept of *mereology* is associated only with composite entities
  - ⋆ and then expresses
  - ⋆ how the entity is composed from subentities.

Characterisation 5.106 By a *value* v_e of an entity we shall loosely understand the following:

- If the entity is an atomic entity,
  - ⋆ then the entire set of identified attributes, $a_{1_e}, a_{2_e}, \ldots, a_{n_e}$, of the entity.
- If the entity is a composite entity (thus consisting, say, of subentities, $e_1, e_2, \ldots, e_m$) then there are three parts to the entity value:
  - ⋆ how it is composed — its mereology m,
  - ⋆ the entire set of identified attributes, $a_{1_e}, a_{2_e}, \ldots, a_{n_e}$, of the entity, and
  - ⋆ (inductively) the identified values, $v_{e_1}, v_{e_2}, \ldots, v_{e_m}$, of respective subentities $(e_1, e_2, \ldots, e_m)$

. ■

Characterisation 5.107 By an *attribute* of an entity we shall loosely understand

- a quality which cannot be separated from the entity

. ■

Example 5.49 *Attributes and Values:*

- The following are some of the attributes of an atomic pencil entity:
  - ⋆ the physical length of the pencil,
  - ⋆ the materials from which it is made,
  - ⋆ the colour of the (lead) pen in the pencil,
  - ⋆ all the other attributes associated with the appearance of the pen (wear and tear),
  - ⋆ its purchase price, etc.
- The value of the atomic pencil is thus
  - ⋆ the (mereological) fact that it is an atomic entity, and
  - ⋆ the sum total of all the above (including "etc.") attributes.

■

Entity Mereology

By mereology we understand a theory of part-hood relations. That is, of the relations of part to whole and the relations of part to part within a whole.

Characterisation 5.108 By the *mereology of an entity* we shall loosely understand

- whether it is atomic,
- or, when it is composite, then how it is made composite
- (i.e., from which kind of subentities it is composed)

. ■

- Please note our attempt to distinguish between
  - ⋆ entities,
  - ⋆ entity values,
  - ⋆ entity attributes and
  - ⋆ entity mereologies.
- Note that we use the term attempt.

Example 5.50 *Mereology Facets of a Railway Net:*

- The mereology of a railway net transpires from the *italicized* terms.
  - ⋆ A net is *composed from*, and hence *decomposable into* a *collection* of lines and a *collection* of stations.
  - ⋆ Any line *links exactly two distinct* stations.
  - ⋆ Any station *is linked to one or more distinct* lines.
  - ⋆ A rail line is *composed from* linear rail units.
  - ⋆ The connectors of a rail unit are (here considered) *inseparable from* the *pairs of* rails (and the ties and their nails, etc.) making up the main part of rail units.
- We have not said anything above about how the collections form nets. But such formation rules are part of the mereology. ■

- There is no end to the kind of attributes and the form of mereology one may eventually associate with an entity.
- And for that matter, one can also associate attributes and mereologies with functions and behaviours, as we shall see.
- It is not productive, we strongly believe, to try enumerate all the possible categories of
  - ⋆ mereologies and
  - ⋆ attributes —
  - ⋆ and hence information —
- that one may associate with entities (functions and behaviours).
- One easily becomes lost in philosophical discourse.
- Our job is mainly constrained by putting whatever we domain-analyse inside the computer.

- Hence, in the final analysis we need just resort to what can be described in terms of abstractions of
  - ⋆ computer data, i.e., values and types,
  - ⋆ computing routines (algorithms) and
  - ⋆ computing processes.
- Thus we shall mainly focus on such mereologies
  - ⋆ which can be informally, but precisely described, in natural English
  - ⋆ and which can be mathematically described,
  - ⋆ or, further constraining the issue of mereologies, which can be represented inside the computer.

Mereologies and Attributes

- So an entity e value v_e is made up, in a loose sense, of
  - ⋆ (i) its mereology —
    - ⋄ (i.a) whether atomic or
    - ⋄ (i.b) composed from subentities, and then how —
  - ⋆ (ii) entity attributes, $a_{1_e}, a_{2_e}, \ldots, a_{n_e}$,
  - ⋆ and (iii), when the entity, e, is composite, then (inductively) the values, $v_{e_1}, v_{e_2}, \ldots, v_{e_m}$, of these entities $(e_1, e_2, \ldots, e_m)$.
- We shall model
  - ⋆ all this information as values of appropriate types.

Model-Oriented Mereologies

- Volumes 1 and 2 of this series of volumes focused on property- and model-oriented ways of specifying mereologies and attributes.
- In those volumes, except in Chap. 2 in Vol. 2 (*hierarchies and composition*), we did not single out the concept of mereology.
- But the model-oriented means of modelling composite entities as
  - ⋆ sets
  - ⋆ lists
  - ⋆ functions
  - Cartesians ⋆
  - maps ⋆
- had that intention.

Model-Oriented Attributes — An Aside

Atomic Types and Values

- We could consider the constraints that might be put on a model-oriented mereology for some entity either as a property of the mereology or as an attribute of the entity.
- In addition to this, there are the end types of the atomic entities that eventually make up any entity.
- The actual values of these atomic types, in our view, constitute the attributes of the entity.

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

Other Attributes

- But, as amply shown in, for example, Vol. 2,
- there are seemingly composite types that model what we should like to classify as attributes rather than as mereologies:
  - ⋆ functions that model temporal progression, e.g., traffic,
  - ⋆ functions that model denotations,
  - ⋆ and so on.
- We shall therefore have to resign, for this volume, and say:
  - ⋆ for a treatment of the proper modelling facets of entities
  - ⋆ we must refer to the entirety of both Vols. 1 and 2 of this series.

4525 3720, Fax: +45 4593 0074 | © Dines Bjørner, Fredsvej 11, DK-2840 Holte, Denmark | E-mails: db@imm.dtu.dk, bjorner@gmail.com, dines@bjorner.biz; URL: www.imm.dtu.dk/~db

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

Entity Properties

- So an entity value is made up, roughly speaking,
  - ⋆ of its mereology,
  - ⋆ its attributes
  - ⋆ and (if composite) the values of all subentities.
- We shall, for convenience, lump the two facets,
  - ⋆ mereologies and
  - ⋆ attributes,

  into one concept: properties.

+45 4525 3720, Fax: +45 4593 0074 | © Dines Bjørner, Fredsvej 11, DK-2840 Holte, Denmark | E-mails: db@imm.dtu.dk, bjorner@gmail.com, dines@bjorner.biz; URL: www.imm.dtu.dk/~db

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

General

- To repeat:
  - ⋆ physically manifest things have values.
    - ◇ That is, we take some values to be values of physically manifest things.
  - ⋆ Physically manifest things, in addition, have mereologies:
    - ◇ the value of a physically manifest thing is the sum total of its mereology and all its attributes.
  - ⋆ Properties (mereology and attributes), like values (in general),
    - ◇ are of types and
    - ◇ are expressed in terms of (usually nonthing) values.

