Software Design, Modelling and Analysis in UML

Lecture 13: Hierarchical State Machines I

2012-01-11

Prof. Dr. Andreas Podelski, **Dr. Bernd Westphal**

Albert-Ludwigs-Universität Freiburg, Germany

Contents & Goals

Last Lecture:

- RTC-Rules: Discard, Dispatch, Commence.
- Step, RTC, Divergence

This Lecture:

- **Educational Objectives:** Capabilities for following tasks/questions.
  - What does this State Machine mean? What happens if I inject this event?
  - Can you please model the following behaviour.
  - What is: initial state.
  - What does this **hierarchical** State Machine mean? What **may happen** if I inject this event?
  - What is: AND-State, OR-State, pseudo-state, entry/exit/do, final state, ...
- **Content:**
  - Putting It All Together
  - Hierarchical State Machines Syntax

Step and Run-to-completion Step

Run-to-Completion Step: Discussion.

What people may **dislike** on our definition of RTC-step is that it takes a **global** and **non-compositional** view. That is:

- In the projection onto a single object we still **see** the effect of interaction with other objects.
- Adding classes (or even objects) may change the divergence behaviour of existing ones.
- Compositional would be: the behaviour of a set of objects is determined by the behaviour of each object "in isolation".
  Our semantics and notion of RTC-step doesn't have this (often desired) property.

Can we give (syntactical) criteria such that any global run-to-completion step is an interleaving of local ones?

Maybe: **Strict interfaces**. (*Proof left as exercise...*)

- **(A)**: Refer to private features only via "self".
  (Recall that other objects of the same class can modify private attributes.)
- **(B)**: Let objects only communicate by events, i.e.
  don't let them modify each other's local state via links **at all**.

Putting It All Together

The Missing Piece: Initial States

Recall: a labelled transition system is $(S, \rightarrow, S_0)$. We **have**

- S: system configurations (σ, ε)
- $\rightarrow$: labelled transition relation $(\sigma, \varepsilon) \xrightarrow[u]{(cons, Snd)} (\sigma', \varepsilon')$.

Wanted: initial states S_0.

Proposal:
Require a (finite) set of **object diagrams** $\mathcal{OD}$ as part of a UML model

$$(\mathscr{CD}, \mathscr{SM}, \mathscr{OD}).$$

And set

$$S_0 = \{(\sigma, \varepsilon) \mid \sigma \in G^{-1}(\mathcal{OD}), \mathcal{OD} \in \mathscr{OD}, \varepsilon \text{ empty}\}.$$

Other Approach: (used by Rhapsody tool) multiplicity of classes.
We can read that as an abbreviation for an object diagram.

Semantics of UML Model — So Far

The **semantics** of the **UML model**

$$\mathcal{M} = (\mathscr{CD}, \mathscr{SM}, \mathscr{OD})$$

where

- some classes in $\mathscr{CD}$ are stereotyped as 'signal' (standard), some signals and attributes are stereotyped as 'external' (non-standard),
- there is a 1-to-1 relation between classes and state machines,
- $\mathscr{OD}$ is a set of object diagrams over $\mathscr{CD}$,

is the **transition system** $(S, \rightarrow, S_0)$ constructed on the previous slide.

The **computations of** $\mathcal{M}$ are the computations of $(S, \rightarrow, S_0)$.

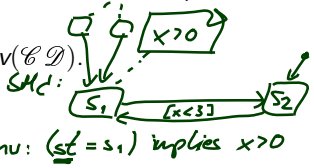
OCL Constraints and Behaviour

- Let $\mathcal{M} = (\mathscr{CD}, \mathscr{SM}, \mathscr{OD})$ be a UML model.
- We call $\mathcal{M}$ **consistent** iff, for each OCL constraint $expr \in \mathit{Inv}(\mathscr{CD})$,

  $\sigma \models expr$ for each "reasonable point" (σ, ε) of computations of $\mathcal{M}$.

  (Cf. exercises and tutorial for discussion of "reasonable point".)

Note: we could define $\mathit{Inv}(\mathscr{SM})$ similar to $\mathit{Inv}(\mathscr{CD})$.



