Software Design, Modelling and Analysis in UML

Lecture 11: Core State Machines I

2014-12-04

Prof. Dr. Andreas Podelski, Dr. Bernd Westphal

Albert-Ludwigs-Universität Freiburg, Germany
Contents & Goals

Last Lecture:
- Associations (up to some rest)

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:**
  - Associations cont’d, back to main track
  - Core State Machines
  - UML State Machine syntax
Associations: The Rest
Recapitulation: Consider the following association:

\[ \langle r : \langle \text{role}_1 : C_1, \mu_1, P_1, \xi_1, \nu_1, o_1 \rangle, \ldots, \langle \text{role}_n : C_n, \mu_n, P_n, \xi_n, \nu_n, o_n \rangle \rangle \]

- **Association name** \( r \) and **role names/types** \( \text{role}_i / C_i \) induce extended system states \( \lambda \).
- **Multiplicity** \( \mu \) is considered in OCL syntax.
- **Visibility** \( \xi \)/**Navigability** \( \nu \): well-typedness.

Now the rest:

- **Multiplicity** \( \mu \): we propose to view them as constraints.
- **Properties** \( P_i \): even more typing.
- **Ownership** \( o \): getting closer to pointers/references.
- **Diamonds**: exercise.
Visibility

Visibility of role-names is treated similar to attributes, by **typing rules**.

**Question**: given

\[
\text{context } C \text{ inv: self}.role.x > 0
\]

is the following OCL expression well-typed or not (wrt. visibility):

\[
\text{role}(w) : \tau_C \rightarrow \tau_D
\]

\[
\text{role}(\text{expr}_1(w)) : \tau_C \rightarrow \tau_D
\]

\[
\langle r : \ldots \langle \text{role} : D, \mu, -, \xi, -, - \rangle, \ldots \langle \text{role'} : C, -, -, -, -, - \rangle, \ldots \rangle \in V
\]
Navigability is similar to visibility: expressions over non-navigable association ends ($\nu = \times$) are **basically** type-correct, but **forbidden**.

**Question**: given

is the following OCL expression well-typed or not (wrt. navigability)?

$$\text{context } D \text{ inv : self.role.x} > 0$$

The standard says: navigation is...

- ’−’: ...possible
- ’>’: ...efficient
- ’×’: ...not possible

**So**: In general, UML associations are different from pointers/references!

**But**: Pointers/references can faithfully be modelled by UML associations.
Recapitulation: Consider the following association:

\[ \langle r : \langle \text{role}_1 : C_1, \mu_1, P_1, \xi_1, \nu_1, o_1 \rangle, \ldots, \langle \text{role}_n : C_n, \mu_n, P_n, \xi_n, \nu_n, o_n \rangle \rangle \]

- **Association name** \( r \) and **role names/types** \( \text{role}_i/C_i \) induce extended system states \( \lambda \).
- **Multiplicity** \( \mu \) is considered in OCL syntax.
- **Visibility** \( \xi \)/**Navigability** \( \nu \): well-typedness.

Now the rest:

- **Multiplicity** \( \mu \): we propose to view them as constraints.
- **Properties** \( P_i \): even more typing.
- **Ownership** \( o \): getting closer to pointers/references.
- **Diamonds**: exercise.
**Recall**: The multiplicity of an association end is a term of the form:

\[ \mu ::= \ast | N | N..M | N..\ast | \mu, \mu \quad (N, M \in \mathbb{N}) \]

**Proposal**: View multiplicities (except $0..1$, $1$) as additional invariants/constraints.

**Recall**: we can normalize each multiplicity $\mu$ to the form

\[ \mu = \underbrace{N_1..N_2, \ldots, N_{2k-1}..N_{2k}} \]

where $N_i \leq N_{i+1}$ for $1 \leq i \leq 2k$, $N_1, \ldots, N_{2k-1} \in \mathbb{N}$, $N_{2k} \in \mathbb{N} \cup \{\ast\}$. 
Multiplicities as Constraints

\[ \mu = N_1..N_2, \ldots, N_{2k-1}..N_{2k} \]

where \( N_i \leq N_{i+1} \) for \( 1 \leq i \leq 2k \), \( N_1, \ldots, N_{2k-1} \in \mathbb{N} \), \( N_{2k} \in \mathbb{N} \cup \{ \ast \} \).