4525 3720, Fax: +45 4593 0074 | © Dines Bjørner, Fredsvej 11, DK-2840 Holte, Denmark | E-mails: db@imm.dtu.dk, bjorner@gmail.com, dines@bjorner.biz; URL: www.imm.dtu.dk/~db

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

"Thing" Properties

- If it helps, you may divide the world of properties into two kinds:
  - ⋆ *thing mereologies:*
    - ◇ a property of a usually composite manifest phenomenon,
    - ◇ that it is a manifest atomic or composite phenomenon itself; and
  - ⋆ *attributes:*
    - ◇ a property of a manifest phenomenon,
    - ◇ which is not a manifest phenomenon. It is more like a property.
  - ⋆ Both kinds of properties are represented by a type and a value.

+45 4525 3720, Fax: +45 4593 0074 | © Dines Bjørner, Fredsvej 11, DK-2840 Holte, Denmark | E-mails: db@imm.dtu.dk, bjorner@gmail.com, dines@bjorner.biz; URL: www.imm.dtu.dk/~db

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

Example 5.51 *"Thing" Properties:*

- A particular railway net is a manifest phenomenon.
- Any
  - ⋆ line or
  - ⋆ station, and, for that matter,
  - ⋆ any smallest rail unit

  and many things in between,
  - ⋆ like platform or siding tracks,

  are also manifest phenomena,
- and are thus *thing properties* of the net phenomenon.

4525 3720, Fax: +45 4593 0074 | © Dines Bjørner, Fredsvej 11, DK-2840 Holte, Denmark | E-mails: db@imm.dtu.dk, bjorner@gmail.com, dines@bjorner.biz; URL: www.imm.dtu.dk/~db

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

- The curvature of a rail unit, we may claim, is an attribute (i.e., also a property, a concept) of that rail unit.
  - ⋆ The curvature itself cannot be taken apart from the rail unit and manifested as such.
- Other examples of railway net *attributes* are:
  - ⋆ the length of a line;
  - ⋆ that a unit of a line is closed for traffic; and
  - ⋆ that a line, when open, can ever only be open in one direction (an up line, as opposed to a down line, for lines connected to a station [into, respectively out from]).

■

+45 4525 3720, Fax: +45 4593 0074 | © Dines Bjørner, Fredsvej 11, DK-2840 Holte, Denmark | E-mails: db@imm.dtu.dk, bjorner@gmail.com, dines@bjorner.biz; URL: www.imm.dtu.dk/~db

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

Real Examples and Our Type System

Set Compounds

Example 5.52 *Communities and Community Networks: Intra and Inter:*

- An informal description:
  - ⋆ A community, C, consists of a finite set of one or more people, P.
  - ⋆ A society, S, consists of a finite set of one or more disjoint communities.
  - ⋆ An intracommunity network, Intra, consists of a finite set of one or more people within a community.
  - ⋆ An intercommunity network, Inter, consists of a finite set of one or more people, exactly one from each of a subset of communities of a society.

4525 3720, Fax: +45 4593 0074 | © Dines Bjørner, Fredsvej 11, DK-2840 Holte, Denmark | E-mails: db@imm.dtu.dk, bjorner@gmail.com, dines@bjorner.biz; URL: www.imm.dtu.dk/~db

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

Then, we give a formal description.

type P
  C = P-**set**
  S = C-**set**
    such that for any two distinct elements c_i, c_j
    of some s in S they share no people in common
  ∀ s:S · ∀ ci,cj:C · {ci,cj}⊆s ∧ ci≠cj ⇒ ci ∩ cj = {}
  Intra = P-**set**
    such that for any i in Intra there exists a community c of some s in S of which i is a subset
  ∀ i:Intra · ∃ s:S · ∃ c:C · c ∈ s ⇒ i⊆c

+45 4525 3720, Fax: +45 4593 0074 | © Dines Bjørner, Fredsvej 11, DK-2840 Holte, Denmark | E-mails: db@imm.dtu.dk, bjorner@gmail.com, dines@bjorner.biz; URL: www.imm.dtu.dk/~db

Inter = P-**set**
  such that for any i in Inter of n elements there exist n
  distinct communities $c_1, c_2, ..., c_n$, of some s in S such
  that each member of $i \equiv$ exactly a member of a respective c_k
 ∀ i:Inter · **card** i = n ⇒
  ∃ s,cs:S · cs⊆s ∧ **card** cs = n ∧ unique(i,cs)
value
 unique: Inter × S → **Bool**
 unique(i,cs) ≡
  ∃ ! p:P,c:C · p ∈ i ∧ c ∈ cs ∧ unique(i \ {p},cs \ {c})
  pre card i = **card** cs

- Let suitably subscripted **p**'s designate distinct persons, then:

(1) {{p1,p2,p3},{p4,p5},{p6}}
(2) {p1,p2}, {p5}, {p6}
(3) {p1,p4}

- Respectively designate a possible:
  - ⋆ (1) society,
  - ⋆ (2) three intracommunity networks and
  - ⋆ (3) an intercommunity network.

- The mereology of the community, society, intracommunity and intercommunity concepts transpire from the above use of the powerset operator (**-set**) and the "such that" constraints.
- The attributes of the community, society, intracommunity and intercommunity concepts amounts to the cardinality of the involved sets.

■

Cartesian Compounds

Next we give an example illustrating Cartesian compounds.

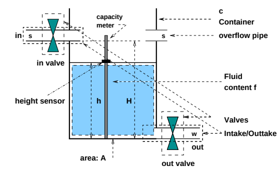

Figure 5.8: A boxlike fluid container system

Example 5.53 *A Fluid Container Configuration:*

- We illustrate a boxlike fluid tank in Fig. 5.8.
- We have the *physically manifest phenomena:*
  - ⋆ The whole tank, as illustrated, is a physically manifest phenomenon.
  - ⋆ We can, and will, claim that so are the
    - ⋄ in(take), overflow and out(take) valves,
    - ⋄ the height sensor, and
    - ⋄ the liquid (if any) in the tank.
- And there are the *attributes:*
  - ⋆ The height (of the liquid) is an attribute,
  - ⋆ and so are the open/closed status of the in(take) and out(take) valves.

- We can speak of the "value" of the entire fluid container system as composed from context and state components.
- The context components are the fixed (valued) attributes of the physical fluid container:
  - ⋆ the width, depth and overall height, i.e., the physical (volume) measurements,
  - ⋆ the height, over the bottom of the container, of the in(take) valve,
  - ⋆ the height, over the bottom of the container, of the out(take) valve, and
  - ⋆ the height, over the bottom of the container, of the overflow pipe.

- The state components are the variable (valued) attributes of the physical fluid container:
  - ⋆ the height, over the bottom of the container, of the liquid level,
  - ⋆ and the open/closed states of the two valves.

We can formalise this:

type
 LCS = CONTEXT × STATE

- The fixed-valued context can be modelled as:

type
 CONTEXT = VOL × VALS × OFLOW
 VOL = wdth:**Real** × dpth:**Real** × hght:**Real**
 VALS = in_valve:(**Real**×DIAM) × out_valve:(**Real**×DIAM)
 OFLOW = **Real** × Diam
 DIAM = **Real**

- The variable-valued state can be modelled as:

type
STATE = liquid:LEVEL × input_valve:OC × output_valve:OC
LEVEL = **Real**
OC == open | closed

- OC stands for the open/close state of a valve.
- open and closed are atomic tokens, i.e., identifiers.
- By being different they denote different "values".

