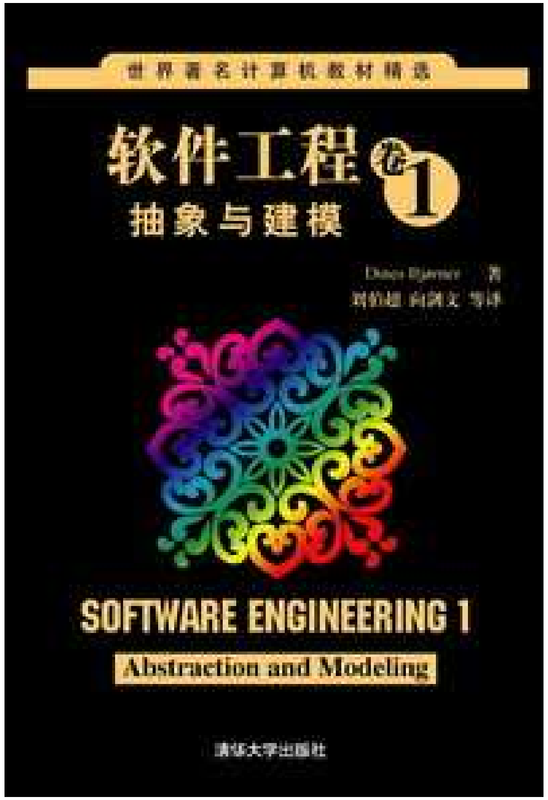

NICE TO SEE YOU BACK

Begin of Lecture 2: Last Session — Discrete Endurant Entities

Parts

FM 2012 Tutorial, Dines Bjørner, Paris, 28 August 2012

Tutorial Schedule

- **Lectures 1–2** 9:00–9:40 + 9:50–10:30

- **Lectures 3–5** 11:00–11:15 + 11:20–11:45 + 11:50–12:30

Lunch **12:30–14:00**

- **Lectures 6–7** 14:00–14:40 + 14:50–15:30

- **Lecture 8–9** 16:00–16:40 + 16:50–17:30

2. Domain Entities

- The world is divisible into two kinds of people:
  - those who divide the population into two kinds of people
  - and the others.
- In this tutorial we shall divide the phenomena we can observe and whose properties we can ascertain into two kinds:
  - the **endurant entities** and
  - the **perdurant entities**.
- Another "division" is of the phenomena and their properties into
  - the **discrete entities** and
  - the **continuous entities**.
- You can have it, i.e., the the analysis and the presentation, either way.

- By a **domain** we shall understand a suitably delineated set of observable entities and abstractions of these, that is, of
  - discrete parts and
  - continuous materials and,
  - discrete actions
    (operation applications causing state changes),
  - discrete events
    ("spurious" state changes not [intentionally] caused by actions),
  - discrete discrete behaviours
    (seen as sets of sequences of actions, events and behaviours) and
  - continuous behaviours
    (abstracted as continuous functions in space and/or time).

2.1. From Observations to Abstractions

- When we observe a domain we observe **instances** of **entities**;
- but when we describe those instances
  - (which we shall call **values**)
  - we describe, not the values,
  - but their **type** and **properties**.
    - Parts and materials have **types** and **values**;
    - actions, events and behaviours, all, have **types** and **values**, namely as expressed by their **signatures**; and
    - actions, events and behaviours have **properties**, namely as expressed by their **function definitions**.

© Dines Bjørner 2012, DTU Informatics, Techn.Univ.of Denmark – July 31, 2012: 09:02

2.2. Algebras

- **Algebra:** Taking a clue from mathematics, an algebra is considered
  - a set of endurants:
    - a set of parts and
    - a set of materials

  and

  - a set of perdurants: operations on entities.

  These operations yield parts or materials.

- With that in mind we shall try view a domain as an algebra, of some kind, of
  - parts and
  - actions, events and behaviours.

2.3. Phenomena

- **Phenomena:** By a **phenomenon** we shall understand
  - something that can be observed by the **human senses**
  - or by **equipment** based on laws of physics and chemistry.
- Those phenomena that can be observed by
  - the human eye or
  - touched, for example, by human hands
  - we call **parts** and **materials**.
- Those phenomena that can be observed of parts and materials
  - can usually be measured
  - and we call them **properties** of these **parts** and those **materials**.

© Dines Bjørner 2012, DTU Informatics, Techn.Univ.of Denmark – July 31, 2012: 09:02

2.4. Entities

- Ontologically we distinguish between two kinds of domain entities:
  - endurant entities and
  - perdurant entities.
- We shall characterise these two terms:
  - endurants on Slide 49 and
  - perdurants on Slide 144.
- This distinction is supported by current literature on ontology [BarrySmith1993].
- In this section of this lecture we shall not enter a discourse on
  - *"things"*,
  - *objects*,
  - *entities*,
  - etcetera.

