

Software Design, Modelling and Analysis in UML

Lecture 15: Hierarchical State Machines I

2013-01-08

Prof. Dr. Andreas Podelski, Dr. Bernd Westphal

Albert-Ludwigs-Universität Freiburg, Germany

- 15 - 2013-01-08 - main -

Contents & Goals

Last Lecture:

- RTC-Rules: Discard, Dispatch, Commence.
- Step, RTC, Divergence
- Putting It All Together - *ODs for initial state*
- Rhapsody Demo

$$(s, \varepsilon) \xrightarrow[u]{\text{rtc}} (s', \varepsilon')$$

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:**
 - Hierarchical State Machines Syntax

- 15 - 2013-01-08 - Prelim -

Hierarchical State Machines

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

[Störrle, 2005]

Client

- States: *abgemeldet*, *angemeldet*
- Transitions: *anmelden()*, *abmelden()*
- Conditions: $[\text{ausstehendeAufrufe} = \text{ausstehendeAufrufe} @ \text{pre} + 1]$, $[\text{ausstehendeAufrufe} > 0]$

ZA Boarding

- States: *Bordkarte einlesen*, *Passagier überprüften*, *Bordkarte akzeptieren*, *Passagier überprüften*, *warten*, *Bordkarte zurückweisen*
- Transitions: *Validität überprüfen*, *Passagier-ID auslesen*, *Passagier überprüften*, *aussetzen*, *warten*, *Passagier überprüften*, *Bordkarte akzeptieren*, *aussetzen*, *warten*, *Bordkarte zurückweisen*
- Conditions: $[\text{Validität} = \text{valid}]$, $[\text{Passagier-ID} = \text{auslesen}]$, $[\text{Passagier überprüften}]$, $[\text{Bordkarte akzeptieren}]$, $[\text{aussetzen}]$, $[\text{warten}]$, $[\text{Bordkarte zurückweisen}]$
- Timeouts: *after(10s) timeout*

ZA Kartenleser

- States: *leer*, *bereit*, *belogen*
- Transitions: *Karte legt an*, *Karte laden*, *Karte auswerfen*, *Karte auslesen*
- Conditions: $[\text{when}(k=1)]$, $[\text{when}(k=0)]$, $[\text{when}(k=2)]$

ZA Boardingautomat (HW)

- States: *gesperrt*, *freigegeben*, *Kartenleser*
- Transitions: *drehkreuz*, *aus*
- Conditions: $[\text{Drehkreuz freigegeben}]$, $[\text{Drehkreuz gesperrt}]$, $[\text{Kreuz dreht sich}]$

Annotations and Explanations:

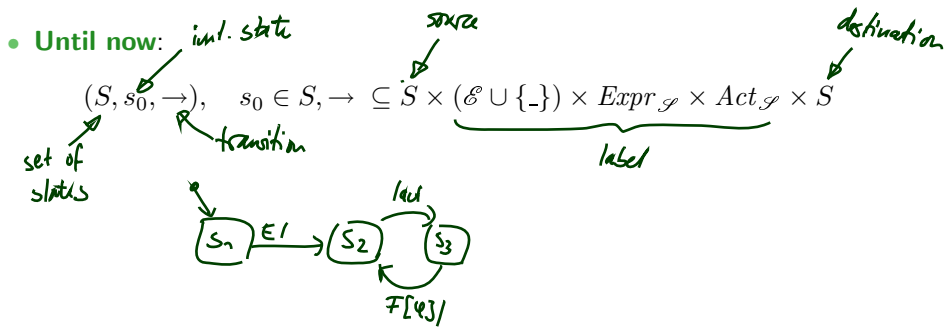
- Client:** "Wenn der Endzustand eines Zustandsautomaten erreicht wird, wird die Region beendet, in der der Endzustand liegt." (Left)
- Client:** "Die Zustandsübergänge von Protokoll-Zustandsautomaten verfügen über eine **Vorbereitung**, einen **Auslöser** und eine **Nachbedingung** (alle optional) – jedoch nicht über einen Effekt." (Right)
- Boarding:** "Protokollzustandsautomaten beschreiben das Verhalten von Softwaresystemen, Reguliäre Beendigung löst ein **completion**-Ereignis aus." (Left)
- Boarding:** "Ein **Eintrittspunkt** definiert, dass ein komplexer Zustand an einer anderen Stelle betrachtet wird, als durch den Anfangszustand definiert ist." (Right)
- Boarding:** "Ein Zustand löst von sich aus bestimmte Ereignisse aus." (Right)
- Boarding:** "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." (Right)
- Boarding:** "Wenn ein **Regionendzustand** erreicht wird, wird der gesamte komplexe Zustand beendet, also auch alle parallelen Regionen." (Right)
- Boarding:** "Ein **verfeinerter Zustand** verweist auf einen Zustandsautomaten (angedeutet von dem Symbol unten links), der..." (Right)
- Boarding:** "Der **Anfangszustand** markiert den vorgestellten Startpunkt von „Boarding“ bzw. „Bordkarte einlesen“." (Left)
- Boarding:** "Das **Zeitereignis** *after(10s)* löst einen Abbruch von „Bordkarte einlesen“ aus." (Left)
- Boarding:** "Der **Gedächtniszustand** sorgt dafür, dass nach dem Wieder aufnehmen der gleiche Zustand wie vor dem Aussetzen eingenommen wird." (Left)
- Boarding:** "Der **Austrittspunkt** erlaubt es, von einem definierten inneren Zustand aus den Oberzustand zu verlassen." (Left)
- Boarding:** "Handwritten notes: 'endy action', 'wichtig', 'wichtig connector', 'AVZ!'." (Center)
- Kartenleser:** "Auch Zeit- und Änderungsereignisse können Zustandsübergänge auslösen: - **after** definiert das Verstreichen eines Intervalls; - **when** definiert einen Zustandswechsel." (Left)
- Kartenleser:** "Zustände und zeitlicher Bezugsrahmen werden über den umgebenden Classifier definiert, hier die Werte der Ports, siehe das Montage diagramm „Abfertigung“ links oben." (Left)

The Full Story

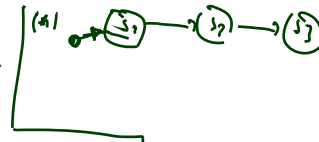
UML distinguishes the following **kinds of states**:

	example		example
simple state <i>(as before)</i>	<p><i>reserved - keyword</i> → entry/act₁^{entry} do/act₁^{do} exit/act₁^{exit} E₁/act_{E1} ... E_n/act_{E_n}</p>	pseudo-state	
final state		initial	
composite state		(shallow) history	
OR		deep history	
AND		fork/join	
		junction, choice	
		entry point	
		exit point	
		terminate	
		submachine state	

Representing All Kinds of States



Representing All Kinds of States

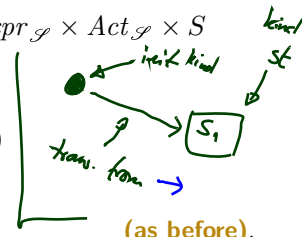


- Until now:

$$(S, s_0, \rightarrow), \quad s_0 \in S, \rightarrow \subseteq S \times (\mathcal{E} \cup \{-\}) \times \text{Expr}_{\mathcal{S}} \times \text{Act}_{\mathcal{S}} \times S$$

- From now on: (hierarchical) state machines

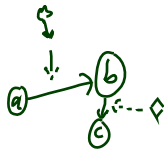
$$(S, \text{kind}, \text{region}, \rightarrow, \psi, \text{annot})$$



where

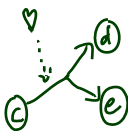
- $S \supseteq \{top\}$ is a finite set of states *(state machine)*
- $kind : S \rightarrow \{st, init, fin, shist, dhist, fork, join, junc, choi, ent, exi, term\}$ is a function which labels states with their **kind**, *(new)*
- $region : S \rightarrow 2^{2^S}$ is a function which characterises the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