⋆ Let the various numerals below designate reals (of unit centimeters), then:

lcs: (ctx,sta)
ctx: (vol,vvs,ofw)
vol: (100.0,100.0,300.0)
vvs: ((250.0,5.0),(10.0,5.0))
ofw: (255.0,5.0)
sta: (221.0,(open,closed))

i.e.,
(((100.0,100.0,300.0),((250.0,5.0),(10.0,5.0)),(255.0,5.0)),(221.0,(open,closed

⋆ exemplifies a configuration.

- We thus consider the fluid container system to be an atomic entity.
- The attributes of the fluid container system amount to container
  - ⋆ width,
  - ⋆ depth and
  - ⋆ height,
- to valve
  - ⋆ height positions
  - ⋆ and diameters,
- to the open or closed state of the input and output valves,
- and to the height of the fluid.

■

List Compounds

The next example illustrates the list compound.

Example 5.54 *Train Journeys:*

- A train journey is an ordered list of two or more station visits.
- A station visit is a grouping of arrival time, station and platform names, and departure time.

type
Time, Sn, Pn
Journey = Sta_Visit*
Sta_Visit = arrival:Time×sta_name:Sn×pla_name:Pn×dept:Time

- The list (and Cartesian) expression:

$$< (a_1, s_1, p_1, d_1), (a_2, s_2, p_2, d_2), (a_3, s_3, p_3, d_3), (a_4, s4,p_4, d_4) >$$

- where suitably subscripted a,s,p and d stand for, respectively, arrival times, station names, platform numbers (or names), and arrival times,
- designate a journey starting at time d1, ending at time a4,
- from station s1 via station s2 and s3, in that order, and ending at station s4.

- The mereology of the train journey concept transpires from the above use of the list and Cartesian type constructors (*, ×).
- We thus consider the train journey concept to be a composite entity
- whose atomic subentities are station visits.
- We consider only one attribute of a train journey,
  - ⋆ namely its number of station visits.
- The attributes of atomic station visits are
  - ⋆ arrival time,
  - ⋆ station name,
  - ⋆ platform number and
  - ⋆ departure time.

■

Map Compounds

We give an example illustrating map compounds.

Example 5.55 *Computer File Directories:* This example is a "classic".

- A directory consists of
  - ⋆ zero, one or more uniquely named files,
  - ⋆ and zero, one or more uniquely named directories.
- So directories map
  - ⋆ file names into files and
  - ⋆ directory names into directories.
- Files, file names and directory names are further unexplained entities.

We can formalise this:

type
File, Fn, Dn
DIR = (Fn $\overrightarrow{m}$ File) × (Dn $\overrightarrow{m}$ DIR)

- Here A $\overrightarrow{m}$B denotes a map from unique A's to not necessarily unique B's.
- A map is like a function:
  - ⋆ Given a map m (say in A $\overrightarrow{m}$B)
  - ⋆ and an a in A, such that
  - ⋆ a is in the definition set of m,
  - ⋆ then m(a) (m applied to a) is some b.

Department of Computer Science and Engineering
Institute of Informatics and Mathematical Modelling
Technical University of Denmark
Richard Petersens Plads, DK-2800 Kgs.Lyngby, Denmark

- Example directories are:

([ ],[ ])
([fn↦file],[ ])
([fn↦file],[dn↦([ ],[ ])])
([fn1↦file1,fn1↦file1],[dn1↦([ ],[ ]),dn2↦([fn↦file],[ ])])
([fn1↦file1,fn1↦file1],[dn1↦([ ],[ ]),dn2↦([fn↦file],[dn↦([ ],[ ])])])

- Here fn, fn1 and fn2 are file names; file, file1 and file2 are files and dn, dn1 and dn2 are directory names.
- The general map expression [a1↦b1,a2↦b2,...,an↦bn],
- where all a1, a2, ... and an are distinct,
- denotes a map from ai to bi.

Department of Computer Science and Engineering
Institute of Informatics and Mathematical Modelling
Technical University of Denmark
Richard Petersens Plads, DK-2800 Kgs.Lyngby, Denmark

- The mereology of the directory concept transpires from the above use of the map and Cartesian type constructors ($\overrightarrow{m}$, ×).
- The attributes of a directory is
  - ⋆ the number and name of files
  - ⋆ and the number and names of subdirectories.

■

Department of Computer Science and Engineering
Institute of Informatics and Mathematical Modelling
Technical University of Denmark
Richard Petersens Plads, DK-2800 Kgs.Lyngby, Denmark

Type Constraints

- In Example 5.52 the informal text lines between the formula lines illustrate the idea of a *type constraint*.
- These type constraints can be considered part of the mereology of the entities modelled.

Characterisation 5.109 By a *type constraint* we shall loosely understand

- some text which delimits a prior type description
- to not contain all the values
- otherwise allowed just by that prior type description

. ■

Department of Computer Science and Engineering
Institute of Informatics and Mathematical Modelling
Technical University of Denmark
Richard Petersens Plads, DK-2800 Kgs.Lyngby, Denmark

Example 5.56 *Type Constraints:*

- We will illustrate some type constraints that should have been expressed in earlier examples:
  - ⋆ *Liquid container system:*
    - ◇ The in(take) valve must be placed higher than the out(take) valve.
    - ◇ The liquid height cannot be lower than the level of the out(take) valve minus the diameter of that valve.
    - ◇ The liquid height cannot be higher than the level of the overflow pipe plus/minus the diameter of that pipe.
    - ◇ The diameter of the out(take) valve should stand in the following relation to the diameter of the in(take) valve: . . . (etc.!).

Department of Computer Science and Engineering
Institute of Informatics and Mathematical Modelling
Technical University of Denmark
Richard Petersens Plads, DK-2800 Kgs.Lyngby, Denmark

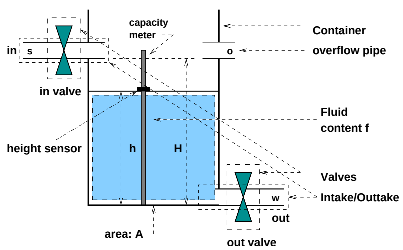

Figure 5.9: A Liquid Container System: The Physics

Department of Computer Science and Engineering
Institute of Informatics and Mathematical Modelling
Technical University of Denmark
Richard Petersens Plads, DK-2800 Kgs.Lyngby, Denmark

- ⋆ *Train journeys:*
  - ◇ Arrival times should be before departure times at any one station,
  - ◇ and their difference should be a minimum of t_{lo} minutes and a maximum of t_{hi} minutes, possibly depending on which station these times relate to.
  - ◇ Departure times at one station should be before arrival times at any next station,
  - ◇ and their difference should be commensurate with the normal travel time between such stations.
  - ◇ The sequence of station names must be commensurate with a route through the railway net.[4]
  - ◇ No station name can occur more than once. (That is, no cycles.)

Department of Computer Science and Engineering
Institute of Informatics and Mathematical Modelling
Technical University of Denmark
Richard Petersens Plads, DK-2800 Kgs.Lyngby, Denmark

- ⋆ *Computer directories:* One might envisage some strictly not necessary constraints on directories:
  - ◇ At any level of a directory, file names (at that level) must be distinct from directory names (at that level).
  - ◇ Given a directory $\Delta = (f, \delta)$, let a valid path p of directory names (of that directory Δ) be a nonempty sequence (i.e., a list) of directory names
    - ∘ such that the first directory name, d, of path p, say $p=\langle d\rangle \frown p'$, is indeed the name of a directory (Δ') of δ, and
    - ∘ such that if the path p is of length two or more, the remaining path p' is a valid path of Δ'.

