Dines Bjørner's MAP-i Lecture # 3

Unique Identifiers, Mereologies and Attributes

Monday, 25 May 2015: 14:30–15:15

1.2.7. Unique Part Identifiers

- Two parts are either identical or a distinct, i.e., unique.
  - Two parts are identical
    - if all their respective qualities
    - have the same values.

    That is, their location in space/time are one and the same.
  - Two parts are distinct
    - even if all the attribute qualities of the two parts,
    - that we have chosen to consider have the same values,
    - if, in that case, their space/time locations are distinct.

- We can assume, without any loss of generality,
  - (i) that all parts, p, of any domain P, have **unique identifiers**,
  - (ii) that **unique identifiers** (of parts p:P) are **abstract values** (of the **unique identifier** sort PI of P),
  - (iii) such that distinct part sorts, P_i and P_j, have distinctly named **unique identifier** sorts, say PI_i and PI_j,
  - (iv) that all π_i:PI_i and π_j:PI_j are distinct, and
  - (v) that the observer function **uid_P** applied to p yields the unique identifier, say π:PI, of p.

Representation of Unique Identifiers:

- Unique identifiers are abstractions.
  - When we endow two parts (say of the same sort) with distinct unique identifiers
  - then we are simply saying that these two parts are distinct.
  - We are not assuming anything about how these identifiers otherwise come about.

Domain Description Prompt 3 . *observe_unique_identifier*:

- *We can therefore apply the* **domain description prompt***:*
  - ***observe_unique_identifier***
- *to parts* ***p:P*** *resulting in the analyser writing down the* ***unique identifier type and observer*** *domain description text according to the following schema:*

3. observe_unique_identifier schema

Narration:

[s] ... narrative text on unique identifier sort ...

[u] ... narrative text on unique identifier observer ...

[a] ... axiom on uniqueness of unique identifiers ...

Formalisation:

type

[s] PI

value

[u] **uid**_P: P $\to$ PI

axiom

[a] $\mathcal{U}$

Example 29 . Unique Transportation Net Part Identifiers:
We continue Example 20 on Slide 123.

30 Links and hubs have unique identifiers

31 and unique identifier observers.

type
30. LI, HI
value
31. **uid**_LI: L → LI
31. **uid**_HI: H → HI
axiom [Well−formedness of Links, L, and Hubs, H]
30. ∀ l,l′:L · l≠l′⇒**uid**_LI(l)≠**uid**_LI(l′),
30. ∀ h,h′:H · h≠h′⇒**uid**_HI(h)≠**uid**_HI(h′) ■

1.2.8. Mereology

- **Mereology** is the study and knowledge of parts and part relations.
  - Mereology as a logical/philosophical discipline
    can perhaps best be attributed to the Polish mathematician/logician
    Stanisław Leśniewski [32, 21].

1.2.8.1. Part Relations

- Which are the relations that can be relevant for part-hood ?
- We give some examples.
  - Two otherwise distinct parts may share attribute values.

Example 30 . Shared Attribute Mereology:

- (i) two or more distinct public transport busses may run according to the same, thus "shared", bus time table;
- (ii) all vehicles in a traffic participate in that traffic, each with their "share", that is, position on links or at hubs – as observed by the (thus postulated, and shared) traffic observer.

etcetera ■

- Two otherwise distinct parts may be said to, for example, be topologically “adjacent” or one “embedded” within the other.

Example 31 . Topological Connectedness Mereology:

- (i) two rail units may be connected (i.e., adjacent),
- (ii) a road link may be connected to two road hubs;
- (iii) a road hub may be connected to zero or more road links;

etcetera. ■

- The above examples are in no way indicative of the “space” of part relations that may be relevant for part-hood.
- The domain analyser is expected to do a bit of experimental research in order to discover necessary, sufficient and pleasing “mereology-hoods” !