Define \( \mu^C_{OCL}(\text{role}) := \) context \( C \) inv :

\[ (N_1 \leq \text{role} \rightarrow \text{size()} \leq N_2) \text{ or } \ldots \text{ or } (N_{2k-1} \leq \text{role} \rightarrow \text{size()} \leq N_{2k}) \]

omit if \( N_{2k} = \ast \)

for each \( \mu \neq 0..1, \mu \neq 1 \),

\[ \langle r : \ldots, \langle \text{role} : D, \mu, \ldots, \ldots \rangle, \ldots, \langle \text{role}' : C, \ldots, \ldots \rangle, \ldots \rangle \in V \text{ or } \]
\[ \langle r : \ldots, \langle \text{role}' : C, \ldots, \ldots \rangle, \ldots, \langle \text{role} : D, \mu, \ldots, \ldots \rangle, \ldots \rangle \in V, \text{ role } \neq \text{ role}' \]

And define

\[ \mu^C_{OCL}(\text{role}) := \text{context } C \text{ inv : not(oclIsUndefined(\text{role}))} \]

for each \( \mu = 1 \).

Note: in \( n \)-ary associations with \( n > 2 \), there is redundancy.
**Multiplicities as Constraints Example**

\[
\mu^C_{OCL}(role) = \text{context } C \text{ inv :}
\]

\[
(N_1 \leq role \rightarrow \text{size()} \leq N_2) \text{ or } \ldots \text{ or } (N_{2k-1} \leq role \rightarrow \text{size()} \leq N_{2k})
\]

\[\text{note: 0..1 is equivalent to 0..0, 1..1}\]

\[\text{CD :}\]

\[
\begin{align*}
\text{role}_1 & \rightarrow C \\
0..1 & \quad v : \text{Int} \\
4, 17 & \quad \text{role}_2 \\
\text{role}_3 & \rightarrow 3..* \\
\end{align*}
\]

\[\text{Inv(CD)} = \]

- \{ context \text{ } C \text{ inv : } 4 \leq \text{role}_2 \rightarrow \text{size()} \leq 4 \text{ or } 17 \leq \text{role}_2 \rightarrow \text{size()} \leq 17, (\ast) \}

- \{ context \text{ } C \text{ inv : } \text{role}_2 \rightarrow \text{size()} = 4 \text{ or } \text{role}_2 \rightarrow \text{size()} = 17 \}

- \{ context \text{ } C \text{ inv : } 3 \leq \text{role}_3 \rightarrow \text{size()} \}

- \{ context \text{ } C \text{ inv : } \} \]
Why Multiplicities as Constraints?

More precise, can’t we just use types? (cf. Slide 26)

- $\mu = 0..1, \mu = 1$: many programming languages have direct correspondences (the first corresponds to type pointer, the second to type reference) — therefore treated specially.

- $\mu = \ast$: could be represented by a set data-structure type without fixed bounds — no problem with our approach, we have $\mu_{\text{OCL}} = \text{true}$ anyway.

- $\mu = 0..3$: use array of size 3 — if model behaviour (or the implementation) adds 5th identity, we’ll get a runtime error, and thereby see that the constraint is violated. **Principally acceptable**, but: checks for array bounds everywhere...?

- $\mu = 5..7$: could be represented by an array of size 7 — but: few programming languages/data structure libraries allow lower bounds for arrays (other than 0). If we have 5 identities and the model behaviour removes one, this should be a violation of the constraints imposed by the model. The implementation which does this removal is **wrong**. How do we see this...?
Well, if the target platform is known and fixed, and the target platform has, for instance,

- reference types,
- range-checked arrays with positions $0, \ldots, N$,
- set types,

then we could simply restrict the syntax of multiplicities to

$$\mu ::= 1 \mid 0..N \mid *$$

and don’t think about constraints (but use the obvious 1-to-1 mapping to types)...