■

Department of Computer Science and Engineering
Institute of Informatics and Mathematical Modelling
Technical University of Denmark
Richard Petersens Plads, DK-2800 Kgs.Lyngby, Denmark

- As the last example shows, it may sometimes be useful to express the constraints by a bit of set, Cartesian, list, map, and, as we shall soon see, function notation.
- Hence we presented some of that set, Cartesian, list and map notation.

[4] That is: Station names must name stations of the rail net, and the journey route must be a route of that net.

Summary: Principles, Techniques and Tools

Principles 5.27 The principle of analysing and modelling *entities* is as follows:

- First decide whether an entity, that is, whether values of a class, i.e., type, of entities are atomic or composite.
- Then, for atomic entities decide on which one or more attributes these atomic entities have.
- And, for composite entities, decide on which of the following two aspects these composite entities have:
  - ⋆ their mereology, i.e., their compositional attributes
  - ⋆ and their subentities

. ■

Recall that the attributes of a composite entity have two facets:

- there are the attributes which indicate how the composite entity is composed;
- and there are other, we could call them the auxiliary attributes.
- The compositional attributes are expressed by saying that the composite entity
  - ⋆ consists of a set
  - ⋆ or a Cartesian
  - ⋆ or a list
  - ⋆ or a set of uniquely identified, that is, a map

  of subentities.
- The auxiliary attributes of a composite entity are very much like the attributes of atomic entities:
  - ⋆ they characterise the values of the entities as such, less their possible subentities.

Techniques 20 *Entities* are modelled as follows:

- Atomic entities by stating an arbitrary aggregation of possibly constrained types (say sorts),
- and composite entities by stating two things:
  - ⋆ an arbitrary aggregation of possibly constrained types (the auxiliary attributes, e.g., sorts), and
  - ⋆ a specific set, Cartesian, list, map (or function) composition of types (the compositional attributes)

. ■

Tools 5.10 Models of *entities* are, for example, expressed using the RSL abstract type (sort) or concrete type concept. ■

Topic 21
Functions

Characterisation 5.110 By a *function* we shall loosely understand

- something, a mathematical quantity (that no one has ever seen), which
- when *applied* to something (else),
- called an *argument* of the function,
- *yields* something (yet else),
- called a *result* of the function for that argument.
- If the function is applied to something which is not a proper *argument* of the function,
- then the totally undefined result, called **chaos,** is yielded

. ■

- The question, for us, when confronted, as we are,
- with phenomena of one kind or another, is
- to decide which phenomena we should model as functions, and which we should not.

Example 5.57 *Functions:* We give some rough sketches of examples:

- To deposit savings in a savings account can be viewed as a function:
  - ⋆ the *deposit* function is applied to two arguments,
    - ◇ the deposit *amount*
    - ◇ and the account *balance.*
  - ⋆ The function yields a new *balance.*

- The above is an appropriate choice of argument and result types if we consider only the account balance (and the amount deposited).
  - ⋆ If, however, we consider the entire bank (and the amount deposited),
  - ⋆ then the *deposit* function is more appropriately applied to the following arguments,
    - ◇ the *bank,*
    - ◇ the *client name,*
    - ◇ the *client account number* (hence the account *balance*) and
    - ◇ the deposit *amount,*
  - ⋆ and yields a new *bank.*

- If you wish to consider
  - ⋆ some response to the client,
    - ◇ for example that the deposit transaction succeeded, or did not succeed,
    - ◇ depending, for example, on the validity of client name and account number,
  - ⋆ then you may wish to add a further, the response, component to the result, for example,
    - ◇ `transaction succeeded` or
    - ◇ `transaction did not succeed`.

Department of Computer Science and Engineering
Institute of Informatics and Mathematical Modelling
Technical University of Denmark
Richard Petersens Plads, DK-2800 Kgs.Lyngby, Denmark

Another example:

- Inquiring about and actually buying an airplane ticket is abstracted as a function. For example, this can be viewed as:
  - ⋆ The *purchase* function
  - ⋆ is applied to three arguments,
    - ◇ *travel information* about from where to where, on which flight, etc.,
    - ◇ the airline *flight reservations register,*
    - ◇ and the *price* (in terms of monies).
  - ⋆ This function yields a "paired" result:
    - ◇ the *ticket,*
    - ◇ and an updated *flight reservations register.*

Department of Computer Science and Engineering
Institute of Informatics and Mathematical Modelling
Technical University of Denmark
Richard Petersens Plads, DK-2800 Kgs.Lyngby, Denmark

Another example:

- To unload a ship in harbour at a quay can be considered a function. For example, viewed as:
  - ⋆ The *unload* function
  - ⋆ is applied to two (composite) arguments,
    - ◇ a *ship* (loaded with cargo destined for)
    - ◇ the *quay* of a harbour.
  - ⋆ This function yields a "paired" result:
    - ◇ the *ship* less its unloaded cargo, and
    - ◇ the *quay* "plus" the unloaded cargo.

Department of Computer Science and Engineering
Institute of Informatics and Mathematical Modelling
Technical University of Denmark
Richard Petersens Plads, DK-2800 Kgs.Lyngby, Denmark

Yet another example:

- To admit a patient at a hospital can be viewed as a function. For example, viewed as,
  - ⋆ the *admission* function
  - ⋆ is applied to a number of arguments,
    - ◇ the *patient,*
    - ◇ the *hospital* (not registering that patient),
    - ◇ the receiving *medical doctor, nurse,* etc.
  - ⋆ This function yields a "composite" result:
    - ◇ an updated *hospital* (now registering that patient).

Department of Computer Science and Engineering
Institute of Informatics and Mathematical Modelling
Technical University of Denmark
Richard Petersens Plads, DK-2800 Kgs.Lyngby, Denmark

A final example:

- To land an aircraft can be viewed as a function. For example, viewed as,
  - ⋆ the *touchdown* function
  - ⋆ is applied to two arguments,
    - ◇ the *aircraft* (which is in a state of flying),
    - ◇ and the *runway* (which is assumed free for landing, i.e., reserved for "that" aircraft).
  - ⋆ This function yields a "composite" result:
    - ◇ the *aircraft* (which is now in a state of running along the runway),
    - ◇ and the *runway* (which is occupied by "that" aircraft).

■

Department of Computer Science and Engineering
Institute of Informatics and Mathematical Modelling
Technical University of Denmark
Richard Petersens Plads, DK-2800 Kgs.Lyngby, Denmark

- An important task in describing, prescribing or specifying (domains, requirements, respectively software design)
- is that of identifying all relevant functions,
- of (i) naming them, their (ii) arguments and (iii) results,
- and of (iv) defining what they "compute".
- In the above example we have covered (i–iii) but not (iv).

Department of Computer Science and Engineering
Institute of Informatics and Mathematical Modelling
Technical University of Denmark
Richard Petersens Plads, DK-2800 Kgs.Lyngby, Denmark

Function Signatures

Characterisation 5.111 By a *function signature* we shall understand the following composite information:

- The name of the function,
- a sequence of names of the types of the arguments and
- a sequence of names of the types of the yielded results

. ■

- A function signature is not a definition of the function.
- But it is a significant indication of what the function "appears to be about".