1.2.8.2. Part Mereology: Types and Functions

Analysis Prompt 13 . *has_mereology*:

- *To discover necessary, sufficient and pleasing "mereology-hoods" the analyser can be said to endow a truth value* **true** *to the* **domain analysis prompt**:
  - *has_mereology*

- When the domain analyser decides that
  - some parts are related in a specifically enunciated mereology,
  - the analyser has to decide on suitable
    - mereology types and
    - mereology (i.e., part relation) observers.
- We can define a **mereology type** as a type $\mathcal{E}$xpression over unique [part] identifier types.
  - We generalise to unique [part] identifiers over a definite collection of part sorts, P1, P2, ..., Pn,
  - where the parts p1:P1, p2:P2, ..., pn:Pn are not necessarily (immediate) sub-parts of some part p:P.

type
  PI1, PI2, ..., PIn
  MT = $\mathcal{E}$(PI1, PI2, ..., PIn),

Domain Description Prompt 4 . *observe_mereology*:

- *If* **has_mereology**(p) *holds for parts p of type* P,
  - *then the analyser can apply the* **domain description prompt**:
    - **observe_mereology**
  - *to parts of that type*
  - *and write down the* mereology types and observers *domain description text according to the following schema:*

4. observe_mereology schema

Narration:

[t] ... narrative text on mereology type ...

[m] ... narrative text on mereology observer ...

[a] ... narrative text on mereology type constraints ...

Formalisation:

type

[t] MT[12]= $\mathcal{E}$(PI1,PI2,...,PIm)

value

[m] **obs_mereo**_P: P $\rightarrow$ MT

axiom [Well−formedness of Domain Mereologies]

[a] $\mathcal{A}$(MT)

[12] MT will be used several times in Sect. .

- *Here* $\mathcal{E}$(PI1,PI2,...,PIm) *is a type expression over possibly all unique identifier types of the domain description,*
- *and* $\mathcal{A}$(MT) *is a predicate over possibly all unique identifier types of the domain description.*
- *To write down the concrete type definition for* **MT** *requires a bit of analysis and thinking.*
- ***has_mereology*** *is a* **prerequisite prompt** *for* `observe_mereology` ■

Example 32 . Road Net Part Mereologies: We continue Example 20 on Slide 123 and Example 29 on Slide 151.

32 Links are connected to exactly two distinct hubs.

33 Hubs are connected to zero or more links.

34 For a given net the link and hub identifiers of the mereology of hubs and links must be those of links and hubs, respectively, of the net.

```
type
32.   LM′ = HI-set, LM = {|his:HI-set • card(his)=2|}
33.   HM = LI-set
value
32.   obs_mereo_L: L → LM
33.   obs_mereo_H: H → HM
axiom [ Well−formedness of Road Nets, N ]
34.   ∀ n:N,l:L,h:H• l ∈ obs_part_Ls(obs_part_LC(n))∧h ∈ obs_part_Hs(ob
34.     let his=mereology_H(l), lis=mereology_H(h) in
34.     his⊆∪{uid_H(h) | h ∈ obs_part_Hs(obs_part_HC(n))}
34.    ∧ lis⊆∪{uid_H(l) | l ∈ obs_part_Ls(obs_part_LC(n))} end
```

Example 33 . Pipeline Parts Mereology:

- We continue Example 27 on Slide 140.
- Pipeline units serve to conduct fluid or gaseous material.
- The flow of these occur in only one direction: from so-called input to so-called output.

35 Wells have exactly one connection to an output unit.

36 Pipes, pumps and valves have exactly one connection from an input unit and one connection to an output unit.

37 Forks have exactly one connection from an input unit and exactly two connections to distinct output units.

38 Joins have exactly one two connection from distinct input units and one connection to an output unit.

39 Sinks have exactly one connection from an input unit.

40 Thus we model the mereology of a pipeline unit as a pair of disjoint sets of unique pipeline unit identifiers.

type
40. UM′=(UI-**set**×UI-**set**)
40. UM={|(iuis,ouis):UI-**set**×UI-**set**·iuis ∩ ouis={}|}
value
40. **obs_mereo**_U: UM
axiom [Well−formedness of Pipeline Systems, PLS (0)]
∀ pl:PL,u:U · u ∈ **obs_part**_Us(pl) ⇒
let (iuis,ouis)=**obs_mereo**_U(u) **in**
case (**card** iuis,**card** ouis) **of**
35. (0,1) → **is**_We(u),
36. (1,1) → **is**_Pi(u)∨**is**_Pu(u)∨**is**_Va(u),
37. (1,2) → **is**_Fo(u),
38. (2,1) → **is**_Jo(u),
39. (1,0) → **is**_Si(u)
end end ■

1.2.8.3. Update of Mereologies

- We normally consider a part's mereology to be constant.
- There may, however, be cases where the mereology of a part changes.
- In order to update mereology values the description language offers the "built-in" operator:

Mereology Update Function

⟐ **upd_mereology**: $P \to M \to P$

for all relevant M and P.

