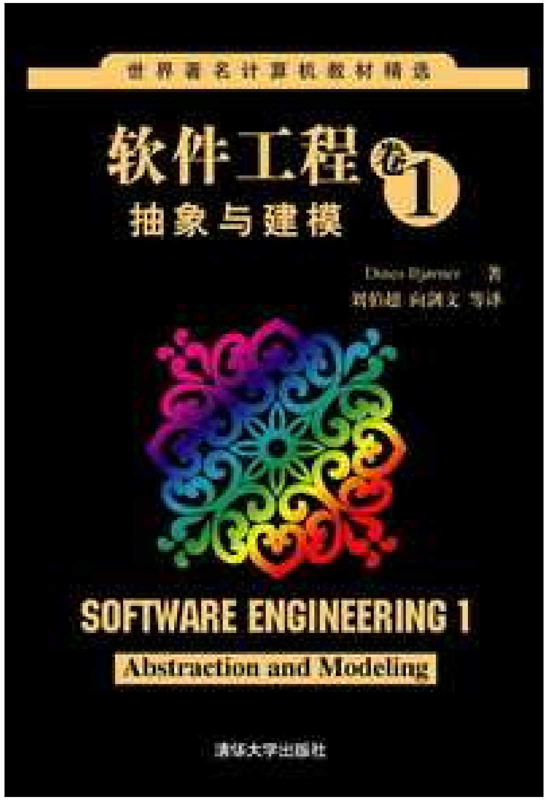

HAPPY TO SEE YOU AGAIN

Begin of Lecture 4: Middle Session — Perdurant Entities

Actions and Events

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

Tutorial Schedule

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

1 **Introduction** Slides 1–35

2 **Endurant Entities: Parts** Slides 36–110

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

3 **Endurant Entities: Materials, States** **Slides 111–142**

$\surd$ **4 Perdurant Entities: Actions and Events** **Slides 143–174**

5 **Perdurant Entities: Behaviours** Slides 175–285

Lunch **12:30–14:00**

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

6 **A Calculus: Analysers, Parts and Materials** Slides 286–339

7 **A Calculus: Function Signatures and Laws** Slides 340–377

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

8 **Domain and Interface Requirements** Slides 378–424

9 **Conclusion: Comparison to Other Work** Slides 428–460

Conclusion: What Have We Achieved Slides 425–427 + 461–472

8. Discrete Perdurants

8.1. General

- From Wikipedia:
  - *Perdurant: Also known as occurrent, accident or happening.*
  - *Perdurants are those entities for which only a fragment exists if we look at them at any given snapshot in time.*
  - *When we freeze time we can only see a fragment of the perdurant.*
  - *Perdurants are often what we know as processes, for example 'running'.*
  - *If we freeze time then we only see a fragment of the running, without any previous knowledge one might not even be able to determine the actual process as being a process of running.*
  - *Other examples include an activation, a kiss, or a procedure.*

- We shall consider **action**s and **event**s
  - to occur instantaneously,
  - that is, in time, but taking no time
- Therefore we shall consider **action**s and **event**s to be **perdurant**s.

8.2. Discrete Actions

- By a **function** we understand
  - a thing
  - which when **applied** to a **value**, called its **argument**,
  - **yield**s a **value**, called its **result**.
- An **action** is
  - a **function**
  - **invoked** on a **state value**
  - and is one that potentially changes that value.

Example: 28 Transport Net and Container Vessel Actions.

- *Inserting* and *removing* hubs and links in a net are considered actions.
- *Setting* the traffic signals for a hub (which has such signals) is considered an action.
- *Loading* and *unloading* containers from or unto the top of a container stack are considered actions. ■

8.2.1. An Aside on Actions

Think'st thou existence doth depend on time?
It doth; but actions are our epochs.
George Gordon Noel Byron,
Lord Byron (1788-1824) Manfred. Act II. Sc. 1.

- *"An action is*
  - *something an agent does*
  - *that was 'intentional under some description'"* [Davidson1980].
- That is, actions are performed by agents.
  - We shall not yet go into any deeper treatment of agency or agents. We shall do so later.
    - Agents will here, for simplicity, be considered behaviours,
    - and are treated later in this lecture.

- As to the relation between **intention** and **action**
  - we note that Davidson wrote: 'intentional under some description'
  - and take that as our cue:
    - the agent follows a script,
    - that is, a behaviour description,
    - and invokes actions accordingly,
    - that is, follow, or honours that script.
- The philosophical notion of 'action' is over-viewed in [sep-action].
- We
  - observe actions in the domain
  - but describe "their underlying" functions.
- Thus we abstract from the **time**s at which actions occur.