Department of Computer Science and Engineering
Institute of Informatics and Mathematical Modelling
Technical University of Denmark
Richard Petersens Plads, DK-2800 Kgs.Lyngby, Denmark

Example 5.58 *Function Signatures:* The signatures corresponding to the functions mentioned in Example 5.57, are:

- Function name: *deposit;* argument types: *amount* and *balance;* and result type: *balance.*
- Function name: *ticket_purchase;* argument types: *travel information, reservation register* and *price;* and result types: *reservation register* and *ticket.*
- Function name: *unload;* argument types: *ship* and *quay;* and result type: *ship,* and *quay.*
- Function name: *admission;* argument types: *patient, hospital* and *medical staff;* and result type: *hospital.*
- Function name: *touchdown;* argument types: *aircraft* and *runway,* and result types: *aircraft* and *runway.*

Department of Computer Science and Engineering
Institute of Informatics and Mathematical Modelling
Technical University of Denmark
Richard Petersens Plads, DK-2800 Kgs.Lyngby, Denmark

We formalise the above:

type
  Amount, Balance
  TravInfo, ReservReg, Price, Ticket
  Ship, Quay
  Patient, Hospital, MedicalStaff
  Aircraft, RunWay
value
  deposit: Amount×Balance → Balance
  ticket_purch: TravInfo×ReservReg×Price → ReservReg×Ticket
  unload: Ship×Quay → Quay×Ship
  admission: Patient×Hospital×MedicalStaff → Hospital
  touch_down: Aircraft×RunWay → Aircraft×RunWay

■

- Once a function is rough-sketch identified
- the developer can determine, at least tentatively,
- the function signatures.
- To do so does indeed often require detailed considerations.
- Delineating the function signatures focuses the developer's mind.
- Many issues surface during this preliminary rough-sketch step.
- Writing down, systematically, whether informally only, or also formally, tends to further focus the development:
- step-by-step "achievements" can be recorded!

Function Definition

- We have not spent much time or space, above, but we have indeed mentioned the concept of function definition.

Characterisation 5.112 By a *function definition* we shall understand

- a description, a prescription, or a specification
- which defines the relationship between
  - ⋆ arguments and
  - ⋆ results
- of a function

. ■

Example 5.59 *Function Definition:* We exemplify function definitions for some of the functions of Examples 5.57 and 5.58.

- The deposit function,
  - ⋆ when applied to an amount (to be deposited) and a(n account) balance,
  - ⋆ yields a new balance (of that account)
  - ⋆ which is the sum of the original balance and the deposited amount.

- The air ticket purchase function,
  - ⋆ when applied to
    - ◇ some (relevant) travel information,
    - ◇ an airline reservations register,
    - ◇ and a(n assumed ticket) price,
  - ⋆ yields
    - ◇ a new airline reservations register
    - ◇ and a ticket
    - ◇ such that the ticket satisfies the travel information
    - ◇ and the ticket (i.e., the reservation that it designates) is properly reflected in the yielded, i.e., new airline reservations register.

  More information must be given about the three entities: travel information, airline reservations register and ticket, in order to define what is meant by "satisfaction" and "proper reflection".

- The unload ship function,
  - ⋆ when applied to
    - ◇ a ship
    - ◇ and a quay,
  - ⋆ yields
    - ◇ a(n updated) quay which,
      - ∘ in addition to the cargo that was already on the quay before unloading,
      - ∘ now also stores that cargo from the ship, which was *destined* for that quay,
    - ◇ and an updated ship
      - ∘ that is less that cargo that has been unloaded,
      - ∘ and only less that cargo!


```
type
  Sn, Qn /∗ Ship and Quay Designators ∗/
  C /∗ Container ∗/
variable
  s_c:C-set
  q_c:C-set
value
  q:Qn
  is_destined: Qn × C → Bool
  unload: C-set × C-set → C-set × C-set
  unload(scs,qcs) as (scs′,qcs′)
    post
      scs \ scs′ = qcs′ \ qcs
      ∀ c:C • c ∈ scs \ scs′ ⇒ is_destined(qn,c) ∧
      ∼∃ c:C • c ∈ scs′ ∧ is_destined(qn,c)
```

■

- We observe that attempts to complete function definitions
- may be frustrated by lack of sufficient details about the types and attributes of arguments and results.
- Another way of formulating the last sentence above is:
  - ⋆ Defining functions help us to "discover"
    - ◇ the type and attributes of arguments and results,
    - ◇ the possible need for auxiliary functions (viz.: 'satisfaction' and 'proper reflection' in Example 5.59 second bullet).
  - ⋆ Often one must be prepared to revise function signatures
  - ⋆ when attempting to complete function definitions.

Algorithms

- Care must be taken to distinguish between
  - ⋆ specifying function definitions, and
  - ⋆ defining algorithms, including coding programs that implement function definitions.
- Function definitions are here thought of as being abstract, as emphasising what is to be computed,
- in contrast to algorithms (and code) which emphasises how a computation might achieve what the function definitions specify.

Characterisation 5.113 By an *algorithm* we shall understand

- a step-by-step specification
- which can serve as prescription for computation by a mechanical device, i.e., a computer,
- such that that computer computes what a function definition otherwise has defined

. ■

Example 5.60 *Algorithm:* We finally indicate rough sketches for algorithms for computing the deposit function (item (i)), respectively the ship unload function (item (iii)), of Example 5.59.

- *A deposit algorithm:*
  - ⋆ Let the amount to be deposited be held in a variable v_d, and let the initial contents of v_d be d.
  - ⋆ Let the balance of the account into which a deposit is to take place be held in a variable v_b, and let the initial contents of v_b be b.
  - ⋆ Now add the contents d of v_d to the contents b of v_b, yielding $d + b$ as the new contents of variable v_b.
  - ⋆ Nothing is said about the final contents of v_d.

- *An unload algorithm:*
  - ⋆ Let the cargo area of ship s be represented by a variable s_c. The contents of s_c is assumed to be a set of containers $\{s_{c_1}, s_{c_2}, \ldots, s_{c_n}\}$.
  - ⋆ Let the cargo area of the quay, q, at which the ship is docked be represented by a variable q_c. The contents of q_c is assumed to be a set of containers $\{q_{c_1}, q_{c_2}, \ldots, q_{c_m}\}$.
  - ⋆ Let an auxiliary predicate function is_destined apply to a quay designator q and ship container s_{c_i}, and let it yield **true** if that container is destined for quay q, **false** otherwise.
  - ⋆ Now, for every ship s container s_{c_j} of s_c for which $q(q, s_{c_j})$ holds, remove s_{c_j} from s_c and add s_{c_j} to q_c.


```
type
  Sn, Qn /* Ship and Quay Designators */
  C /* Container */
variable
  s_c:C-set
  q_c:C-set
value
  q:Qn
  is_destined: Qn × C → Bool
  unload: C-set × C-set → C-set × C-set
  unload(s_c,q_c) ≡
    while ∃ c:C•c ∈ s_c ∧ is_destined(qn,c) do
      let c:C•c ∈ s_c ∧ is_destined(qn,c) in
      s_c := s_c \ {c} ‖ q_c := q_c ∪ {c}
    end end
```


Our informal algorithm presentation could have been presented in the form of a pseudo-program:

```
action:
  name: unload
global variables:
  q: Quay_name,
  s_cs: Ship_containers,
  q_cs: Quay_containers.
pseudo-program:
  while there exists containers in s_cs destined for quay q
    do
      let c be a container in s_cs destined for q:
      remove c from s_cs;
      add c to q_cs
    end
```

Pseudo-programs can be expressed in a rich variety of forms. ■

- We leave it to the course student to examine the three definitions,
  - ⋆ that of the unload function,
  - ⋆ that of the unload algorithm and
  - ⋆ that of the unload pseudo-program.
- We believe that the difference between those definitions
- shows aspects of the difference, in general, between
  - ⋆ function definitions and
  - ⋆ algorithm specifications.