- The meaning of **upd_mereology** is, informally:

type
  P, M
value
  upd_mereology: P → M → P
  upd_mereology(p)(m) **as** p′
    post: **obs_mereo**_H(p′) = m

- The above is a simplification.
  - It lacks explaining that all other aspects of the part p:P are left unchanged.
  - It also omits mentioning some proof obligations.
    - The updated mereology must, for example,
    - only specify such unique identifiers of parts
    - that are indeed existing parts.
  - A proper formal explication requires
  - that we set up a formal model of the
  - domain/method/analyser/description quadrangle.

Example 34 . Mereology Update:

- The example is that of updating the mereology of a hub.
- Cf. Example 32 on Slide 160.

41 Inserting a link, l:L, between two hubs, ha:H,hb:H require the update of the mereologies of these two existing hubs.

42 The unique identifier of the inserted link, l:L, is li, li=**uid**_L(l) and h is either ha or hb;

43 li is joined to the mereology of both ha or hb; and respective hubs are updated accordingly.

```
     value
41.  update_hub_mereology: H → LI → H
42.  update_hub_mereology(h)(li) ≡
43.      let m = {li} ∪ obs_mereo_H(h) in upd_mereology(h)(m) end
```

1.2.8.4. Formulation of Mereologies

- The observe_mereology domain descriptor, Slide 158,
  - may give the impression that the mereo type MT can be described
  - "at the point of issue" of the observe_mereology prompt.
  - Since the MT type expression may, in general, depend on any part sort
  - the mereo type MT can, for some domains,
  - "first" be described when all part sorts have been dealt with.
- In *Domain Analysis: Endurants – An Analysis & Description Process Model* we we present a model of one form of evaluation of the TripTych analysis and description prompts.

1.2.9. Part Attributes

1.2.9.1. Inseparability of Attributes from Endurants

- Parts are
  - typically recognised because of their spatial form
  - and are otherwise characterised by their intangible, but measurable attributes.
- We learned from our exposition of *formal concept analysis* that
  - a formal concept, that is, a type, consists of all the entities
  - which all have the same qualities.
- Thus removing a quality from an entity makes no sense:
  - the entity of that type
  - either becomes an entity of another type
  - or ceases to exist (i.e., becomes a non-entity) !

1.2.9.2. Attribute Quality and Attribute Value

- We distinguish between
  - an attribute, as a logical proposition and
  - an attribute value as a value in some value space.

Example 35 . Attribute Propositions and Other Values:

- A particular street segment (i.e., a link), say ℓ,
  - satisfies the proposition (attribute) has_length, and
  - may then have value length 90 meter for that attribute.
- A particular road transport domain, δ,
  - has three immediate sub-parts: net, n, fleet, f, and monitor m;
  - typically nets `has_net_name` and `has_net_owner` proposition attributes
  - with, for example, `US Interstate Highway System` respectively `US Department of Transportation` as values for those attributes ■

1.2.9.3. Endurant Attributes: Types and Functions

- Let us recall that attributes cover qualities other than unique identifiers and mereology.
- Let us then consider that parts have one or more attributes.
  - These attributes are qualities
  - which help characterise "what it means" to be a part.

Example 36 . Atomic Part Attributes:

- Examples of attributes of atomic parts such as a human are:
  - *name*,
  - *gender*,
  - *birth-date*,
  - *birth-place*,
  - *nationality*,
  - *height*,
  - *weight*,
  - *eye colour*,
  - *hair colour*,

  etc.
- Examples of attributes of transport net links are:
  - *length*,
  - *location*,
  - *1 or 2-way link*,
  - *link condition*,

  etc. ■

Example 37 . Composite Part Attributes:

- Examples of attributes of composite parts such as a road net are:
  - *owner,*
  - *public or private net,*
  - *free-way or toll road,*
  - *a map of the net,*

  etc.

- Examples of attributes of a group of people could be: *statistic distributions of*
  - *gender,*
  - *age,*
  - *income,*
  - *education,*
  - *nationality,*
  - *religion,*

  etc. ■

- We now assume that all parts have attributes.
- The question is now, in general, how many and, particularly, which.