In general, unfortunately, we don’t know.
Recall/Later:

\[ \mathcal{C} \mathcal{D} = \{ CD_1, \ldots, CD_n \} \]

signature \( I(\mathcal{C} \mathcal{D}) \)

invariants \( Inv(\mathcal{C} \mathcal{D}) \)

basic (classes and attributes)

distinguish

extended (visibility)

From now on: \( Inv(\mathcal{C} \mathcal{D}) = \{ \text{constraints occurring in notes} \} \cup \{ \mu_{OCL}(role) \mid \langle r : \ldots, \langle \text{role} : D, \mu, _-, _, _- \rangle, \ldots, \langle \text{role}' : C, _, _, _, _- \rangle, \ldots \rangle \in V \text{ or } \langle r : \ldots, \langle \text{role}' : C, _, _, _, _- \rangle, \ldots, \langle \text{role} : D, \mu, _-, _- \rangle, \ldots \rangle \in V, \text{ role } \neq \text{ role'}, \mu \notin \{0..1\} \} \).
We don’t want to cover association **properties** in detail, only some observations (assume binary associations):

<table>
<thead>
<tr>
<th>Property</th>
<th>Intuition</th>
<th>Semantical Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>unique</strong></td>
<td>one object has <strong>at most one</strong> ( r )-link to a single other object</td>
<td><strong>current setting</strong></td>
</tr>
<tr>
<td><strong>bag</strong></td>
<td>one object may have <strong>multiple</strong> ( r )-links to a single other object</td>
<td>have ( \lambda(r) ) yield multi-sets</td>
</tr>
<tr>
<td><strong>ordered, sequence</strong></td>
<td>an ( r )-link is a <strong>sequence</strong> of object identities (possibly including duplicates)</td>
<td>have ( \lambda(r) ) yield sequences</td>
</tr>
</tbody>
</table>

### Diagram

![Diagрам](image-url)
We don’t want to cover association properties in detail, only some observations (assume binary associations):

<table>
<thead>
<tr>
<th>Property</th>
<th>Intuition</th>
<th>Semantical Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>unique</td>
<td>one object has <strong>at most one</strong> ( r )-link to a single other object</td>
<td><strong>current setting</strong></td>
</tr>
<tr>
<td>bag</td>
<td>one object may have <strong>multiple</strong> ( r )-links to a single other object</td>
<td>have ( \lambda(r) ) yield multi-sets</td>
</tr>
<tr>
<td>ordered, sequence</td>
<td>an ( r )-link is a <strong>sequence</strong> of object identities (possibly including duplicates)</td>
<td>have ( \lambda(r) ) yield sequences</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Property</th>
<th>OCL Typing of expression ( role(expr) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>unique</td>
<td>( \tau_D \rightarrow Set(\tau_C) )</td>
</tr>
<tr>
<td>bag</td>
<td>( \tau_D \rightarrow Bag(\tau_C) )</td>
</tr>
<tr>
<td>ordered, sequence</td>
<td>( \tau_D \rightarrow Seq(\tau_C) )</td>
</tr>
</tbody>
</table>

For **subsets**, **redefines**, **union**, etc. see [OMG, 2007a, 127].
Ownership

Intuitively it says:

Association $r$ is **not a “thing on its own”** (i.e. provided by $\lambda$), but association end ‘role’ is **owned** by $C$ (!). (That is, it’s stored inside $C$ object and provided by $\sigma$).

**So:** if multiplicity of role is 0..1 or 1, then the picture above is very close to concepts of pointers/references.

Actually, ownership is seldom seen in UML diagrams. Again: if target platform is clear, one may well live without (cf. [OMG, 2007b, 42] for more details).

**Not clear to me:**
Back to the Main Track
**Recall:** on some earlier slides we said, the extension of the signature is **only** to study associations in “full beauty”.
For the remainder of the course, we should look for something simpler...

**Proposal:**

- **from now on**, we only use associations of the form

  (i) \[ C \times^{0..1} \text{role} \rightarrow D \]

  (ii) \[ C \times^{0..*} \text{role} \rightarrow D \]

  (And we may omit the non-navigability and ownership symbols.)

- Form (i) introduces \( \text{role} : C_{0,1} \), and form (ii) introduces \( \text{role} : C_* \) in \( V \).

- In both cases, \( \text{role} \in atr(C) \).

- We drop \( \lambda \) and go back to our nice \( \sigma \) with \( \sigma(u)(\text{role}) \subseteq \mathcal{B}(D) \).
OCL Constraints in (Class) Diagrams
Two options:

(i) Notes.

(ii) Particular dedicated places.

(i) Notes:

A UML note is a picture of the form

\[
\text{[text]}
\]

\text{text} can principally be everything, in particular comments and constraints.