Topic 22
Events and Behaviours
States, Actions, Events and Behaviours

Characterisation 5.114 By a *state* we shall loosely understand

- a collection of one or more entities
- whose value may change

. ■

Characterisation 5.115 By an *action* we shall loosely understand

- something which changes a state

. ■

Characterisation 5.116 By an *event* we shall loosely understand

- the occurrence of something
- that may either trigger an action,
- or is triggered by an action,
- or alter the course of a behaviour,
- or a combination of these

. ■

Characterisation 5.117 By a *behaviour* we shall loosely understand

- a sequence of actions and events

. ■

Example 5.61 *States, Actions, Events and Behaviours:* We show some examples where the four concepts "intermingle". The below examples relate to subexamples (i, ii, iii) of Examples 5.57–5.60.

- *Clients and bank accounts:*
  - ⋆ The balance of some client's bank account forms a state.
  - ⋆ Carrying out the functions of depositing (and withdrawing) monies into (respectively from) the account amounts to actions.
  - ⋆ The decisions by a client to make deposits and withdrawals amount to events that trigger respective actions.

- ⋄ Assuming that there is a lower credit limit, the situation where a withdrawal action results in a bank account balance that exceeds the credit limit amounts to an event.
- ⋄ That event may or may not trigger an action, or may do so in a delayed fashion.

⋆ The sequence of a specific series of deposit and withdrawal events and actions forms a behaviour. For any given client and bank account many such behaviours are possible.

- *Airline ticket purchase:*
  - ⋆ The airline seat reservation register forms a state.
  - ⋆ Carrying out the functions of actually buying a (or cancelling an already bought) ticket amounts to actions.
  - ⋆ The decisions to buy (reserve without paying, or actually reserve and pay), respectively cancel, a ticket are events that trigger respective actions.
    - ⋄ Having reserved, without actually paying for a ticket, and then not paying for the ticket before a certain date, amounts to an event, which may, or may not trigger an action.
  - ⋆ The sequence of a specific series of one or more ticket purchases and zero, one or more ticket cancellation events and actions forms a behaviour. For any given airline reservation system and potential passengers many such interleave and/or concurrent behaviours are possible.

- *Ship unloading and quay loadings:*
  - ⋆ The ship cargo and the quay cargo storage can be considered either as a combined state, or, respectively, as the states, of a ship and a quay.
  - ⋆ Carrying out the function of unloading a ship amounts to an action.
  - ⋆ The decision to start unloading amounts to an event that triggers the unload action.
    - ⋄ The phenomenon that there is no more cargo to unload amounts to an event, which may or may not trigger an action.

- ⋆ Behaviours:
  - ⋄ The quay behaviour:
    - ◦ Seen from the point of view of a quay, the sequence of a specific series of unload events and actions, with respect to possibly different ships, forms a behaviour.
    - ◦ For any given quay many such behaviours are possible.
  - ⋄ The ship behaviour:
    - ◦ Seen from the point of view of a ship, the sequence of a specific series of unload events and actions, with respect to possibly different quays, forms a behaviour.
    - ◦ For any given ship many such behaviours are possible.
  - ⋄ The combined quay/ship behaviour:
    - ◦ Any set of pairs of commensurate quay and ship behaviours forms a behaviour.

■

Synchronisation and Communication

- From the above we observe two closely related phenomena:
  - ⋆ That behaviours may *communicate,*
  - ⋆ and that the communications may take place
    - ⋄ *synchronously,* or
    - ⋄ *asynchronously.*

Characterisation 5.118 By *communication* we loosely mean

- the exchange of entities between behaviours,
- from one to the other, or both ways

. ■

- A communication may involve one "sender", i.e., output,
- and one or more "receivers", i.e., inputs.

Characterisation 5.119 By *synchronous communication* we loosely mean

- the simultaneous communication
- between behaviours
- of one entity from one sender to one or more receivers

. ■

Department of Computer Science and Engineering
Institute of Informatics and Mathematical Modelling
Technical University of Denmark
Richard Petersens Plads, DK-2800 Kgs.Lyngby, Denmark

Characterisation 5.120 By a *shared event* we mean

- the simultaneous occurrence of one output event in one sender behaviour
- with its one or more synchronously communicating input events in one or more receiver behaviours

. ■

- Thus synchronous communication can be said to be via a zero-capacity buffer between the sending and the receiving behaviours.
- Whether the synchronous communication is between one sender and one receiver, or several receivers is not stated here —
- but any description (prescription, specification) must state that,
- as it must state how the identification of the sending and receiving behaviours is accomplished.
- The sender behaviour is thus expected to be "held up" from the moment it wishes to synchronously communicate till the moment all communications have been accomplished.

Characterisation 5.121 By *asynchronous communication* we shall loosely understand

- the possibly delayed, e.g., buffered, communication
- between behaviours
- of one entity from one sender to one or more receivers

. ■

- Thus asynchronous communication can be said to be via a non-zero-capacity buffer between the sending and the receiving behaviours.
- Whether the buffer acts like a queue or like a heap we do not specify here —
- but any description (prescription, specification) must state that,
- as it must state how the identification of the sending and receiving behaviours is accomplished.
- The sender behaviour is thus expected to be able to proceed with its "own" actions once it has placed its asynchronous communication.

Characterisation 5.122 By *synchronisation* we shall loosely understand

- the explicitly expressed (i.e., controlled)
- simultaneous occurrence of an
- event in two or more behaviours

. ■

- Thus we consider the output communication in (i.e., from) one sender behaviour
- to designate the same event
- as the simultaneous input communication(s) in one (or more) receiver behaviour(s).
- We say that synchronisation reflects shared events.

Example 5.62 *Communications:* We continue our line of examples with subexamples (i, iii, v) of Examples 5.57–5.61.

- When the bank account holder, i.e., the client,
  - ⋆ hands over the monies to be deposited to the bank teller,
  - ⋆ then we say that an output/input event has occurred,
  - ⋆ that the monies are being communicated
  - ⋆ and that the communication is synchronous.

- When a ship
  - ⋆ unloads a (piece of) cargo (i.e., a container) onto a quay,
  - ⋆ then we say that an output/input event has occurred,
  - ⋆ that the cargo is being communicated
  - ⋆ and that the communication is synchronous.

Department of Computer Science and Engineering
Institute of Informatics and Mathematical Modelling
Technical University of Denmark
Richard Petersens Plads, DK-2800 Kgs.Lyngby, Denmark

- When a landing aircraft
  - ⋆ touches the ground,
  - ⋆ then we say that an output/input event has occurred,
  - ⋆ that the concept "touchdown" is being communicated
  - ⋆ and that the communication is synchronous.