Analysis Prompt 14 . *attribute_names:*

- *The* **domain analysis prompt** ***attribute_names***
  - *when applied to a part p*
  - *yields the set of names of its attribute types:*
  - ***attribute_names****(p):* $\{\eta A_1, \eta A_2, ..., \eta A_n\}$.
- η *is a type operator. Applied to a type A it yields is name*[13] ■

[13] Normally, in non-formula texts, type A is referred to by ηA. In formulas A denote a type, that is, a set of entities. Hence, when we wish to emphasize that we speak of the name of that type we use ηA. But often we omit the distinction

- We cannot automatically, that is, syntactically, guarantee that our domain descriptions secure that
  - the various attribute types
  - for an emerging part sort
  - denote disjoint sets of values.

  Therefore we must prove it.

1.2.9.3.1 The Attribute Value Observer

- The "built-in" description language operator
  - **attr**_A
- applies to parts, p:P, where ηA∈attribute_names(p).
- It yields the value of attribute A of p.

Domain Description Prompt 5 . *observe_attributes*:

- *The domain analyser experiments, thinks and reflects about part attributes.*
- *That process is initated by the* **domain description prompt**:
  - ⟐ ***observe_attributes***.
- *The result of that* **domain description prompt** *is that the domain analyser cum describer writes down the* **attribute (sorts or) types and observers** *domain description text according to the following schema:*

5. observe_attributes schema

Narration:

[t] ... narrative text on attribute sorts ...
[o] ... narrative text on attribute sort observers ...
[i] ... narrative text on attribute sort recognisers ...
[p] ... narrative text on attribute sort proof obligations ...

Formalisation:

type
[t] A_i [$1 \leq i \leq n$]
value
[o] **attr**_A_i:P→A_i [$1 \leq i \leq n$]
[i] **is**_A_i:A_i→**Bool** [$1 \leq i \leq n$]
proof obligation [Disjointness of Attribute Types]
[p] $\forall\ \delta:\Delta$
[p] **let** P be any part sort **in** [the Δ domain description]
[p] **let** a:(A_1|A_2|...|A_n) **in** **is**_A_i(a) $\neq$ **is**_A_j(a) **end end** [$i \neq j$, $1 \leq i,j \leq n$]

- *The* **type** *(or rather sort) definitions:* A_1, A_2, ..., A_n *inform us that the domain analyser has decided to focus on the distinctly named* A_1, A_2, ..., A_n *attributes.*
- *And the* **value** *clauses*
  - **attr**_A_1:$P \rightarrow A_1$,
  - **attr**_A_2:$P \rightarrow A_2$,
  - ...,
  - **attr**_A_n:$P \rightarrow A_n$

  are then "automatically" given:
  - *if a part (type* P*) has an attribute* A_i
  - *then there is postulated, "by definition" [eureka] an attribute observer function* **attr**_A_i:$P \rightarrow A_i$ *etcetera* ■

- The fact that, for example, A_1, A_2, ..., A_n are attributes of **p**:P, means that the propositions
  - has_attribute_A$_1(p)$,
    has_attribute_A$_2(p)$,
    ..., and
    has_attribute_A$_n(p)$

  holds.
- Thus the observer functions **attr**_A$_1$, **attr**_A$_2$, ..., **attr**_A$_n$
  - can be applied to p in P
  - and yield attribute values a$_1$:A$_1$, a$_2$:A$_2$, ..., a$_n$:A$_n$ respectively.

Example 38 . Road Hub Attributes: After some analysis a domain analyser may arrive at some interesting hub attributes:

44 hub state: from which links (by reference) can one reach which links (by reference),

45 hub state space: the set of all potential hub states that a hub may attain,

46 such that

a. the links referred to in the state are links of the hub mereology

b. and the state is in the state space.

47 Etcetera — i.e., there are other attributes not mentioned here.

type
44. $\mathsf{H\Sigma} = (\mathsf{LI}\times\mathsf{LI})$**-set**
45. $\mathsf{H\Omega} = \mathsf{H\Sigma}$**-set**
value
44. **attr**_$\mathsf{H\Sigma}$:H→$\mathsf{H\Sigma}$
45. **attr**_$\mathsf{H\Omega}$:H→$\mathsf{H\Omega}$
axiom [Well−formedness of Hub States, $\mathsf{H\Sigma}$]
46. ∀ h:H · **let** lis = **obs_mereo**_H(h) **in**
46. **let** hσ = **attr**_$\mathsf{H\Sigma}$(h) **in**
46a.. {li,li′|li,li′:LI•(li,li′)∈ hσ}⊆lis
46b.. ∧ hσ ∈ **attr**_$\mathsf{H\Omega}$(h)
46. **end end**
type
47. ..., ...
value
47. **attr**_..., ...