2.4.1. A Description Bias

- One of several "twists"
  - that make the TripTych form of domain engineering
  - distinct from that of **ontological engineering**
  - is that we use a **model-oriented formal specification approach**[12]
  - where usual **ontology formalisation language**s are variants of Lisp's [Lisp1] S-expressions.
  - KIF: Knowledge Interchange Format, http://www.ksl.stanford.edu/knowledge-sharing/kif/ is a leading example.

[12] RAISE [RaiseMethod]. Our remarks in this section apply equally well had we instead chosen either of the Alloy [alloy], Event B [JRAbrial:TheBBooks], VDM [e:db:Bj78bwo,e:db:Bj82b,jf-pgl-97] or Z [m:z:jd+jcppw96] **formal specification languages**.

© Dines Bjørner 2012, DTU Informatics, Techn.Univ.of Denmark – July 31, 2012: 09:02

- The bias is now this:
  - The **model-oriented languages** mentioned in this section all share the following:
    - (a) a **type concept** and facilities for defining types, that is: endurants (parts), and
    - (b) a **function concept** and facilities for defining functions (notably including **predicates**), that is: perdurants (actions and events).
    - (c) RSL further has constructs for defining **processes**, which we shall use to model **behaviours**.

2.4.2. An 'Upper Ontology'

- By an **upper ontology** we shall understand
  - a relatively small, ground set of ontology expressions
  - which form a basis for a usually very much larger set of ontology expressions.

© Dines Bjørner 2012, DTU Informatics, Techn.Univ.of Denmark – July 31, 2012: 09:02

- The need for introducing the notion of an **upper ontology** arose, in the late 1980s to early 1990s as follows:
  - usually an ontology was (is) expressed in some very basic language, viz., `Lisp`-like `S`-expressions[13].
  - This was necessitated by the desire to be able to share ontologies between many computing applications worldwide.
  - Then it was found that several ontologies shared initial bases in terms of which the rest of their ontologies were formulated.
  - These shared bases were then referred to as **upper ontologies** — and a need to "standardise" these arose [ontology:guarino97a,StaabStuder2004].

[13] Ontology languages: `KIF` http://www.ksl.stanford.edu/knowledge-sharing/kif/-#manual, `OWL` [Ontology Web Language] [OWL:2009], ISO Common Logic [ISO:CL:2007]

- We therefore consider the following **model-oriented specification language** constructs as forming an **upper ontology**:
  - **type**s, **ground type**s, **type expression**s and **type definition**s;
  - **function**s, **function signature**s and **function definition**s;
  - **process**es, **process signature**s and **process definition**s,

  as constituting an **upper level ontology** for `TripTych` **domain description**s.
- That is, every domain description is structured with respect to:
  - **part**s and **material**s using **type**s,
  - **action**s using **function**s,
  - **event**s using **predicate**s,
  - **discrete**behaviours using **process**es and
  - **continuous behaviours** using **partial differential**equations.

© Dines Bjørner 2012, DTU Informatics, Techn.Univ.of Denmark – July 31, 2012: 09:02

3. Endurants

- There is sort of a dichotomy buried in our treating endurants before perdurants. The dichotomy is this:
  - ❖ one could claim that the perdurants,
    i.e., the actions, events and behaviours
    is *"what it, the domain, is all about"*;
- To describe these, however, we need refer to endurants!

3.1. General

Wikipedia:

- *By an* **endurant** *(also known as a* **continuant** *or a* **substance***) we shall understand an entity*
  - *that can be observed, i.e., perceived or conceived,*
  - *as a complete concept,*
  - *at no matter which given snapshot of time.*
- *Were we to freeze time*
  - *we would still be able to observe the entire endurant.*

3.2. Discrete and Continuous Endurants

- We distinguish between
  - **discrete endurant**s, which we shall call **part**s, and
  - **continuous endurant**s, which we shall call **material**s.

© Dines Bjørner 2012, DTU Informatics, Techn.Univ.of Denmark – July 31, 2012: 09:02

We motivate and characterise this distinction.

- By a **discrete endurant**, that is, a **part**, we shall understand something which is
  - separate or distinct in form or concept,
  - consisting of distinct or separate parts.
- By a **continuous endurant**, that is, a **material**, we shall understand something which is
  - prolonged without interruption,
  - in an unbroken series or pattern.
- We shall
  - first treat the idea of **discrete endurant**, that is, a **part** (Slides 51–110),
  - then the idea of **continuous endurant**, that is, a **material** (Slides 112–134).

4. Discrete Endurants: Parts

4.1. Atomic and Composite Parts

- **Parts** may be analysed into disjoint sets of
- **atomic parts** and
- **composite parts**.
- **Atomic parts** are those which,
  - in a given context,
  - are deemed *not* to consist of meaningful, separately observable proper **sub-parts**.
- **Composite parts** are those which,
  - in a given context,
  - are deemed to *indeed* consist of meaningful, separately observable proper **sub-parts**.
- A **sub-part** is a **part**.

© Dines Bjørner 2012, DTU Informatics, Techn.Univ.of Denmark – July 31, 2012: 09:02

Example: 4 **Atomic and/or Composite Parts.** To one person a part may be atomic; to another person the same part may be composite.

