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** $(S, \to, S_0)$ constructed on the previous slide.

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

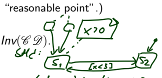
OCL Constraints and Behaviour

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

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

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

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



context C inv: (st = s1) implies x>0

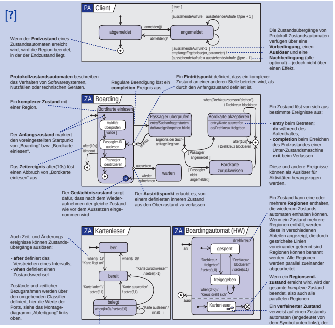
Pragmatics:

- 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?


[?]
PA Client
abgemeldet
angemeldet
anmelden()/
abmelden()/
[true]
/
[ausstehendeAufrufe = ausstehendeAufrufe @pre + 1]
[ausstehendeAufrufe>1]
empfangeErgebnisse(nr, parameter) /
[ausstehendeAufrufe = ausstehendeAufrufe @pre - 1]
ZA Boarding
Bordkarte einlesen
Validität überprüfen
[valide]
Passagier-ID auslesen
Passagier identifizieren
after(10s) /timeout
/timeout
Passagier überprüfen
entry/Suchanfrage starten
do/Anzeigelämpchen blinkt
Ergebnis der Suchanfrage liegt vor
[Passagier angemeldet]
[Passagier nicht angemeldet]
Bordkarte akzeptieren
entry/Karte auswerfen
do/Drehkreuz freigeben
when(Drehkreuzsensor="drehen") / Drehkreuz blockieren
after(10s) / Drehkreuz blockieren
Bordkarte zurückweisen
aussetzen
wieder aufnehmen
warten
H
ZA Kartenleser
leer
bereit
belegt
when(k=1)/ "Karte liegt an"
when(k=0)/
"Karte zurückweisen" / setze(f,-1)
"Karte laden" / setze(f,1)
"Karte auswerfen" / setze(f,1)
when(k=0) / setze(f,0)
"Karte auslesen" / inhalt = i
ZA Boardingautomat (HW)
drehkreuz
an/
aus/
gesperrt
freigegeben
"Drehkreuz freigeben" / setze(s,0)
"Drehkreuz blockieren" / setze(s,1)
when(d>0) / "Kreuz dreht sich"
Kartenleser

Wenn der **Endzustand** eines Zustandsautomaten erreicht wird, wird die Region beendet, in der der Endzustand liegt.

Die Zustandsübergänge von Protokoll-Zustandsautomaten verfügen über eine **Vorbedingung**, einen **Auslöser** und eine **Nachbedingung** (alle optional) – jedoch nicht über einen Effekt.

Protokollzustandsautomaten beschreiben das Verhalten von Softwaresystemen, Nutzfällen oder technischen Geräten.

Reguläre Beendigung löst ein **completion**-Ereignis aus.

Ein **Eintrittspunkt** definiert, dass ein komplexer Zustand an einer anderen Stelle betreten wird, als durch den Anfangszustand definiert ist.

Ein **komplexer Zustand** mit einer Region.

Der **Anfangszustand** markiert den voreingestellten Startpunkt von „Boarding" bzw. „Bordkarte einlesen".

Das **Zeitereignis** after(10s) löst einen Abbruch von „Bordkarte einlesen" aus.

Ein Zustand löst von sich aus bestimmte Ereignisse aus:
- **entry** beim Betreten;
- **do** während des Aufenthaltes;
- **completion** beim Erreichen des Endzustandes einer Unter-Zustandsmaschine
- **exit** beim Verlassen.

Diese und andere Ereignisse können als Auslöser für Aktivitäten herangezogen werden.

Der **Gedächtniszustand** sorgt dafür, dass nach dem Wiederaufnehmen der gleiche Zustand wie vor dem Aussetzen eingenommen wird.

Der **Austrittspunkt** erlaubt es, von einem definierten inneren Zustand aus den Oberzustand zu verlassen.

Ein Zustand kann eine oder mehrere **Regionen** enthalten, die wiederum Zustandsautomaten enthalten können. Wenn ein Zustand mehrere Regionen enthält, werden diese in verschiedenen Abteilen angezeigt, die durch gestrichelte Linien voneinander getrennt sind. Regionen können benannt werden. Alle Regionen werden parallel zueinander abgearbeitet.