1.2.9.4. Attribute Categories

- One can suggest a hierarchy of part attribute categories:
  - static or
  - dynamic values — and within the dynamic value category:
    - inert values or
    - reactive values or
    - active values — and within the dynamic active value category:
      - autonomous values or
      - biddable values or
      - programmable values.
- We now review these attribute value types.

Part attributes are either constant or varying, i.e., **static** or **dynamic** attributes.

- By a **static attribute**, `is_static_attribute`, we shall understand an attribute whose values
  - are constants,
  - i.e., cannot change.
- By a **dynamic attribute**, `is_dynamic_attribute`, we shall understand an attribute whose values
  - are variable,
  - i.e., can change.

Dynamic attributes are either inert, reactive or active attributes.

- By an **inert attribute**, `is_inert_attribute`,
  we shall understand a dynamic attribute whose values
  - only change as the result of external stimuli where
  - these stimuli prescribe properties of these new values.
- By a **reactive attribute**, `is_reactive_attribute`,
  we shall understand a dynamic attribute whose values,
  - if they vary, change value in response to
  - the change of other attribute values.
- By an **active attribute**, `is_active_attribute`,
  we shall understand a dynamic attribute whose values
  - change (also) of its own volition.

Example 39 . Inert and Reactive Attributes:

- Buses (i.e., vehicles) have a *timetable* attribute which is dynamic, i.e., can change, namely when the operator of the bus decides so, thus the bus timetable attribute is inert.
- Pipeline valve units include the two attributes of *valve opening* (`open`, `close`) and *internal flow* (measured, say `gallons` per `second`).
  - The valve opening attribute is of the programmable attribute category.
  - The flow attribute is reactive (flow changes with valve opening/closing)

Active attributes are either autonomous, biddable or programmable attributes.

- By an **autonomous attribute**, `is_autonomous_attribute`, we shall understand a dynamic active attribute
  - whose values change value only "on their own volition".[14]
- By a **biddable attribute**, `is_biddable_attribute`, (of a part) we shall understand a dynamic active attribute whose values
  - may be subject to a contract
  - as to which values it is expected to exhibit.
- By a **programmable attribute**, `is_programmable_attribute`, we shall understand a dynamic active attribute whose values
  - can be accurately prescribed.

[14] The values of an autonomous attributes are a "law onto themselves and their surroundings".

Example 40 . Static, Programmable and Inert Link Attributes:

48 Some link attributes

a. length, b. name,

can be considered static,

49 whereas other link attributes

a. state, b. state space

can be considered programmable,

50 Finally link attributes

a. link state–of–repair, b. date last maintained,

can be considered inert.

type
48a.. LEN
value
48a.. **obs_part**_LEN: L → LEN
type
48b.. Name
value
48b.. **obs_part**_Name: L → Name
type
49a.. $L\Sigma'$=(HI×HI)-**set**
49a.. $L\Sigma$={|$l\sigma$:$L\Sigma$ · **card** $l\sigma \leq 2$|}
value
49a.. **obs_part**_$L\Sigma$: L → $L\Sigma$
type
49b.. $L\Omega'$=$L\Sigma$-**set**
49b.. $L\Omega$={|$l\omega$:$L\Omega$ · **card** $l\omega = 1$|}
value
49b.. **obs_part**_$L\Omega$: L → $L\Omega$
type
50a.. LSoR
50b.. DLM
value
50a.. **obs_part**_LSoR: L → LSoR
50b.. **obs_part**_DLM: L → DLM ■

Example 41 . Autonomous and Programmable Hub Attributes:
We continue Example **??**.

- Time progresses autonomously,
- Hub states are programmed (*traffic signals*):
  - changing
    - from red to green via yellow,
    - in one pair of (co-linear) directions,
  - while changing, in the same time interval,
    - from green via yellow to red
    - in the "perpendicular" directions ■

- **External Attributes:** By an **external attribute** we shall understand
  - either a inert,
  - or a reactive,
  - or an autonomous,
  - or a biddable

  attribute ■
- Thus we can define the domain analysis prompt:
  - `is_external_attribute`,
  - as:

value
is_external_attribute: P → **Bool**
is_external_attribute(p) ≡
is_dynamic_attribute(p) ∧ ∼is_programmable_attribute(p)
pre: is_endurant(p) ∧ is_discrete(p)

- Figure 2 captures the attribute value ontology.