- It is the domain describer who decides the outcome of this aspect of domain analysis.
  - In some domain analysis a ‘person’ may be considered an atomic part.
    - For the domain of ferrying cars with passengers
    - persons are considered parts.
  - In some other domain analysis a ‘person’ may be considered a composite part.
    - For the domain of medical surgery
    - persons may be considered composite parts. ■

Example: 5 Container Lines.

- We shall presently consider **containers** (as used in container line shipping) to be atomic parts.
- And we shall consider a **container vessel** to be a composite part consisting of
  - an **indexed set** of **container bays**
  - where each container bay consists of **indexed set** of **container rows**
  - where each container row consists of **indexed set** of **container stacks**
  - where each container stack consists of a **linearly indexed sequence** of containers.
- Thus container vessels, container bays, container rows and container stacks are composite parts. ■

© Dines Bjørner 2012, DTU Informatics, Techn.Univ.of Denmark – July 31, 2012: 09:02

4.1.1. Atomic Parts

- When we observe
  - what we have decided, i.e., analysed, to be an endurant,
  - more specifically an **atomic part**, of a domain,
  - we are observing an **instance** of an atomic part.
- When we describe those instances
  - we describe, not their **value**s, i.e., the instances,
  - but their
    - **type** and
    - **properties**.

© Dines Bjørner 2012, DTU Informatics, Techn.Univ.of Denmark – July 31, 2012: 09:02

- In this section on **endurant entities**
  we shall unfold what these properties might be.
- But, for now, we focus on the **type** of the observed **atomic part**.
- So the situation is that we are observing a number of atomic parts
  - and we have furthermore decided that
  - they are all of *"the same kind"*.

- What does it mean for a number of atomic parts to be of *"the same kind"* ?
  - It means
    - that we have decided,
    - for any pair of **part**s considered of the same kind,
    - that the kinds of properties,
      - for such two **part**s,
    - are *"the same"*,
      - that is, of the same **type**, but possibly of different **value**s,
    - and that a number of different, other "facets",
    - are not taken into consideration.

- That is,
  - we **abstract** a collection of atomic parts
  - to be of the same kind,
  - thereby “dividing the domain of endurants” into possibly two distinct sets
    - those that are of the analysed kind, and
    - those that are not.

© Dines Bjørner 2012, DTU Informatics, Techn.Univ.of Denmark – July 31, 2012: 09:02

- It is now our description choice to associate with a set of **atomic parts** of *"the same kind"*
  - a **part type** (by suggesting a name for that **type**, for example, T) and
  - a set of **properties** (of its values):
    - **unique identifier**,
    - **mereology** and
    - **attributes**.

- Later we shall introduce discrete perdurants (actions, events and behaviours) whose signatures involves (possibly amongst others) type T.
- Now we can characterise *"of the same kind"* atomic part facets[14]
  - being of the same, named part type,
  - having the same unique identifier type,
  - having the same mereology (but not necessarily the same mereology values), and
  - having the same set of attributes (but not necessarily of the same attribute values),
- The *"same kind"* criteria apply equally well to composite part facets.

[14] as well as "of the same kind" composite part facets.

Example: 6 Transport Nets: Atomic Parts (I).

- The types of atomic transportation net parts are:
  - hubs, say of type H, and
  - links, say of type L.
- The chosen mereology associates with every hub and link a
  - distinct unique identifiers
  - (of types HI and LI respectively), and, *vice versa*,
  - how hubs and links are connected:
    - hubs to any number of links and
    - links to exactly two distinct hubs.

- The chosen attributes of
  - hubs include
    - hub location,
    - hub design[15],
    - hub traffic state[16],
    - hub traffic state space[17], etc.;
  - and of links include
    - link location,
    - link length,
    - link traffic state[18],
    - link traffic state space[19], etc.
- With these mereologies and attributes we see that we can consider hubs and links as different kinds of atomic parts. ■

[15]Design: simple crossing, freeway "cloverleaf" interchange, etc.

[16]A hub traffic state is (for example) a set of pairs of link identifiers where each such pair designates that traffic can move from the first designated link to the second.

[17]A hub state space is (for example) the set of all hub traffic states that a hub may range over.

[18]A link traffic state is (for example) a set of zero to two distinct pairs of the hub identifiers of the link mereology.

[19]A link traffic state space is (for example) the set of all link traffic states that a link may range over.

 © Dines Bjørner 2012, DTU Informatics, Techn.Univ.of Denmark – July 31, 2012: 09:02

Observers for Atomic Parts

- Let the domain describer decide
  - that a **type**, A (or Δ), is **atomic**,
  - hence that it does not consists of sub-parts.
- Hence there are no **observer** to be associated with A (or Δ).

4.1.2. Composite Parts

- The domain describer has chosen to consider
  - a part (i.e., a **part type**)
  - to be a composite part (i.e., a **composite part type**).
- Now the domain describer has to analyse the types of the sub-parts of the composite part.
  - There may be just one *"kind of"* sub-part of a composite part[20],
  - or there may be more than one *"kind of"*[21].
- For each such **sub-part type**
  - the domain describer decides on
  - an appropriate, distinct **type name** and
  - a **sub-part observer** (i.e., a **function signature**).