■

Processes

Characterisation 5.123 By a *process* we shall loosely understand

- the same as a behaviour

. ■

- We shall, at times, make a pragmatic distinction:
  - ⋆ behaviours are what we observe, in the domain,
  - ⋆ while processes are what computers facilitate.
- We may implement a few, many, or all observable actions and events of behaviours in terms of computer processes.

Example 5.63 *Processes:*

- We focus just on the ship unloading example:
  - ⋆ We now consider the unloading as that of the interaction between two behaviours, two processes: ship and quay.
  - ⋆ The ship:
    - ◇ The ship recurrently examines all its cargo.
    - ◇ For every cargo destined for a designated quay,
    - ◇ that cargo is removed from the ship's cargo,
    - ◇ and communicated to the quay.
    - ◇ When it contains no more such cargo, the ship "goes on to do other business" — which is not specified.

- ⋆ The quay:
  - ◇ The quay recurrently accepts all communications of cargo,
  - ◇ from whichever ship.
  - ◇ And it keeps doing that "ad infinitum"!


```
type C, Qn
channel sq:C
value
  qn:Qn
  is_destined: Qn × C → Bool
  ship: C-set → out sq Unit
  ship(cs) ≡
    if ∃ c:C • c ∈ cs ∧ is_destined(qn,c)
      then let c:C • c ∈ cs ∧ is_destined(qn,c) in sq ! c; ship(cs\{c}) end
      else skip
    end
  quay: C-set → in sq Unit
  quay(cs) ≡ let c = sq ? in quay(cs ∪ {c}) end
```

Please observe the close correspondence between the informal and the formal explication above. ■

Traces

Characterisation 5.124 By a *trace* we shall loosely understand

- almost the same as a behaviour:
  - ⋆ a sequence of actions (usually action identifications)
  - ⋆ and (usually action identifications) events
  - ⋆ of a single process.

- Since, say, the parallel composition of two or more behaviours also forms a behaviour, hence a process, we need to clarify the notion of trace for such composite behaviours.
  - ⋆ Either, in a trace, we focus on the events shared between two or more behaviours,
  - ⋆ or we also include the actions of each of these behaviours

. ■

Example 5.64 *Traces:* We continue subexample (iii) of Examples 5.57–5.63.

- The event trace of a ship in harbour, more precisely, during unloading, is a finite sequence of zero, one or more unload events.
- The event trace of a harbour quay, with respect to ship unloadings, is an indefinite length sequence of zero, one or more unload events.

■

Department of Computer Science and Engineering
Institute of Informatics and Mathematical Modelling
Technical University of Denmark
Richard Petersens Plads, DK-2800 Kgs.Lyngby, Denmark

Process Definition Languages

- The formal part of Example 5.63 illustrated textual definitions of processes.
- There are also a number of two-dimensional, diagrammatic ways of "rendering" the progress and interaction of two or more processes.
- These were covered extensively in Vol. 2's Chaps. 12–14:
  - ⋆ Various forms of *Petri nets* (*condition event nets, place transition nets,* and *coloured Petri nets*),
  - ⋆ *message* and *live sequence charts* (MSCs and LCSs) and
  - ⋆ *statecharts*.
- Variations of these diagrammatic forms are currently embodied in `UML`.

Department of Computer Science and Engineering
Institute of Informatics and Mathematical Modelling
Technical University of Denmark
Richard Petersens Plads, DK-2800 Kgs.Lyngby, Denmark

Topic 23
Choice on Modelling Phenomena and Concepts

- On one hand, we have a universe of discourse conceived of in terms of phenomena and concepts.
- On the other hand, we have some principles, techniques and tools for modelling phenomena and concepts.
- Previous lectures have covered the model concepts of
  - ⋆ entities,
  - ⋆ functions,
  - ⋆ events, and
  - ⋆ behaviours.

Department of Computer Science and Engineering
Institute of Informatics and Mathematical Modelling
Technical University of Denmark
Richard Petersens Plads, DK-2800 Kgs.Lyngby, Denmark

- Confronted with a description (prescription, specification) task there are a number of modelling questions.
- Some we covered in other lecture series.
- Some of these will be covered in this lecture.
- Others will be covered in several of the remaining lectures.

Department of Computer Science and Engineering
Institute of Informatics and Mathematical Modelling
Technical University of Denmark
Richard Petersens Plads, DK-2800 Kgs.Lyngby, Denmark

Qualitative Characteristics

- In which order do we describe (prescribe, specify) phenomena and concepts?
- Do we model them according to whether they are modelled as
  - ⋆ entities, or functions, or events or behaviours?
- The answer to these questions is relatively simple:
  - ⋆ It depends on which phenomena and concepts that
  - ⋆ best "characterise" the universe of discourse being described (prescribed, specified)!

Department of Computer Science and Engineering
Institute of Informatics and Mathematical Modelling
Technical University of Denmark
Richard Petersens Plads, DK-2800 Kgs.Lyngby, Denmark

Quantitative Characteristics

- Thus the describer (prescriber, specifier) is confronted with another question:
  - ⋆ What does it mean that
    - ⋄ a phenomenon or concept
    - ⋄ characterises a universe of discourse
    - ⋄ "better" than another phenomenon or concept?
- The answer is, basically, a matter of style and of taste.
- But we shall try formulate some guidelines.

Department of Computer Science and Engineering
Institute of Informatics and Mathematical Modelling
Technical University of Denmark
Richard Petersens Plads, DK-2800 Kgs.Lyngby, Denmark

- To do so we introduce a notion of universe of discourse "intensity".
- The background for our "intensity" notion is the qualitative characterisation of a phenomena or a concept as being (modelled as):
  - ⋆ an entity
  - ⋆ a function
  - ⋆ an event
  - ⋆ a behaviour

Department of Computer Science and Engineering
Institute of Informatics and Mathematical Modelling
Technical University of Denmark
Richard Petersens Plads, DK-2800 Kgs.Lyngby, Denmark

- Correspondingly we postulate that one can quantitatively characterise a phenomenon or a concept as one or more of:
  - ⋆ *information-intensive*
  - ⋆ *function-intensive*
  - ⋆ *event-intensive*
  - ⋆ *process-intensive*
- Usually it only makes sense to speak of intensity if a phenomenon or a concept is predominantly one of the above, or, at the very most, two.
- If it is three, or all, then we would claim that it has no intensity!

Department of Computer Science and Engineering
Institute of Informatics and Mathematical Modelling
Technical University of Denmark
Richard Petersens Plads, DK-2800 Kgs.Lyngby, Denmark

Information-Intensive Universe of Discourse

Characterisation 5.125 By a universe of discourse being said to be *information-intensive* we roughly mean that

- entities, their number and type
- play a central role in understanding that universe of discourse

. ∎

Department of Computer Science and Engineering
Institute of Informatics and Mathematical Modelling
Technical University of Denmark
Richard Petersens Plads, DK-2800 Kgs.Lyngby, Denmark

Example 5.65 *Information-Intensive Phenomena:*

- An insurance system, with its records of insurances, claims and decisions (wrt. claims), can be said to be an information-intensive system.
- A cadastral and cartographic system, with its many maps on properties and geographical areas, can be said to be an information-intensive system.
- The military intelligence agency of a country, with its many reports on supposed (military strengths and weaknesses, strategies and tactics, etc., of) enemies, can be said to be an information-intensive system.