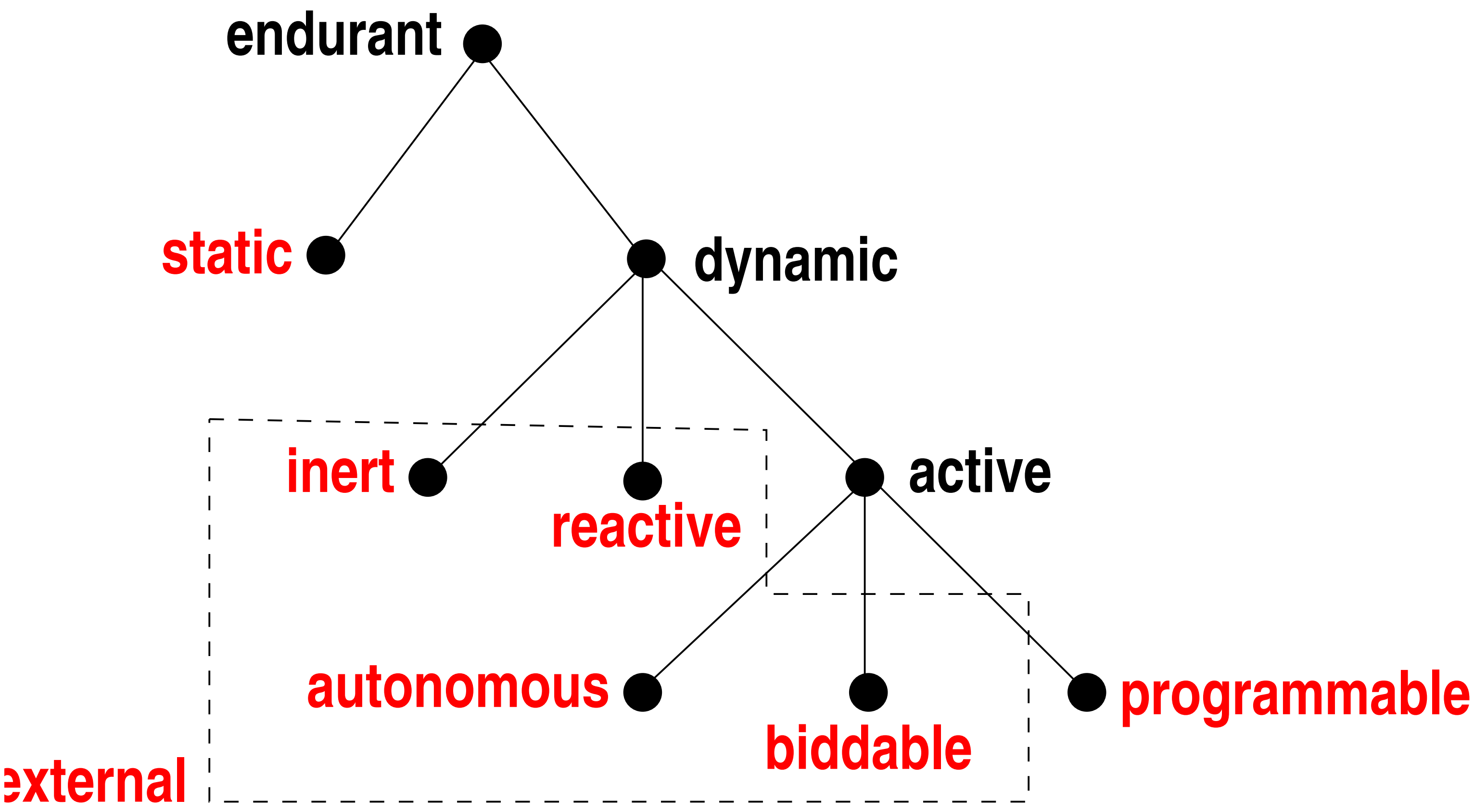

Figure 2: Attribute Value Ontology

1.2.9.5. Access to Attribute Values

- In an action, event or a behaviour description
  - static values of parts, p,
  - (say of type A)
  - can be "copied", **attr**_A(p),
  - and still retain their (static) value.
- But, for action, event or behaviour descriptions,
  - dynamic values of parts, p,
  - cannot be "copied",
  - but **attr**_A(p) must be "performed"
  - every time they are needed.