[20] that is, only one sub-part type
[21] that is, more than one sub-part type

© Dines Bjørner 2012, DTU Informatics, Techn.Univ.of Denmark – July 31, 2012: 09:02

Example: 7 Container Vessels: Composite Parts. We bring pairs of informal, narrative description texts and formalisations.

- For a container vessel, say of type V, we have
  - *Narrative:*
    - A container vessel, v:V, consists of container bays, bs:BS.
    - A container bay, b:B, consists of container rows, rs:RS.
    - A container row, r:R, consists of container stacks, ss:SS.
    - A container stack, s:S, consists of a linearly indexed sequence of containers.
  - *Formalisation:*

type V,BS, **value** obs_BS: V→BS,
type B,RS, **value** obs_RS: B→RS,
type R,SS, **value** obs_CS: R→SS,
type SS,S, **value** obs_S: SS→S,
type S = C*. ■

4.1.3. Abstract Types, Sorts, and Concrete Types

- By an **abstract type**, or a **sort**, we shall understand a type
  - which has been given a name
  - but is otherwise undefined, that is,
    - is a space of undefined mathematical quantities,
      - where these are given properties
      - which we may express in terms of **axiom**s over sort (including **property**) values.

© Dines Bjørner 2012, DTU Informatics, Techn.Univ.of Denmark – July 31, 2012: 09:02

- By a **concrete type** we shall understand a type, T,
  - which has been given both a name
  - and a defining type expression of, for example the form
    - T = A-**set**,
    - T = A-**infset**,
    - $\mathsf{T} = \mathsf{A}\times\mathsf{B}\times\cdots\times\mathsf{C}$,
    - $\mathsf{T} = \mathsf{A}^*$,
    - $\mathsf{T} = \mathsf{A}^\omega$,
    - $\mathsf{T} = \mathsf{A} \underset{m}{\to} \mathsf{B}$,
    - $\mathsf{T} = \mathsf{A}\to\mathsf{B}$,
    - $\mathsf{T} = \mathsf{A}\overset{\sim}{\to}\mathsf{B}$, or
    - $\mathsf{T} = \mathsf{A}|\mathsf{B}|\cdots|\mathsf{C}$.
  - where A, B, ..., C are **type name**s or **type expression**s.

Example: **8** **Container Bays.** We continue Example 7 on page 64.

type Bs = BId $\overrightarrow{m}$ B,
value obs_Bs: BS→Bs,

type Rs = RId $\overrightarrow{m}$ R,
value obs_Rs: B→Rs,

type Ss = SId $\overrightarrow{m}$ S,
value obs_Ss: R→Ss,

type S = C*. ■

© Dines Bjørner 2012, DTU Informatics, Techn.Univ.of Denmark – July 31, 2012: 09:02

Observers for Composite Parts I/II

- Let the domain describer decide
  - that a type, A (or Δ), is composite
  - and that it consists of sub-parts of types B, C, ..., D.
- We can initially consider these types B, C, ..., D, as **abstract types**, or **sorts**, as we shall mostly call them.
- That means that there are the following formalisations:
  - **type** A, B, C, ..., D;
  - **value** obs_B: A→B, obs_C: A→C, ..., obs_D: A→D.

Observers for Composite Parts II/II

- We can also consider the types B, C, . . . , D, as **concrete types**,
  - **type** Bc = TypBex, Cc = TypCex, ..., Dc = TypDex;
  - **value** obs_Bc: B→Bc, obs_Cc: C→Cc, . . . , obs_Dc: D→Dc,
  - where TypBex, TypCex, . . . , TypDex are type expressions as, for example, hinted at above.
- The prefix obs_ distinguishes **part observers**
  - from **mereology observers** (uid_, mereo_) and
  - **attribute observers** (attr_).

© Dines Bjørner 2012, DTU Informatics, Techn.Univ.of Denmark – July 31, 2012: 09:02

4.2. Properties

- Endurants have **properties**.
  - Properties are
    - what makes up a parts (and materials) and,
    - with **property values** distinguishes
      one part from another part and
      one material from another material.
  - We name properties.
    - **Properties** of **parts** and **materials** can be given distinct names.
    - We let these names also be the **property type name**.
    - Hence two parts (materials) of the same **part type** (**material type**)
      have the same set of **property type names**.