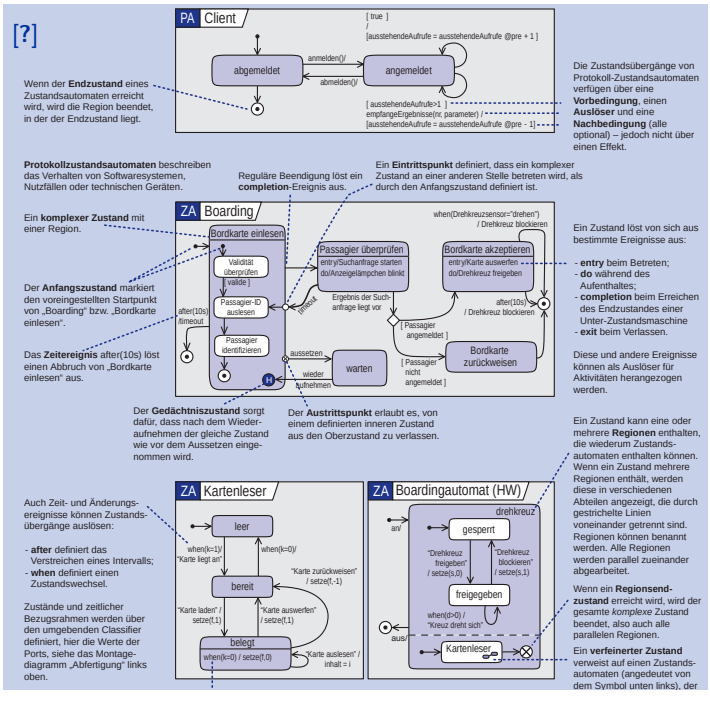
Pragmatics:

- In **UML-as-blueprint mode**, if $\mathscr{SM}$ doesn't exist yet, then $\mathcal{M} = (\mathscr{CD}, \emptyset, \mathscr{OD})$ is typically asking the developer to provide $\mathscr{SM}$ such that $\mathcal{M}' = (\mathscr{CD}, \mathscr{SM}, \mathscr{OD})$ is consistent.

  If the developer makes a mistake, then $\mathcal{M}'$ is inconsistent.
- **Not common**: if $\mathscr{SM}$ is given, then constraints are also considered when choosing transitions in the RTC-algorithm. In other words: even in presence of mistakes, the $\mathscr{SM}$ never move to inconsistent configurations.

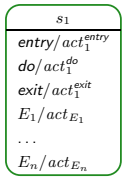
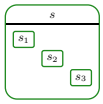
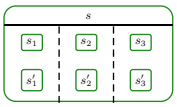
Hierarchical State Machines

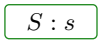
UML State-Machines: What do we have to cover?



The Full Story

UML distinguishes the following **kinds of states**:

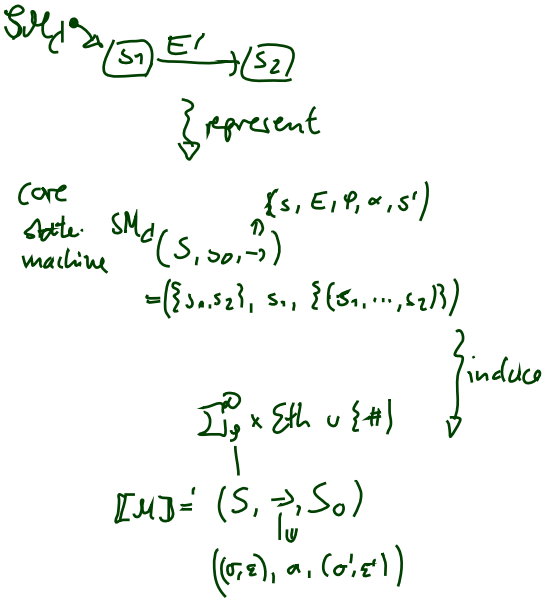
	example
simple state	
final state	
composite state	
OR	
AND	

	example
pseudo-state	
initial	
(shallow) history	
deep history	
fork/join	
junction, choice	
entry point	
exit point	
terminate	
submachine state	

Representing All Kinds of States

- **Until now**:

$$(S, s_0, \rightarrow), \quad s_0 \in S, \rightarrow \subseteq S \times (\mathscr{E} \cup \{\_\}) \times Expr_{\mathscr{S}} \times Act_{\mathscr{S}} \times S$$



Representing All Kinds of States

- **Until now**:

$$(S, s_0, \rightarrow), \quad s_0 \in S, \rightarrow \subseteq S \times (\mathscr{E} \cup \{\_\}) \times Expr_{\mathscr{S}} \times Act_{\mathscr{S}} \times S$$

- **From now on**: (**hierarchical**) **state machines**

$$(S, kind, region, \Rightarrow, \psi, annot)$$

where

- $S \supseteq \{top\}$ is a finite set of states **(as before)**,
- $kind : S \to \{st, init, fin, shist, dhist, fork, join, junc, choi, ent, exi, term\}$ is a function which labels states with their **kind**, **(new)**
- $region : S \to 2^{2^S}$ is a function which characterises the **regions** of a state, **(new)**
- $\Rightarrow$ is a set of transitions, **(changed)**
- $\psi : (\Rightarrow) \to 2^S \times 2^S$ is an **incidence function**, and **(new)**
- $annot : (\Rightarrow) \to (\mathscr{E} \cup \{\_\}) \times Expr_{\mathscr{S}} \times Act_{\mathscr{S}}$ provides an annotation for each transition. **(new)**

(s_0 is then redundant — replaced by proper state (!) of kind '*init*'.)