- That is:
  - **static values** require at most one domain access,
  - whereas **dynamic values** require repeated domain accesses.
- We shall return to the issue of **attribute value access** in Sect. 1.3.8.

1.2.9.6. Shared Attributes

- Normally part attributes of different part sorts are distinctly named.
- If, however, `observe_attributes`(p_{ik}:P_i) and `observe_attributes`($p_{j\ell}$:
  - for any two distinct part sorts, P_i and P_j, of a domain,
  - "discovers" identically named attributes, say A,
  - then we say that parts p_i:P_i and p_j:P_j **share** attribute A.
  - that is, that a:**attr**_A(p_i) (and a′:**attr**_A(p_j)) is a **shared attribute**
  - (with a=a′ always (□) holding).

Attribute Naming:

- Thus the domain describer has to exert great care when naming attribute types.
  - If P_i and P_j are two distinct types of a domain
  - then if and only if an attribute of P_i is to be shared with an attribute of P_j
  - must that attribute be identically named in the description of P_i and P_j.

Example 42. Shared Attributes. Examples of shared attributes:

- Bus **timetable attributes** have the same value as the regional transport system **timetable attribute**.
- Bus **clock attributes** have the same value as the regional transport system **clock attribute**.
- Bus **owner attributes** have the same value as the regional transport system **owner attribute**.
- Bank customer **passbooks** record bank transactions on, for example, demand/deposit accounts share values with the bank general ledger **passbook** entries.
- A link incident upon or emanating from a hub shares the **connection** between that link and the hub as an attribute.
- Two pipeline units[15], p_i, p_j, that are **connected**, such that an outlet π_j of p_i "feeds into" an inlet π_i of p_j, are said to share the connection (modeled by, e.g., $\{(\pi_i, \pi_j)\}$. ■

[15] See upcoming Example 33 on Slide 162

Example 43 . Shared Timetables:

- The fleet and vehicles of Example 20 on Slide 123 and Example 21 on Slide 130 is that of a bus company.

51 From the fleet and from the vehicles we observe unique identifiers.

52 Every bus mereology records the same one unique fleet identifier.

53 The fleet mereology records the set of all unique bus identifiers.

54 A bus timetable is a share fleet and bus attribute.

type

51. FI, VI, BT

value

51. **uid**_F: F → FI
51. **uid**_V: V → VI
52. **obs_mereo**_F: F → VI-**set**
53. **obs_mereo**_V: V → FI
54. **attr**_BT: (F|V) → BT

axiom

$\square$ ∀ f:F ⇒
∀ v:V · v ∈ **obs_part**_Vs(**obs_part**_VC(f)) · **attr**_BT(f) = **attr**_BT(v)

[which is the same as]

$\square$ ∀ f:F ⇒
{**attr**_BT(f)}={**attr**_BT(v):v:V·v ∈ **obs_part**_Vs(**obs_part**_VC(f))}

- Part attributes of one sort, P_i, may be simple type expressions such as
  - A-**set**,
  - where A may be an attribute of some other part sort, P_j,
  - in which case we say that part attributes
    - A-**set** and
    - A

    are shared.

Example 44 . Shared Passbooks:

55 A banking system contains

- an administration and
- a set of customers.

56 The administration contains a general ledger.

57 An attribute of a general ledger is a set of passbooks.

58 An attribute of a customer is that of a passbook.

59 Passbooks are uniquely identified by unique customer identifiers.

type

55. [parts] BS, AD, GL, CS, Cs = C-**set**

58. [attributes] PB

value

55. **obs_part**_AD: BS → AD

56. **obs_part**_GL: AD → GL

57. **attr**_PBs: GL → PB-**set**

55. **obs_part**_CS: BS → CS

55. **obs_part**_Cs: BS → Cs

58. **attr**_PB: C → PB

59. **uid**_PB: PB → PBI

axiom

□ ∀ bs:BS ·

attr_PBs(**attr**_GL(**obs_part**_AD(bs)))

= {**attr**_PB(c)|c:C·c ∈ **obs_part**_Cs(**obs_part**_CS(bs))} ■

Dines Bjørner's MAP-i Lecture # 3

End of MAP-i Lecture # 3: Unique Identifiers, Mereologies and Attributes

Monday, 25 May 2015: 14:30–15:15