8.2.2. Action Signatures

- By an **action signature** we understand a quadruple:
  - a function name,
  - a function definition set type expression,
  - a total or partial function designator ($\rightarrow$, respectively $\stackrel{\sim}{\rightarrow}$), and
  - a function image set type expression:
    fct_name: A $\rightarrow$ Σ ($\rightarrow$|$\stackrel{\sim}{\rightarrow}$) Σ [$\times$ R],

  where $(X \mid Y)$ means either X or Y, and $[Z]$ means optional Z.

Example: 29 Action Signatures: Nets and Vessels.

insert_Hub: N$\rightarrow$H$\stackrel{\sim}{\rightarrow}$N;
remove_Hub: N$\rightarrow$HI$\stackrel{\sim}{\rightarrow}$N;
set_Hub_Signal: N$\rightarrow$HI$\stackrel{\sim}{\rightarrow}H\Sigma\stackrel{\sim}{\rightarrow}$N
load_Container: V$\rightarrow$C$\rightarrow$StackId$\stackrel{\sim}{\rightarrow}$V; and
unload_Container: V$\rightarrow$StackId$\stackrel{\sim}{\rightarrow}$(V$\times$C). ■

8.2.3. Action Definitions

- There are a number of ways in which to characterise an action.
- One way is to characterise its underlying function by a pair of predicates:
  - precondition: a predicate over function arguments — which includes the state, and
  - postcondition: a predicate over function arguments, a proper argument state and the desired result state.
  - If the precondition holds, i.e., is **true**, then the arguments, including the argument state, forms a proper 'input' to the action.
  - If the postcondition holds, assuming that the precondition held, then the resulting state [and possibly a yielded, additional "result" (R)] is as they would be had the function been applied.

Example: 30 **Transport Nets: Insert Hub Action.** We give one example.

19. The insert action applies to a net and a hub and conditionally yields an updated net.
    (a) The condition is that there must not be a hub in the "argument" net with the same unique hub identifier as that of the hub to be inserted and
    (b) the hub to be inserted does not initially designate links with which it is to be connected.
    (c) The updated net contains all the hubs of the initial net "plus" the new hub.
    (d) and the same links.

value

19. insert_H: N $\rightarrow$ H $\stackrel{\sim}{\rightarrow}$ N

19. insert_H(n)(h) **as** n$'$, **pre**: pre_insert_H(n)(h), **post**: post_insert_H(n)(h)

19(a). pre_insert_H(n)(h) $\equiv$

19(a). $\sim\exists$ h$'$:H $\cdot$ h$'$ $\in$ obs_Hs(n) $\wedge$ uid_HI(h)=uid_HI(h$'$)

19(b). $\wedge$ mereo_H(h) = {}

19(c). post_insert_H(n)(h)(n$'$) $\equiv$

19(c). (n$'$)obs_Hs(n) $\cup$ {h} = obs_Hs(n$'$)

19(d). $\wedge$ obs_Ls(n) = obs_Ls(n$'$)

- We refer to the notes accompanying these lectures.
- There you will find definitions of insert_link, remove_hub and remove_link action functions.

- What is not expressed, but tacitly assume in the above pre- and post-conditions is
  - that the state, here n, satisfy invariant criteria before (i.e. n) and after (i.e., n') actions,
  - whether these be implied by axioms
  - or by well-formedness predicates.

  over parts.
- This remark applies to any definition of actions, events and behaviours.

Example: 31 Action: Remove Container from Vessel. We give the second of two examples.

20. The remove_Container_from_Vessel action applies to a vessel and a stack address and conditionally yields an updated vessel and a container.
    (a) We express the 'remove from vessel' function primarily by means of an auxiliary function remove_C_from_BS, remove_C_from_BS(obs_BS(v))(stid), and some further post-condition on the before and after vessel states (cf. Item 20(d)).
    (b) The remove_C_from_BS function yields a pair: an updated set of bays and a container.
    (c) When obs_erving the BayS from the updated vessel, v′, and pairing that with what is assumed to be a vessel, then one shall obtain the result of remove_C_from_BS(obs_BS(v))(stid).
    (d) Updating, by means of remove_C_from_BS(obs_BS(v))(stid), the bays of a vessel must leave all other properties of the vessel unchanged.