■

Function-Intensive Universe of Discourse

Characterisation 5.126 By a universe of discourse being said to be *function-intensive* we roughly mean

- that functions, their definition, their application and their composition,
- play a central role in understanding that universe of discourse

. ■

Example 5.66 *Function-Intensive Phenomena:*

- Most resource planning, scheduling and allocation, and rescheduling systems (say, for production and for transport) can be said to be function-intensive.

■

Event-Intensive Universe of Discourse

Characterisation 5.127 By a universe of discourse being said to be *event-intensive* we roughly mean

- that events, that is, the synchronous or asynchronous exchange of entities between behaviours,
- play a central role in understanding that universe of discourse

. ■

Example 5.67 *Event-Intensive Phenomena:*

- A railway train traffic system with its movement of trains, their arrival and departure from stations, their reaching intermediate positions along a line, and the attendant rail-point interlocking and signal settings, can be said to be an event-intensive system.
- Similarly for air traffic systems: The arrival and departure of aircraft, in and out of ground, approach and regional monitoring and control zones, can be said to be event-intensive systems.
- Telephone exchange systems, with the handling of single- and multiple-party telephone calls, hang-ups, simple service-oriented inquiries, etc., can be said to be event-intensive systems.

■

Process Intensive Universe of Discourse

Characterisation 5.128 By a universe of discourse being said to be *process-intensive* we roughly mean

- that behaviours
- play a central role in understanding that universe of discourse

. ■

Example 5.68 *Behaviour-Intensive Phenomena:*

- Freight logistics systems with their handling and actual flow of freight by senders and logistics firms, and from senders via logistics firms, transport hubs, and transport conveyors to receivers, can be said to be behaviour-intensive systems.
- Healthcare systems, with their flow of people (patients and staff), material (medicine, etc.), information (patient medical records), and control, can be said to be behaviour-intensive systems.

- Harbours, railway stations and airports, with their handling of ship, train and aircraft arrivals and departures, assignment to quays, platforms and gates, unloading and loading, and servicing of a multitude of kinds: baggage, catering, fuel, cleaning, etc., can be said to be behaviour-intensive systems.

■

Discussion 5.18 Since events and behaviours are closely related, we shall find that both event- and behaviour-intensive systems give rise to implementations in terms of processes. ■

Principles, Techniques and Tools

Principles 5.28 • When describing a universe of discourse focus on identifying and describing *phenomena* and *concepts*

. ■

Techniques 21 • Analyse the identified *phenomenon* or *concept* and

• decide whether to model as an

⋆ entity, including as a set (i.e., type) of entities,

⋆ or as a function,

⋆ or as an event,

⋆ or as a behaviour.

• Then use the elsewhere given techniques (and tools) for respectively modelling entities, functions, events and behaviours

. ■

Discussion 5.19 • There are some dualities.

⋆ An entity can possibly be modelled as a function,

⋆ or an entity can possibly be modelled as a behaviour,

⋆ etc.

• We can hint at this as follows.

⋆ *Entity as function:* you may consider the phenomenon, p,

⋄ for example, modelled as an integer, as an entity, i_p,

⋄ or as the function, f_p.

⋄ As an entity i_p denotes a value.

⋄ As a function f_p is to be applied to an empty argument, $f_p()$, and then yields the value.

• *Entity as behaviour:* you may consider the phenomenon, p,

⋆ for example, modelled as an integer, as an entity, i_p,

⋆ or as the behaviour, b_p.

⋆ As an entity i_p denotes a value.

⋆ As a behaviour b_p is a behaviour composed, in parallel, with that of the phenomenon, i.e., an inquiring behaviour, q_p, which wishes to know, to use, the value of the integer phenomenon.

⋆ This is modelled as follows:

⋄ There are two behaviours,

∘ the inquiring behaviour q_p,

∘ and the integer behaviour b_p.

⋄ To ascertain the value of p behaviour q_p synchronises and communicates with b_p

⋄ by requesting the value of the integer phenomenon and

⋄ obtaining, in return, that value.

• Thus, whether we model a phenomenon, that appears to be an entity, as an entity, or as a function, or as a behaviour, is not always that straightforward.

⋆ If the phenomenon is otherwise simple, i.e., is not subject to the constraints listed for the next two alternatives, then model it as an entity.

⋆ If instead the phenomenon is shared among many behavioural phenomena then model it as a behaviour

. ■

Example 5.69 *Shared Documents:*

• A (perhaps even information-intensive) phenomenon can be thought of as a large collection of uniquely identified documents.

• Users of these documents may wish to read, copy or edit some document(s).

• We consider these users as a set of many behavioural phenomena,

• hence we model the document collection as a behaviour.

• This document collection behaviour is centred around an internal entity: the document storage.

• User behaviours now communicate with the document collection behaviour,

⋆ by requesting copies of documents for reading, and hence no return,

⋆ by requesting copies of documents for copying, and hence no return, or

⋆ by requesting original documents for editing, and hence for subsequent return.

• Requests are communicated from a user behaviour to the document collection behaviour.

• Responses are communicated from the document behaviour to the requesting user behaviour.

• Document returns are communicated from a user behaviour to the document collection behaviour.

■

Tools 5.11 • The tools for *modelling entities* are those of sorts and concrete types, as well as axioms, i.e., constraints over these.

• The tools for modelling functions are those of function signatures and function definitions, whether by explicit (i.e., function body) definition, or by pre/postconditions, or by axioms.

Department of Computer Science and Engineering
Institute of Informatics and Mathematical Modelling
Technical University of Denmark
Richard Petersens Plads, DK-2800 Kgs.Lyngby, Denmark

- The tools for modelling behaviours are either textual or diagrammatic, or a combination of these.
  - ⋆ Typically the textual tool is, as in these lectures, the `RSL/CSP` specification language.
  - ⋆ The diagrammatic tools are either one or more of
    - ◇ various forms of *Petri nets* (*condition event nets, place transition nets,* and *coloured Petri nets*),
    - ◇ *message* and *live sequence charts* and
    - ◇ *statecharts*

. ■

Department of Computer Science and Engineering
Institute of Informatics and Mathematical Modelling
Technical University of Denmark
Richard Petersens Plads, DK-2800 Kgs.Lyngby, Denmark

Topic 24
Discussion
Entities, Functions, Events and Behaviours

- We have postulated that the concepts of
- entities, functions, events and behaviours
- offer a suitable complement of choices
- when modelling phenomena and concepts.
- How do we know that?
- From where do we know that?
- Well, pragmatics, i.e., experience, has shown that they suffice, to a large extent.
- And the four notions each fit into nicely complementing theories:
  - ⋆ of data types, of recursive function theory and of process algebras.

Department of Computer Science and Engineering
Institute of Informatics and Mathematical Modelling
Technical University of Denmark
Richard Petersens Plads, DK-2800 Kgs.Lyngby, Denmark

Intensity and Problem Frames

- The notion of intensity is very much a part of that of Michael Jackson's idea of *problem frame*.
- We consider the notion of intensity to form one dimension along which the broader notion of problem frames can be characterised.
- In the above intensity classes we did not, for example, distinguish between the types of entities, within an entity-intensive phenomenon or within any of the other intensity phenomena.
- Once we add that dimension to our categorisation, then we start moving closer to Michael Jackson's notion of problem frames.
- We shall take a deeper look at problem frames in a later lecture.