- **Properties** are all that distinguishes **parts** (and **materials**).
  - The **part types** (**material types**) in themselves do not express properties.
  - They express a class of parts (respectively materials).
  - All parts (materials) of the same type
  - have the same **property types**.
  - **Parts** (**materials**) of the different **types** have different sets of **property types**,

© Dines Bjørner 2012, DTU Informatics, Techn.Univ.of Denmark – July 31, 2012: 09:02

- For pragmatic reasons we distinguish between three kinds of properties:
  - unique identifiers,
  - mereology, and
  - attributes.
- If you "remove" a property from a part
  - it "looses" its (former) part type,
  - to, in a sense, attain another part type:
    - perhaps of another, existing one,
    - or a new "created" one.
- *But we do not know* how to model *removal of a property* from an endurant value ![22]

[22] And we see no need for describing such type-changes. Crude oil does not "morph" into fuel oil, diesel oil, kerosene and petroleum. Crude oil is consumed and the fractions result from distillation, for example, in an oil refinery.

Example: 9 Atomic Part Property Kinds.

- We distinguish between two kinds of persons:
  - ‘living persons’ and ‘deceased persons’;
  - they could be modelled by two different **part types**:
    - LP: living person, with a set of properties,
    - DP: deceased person, with a, most likely, different set of properties.
- All persons have been born, hence have a birth date (**static attributes**).
- Only deceased persons have a (well-defined) death date.

© Dines Bjørner 2012, DTU Informatics, Techn.Univ.of Denmark – July 31, 2012: 09:02

- All persons also have height and weight profiles (i.e., with dated values, i.e., **dynamic attributes**).
- One can always associate a **unique identifier** with each person.
- Persons are related, family-wise:
  - have parents (living or deceased),
  - (up to four known) grandparents, etc.,
  - may have brothers and sisters (zero or more),
  - may have children (zero or more), etc.
  - These family-relations can be considered the **mereology** for living persons. ■

© Dines Bjørner 2012, DTU Informatics, Techn.Univ.of Denmark – July 31, 2012: 09:02

4.2.1. Unique Identification

- We can assume that all parts
  - of the same part type
  - can be uniquely distinguished,
  - hence can be given unique identifications.

© Dines Bjørner 2012, DTU Informatics, Techn.Univ.of Denmark – July 31, 2012: 09:02

Unique Identification

- With every **part**, whether **atomic** or **composite** we shall associate a **unique part identifier**, of just **unique identifier**.
- Thus we shall associate with **part type** T
  - the **unique part type identifier type** TI,
  - and a **unique part identifier observer** function, uid_TI: T→TI.
- These associations (TI and uid_TI) are, however,
  - usually expressed explicitly,
  - whether they are ("subsequently") needed !

- The **unique identifier** of a part
  - can not be changed;
  - hence we can say that
    - no matter what a given **part**'s **property values** may take on,
    - that **part** cannot be confused with any other part.
- Since we can talk about this concept of **unique identification**,
  - we can **abstract**ly describe it —
    - and do not have to bother about any representation,
    - that is, whether we can humanly observe **unique identifier**s.

© Dines Bjørner 2012, DTU Informatics, Techn.Univ.of Denmark – July 31, 2012: 09:02

4.2.2. Mereology

- Mereology [CasatiVarzi1999][23] (from the Greek $\mu\epsilon\rho o\varsigma$ 'part') is
  - *the theory of part-hood relations:*
  - *of the relations of part to whole and*
  - *the relations of part to part within a whole.*

[23] Achille Varzi: Mereology, http://plato.stanford.edu/entries/mereology/

- For pragmatic reasons we choose to model the mereology of a domain in either of two ways
  - either by defining a **concrete type**
    as a model of the composite type,
  - or by endowing the sub-parts of the composite part with structures of **unique part identifier**s.

  or by suitable combinations of these.

© Dines Bjørner 2012, DTU Informatics, Techn.Univ.of Denmark – July 31, 2012: 09:02

Example: 10 **Container Bays, Etcetera: Mereology.** First we show how to model **indexed set of container bays**, **rows** and **stacks** for the previous example.

- *Narrative:*
  - (i) An **indexed set**, bs:BS, of bays is a **bijective map** from unique bay identifiers, bid:BId, to bays, b:B.
  - (ii) An **indexed set**, rs:RS, of rows is a **bijective map** from unique row identifiers, rid:RId, to rows, r:R.
  - (iii) An **indexed set**, ss:SS, of stacks is a **bijective map** from unique stack identifiers, sid:SId, to stacks, s:S.
  - (iv) A stack is a **linear indexed sequence** of containers, c:C.

- *Formalisation:*
  - (i) **type** BS, B, BId,
    Bs=BId $\overrightarrow{m}$ B,
    value obs_Bs: BS→Bs
    (or obs_Bs: BS→(BId $\overrightarrow{m}$ B));
  - (ii) **type** RS, R, RId,
    Rs=RId $\overrightarrow{m}$ R,
    value obs_Rs: RS→Rs
    (or obs_Rs: RS→(RId $\overrightarrow{m}$ R));
  - (iii) **type** SS, S, SId,
    Ss=SId $\overrightarrow{m}$ S;
  - (iv) **type** C,
    S=C*.

Example: 11 Transport Nets: Mereology.

- We show how to model a **mereology**
  - for a **transport net** of **links** and **hubs**.
- *Narrative:*

(i) Hubs and links are endowed with unique hub, respectively link identifiers.