21. The pre-condition for remove_C_from_BS(bs)(stid) is
    (a) that stid is a valid_address in bs, and
    (b) that the stack in bs designated by stid is non_empty.
22. The post-condition for remove_C_from_BS(bs)(stid) wrt. the updated bays, bs′, is
    (a) that the yielded container, i.e., c, is obtained, get_C(bs)(stid), from the top of the non-empty, designated stack,
    (b) that the mereology of bs′ is unchanged, unchanged_mereology(bs,bs′). wrt. bs. ,
    (c) that the stack designated by stid in the "input" state, bs, is popped, popped_designated_stack(bs,bs′)(stid), and
    (d) that all other stacks are unchanged in bs′ wrt. bs, unchanged_non_designated_stacks(bs,bs′)(stid).

value

20. remove_C_from_V: V $\rightarrow$ StackId $\stackrel{\sim}{\rightarrow}$ (V$\times$C)
20. remove_C_from_V(v)(stid) **as** (v$'$,c)
20(c). (obs_BS(v$'$),c) = remove_C_from_BS(obs_BS(v))(stid)
20(d). $\wedge$ props(v)=props(v$''$)

20(b). remove_C_from_BS: BS $\rightarrow$ StackId $\rightarrow$ (BS$\times$C)
20(a). remove_C_from_BS(bs)(stid) **as** (bs$'$,c)
21(a). **pre**: valid_address(bs)(stid)
21(b). $\wedge$ non_empty_designated_stack(bs)(stid)
22(a). **post**: c = get_C(bs)(stid)
22(b). $\wedge$ unchanged_mereology(bs,bs$'$)
22(c). $\wedge$ popped_designated_stack(bs,bs$'$)(stid)
22(d). $\wedge$ unchanged_non_designated_stacks(bs,bs$'$)(stid)

- This example hints at *a theory of container vessel bays, rows and stacks*.
- More on that is found in Appendix C. ■

- There are other ways of defining functions.
- But the form of these are not material to the aims of this tutorial.

Modelling Actions, I/III

- The **domain describer** has decided that an **entity** is a **perdurant** and is, or represents an **action**: was *"done by an agent and intentionally under some description"* [Davidson1980].
  - The domain describer has further decided that the observed action is of a class of actions — of the "same kind" — that need be described.
  - By actions of the 'same kind' is meant that these can be described by the same **function signature** and **function definition**.

Modelling Actions, II/III

- First the domain describer must decide on the underlying **function signature**.
  - The **argument type** and the **result type** of the signature are those of either previously identified
    - parts and/or materials,
    - unique part identifiers, and/or
    - attributes.

Modelling Actions, III/III

- Sooner or later the domain describer must decide on the **function definition**.
  - The form must be decided upon.
  - For pre/post-condition forms it appears to be convenient to have developed, "on the side", a **theory of mereology** for the part types involved in the function signature.

8.3. Discrete Events

- By an **event** we understand
  - a state change
  - resulting indirectly from an unexpected application of a function,
  - that is, that function was performed "surreptitiously".
- Events can be characterised by a pair of (before and after) states, a predicate over these and, optionally, a **time** or **time interval**.
- Events are thus like actions:
  - change states,
  - but are usually
    - either caused by "previous" actions,
    - or caused by "an outside action".

Example: 32 Events.

- *Container vessel:* A container falls overboard sometimes between times t and t'.
- *Financial service industry:* A bank goes bankrupt sometimes between times t and t'.
- *Health care:* A patient dies sometimes between times t and t'.
- *Pipeline system:* A pipe breaks sometimes between times t and t'.
- *Transportation:* A link "disappears" sometimes between times t and t'.

8.3.1. An Aside on Events

- We may observe an event, and
  - then we do so at a specific time or
  - during a specific time interval.
- But we wish to describe,
  - not a specific event
  - but a class of events of "the same kind".
- In this tutorial
  - we therefore do not ascribe
  - time points or time intervals
  - with the occurrences of events.

8.3.2. Event Signatures

- An event signature
  - is a predicate signature
  - having an event name,
  - a pair of state types $(\Sigma \times \Sigma)$,
  - a total function space operator $(\rightarrow)$
  - and a **Bool**ean type constant:
  - evt: $(\Sigma\times\Sigma) \rightarrow$ **Bool**.
- Sometimes there may be a good reason
  - for indicating the type, ET, of an event cause value,
  - if such a value can be identified:
  - evt: ET $\times$ $(\Sigma \times \Sigma) \rightarrow$ **Bool**.

8.3.3. Event Definitions

- An event definition takes the form of a predicate definition:
  - A predicate name and argument list, usually just a state pair,
  - an existential quantification
    - over some part (of the state) or
    - over some dynamic attribute of some part (of the state)
    - or combinations of the above
  - a pre-condition expression over the input argument(s),
  - an implication symbol ($\Rightarrow$), and
  - a post-condition expression over the argument(s).
- evt(σ,σ') = $\exists$ (ev:ET) • pre_evt(ev)(σ) $\Rightarrow$ post_evt(ev)(σ,σ').
- There may be variations to the above form.

Example: 33 Narrative of Link Event. The disappearance of a link in a net, for example due to a mud slide, or a bridge falling down, or a fire in a road tunnel, can, for example be described as follows:

23. Link disappearance is expressed as a predicate on the “before” and “after” states of the net. The predicate identifies the “missing” ℓink (!).