From UML to Hierarchical State Machines: By Example

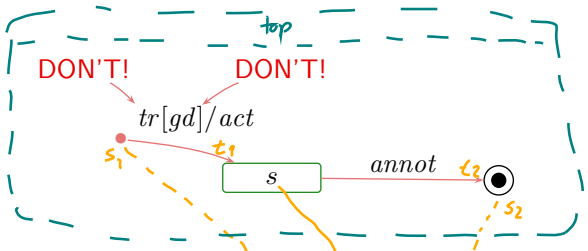
$(S, kind, region, \rightarrow, \psi, annot)$

	example	$\in S$	$kind$	$region$
simple state	s	s	st	$\emptyset$
final state	◉	q	fin	$\emptyset$
composite state				
OR	s with s_1, s_2, s_3	s	st	$\{\{s_1, s_2, s_3\}\}$
AND	s with s_1, s_2, s_3 / s'_1, s'_2, s'_3	s	st	$\{\{s_1, s'_1\}, \{s_2, s'_2\}, \{s_3, s'_3\}\}$
submachine state	(later)	-	-	
pseudo-state	●, Ⓗ, ...	q, q'	$init, hist, \ldots$	$\emptyset$

region

$(s, kind(s))$ for short

From UML to Hierarchical State Machines: By Example



... translates to $(S, kind, region, \rightarrow, \psi, annot) =$

$(\underbrace{\{(top, st), (s_1, init), (s, st), (s_2, fin)\}}_{S, kind},$

$\underbrace{\{s_1 \mapsto \emptyset, s_2 \mapsto \emptyset, s \mapsto \emptyset, top \mapsto \{\{s, s_1, s_2\}\}\}}_{region},$

$\underbrace{\{t_1, t_2\}}_{\rightarrow}, \underbrace{\{t_1 \mapsto (\{s_1\}, \{s\}), t_2 \mapsto (\{s\}, \{s_2\})\}}_{\psi},$

$\underbrace{\{t_1 \mapsto (tr, gd, act), t_2 \mapsto annot\}}_{annot})$

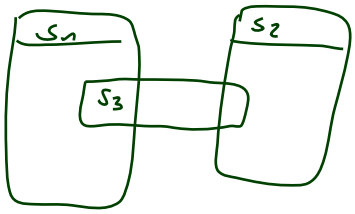
Well-Formedness: Regions (follows from diagram)

Name Def. ↓	$\in S$	$kind$	$region \subseteq 2^S, S_i \subseteq S$	$child_1 \subseteq S$
simple state	s	*st*	$\emptyset$	$\emptyset$
final state	s	*fin*	$\emptyset$	$\emptyset$
composite state	s	*st*	$\{S_1, \ldots, S_n\}, n \geq 1$	$S_1 \cup \cdots \cup S_n$
pseudo-state	s	*init*, ...	$\emptyset$	$\emptyset$
implicit top state	top	*st*	$\{S_1\}$	S_1

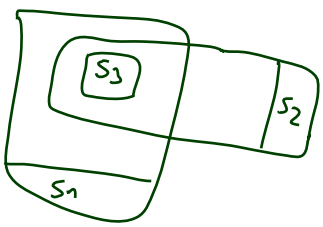
WFR / Observations:

- Each state (except for top) lies in exactly one region,
- States $s \in S$ with $kind(s) = st$ **may comprise** regions.
  - No region: simple state.
  - One region: OR-state.
  - Two or more regions: AND-state.
- Final and pseudo states **don't comprise** regions.
- The region function induces a **child** function.

Each state (exc. top) lies in exactly one region because we may not draw



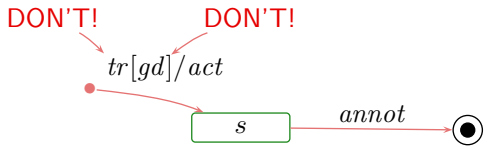
or



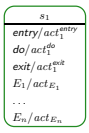
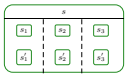
Well-Formedness: Initial State (requirement on diagram)

- Each non-empty region has a reasonable initial state and at least one transition from there, i.e.
  - for each $s \in S$ with $region(s) = \{S_1, \ldots, S_n\}$, $n \geq 1$, for each $1 \leq i \leq n$,
  - there exists exactly one initial pseudo-state $(s_1^i, \mathit{init}) \in S_i$ and at least one transition $t \in \longrightarrow$ with s_1^i as source
  - and such transition's target s_2^i is in S_i, and (**for simplicity!**) $kind(s_2^i) = \mathit{st}$, and $annot(t) = (\_, \mathit{true}, act)$.
- No ingoing transitions to initial states.
- No outgoing transitions from final states.