(ii) Each hub is furthermore endowed with a hub mereology which lists the unique link identifiers of all the links attached to the hub.

(iii) Each link is furthermore endowed with a link mereology which lists the set of the two unique hub identifiers of the hubs attached to the link.

(iv) Link identifiers of hubs and hub identifiers of links must designate hubs, respectively links of the net.

- *Formalisation:*

(i) **type** H, HI, L, LI;

value

(ii) uid_HI:H→HI, uid_LI:L→LI,

mereo_H:H→LI-**set**, mereo_L:L→HI-**set**,

axiom

(iii) $\forall$ l:L · **card** mereo_L(l) = 2

(iv) $\forall$ n:N, l:L, h:H · l $\in$ obs_Ls(obs_LS(n)) $\wedge$ h $\in$ obs__Hs(obs_HS(n))

$\forall$ hi:HI · hi $\in$ mereo_L(l) $\Rightarrow$

$\exists$ h′:H·h′ $\in$ obs_Hs(obs_HS(n)) $\wedge$ uid_HI(h)=hi

$\wedge$ $\forall$ li:LI · li $\in$ mereo_H(h) $\Rightarrow$

$\exists$ l′:L·l′ $\in$ obs_Ls(obs_LS(n)) $\wedge$ uid_LI(l)=li

© Dines Bjørner 2012, DTU Informatics, Techn.Univ.of Denmark – July 31, 2012: 09:02

Concrete Models of Mereology

The concrete mereology example models above illustrated maps and sequences as such models.

- In general we can model mereologies in terms of
  - (i) sets: **A-set**,
  - (ii) Cartesians: $A_1 \times A_2 \times \ldots \times A_m$,
  - (iii) lists: A^*, and
  - (iv) maps: $A \xrightarrow[m]{} B$,

  where A, A_1, A_2,...,A_m and B are types [we assume that they are type names] and where the A_1, A_2,...,A_m type names need not be distinct.
- Additional **concrete types**, say D, can be defined by **concrete type definitions**, D=E, where E is either of the **type expressions** (i–iv) given above or (v) $E_i|E_j$, or (vi) (E_i). where E_k (for suitable k) are either of (i–vi).
- Finally it may be necessary to express well-formedness predicates for concretely modelled mereologies.

Abstract Models of Mereology

Abstractly modelling mereology of parts, to us, means the following.

- With part types P_1, P_2, ..., P_n
  - is associated the unique part identifier types, Π_1, Π_2, ..., Π_n,
  - that is $\mathsf{uid\_}\Pi_i$: $\mathsf{P}_i \to \Pi_i$ for $i \in \{1..n\}$,
- and with each part type, P_i,
  - is then associated a **mereology** observer,
  - $\mathsf{mereo\_P}_i$: $\mathsf{P}_i \to \Pi_j$**-set**$\times\Pi_k$**-set**$\times\ldots\times\Pi_\ell$**-set**,
- such that for all p:Pi we have that
  - if $\mathsf{mereo\_P}_i(\mathsf{p}) = (\{..., \pi_{j_a}, ...\},\{..., \pi_{k_b}, ...\},...,\{..., \pi_{\ell_c}, ...\})$
  - for $i, j, k, ...\ell \in \{1..n\}$
  - then part p:P_i is connected (related) to the parts identified by ..., π_{j_a}, ... π_{k_b}, ..., π_{ℓ_c}, ...
- Finally it may be necessary to express axioms for abstractly modelled mereologies.

 © Dines Bjørner 2012, DTU Informatics, Techn.Univ.of Denmark – July 31, 2012: 09:02

- How parts are related to other parts
  - is really a modelling choice, made by the domain describer.
  - It is not necessarily something
    that is obvious
    from observing the parts.

Example: 12 Pipelines: A Physical Mereology.

- Let pipes of a pipe line be composed with valves, pumps, forks and joins of that pipe line.
- Pipes, valves, pumps, forks and joins (i.e., pipe line units) are given unique pipe, valve, pump, fork and join identifiers.
- A mereology for the pipe line could now endow pipes, valves and pumps with
  - one input unique identifier, that of the predecessor successor unit, and
  - one output unique identifier, that of the successor unit.
- Forks would then be endowed with
  - two input unique identifiers, and
  - one out put unique identifier;
- and joins "the other way around".

Example: 13 Documents: A Conceptual Mereology.

- The mereology of, for example, this document,
  - that is, of the tutorial slides,

  is determined by the author.
- There unfolds, while writing the document,
  - a set of unique identifiers
  - for section, subsection, sub-subsection, paragraph, etc., units. and
  - between texts of a “paper version” of the document and slides of a “slides version” of the document.

- This occurs as the author necessarily
  - inserts cross-references,
    - in unit texts to other units, and
    - from unit texts to other documents (i.e., 'citations');
  - and while inserting "page" shifts for the slides.
- From those inserted references
  there emerges what we could call the document mereology. ■

- So the determination of a, or the, mereology of composite parts
  - is either given by physical considerations,
  - or are given by (more-or-less) logical (or other) considerations,
  - or by combinations of these.
- The "design" of mereologies improves with experience.