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

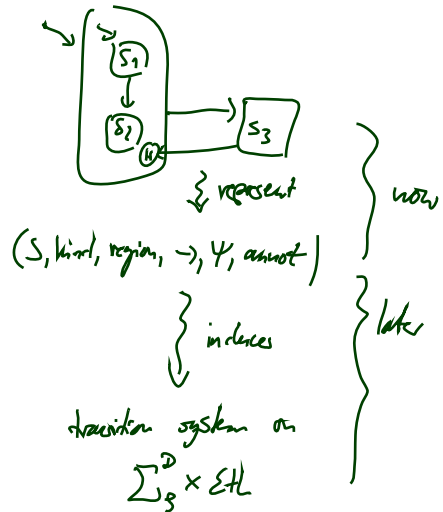
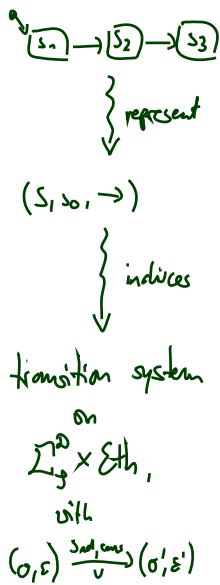


- $(\{a, b, c\}, \{(a, b), (b, c)\})$

- $(\{a, b, c\}, \{\heartsuit, \diamond\}, \{\heartsuit \mapsto (a, b), \diamond \mapsto (b, c)\})$



- $(\{c, d, e\}, \{\heartsuit\}, \{\heartsuit \mapsto (\{c\}, \{d, e\})\})$



From UML to Hierarchical State Machines: By Example

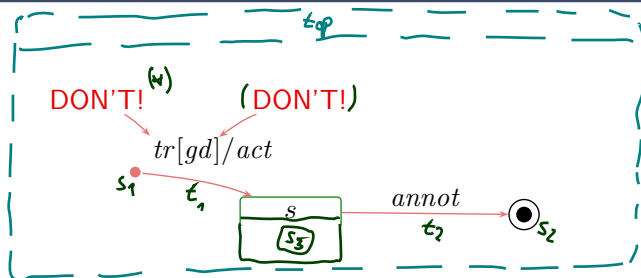
$(S, kind, region, \rightarrow, \psi, annot)$

	example	$\in S$	kind	region
simple state <i>(nothing nested)</i>	<div style="border: 1px solid black; width: 40px; height: 20px; margin: auto; display: flex; align-items: center; justify-content: center;">s</div>	s	st	\emptyset
final state	<div style="display: flex; align-items: center; justify-content: center;"> <div style="border: 1px solid black; width: 15px; height: 15px; border-radius: 50%; margin-right: 5px;"></div> <div style="border: 1px solid black; width: 40px; height: 20px; display: flex; align-items: center; justify-content: center;">s</div> </div>	q	fin	\emptyset
composite state	<div style="border: 1px solid black; width: 60px; height: 40px; margin: auto; display: flex; align-items: center; justify-content: center;"> <div style="border: 1px solid black; width: 20px; height: 15px; margin-right: 5px;">s₁</div> <div style="border: 1px solid black; width: 20px; height: 15px; margin-right: 5px;">s₂</div> <div style="border: 1px solid black; width: 20px; height: 15px;">s₃</div> </div>	s	st	$\{\{s_1, s_2, s_3\}\}$
OR				
AND	<div style="border: 1px solid black; width: 60px; height: 40px; margin: auto; display: flex; align-items: center; justify-content: center;"> <div style="border: 1px solid black; width: 20px; height: 15px; margin-right: 5px;">s₁</div> <div style="border: 1px solid black; width: 20px; height: 15px; margin-right: 5px;">s₂</div> <div style="border: 1px solid black; width: 20px; height: 15px; margin-right: 5px;">s₃</div> </div>	s	st	$\{\{s_1, s_1'\}, \{s_2, s_2'\}, \{s_3, s_3'\}\}$
submachine state	(later)			
pseudo-state	<div style="display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; border-radius: 50%; background-color: red; margin-right: 5px;"></div> <div style="border: 1px solid black; width: 15px; height: 15px; border-radius: 50%; margin-right: 5px;"></div> <div style="border: 1px solid black; width: 15px; height: 15px; border-radius: 50%; margin-right: 5px;"></div> </div>	q	init, shift, ..	\emptyset

