

Software Design, Modelling and Analysis in UML

Lecture 12: Core State Machines III

2011-12-11

Prof. Dr. Andreas Podolski, Dr. Bernd Westphal
 Albert-Ludwigs-Universität Freiburg, Germany

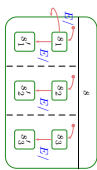
Contents & Goals

- Last Lecture:**
 - The basic causality model
 - Ether
- 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: Signal, Event, Ether, Transformer, Step, RTC.
 - Content:**
 - System Configuration, Transformer
 - Examples for transformer
 - Run- to-completion Step
 - Putting It All Together

And/?

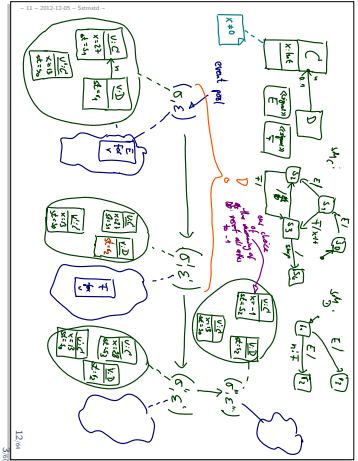


- ...:
- We have to formally define what event occurrence is.
- We have to define where events are stored – what the event pool is.
- We have to explain how transitions are chosen – “matching”.
- We have to explain what the effect of actions is – on state and event pool.
- Write-completion steps (aka super-steps).
- We have to formally define a notion of stability and RTC-step completion.
- And then: Hierarchical state machines.



Roadmap: Chronologically

- What do we (have to) cover? UML State Machine Diagrams Syntax
 - Def.: Signature with signals
 - Def.: Core state machine.
 - Map UML State Machine Diagrams to core state machines.
- Semantics:**
- Def.: Ether (aka, event pool)
 - Def.: System configuration
 - Def.: Event
 - Def.: Transformer
 - Def.: Transition system, computation
 - Def.: step, run-to-completion step
 - Later: Hierarchical state machines



System Configuration, Ether, Transformer

Definition. Let $\mathcal{S} = (\mathcal{S}_0, V, \text{attr}, \mathcal{E})$ be a signature with signals and \mathcal{S} a structure.

We call a **signature** $(Eh, \text{ready}, \oplus, \ominus, \lfloor \cdot \rfloor)$ an ether over \mathcal{S} and \mathcal{S} if and only if it provides:

- a ready operation which yields a set of events that are ready for a given object, i.e. $\text{ready} : Eh \times \mathcal{S}(c) \rightarrow 2^{\mathcal{S}(c)}$
- an operation to insert an event, $\text{insert} : Eh \times \mathcal{S}(c) \rightarrow \mathcal{S}(c)$
- an operation to remove an event, i.e. $\ominus : Eh \times \mathcal{S}(c) \rightarrow Eh$
- an operation to clear the ether for a given object, i.e. $\lfloor \cdot \rfloor : Eh \times \mathcal{S}(c) \rightarrow Eh$.

The standard distinguishes (among others)

- **SignalEvent** [OMG, 2007b, 450] and **Reception** [OMG, 2007b, 447].

On **SignalEvent**, it says *are computed for a signal/trigger* *has a name*

A **signal event** represents the receipt of an asynchronous signal change. A signal event is for example, state a state machine to trigger a transition [OMG, 2007b, 449]

Semantic Variation Points

The means by which (events) are transported to their target depend on the type of requesting action, the target, the properties of the communication medium, and numerous other factors.

In some cases, this is instantaneous and completely reliable while in others it may involve transmission delays of variable duration, loss of requests, reordering, or duplication.

(See also the discussion on page 421) [OMG, 2007b, 450]

Our ether is a general representation of the possible choices.

Often seen minimal requirement: order of sending by one object is preserved.

But: we'll later briefly discuss "discarding" of events.