Sometimes, content is explicitly classified for clarity:

\[
\text{OCL: expr}
\]
stands for

\textit{expr}
(ii) **Particular dedicated places** in class diagrams:  

For simplicity, we view the above as an abbreviation for

\[
C
\]

\[
\xi \ v : \tau \ \{p_1, \ldots, p_n\} \ \{\text{expr}\}
\]

\[
\xi \ f(v_1 : \tau, \ldots, v_n : \tau_n) : \tau \ \{p_1, \ldots, p_n\} \ \{\text{pre} : \text{expr}_1
\]

\[
\text{post} : \text{expr}_2\}
\]
Let $\mathcal{CD}$ be a class diagram.

As we (now) are able to recognise OCL constraints when we see them, we can define

$$\text{Inv}(\mathcal{CD})$$

as the set $\{\varphi_1, \ldots, \varphi_n\}$ of OCL constraints \textbf{occurring} in notes in $\mathcal{CD}$ — after \textbf{unfolding} all abbreviations (cf. next slides).

As usual: $\text{Inv}(\mathcal{C} \mathcal{D}) := \bigcup_{\mathcal{CD} \in \mathcal{C} \mathcal{D}} \text{Inv}(\mathcal{CD})$.

\textbf{Principally clear}: $\text{Inv}(\cdot)$ for any kind of diagram.
If $\mathcal{CD}$ consists of only $CD$ with the single class $C$, then

- $\text{Inv}(\mathcal{CD}) = \text{Inv}(CD) = \ldots$
Definition. Let $\mathcal{CD}$ be a set of class diagrams. We say, the semantics of $\mathcal{CD}$ is the signature it induces and the set of OCL constraints occurring in $\mathcal{CD}$, denoted

$$\llbracket \mathcal{CD} \rrbracket := \langle \mathcal{I}(\mathcal{CD}), \text{Inv}(\mathcal{CD}) \rangle.$$ 

Given a structure $\mathcal{D}$ of $\mathcal{I}$ (and thus of $\mathcal{CD}$), the class diagrams describe the system states $\Sigma^\mathcal{D}$, of which some may satisfy $\text{Inv}(\mathcal{CD})$.

In pictures:

- $\mathcal{CD} = \{\mathcal{CD}_1, \ldots, \mathcal{CD}_n\}$
- signature $\mathcal{I}(\mathcal{CD})$
- invariants $\text{Inv}(\mathcal{CD})$
- distinguish
- extended (visibility)
- basic (classes and attributes)
Recall: a UML model is an image or pre-image of a software system.

A set of class diagrams $\mathcal{CD}$ with invariants $\text{Inv}(\mathcal{CD})$ describes the structure of system states.
Together with the invariants it can be used to state:

- **Pre-image**: Dear programmer, please provide an implementation which uses only system states that satisfy $\text{Inv}(\mathcal{CD})$.
- **Post-image**: Dear user/maintainer, in the existing system, only system states which satisfy $\text{Inv}(\mathcal{CD})$ are used.

(The exact meaning of “use” will become clear when we study behaviour — intuitively: the system states that are reachable from the initial system state(s) by calling methods or firing transitions in state-machines.)

**Example**: highly abstract model of traffic lights controller.

![Traffic Lights Controller Diagram]

\[
\text{TLCtrl} \\
\text{red : Bool} \\
\text{green : Bool} \\
\text{not(red and green)}
\]
Find the 10 differences:

\[
\begin{array}{|c|}
\hline
C \\
\hline
x : Int \{x = 3 \lor x > 17\} \\
\hline
\end{array}
\]

\[
\begin{array}{|c|}
\hline
C \\
\hline
x : T \\
\hline
\end{array}
\]

\[
\mathcal{D}(T) = \{3\} \cup \{n \in \mathbb{N} | n > 17\}
\]

- \( x = 4 \) is well-typed in the left context, a system state satisfying \( x = 4 \) violates the constraints of the diagram.
- \( x = 4 \) is not even well-typed in the right context, there cannot be a system state with \( \sigma(u)(x) = 4 \) because \( \sigma(u)(x) \) is supposed to be in \( \mathcal{D}(T) \) (by definition of system state).

**Rule-of-thumb:**

- If something “feels like” a type (one criterion: has a natural correspondence in the application domain), then make it a type.
- If something is a requirement or restriction of an otherwise useful type, then make it a constraint.
UML State Machines
Brief History:

- Rooted in Moore/Mealy machines, Transition Systems
- [Harel, 1987]: Statecharts as a concise notation, introduces in particular hierarchical states.
- Manifest in tool Statemate [Harel et al., 1990] (simulation, code-generation); nowadays also in Matlab/Simulink, etc.
- From UML 1.x on: State Machines (in State Chart Diagrams) (not the official name, but understood: UML-Statecharts)
- Late 1990’s: tool Rhapsody with code-generation for state machines.