© Dines Bjørner 2012, DTU Informatics, Techn.Univ.of Denmark – July 31, 2012: 09:02

Example: 14 Pipelines: Mereology.

- We divert from our line of examples centered around
  - transport nets and, to some degree,
  - container transport,
- to bring a second, in a series of examples
  - on pipelines
  - (for liquid or gaseous material flow).

1. A pipeline consists of connected units, u:U.
2. Units have unique identifiers.
3. And units have mereologies, ui:UI:
   (a) pump, pu:Pu, pipe, pi:Pi, and valve, va:Va, units have one input connector and one output connector;
   (b) fork, fo:Fo, [join, jo:Jo] units have one [two] input connector[s] and two [one] output connector[s];
   (c) well, we:We, [sink, si:Si] units have zero [one] input connector and one [zero] output connector.
   (d) Connectors of a unit are designated by the unit identifier of the connected unit.
   (e) The auxiliary sel_UIs_in selector funtion selects the unique identifiers of pipeline units providing input to a unit;
   (f) sel_UIs_out selects unique identifiers of output recipients.

type
1. U = Pu | Pi | Va | Fo | Jo | Si | We
2. UI

value
2. uid_U: U → UI
3. mereo_U: U → UI-**set** × UI-**set**
3. wf_mereo_U: U → **Bool**
3. wf_mereo_U(u) ≡
3(a). is_(Pu|Pi|Va)(u) → **card** iusi = 1 = **card** ouis,
3(b). is_Fo(u) → **card** iuis = 1 ∧ **card** ouis = 2,
3(b). is_Jo(u) → **card** iuis = 2 ∧ **card** ouis = 1,
3(c). is_We(u) → **card** iuis = 0 ∧ **card** ouis = 1,
3(d). is_Si(u) → **card** iuis = 1 ∧ **card** ouis = 0

3(e). sel_UIs_in
3(e). sel_UIs_in(u) ≡ **let** (iuis,_)=mereo_U(u) **in** iuis **end**
3(f). sel_out: U → UI-**set**
3(f). sel_UIs_out(u) ≡ **let** (_,ouis)=mereo_U(u) **in** ouis **end**

- We omit treatment of axioms for pipeline units
  - being indeed connected to existing other pipeline units.
  - We refer to Example 22 on page 119 and 23 on page 123. ■

4.2.3. Attributes

- By an **attribute** of a part, p:P, we shall understand
  - some **observable property**, some **phenomenon**,
  - that is not a **sub-part** of p
  - but which characterises p
  - such that all parts of type P have that attribute and
  - such that "removing" that attribute from p
    (if such was possible)
    "renders" the type of p undefined.
- We ascribe types to attributes — not, therefore, to be confused with types of (their) parts.

Example: 15 Attributes.

- Example attributes of links of a transport net are:
  - length LEN,
  - location LOC,
  - state $\mathsf{L}\Sigma$ and
  - state space $\mathsf{L}\Omega$,
- Example attributes of a person could be:
  - name NAM,
  - birth date BID,
  - gender GDR,
  - weight WGT,
  - height HGT and
  - address ADR.

© Dines Bjørner 2012, DTU Informatics, Techn.Univ.of Denmark – July 31, 2012: 09:02

- Example attributes of a transport net could be:
  - name of the net,
  - legal owner of the net,
  - a map of the net,
  - etc.
- Example attributes of a container vessel could be:
  - name of container vessel,
  - vessel dimensions,
  - vessel tonnage (TEU),
  - vessel owner,
  - current stowage plan,
  - current voyage plan, etc.

4.2.3.1 Static and Dynamic Attributes

- By a **static attribute** we mean an attribute (of a part) whose value remains fixed.
- By a **dynamic attribute** we mean an attribute (of a part) whose value may vary.

 © Dines Bjørner 2012, DTU Informatics, Techn.Univ.of Denmark – July 31, 2012: 09:02

Example: 16 Static and Dynamic Attributes.

- The length and location attributes of links are static.
- The state and state space attributes of links and hubs are dynamic.
- The birth-date attribute of a person is considered static.
- The height and weight attributes of a person are dynamic.
- The map of a transport net may be considered dynamic.
- The current stowage and the current voyage plans of a vessel should be considered dynamic. ■

Attribute Types and Observers, I/II

- Let the domain describer decide that parts of type P
- have attributes of types A_1, A_2, ..., A_t.
- This means that the following two formal clauses arise:
  - P, A_1, A_2, ..., A_t and
  - attr_A_1:P→A_1, attr_A_2:P→A_2, ..., attr_A_t:P→A_t

© Dines Bjørner 2012, DTU Informatics, Techn.Univ.of Denmark – July 31, 2012: 09:02

Attribute Types and Observers, II/II

- We may wish to annotate the list of **attribute type names** as to whether they are static or dynamic, that is,
  - whether **values** of some **attribute type**
  - vary or
  - remain fixed.
- The prefix **attr_** distinguishes **attribute observers** from **part observers** (**obs_**) and **mereology observers** (**uid_**, **mereo_**).