Definition. Let $\mathcal{S}_0 = (\mathcal{S}_0, \mathcal{E}_0, V_0, \text{attr}_0, \mathcal{E})$ be a signature with signals, \mathcal{S}_0 a structure of \mathcal{S}_0 , $(Eh, \text{ready}, \oplus, \ominus, \lfloor \cdot \rfloor)$ an ether over \mathcal{S}_0 and \mathcal{S}_0 . Furthermore assume there is one core state machine M_C per class $C \in \mathcal{E}$.

A **system configuration** over \mathcal{S}_0 , \mathcal{S}_0 and Eh is a pair (σ, \mathcal{C}) where σ is a mapping from the set of objects to the set of states of the state machines M_C .

where

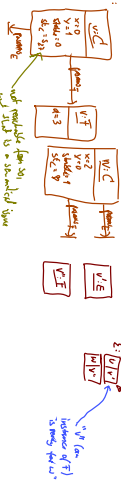
- $\mathcal{S} = (\mathcal{S}_0 \cup \{StMc, \cdot\} \mid C \in \mathcal{E}, \mathcal{E}_0)$
- $V_0 \cup \{StMc, \cdot\} \mid C \in \mathcal{E}, \mathcal{E}_0$
- $\{StC : StC, +, st(0)\} \mid C \in \mathcal{E}$
- $\cup \{\text{params } s : Eh, +, \emptyset\} \mid E \in \mathcal{E}_0\}$
- $\{C \mapsto \text{attr}(C)\}$
- $\cup \{\text{stable}, stC\} \cup \{\text{params } s \mid E \in \mathcal{E}_0\} \mid C \in \mathcal{E}, \mathcal{E}_0$
- $\mathcal{S} = \mathcal{S}_0 \cup \{StC \mapsto St(M_C) \mid C \in \mathcal{E}\}$, and σ is a mapping from the set of objects to the set of states of the state machines M_C .
- $\sigma(u)(C) \cap \mathcal{S}(u)(\mathcal{E}_0) = \emptyset$ for each $u \in \text{dom}(\sigma)$ and $v \in V_{\mathcal{S}_0}$.



$\mathcal{S}_0 = \{v_1, v_2, v_3, \{C, \mathcal{E}\}, \{m_1, m_2, m_3\}, \{s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9, s_{10}, s_{11}, s_{12}, s_{13}, s_{14}, s_{15}, s_{16}, s_{17}, s_{18}, s_{19}, s_{20}, s_{21}, s_{22}, s_{23}, s_{24}, s_{25}, s_{26}, s_{27}, s_{28}, s_{29}, s_{30}, s_{31}, s_{32}, s_{33}, s_{34}, s_{35}, s_{36}, s_{37}, s_{38}, s_{39}, s_{40}, s_{41}, s_{42}, s_{43}, s_{44}, s_{45}, s_{46}, s_{47}, s_{48}, s_{49}, s_{50}, s_{51}, s_{52}, s_{53}, s_{54}, s_{55}, s_{56}, s_{57}, s_{58}, s_{59}, s_{60}, s_{61}, s_{62}, s_{63}, s_{64}, s_{65}, s_{66}, s_{67}, s_{68}, s_{69}, s_{70}, s_{71}, s_{72}, s_{73}, s_{74}, s_{75}, s_{76}, s_{77}, s_{78}, s_{79}, s_{80}, s_{81}, s_{82}, s_{83}, s_{84}, s_{85}, s_{86}, s_{87}, s_{88}, s_{89}, s_{90}, s_{91}, s_{92}, s_{93}, s_{94}, s_{95}, s_{96}, s_{97}, s_{98}, s_{99}, s_{100}\}$