Auch Zeit- und Änderungsereignisse können Zustandsübergänge auslösen:

- **after** definiert das Verstreichen eines Intervalls;
- **when** definiert einen Zustandswechsel.

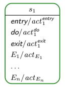
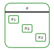
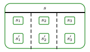
Zustände und zeitlicher Bezugsrahmen werden über den umgebenden Classifier definiert, hier die Werte der Ports, siehe das Montagediagramm „Abfertigung" links oben.

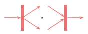
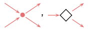
Wenn ein **Regionsendzustand** erreicht wird, wird der gesamte *komplexe* Zustand beendet, also auch alle parallelen Regionen.

Ein **verfeinerter Zustand** verweist auf einen Zustandsautomaten (angedeutet von dem Symbol unten links), der

The Full Story

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

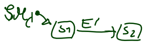
	example
simple state	
final state	◉
composite state	
OR	
AND	

	example
pseudo-state	
initial	●
(shallow) history	H
deep history	H*
fork/join	
junction, choice	
entry point	○
exit point	⊗
terminate	×
submachine state	$S : s$

Representing All Kinds of States

- **Until now**:

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



} represent

core state machine SM_d $(S, s_0, \overset{\{s, E, \varphi, \alpha, s'\}}{\rightarrow})$

$= (\{s_1, s_2\}, s_1, \{(s_1, \dots, s_2)\})$

} induce

$\Sigma_{\mathscr{S}}^{\mathscr{D}} \times Eth \cup \{\#\}$

$[\![M]\!] = (S, \Rightarrow, S_0)$

$\Rightarrow \ni ((\sigma, \varepsilon), \alpha, (\sigma', \varepsilon'))$

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**

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

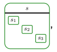
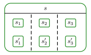
where (state machine)

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

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

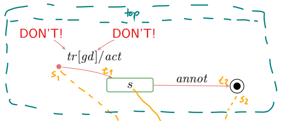
	example	$\in S$	*kind*	*region*
simple state	s	s	st	$\emptyset$
final state	◉	q	fin	$\emptyset$
composite state				
OR		s	st	$\{\{s_1, s_2, s_3\}\}$
AND		s	st	$\{\{s_1, s_1'\}, \{s_2, s_2'\}, \{s_3, s_3'\}\}$
submachine state	(later)			
pseudo-state	●, Ⓗ, ..	q, q'	init, shist, ..	$\emptyset$

some fresh value

region

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

From UML to Hierarchical State Machines: By Example



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

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

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

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

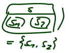
$$\underbrace{\{t_1 \mapsto (tr, gd, act), t_2 \mapsto annot\}}_{annot})$$

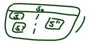
Well-Formedness: Regions (follows from diagram)

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

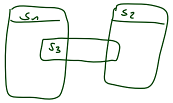
WFR / Observations:

- 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.

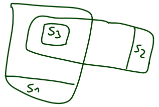
child() $= \{s_1, s_2\}$

child() $= \{s, s', s''\}$

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



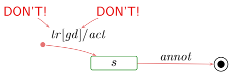
or



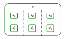
Well-Formedness: Initial State (requirement on diagram)

- Each non-empty region has a reasonable initial state and at least one transition from there, i.e.
  - for each $s \in S$ with $region(s) = \{S_1, \dots, S_n\}$, $n \geq 1$, 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 $t \in \to$ 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:



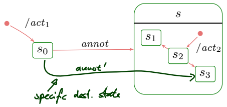
Plan

	example		example
simple state	s_1 *entry*$/act_1^{\mathit{entry}}$ *do*$/act_1^{\mathit{do}}$ *exit*$/act_1^{\mathit{exit}}$ E_1/act_{E_1} ... E_n/act_{E_n}	**pseudo-state**	
		initial	●
		(shallow) history	Ⓗ
		deep history	H*
		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



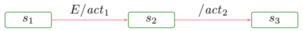
Principle:

- 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),

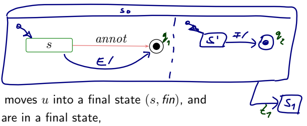
(iii) remove event from the ether,

(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

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

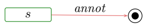
: Example

σ: $st = s, s' \xrightarrow{E} q_1, s' \xrightarrow{F} q_1, q_2 \xrightarrow{\text{"}s_0\text{ compl"}} s_1$

ε: E, F

conceptual compl. event enables t_1

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

  $\leadsto$ 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