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

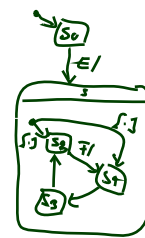
$c \rightarrow (q, q', (s_1, s_2))$

From UML to Hierarchical State Machines: By Example



... translates to $(S, kind, region, \rightarrow, \psi, annot) = (s_3, st),$
 $(\{top, st\}, (s, st), (s_1, init), (s_2, fin)\},$
 $\underbrace{\{top \mapsto \{\{s, s_1, s_2\}\}, s_1 \mapsto \emptyset, s_2 \mapsto \emptyset, s_3 \mapsto \{\{s_3\}\}, s_3 \mapsto \emptyset\}}_{S, kind}$
 $\underbrace{\{t_1, t_2\}, \{t_1 \mapsto (\{s_1, \{s\}\}, t_2 \mapsto (\{s\}, \{s_2\})\}}_{region}$
 $\rightarrow \underbrace{\{t_1 \mapsto (tr, gd, act), t_2 \mapsto annot\}}_{\psi}$
 $\underbrace{\quad}_{annot}$

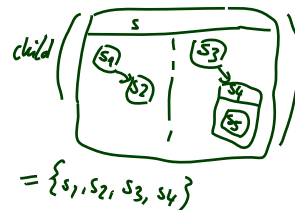
(A) because



Well-Formedness: Regions (follows from diagram)

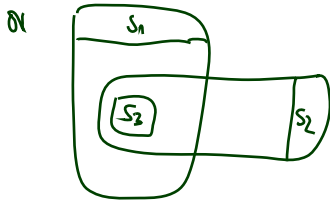
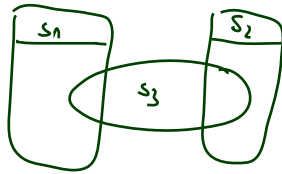
	$\in S$	kind	region $\subseteq 2^S, S_i \subseteq S$	child $\subseteq S$
simple state	s	st	\emptyset	\emptyset
final state	s	fin	\emptyset	\emptyset
composite state	s	st	$\{S_1, \dots, S_n\}, n \geq 1$	$S_1 \cup \dots \cup S_n$
pseudo-state	s	init, ...	\emptyset	\emptyset
implicit top state	top	st	$\{S_1\}$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

- 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.



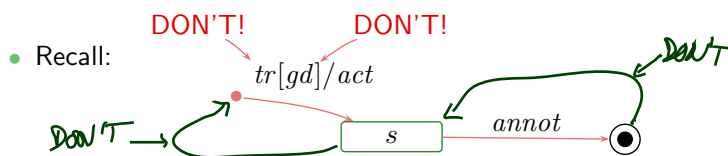
$$region(s) = \{ \{s_1, s_2\}, \{s_3, s_4\} \}$$

Each state (except for top) lies in exactly one region.
 Follows from diagrams because we may not draw:

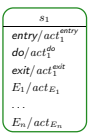







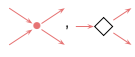






Well-Formedness: Initial State (requirement on diagram)

- 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 $(s_1^i, init) \in S_i$ and at least one transition $t \in \rightarrow$ with s_1^i as source,
 - and such transition's target s_2^i is in S_i , and **(for simplicity!)** $kind(s_2^i) = st$, and $annot(t) = (-, true, act)$.
- No ingoing transitions to initial states.
- No outgoing transitions from final states.