$\mathcal{D}(u_1) = \{s_{10}, s_{20}, s_{30}\}$

$\mathcal{D}(u_2) = \{s_{40}, s_{50}, s_{60}\}$



System Configuration Step-by-Step

- We start with some signature with signals $\mathcal{S}_0 = (\mathcal{S}_0, \mathcal{E}_0, V_0, \text{attr}_0, \mathcal{E})$.
- A **system configuration** is a pair (σ, \mathcal{C}) which comprises a system state σ wrt. \mathcal{S} (not wrt. \mathcal{S}_0).
- Such a system state σ wrt. \mathcal{S} provides, for each object $u \in \text{dom}(\sigma)$,
 - values for the **explicit attributes** in V_0 ,
 - values for a number of **implicit attributes**, namely
 - a **stability flag**, i.e. $\sigma(u)(\text{stable})$ is a boolean value,
 - a **current (state machine) state**, i.e. $\sigma(u)(s)$ denotes one of the states of core state machine M_C .
 - a **temporary association to access event parameters** for each class, i.e. $\sigma(u)(\text{params } s)$ is defined for each $E \in \mathcal{E}$.
- For convenience require: there is **no link** to an event except for **params**.

Stability

Definition. Let (σ, \mathcal{C}) be a system configuration over some $\mathcal{S}_0, \mathcal{S}_0, Eh$. We call an object $u \in \text{dom}(\sigma) \cap \mathcal{S}(u)(\mathcal{E}_0)$ **stable** in σ if and only if $\sigma(u)(\text{stable}) = \text{true}$.

Definition. Let \mathcal{S}_0 be a structure of the signature with signals

$\mathcal{S}_0 = (\mathcal{O}_0, \mathcal{S}_0, \mathcal{V}_0, \text{arr}, \sigma)$ and let $E \in \mathcal{E}_0$ be a signal.

Let $\text{arr}(E) = \{v_1, \dots, v_n\}$. We call

$$e = (E, \{v_i \mapsto d_i, \dots, v_n \mapsto d_n\}),$$

or shorter (if mapping is clear from context)

$$(E, (d_1, \dots, d_n)) \text{ or } (E, \vec{d}),$$

an event (or an instance) of signal E (if type-consistent).

We use $\text{Inst}(\mathcal{E}_0, \mathcal{S}_0)$ to denote the set of all events of all signals in \mathcal{S}_0 wrt. \mathcal{S}_0 .

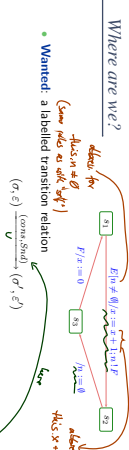
- As we always try to maximize confusion...
- By our existing naming convention, $w \in \mathcal{W}(E)$ is also called instance of the (signal) class E in system configuration (σ, \mathcal{E}) if $w \in \text{dom}(\sigma)$.
- The corresponding event is then $(E, \sigma(w))$.

The idea is the following:

- Signals are types (classes).
- Instances of signals (in the standard sense) are kept in the system state component σ of system configurations (σ, \mathcal{E}) .
- Identities of signal instances are kept in the ether.
- Each signal instance is in particular an event — somehow “a recording that this signal occurred” (without caring for its identity)
- The main difference between signal instance and event: Events don't have an identity.

- Why is this useful? In particular for reflective descriptions of behaviour, we are typically not interested in the identity of a signal instance, but only whether it is an “E” or “F”, and which parameters it carries.

Where are we?



- Wanted: a labelled transition relation on system configuration, labelled with the consumed and sent events, (σ, \mathcal{E}) being the result (or effect) of one object v , taking a transition of its state machine from the current state $\sigma(v)$ to $\sigma'(v)$.
- Have: system configuration, (σ, \mathcal{E}) comprising current state machine state and stability flag for each object, and the ether.

- Plan:
 - Introduce transformer as the semantics of action annotations. Intuitively, (σ', \mathcal{E}') is the effect of applying the transformer of the taken transition.
 - Explain how to choose transitions depending on σ and when to stop taking transitions — the run-to-completion “algorithm”.