- Recall:



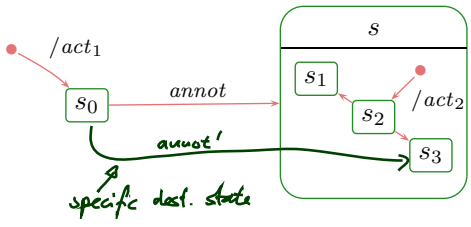
Plan

	example		example
simple state		pseudo-state initial	
		(shallow) history	
		deep history	
		fork/join	
final state			
composite state		junction, choice	
OR		entry point	
		exit point	
AND		terminate	
		submachine state	$S:s$

- Initial pseudostate, final state.
- Composite states.
- Entry/do/exit actions, internal transitions.
- History and other pseudostates, the rest.

Initial Pseudostates and Final States

Initial Pseudostate



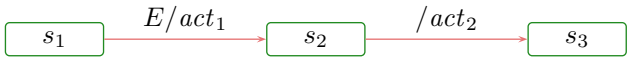
Principle:

- when entering a region **without** a specific destination state,
- then go to a state which is destination of an initiation transition,
- execute the action of the chosen initiation transitions **between** exit and entry actions.

Special case: the region of top.

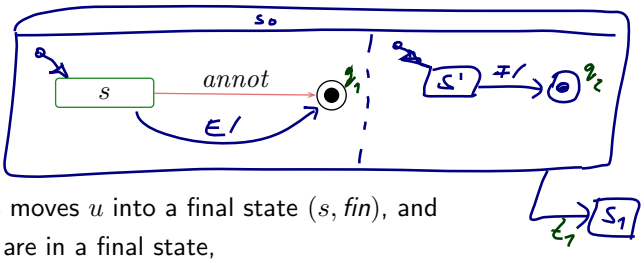
- If class C has a state-machine, then "create-C transformer" is the concatenation of
  - the transformer of the "constructor" of C (here not introduced explicitly) and
  - a transformer corresponding to ⟨one⟩ initiation transition of the top region.

Towards Final States: Completion of States



- Transitions without trigger can **conceptionally** be viewed as being sensitive for the "completion event".
- Dispatching (here: E) **can then alternatively** be **viewed** as
  - (i) fetch event (here: E) from the ether,
  - (ii) take an enabled transition (here: to s_2),
  - (iii) remove event from the ether,
  - (iv) after having finished entry and do action of current state (here: s_2) — the state is then called **completed** —,
  - (v) raise a **completion event** — with strict priority over events from ether!
  - (vi) if there is a transition enabled which is sensitive for the completion event,
    - then take it (here: (s_2, s_3)).
    - otherwise become stable.

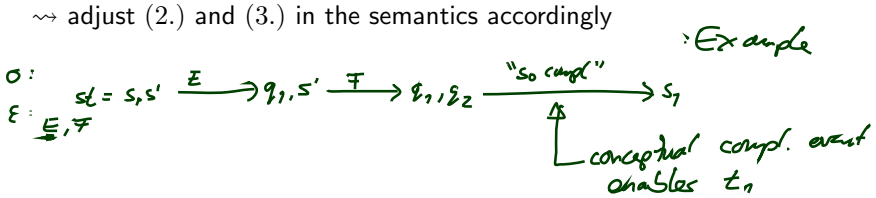
Final States



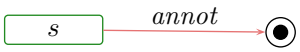
- If
  - a step of object u moves u into a final state (s, fin), and
  - all sibling regions are in a final state,

  then (conceptionally) a completion event for the current composite state s is raised.
- If there is a transition of a **parent state** (i.e., inverse of $child$) of s enabled which is sensitive for the completion event,
  - then take that transition,
  - otherwise kill u

  $\rightsquigarrow$ adjust $(2.)$ and $(3.)$ in the semantics accordingly



Final States



- If
  - a step of object u moves u into a final state (s, fin), and
  - all sibling regions are in a final state,

  then (conceptionally) a completion event for the current composite state s is raised.
- If there is a transition of a **parent state** (i.e., inverse of $child$) of s enabled which is sensitive for the completion event,
  - then take that transition,
  - otherwise kill u

  $\rightsquigarrow$ adjust $(2.)$ and $(3.)$ in the semantics accordingly
- **One consequence**: u never survives reaching a state (s, fin) with $s \in child(top)$.
- **Now:** in Core State Machines, there is no parent state.
- **Later:** in Hierarchical ones, there may be one.

References