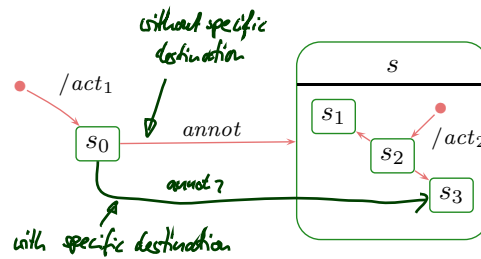
Plan

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

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

Initial Pseudostates and Final States

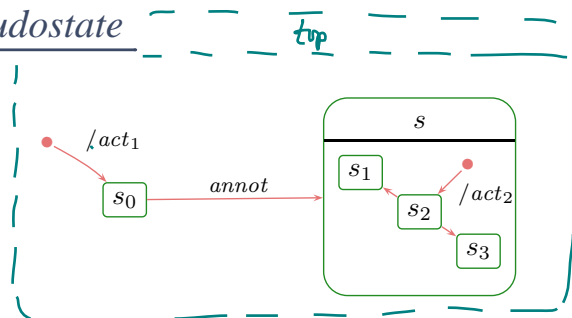
Initial Pseudostate



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. *of source and destination (leaves).*

Initial Pseudostate



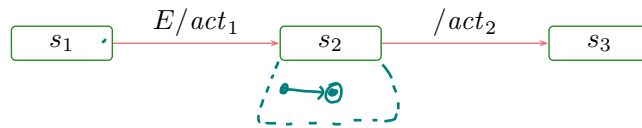
Principle:

- 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

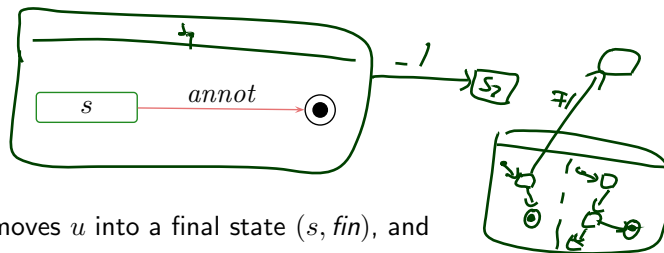


- Transitions without trigger can **conceptionally** be viewed as being sensitive for the “completion event”.
- Dispatching (here: E) can then **alternatively** be **viewed** as
 - (i) fetch event (here: E) from the ether,
 - (ii) take an enabled transition (here: to s_2),
 - (iii) remove event from 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.

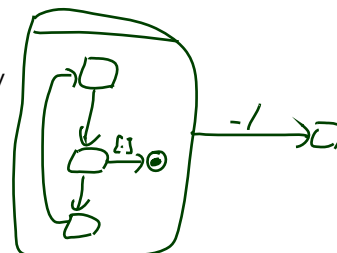
- 15 - 2013-01-08 - Sinitfin -

14/54

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



- 15 - 2013-01-08 - Sinitfin -

15/54

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

References

- [Crane and Dingel, 2007] Crane, M. L. and Dingel, J. (2007). UML vs. classical vs. rhapsody statecharts: not all models are created equal. *Software and Systems Modeling*, 6(4):415–435.
- [Damm et al., 2003] Damm, W., Josko, B., Votintseva, A., and Pnueli, A. (2003). A formal semantics for a UML kernel language 1.2. IST/33522/WP 1.1/D1.1.2-Part1, Version 1.2.
- [Fecher and Schönborn, 2007] Fecher, H. and Schönborn, J. (2007). UML 2.0 state machines: Complete formal semantics via core state machines. In Brim, L., Haverkort, B. R., Leucker, M., and van de Pol, J., editors, *FMICS/PDMC*, volume 4346 of *LNCS*, pages 244–260. Springer.
- [Harel and Gery, 1997] Harel, D. and Gery, E. (1997). Executable object modeling with statecharts. *IEEE Computer*, 30(7):31–42.
- [Harel and Kugler, 2004] Harel, D. and Kugler, H. (2004). The rhapsody semantics of statecharts. In Ehrig, H., Damm, W., Große-Rhode, M., Reif, W., Schnieder, E., and Westkämper, E., editors, *Integration of Software Specification Techniques for Applications in Engineering*, number 3147 in *LNCS*, pages 325–354. Springer-Verlag.
- [OMG, 2007] OMG (2007). Unified modeling language: Superstructure, version 2.1.2. Technical Report formal/07-11-02.
- [Störrle, 2005] Störrle, H. (2005). *UML 2 für Studenten*. Pearson Studium.