24. Before the disappearance of link ℓ in net n
    (a) the hubs h' and h'' connected to link ℓ
    (b) were connected to links identified by $\{l'_1, l'_2, \ldots, l'_p\}$ respectively $\{l''_1, l''_2, \ldots, l''_q\}$
    (c) where, for example, l'_i, l''_j are the same and equal to uid_$\Pi(\ell)$.

25. After link ℓ disappearance there are instead
    (a) two separate links, ℓ_i and ℓ_j, "truncations" of ℓ
    (b) and two new hubs h''' and h''''
    (c) such that ℓ_i connects h' and h''' and
    (d) ℓ_j connects h'' and h'''';
    (e) Existing hubs h' and h'' now have mereology
        i. $\{l'_1, l'_2, \ldots, l'_p\} \setminus \{\mathsf{uid\_}\Pi(\ell)\} \cup \{\mathsf{uid\_}\Pi(\ell_i)\}$ respectively
        ii. $\{l''_1, l''_2, \ldots, l''_q\} \setminus \{\mathsf{uid\_}\Pi(\ell)\} \cup \{\mathsf{uid\_}\Pi(\ell_j)\}$
26. All other hubs and links of n are unaffected. ■

Example: 34 Formalisation of Link Event. Continuing Example 33 above:

23. link_disappearance: N $\times$ N $\rightarrow$ **Bool**
23. link_disappearance(n,n$'$) $\equiv$
23. $\exists$ ℓ:L $\cdot$ pre_link_dis(n,ℓ) $\Rightarrow$ post_link_dis(n,ℓ,n$'$)

24. pre_link_dis: N $\times$ L $\rightarrow$ **Bool**
24. pre_link_dis(n,ℓ) $\equiv$ ℓ $\in$ obs_Ls(n)

27. We shall “explain” *link disappearance* as the combined, instantaneous effect of

    (a) first a **remove link** “event” where the **removed link** connected **hubs** hi_j and hi_k;

    (b) then the **insertion** of two new, “fresh” **hubs**, h_α and h_β;

    (c) “followed” by the **insertion** of two new, “fresh” links $\mathsf{l}_{j\alpha}$ and $\mathsf{l}_{k\beta}$ such that

        i. $\mathsf{l}_{j\alpha}$ connects hi_j and h_α and

        ii. $\mathsf{l}_{k\beta}$ connects hi_k and $\mathsf{h}_{k\beta}$

value

27. post_link_dis(n,ℓ,n$'$) $\equiv$
27(a). **let** n$''$ = remove_link(n)(uid_L(ℓ)) **in**
27(b). **let** h$_\alpha$,h$_\beta$:H $\cdot$ $\{$h$_\alpha$,h$_\beta\}$ $\cap$ obs_Hs(n)=$\{\}$ **in**
27(b). **let** n$'''$ = insert_H(n$''$)(h$_\alpha$) **in**
27(b). **let** n$''''$ = insert_H(n$'''$)(h$_\beta$) **in**
27(c). **let** l$_{j\alpha}$,l$_{k\beta}$:L $\cdot$ $\{$l$_{j\alpha}$,l$_{k\beta}\}$ $\cap$ obs_Ls(n)=$\{\}$ **in**
27((c))i. **let** n$'''''$ = insert_L(n$''''$)(l$_{j\alpha}$) **in**
27((c))ii. n$'$ = insert_L(n$'''''$)(l$_{k\beta}$) **end end end end end end**

- We refer to the notes accompanying these lectures.
- There you will find definitions of insert_link, remove_hub and remove_link action functions.

Modelling Events I/II

- The **domain describer** has decided that an **entity** is a **perdurant** and is, or represents an **event**: occurred surreptitiously, that is, was not an action that was *"done by an agent and intentionally under some description"* [Davidson1980].
  - The domain describer has further decided that the observed event is of a class of events — of the "same kind" — that need be described.
  - By events of the 'same kind' is meant that these can be described by the same **predicate function signature** and **predicate function definition**.

Modelling Events, II/II

- First the domain describer must decide on the underlying **predicate function signature**.
  - The **argument type** and the **result type** of the signature are those of either previously identified
    - parts,
    - unique part identifiers, or
    - attributes.
- Sooner or later the domain describer must decide on the **predicate function definition**.
  - For predicate function definitions it appears to be convenient to have developed, "on the side", a **theory of mereology** for the part types involved in the function signature.

End of Lecture 4: Middle Session — Perdurant Entities

Actions and Events

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


Software
Engineering 1


Software
Engineering 2


Software
Engineering 3

MINI BREAK