4.3. Shared Attributes and Properties

- Shared attributes and shared properties
  - play an important rôle in understanding domains.

4.3.1. Attribute Naming

- We now *impose a restriction* on the naming of part attributes.
  - If attributes
    - of two different parts
    - of different part types
    - are identically named
    - then attributes must be somehow related, over time!
  - The "somehow" relationship must be described.

© Dines Bjørner 2012, DTU Informatics, Techn.Univ.of Denmark – July 31, 2012: 09:02

Example: 17 Shared Bus Time Tables.

- Let our domain include that of *bus time tables* for *busses* on a *bus transport net* as described in many examples in this tutorial.
- We can then imagine a *bus transport net* as containing the following parts:
  - a *net*,
  - a *management system*,
  - a set of *busses*.
- For the sake of argument we consider a *bus time table* to be an attribute of the *bus management system*.
- And we also consider *bus time tables* to be attributes of *busses*.

- We think of the *bus time table* of a *bus*
  - to be that subset of the *bus management system bus time table*
  - which corresponds to the *bus*' *line number*.
- By saying that *bus time tables*
  - "corresponds" to well-defined subsets of
  - the *bus management system bus time table*

  we mean the following
  - The value of the *bus bus time table*
  - must at every time
  - be equal to the corresponding *bus line entry* in the *bus management system bus time table*.

© Dines Bjørner 2012, DTU Informatics, Techn.Univ.of Denmark – July 31, 2012: 09:02

4.3.2. Attribute Sharing

- We say that two **parts**,
  - of no matter what **part type**,
  - *share* an **attribute**,
  - if the following is the case:
    - the corresponding **part types** (and hence the **parts**)
    - have **identically named attributes**.
    - We say that **identically named attributes** designate **shared attributes**.
  - We do not present the corresponding invariants over **parts** with **identically named attributes**.

4.4. Shared Properties

- We say that two **parts**,
  - of no matter what **part type**,
  - *share* a **property**,
  - if either of the following is the case:
    - (i) either the corresponding **part types** (and hence the **parts**) have **shared attributes**;
    - (ii) or the **unique identifier type** of one of the **parts** potentially is in the **mereology type** of the other **part**;
    - (iii) or both.
  - We do not present the corresponding **invariants** over **parts** with **shared properties**.

© Dines Bjørner 2012, DTU Informatics, Techn.Univ.of Denmark – July 31, 2012: 09:02

4.5. Summary of Discrete Endurants

- We have introduced the **endurant** notions of **atomic parts** and **composite parts**:
  - part types,
  - part observers (**obs_**),
    - sort observers, and
    - concrete type observers;
  - part properties:
    - unique identifiers:
      - unique part identifier observers (**uid_**),
      - unique part identifier types,
    - mereology:
      - part mereologies,
      - part mereology observers (**mereo_**);

      and
    - attributes:
      - attribute observers (**attr_**) and
      - attribute types.

- The **unique identifier** property cannot necessarily be observed:
  - it is an **abstract concept** and
  - can be objectively "assigned".

  That is: **unique identifier**s are not required to be manifest.
- The **mereology** property also cannot usually be observed:
  - it is also an **abstract concept**,
  - but can be deduced from careful analysis.

  That is: **mereology** is not required to be manifest.
- The **attribute**s can be observed:
  - usually by simple physical measurements,
  - or by deduction from (conceptual) facts,

  That is: **attribute**s are usually only "indirectly" manifest.

© Dines Bjørner 2012, DTU Informatics, Techn.Univ.of Denmark – July 31, 2012: 09:02

Discrete Endurant Modelling I/II

Faced with a **phenomenon** the **domain analyser** has to decide

- whether that **phenomenon** is an **entity** or not, that is, whether
  - an **endurant** or
  - a **perdurant** or
  - neither.
- If **endurant** and if **discrete**, then whether it is
  - an atomic part or
  - a composite part.
- Then the **domain analyser** must decide on its type,
  - whether an **abstract type** (a **sort**)
  - or a **concrete type**, and, if so, which concrete form.

Discrete Endurant Modelling II/II

- Next the **unique identifier** and the **mereology** of the **part type** (e.g., P) must be dealt with:
  - **type name** (e.g., PI) for and, hence, **unique identifier observer name** (uid_PI) of **unique identifiers** and the
  - **part mereology types** and **mereology observer name** (mereo_P).
- Finally the designer must decide on the **part type attributes** for parts p:P:
  - for each such a suitable **attribute type name**, for example, A_i for suitable i,
  - a corresponding **attribute observer signature**, attr_A_i:P→A_i,
  - and whether an attribute is considered **static** or **dynamic**.

End of Lecture 2: Last Session — Discrete Endurant Entities

Parts

FM 2012 Tutorial, Dines Bjørner, Paris, 28 August 2012


Software
Engineering 1


Software
Engineering 2


Software
Engineering 3

LONG BREAK