Transformer

because of readability

Definition. Let $\mathcal{S}_0^{\mathcal{E}}$ the set of system configurations over some $\mathcal{S}_0, \mathcal{E}_0, E, H$. We call a **rigidification** the typed recording τ of some object v in $\mathcal{S}_0^{\mathcal{E}}$ a (system configuration) **transformer**: system configuration τ .

- In the following, we assume that each application of a transformer τ to some system configuration (σ, \mathcal{E}) for object v , is associated with a set of observations
- $$\text{Obs}(v, \tau)(\sigma, \mathcal{E}) \in 2^{\mathcal{O}(\mathcal{E}) \times \text{Inst}(\mathcal{E}_0, \mathcal{S}_0) \cup \{ \perp \} \times 2^{\mathcal{O}(\mathcal{E})}}$$
- Handwritten notes: τ gives concrete object v and $\text{Obs}(v, \tau)$ receives a set of observations.
- An observation $(v, \text{sent}, u, (E, \vec{d}), u, a) \in \text{Obs}(v, \tau)(\sigma, \mathcal{E})$ represents the information that, as a “side effect” of v , executing τ , an event (E, \vec{d}) has been sent from v to u .
 - Special cases: creation / destruction.

Why Transformers?

- Recall the (simplified) syntax of transition annotations:


```
annot ::= [ (event) [! (guard) ?] [! (F' (action)) ] ]
```
- Clear: (event) is from \mathcal{E} of the corresponding signature.
- But: What are (guard) and (action)?
- UML can be viewed as being parameterized in expression language (providing (guard)) and action language (providing (action)).
- Examples:
 - Expression Language:
 - OCL
 - Java, C++, ... expressions
 - ...
 - Action Language:
 - UML Action Semantics, “Executable UML”
 - Java, C++, ... statements (plus some event send action)
 - ...

↳ the following are candidates

$\text{Act}_\sigma ::= \{ \text{skip} \}$
 $v \{ \text{update}(\text{expr}_i, v, \text{expr}_i) \mid \text{expr}_i, \text{expr}_i \in \text{OCLExp}, v \in V \}$
 $v \{ \text{send}(\text{expr}_i, E, \text{expr}_i) \mid \text{expr}_i, \text{expr}_i \in \text{OCLExp}, E \in \mathcal{E} \}$
 $v \{ \text{create}(C, \text{expr}, v) \mid \text{expr} \in \text{OCLExp}, C \in \mathcal{C}, v \in V \}$
 $v \{ \text{destroy}(\text{expr}) \mid \text{expr} \in \text{OCLExp} \}$
 Empty: OCL expressions over \mathcal{V}

Transformers as Abstract Actions!

example OCL:

In the following, we assume that we're given

- an expression language $Expr$ for guards, and
- an action language Act for actions,

and that we're given

- a semantics for boolean expressions in form of a partial function

$$I \llbracket \cdot \rrbracket : Expr \rightarrow (\mathbb{Z}^S \times (\mathbb{Z}^S \times BH) \times (\mathbb{Z}^S \times BH)) \rightarrow \mathbb{B}$$

which evaluates expressions in a given system configuration,

Assuming I to be partial is a way to treat "undefined" during runtime. If I is not defined (for instance because of dangling-reference navigation or division-by-zero), we want to go to a designated "error" system configuration.

a transformer for each action: for each $act \in Act$, we assume to have

$$t_{act} \subseteq \mathcal{P}(\mathcal{C}) \times (\mathbb{Z}^S \times BH) \times (\mathbb{Z}^S \times BH)$$

Expression/Action Language Examples

We can make the assumptions from the previous slide because instances exist:

- for OCL, we have the OCL semantics from Lecture 03. Simply remove the preimages which map to \perp .
 - for Java, the operational semantics of the SWT lecture uniquely defines transformers for sequences of Java statements.
- We distinguish the following kinds of transformers:
- skip**: do nothing — recall: this is the default action
 - send**: modifies ε — interesting ε , because state machines are built around sending/consuming events
 - create/delete**: modify domain of σ — not specific to state machines, but let's discuss them here as we're at it
 - update**: modify own or other objects' local state — boring

Transformer Examples: Presentation

abstract syntax	concrete syntax
op	
infinite semantics	...
well-typedness	...
semantics	$((\alpha, \varepsilon), (\alpha', \varepsilon')) \in \text{Ker} \llbracket \cdot \rrbracket$ iff ... or $\text{Ker} \llbracket \cdot \rrbracket \llbracket \alpha, \beta \rrbracket = \{(\alpha', \varepsilon')\}$ where ...
observables	$Obs_{\text{skip}} \llbracket \alpha, \beta \rrbracket = \{\dots\}$; not a relation, depends on choice
(error) conditions	Not defined if ...

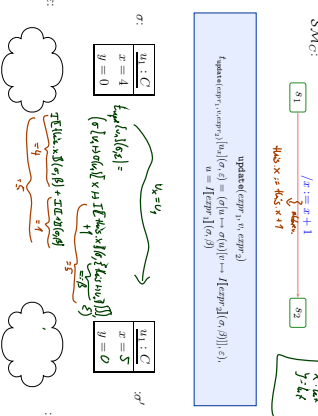
Transformer: Skip

abstract syntax	skip	concrete syntax	skip
infinite semantics	do nothing		
well-typedness	./.		"/. v_1, v_2 remain skip on (α, ε) , $(\alpha, \varepsilon) \mapsto (\alpha, \varepsilon)$!
semantics	$\llbracket \text{skip} \rrbracket \llbracket \alpha, \varepsilon \rrbracket = \{(\alpha, \varepsilon)\}$		
observables	$Obs_{\text{skip}} \llbracket \alpha, \beta \rrbracket \llbracket \alpha, \varepsilon \rrbracket = \emptyset$		
(error) conditions			

Transformer: Update

abstract syntax	$\text{update}(expr_1, v, expr_2)$	concrete syntax	$v \in \text{Expr}$ $expr_1, v, expr_2$
infinite semantics	Update attribute v in the object denoted by $expr_1$ to the value denoted by $expr_2$.		
well-typedness	$expr_1 : TC$ and $v : \tau \in \text{Var}(C)$; $expr_2 : \tau$ $expr_1, expr_2$ obey visibility and nonnullability		
semantics	$\llbracket \text{update}(expr_1, v, expr_2) \rrbracket \llbracket \alpha, \varepsilon \rrbracket = \{(\alpha', \varepsilon')\}$ where $\alpha' = \sigma[\alpha]_{v \mapsto \llbracket expr_2 \rrbracket \llbracket \alpha, \beta \rrbracket}$ with $n = \llbracket expr_1 \rrbracket \llbracket \alpha, \beta \rrbracket$		"denote how v change" (in our context) "denote how α change" (in our context)
observables	$Obs_{\text{update}(expr_1, v, expr_2)} \llbracket \alpha, \beta \rrbracket \llbracket \alpha, \varepsilon \rrbracket$ not defined.		"denote how α change" (in our context)
(error) conditions	Not defined if $\llbracket expr_1 \rrbracket \llbracket \alpha, \beta \rrbracket$ or $\llbracket expr_2 \rrbracket \llbracket \alpha, \beta \rrbracket$ not defined.		

Update Transformer Example



References

59/60

References

[Harel and Gery, 1997] Harel, D. and Gery, E. (1997). Executable object modeling with statecharts. *IEEE Computer*, 30(7):31–42.

[OMG, 2007a] OMG (2007a). Unified modeling language: Infrastructure, version 2.1.2. Technical Report formal/07-11-04.

[OMG, 2007b] OMG (2007b). Unified modeling language: Superstructure, version 2.1.2. Technical Report formal/07-11-02.

60/60