Note: there is a common core, but each dialect interprets some constructs subtly different [Crane and Dingel, 2007]. (Would be too easy otherwise...)
Roadmap: Chronologically

(i) What do we (have to) cover? UML State Machine Diagrams **Syntax**.

(ii) Def.: Signature with **signals**.

(iii) Def.: **Core state machine**.

(iv) Map UML State Machine Diagrams to core state machines.

**Semantics:**
The Basic Causality Model

(v) Def.: **Ether** (aka. event pool)

(vi) Def.: **System configuration**.

(vii) Def.: **Event**.

(viii) Def.: **Transformer**.

(ix) Def.: **Transition system**, computation.

(x) Transition relation induced by core state machine.

(xi) Def.: **step, run-to-completion step**.

(xii) Later: Hierarchical state machines.
UML State Machines: Syntax
UML State-Machines: What do we have to cover?

[Störrle, 2005]

When the Endzustand of a Protokollzustandsautomaten is reached, the region is ended, in which the Endzustand lies.

Protokollzustandsautomaten describe the behavior of software systems, in particular with regard to failures or technical devices.

A complex state consists of a region.

The transition event after(10s) starts a load of the "Boarding" control.

The memory state ensures that after an interruption the same state is taken over as before.

The exit point allows to leave a defined state in the state machine.

The start state marks the reference point of "Boarding" or "Boardkarte einlesen".

The entry point defines that a complex state is entered at a different location than defined by the start state.

A start state defines the state machine and the behavior of the system.

Regulatory completion defines a complex state at a different position.

Ereignisse können innerhalb eines Zustands Aktionen auslösen.

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


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.

Ein Zeitereignis after(10s) löst einen Abbruch von "Boardkarte einlesen" aus.

Reguläre Beendigung löst ein completion-Ereignis 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.


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

Ein verfeinerter Zustand verweist auf einen Zustandsautomaten (angedeutet von dem Symbol unten links), der das Verhalten des Zustandes definiert.

 Ereignisse können innerhalb eines Zustands Aktionen auslösen.
Proven approach:

Start out simple, consider the essence, namely

- basic/leaf states
- transitions,

then extend to cover the complicated rest.
Definition. A tuple

\[ \mathcal{I} = (\mathcal{I}, \mathcal{C}, V, atr, \mathcal{E}), \quad \mathcal{E} \subseteq \mathcal{C} \text{ a set of signals}, \]

is called signature (with signals) if and only if

\[ (\mathcal{I}, \mathcal{C}, V, atr) \]

is a signature (as before).

Note: Thus conceptually, a signal is a class and can have attributes of plain type and associations.
\[ Y = (\mathcal{T}, \{C, E, F, G\}, \{x : \text{int}, c : C_0\}, \{C \land \sigma, E \lor \sigma, \neg F \Rightarrow \{x : \text{int}, G \Rightarrow \exists c : C_0\}\}, \{E, F, G\}) \]
Definition.
A core state machine over signature $\mathcal{S} = (\mathcal{T}, \mathcal{C}, \mathcal{V}, \text{attr}, \mathcal{E})$ is a tuple

$$M = (S, s_0, \rightarrow)$$

where

- $S$ is a non-empty, finite set of (basic) states,
- $s_0 \in S$ is an initial state,
- and

$$\rightarrow \subseteq S \times (\mathcal{E} \cup \{-\}) \times \mathcal{Expr}_{\mathcal{S}} \times \mathcal{Act}_{\mathcal{S}} \times S$$

is a labelled transition relation.

We assume a set $\mathcal{Expr}_{\mathcal{S}}$ of boolean expressions (may be OCL, may be something else) and a set $\mathcal{Act}_{\mathcal{S}}$ of actions over $\mathcal{S}$.
UML state machine diagram $SM$:

\[
\text{annot ::= } [ \langle \text{event} \rangle \langle . \rangle \langle \text{event} \rangle^* \left[ \langle [ \langle \text{guard} \rangle \langle . \rangle \rangle \right] \left[ \langle / \rangle \langle \text{action} \rangle \right] ]
\]

with

- $\text{event} \in \mathcal{E}$,
- $\text{guard} \in \text{Expr}_\mathcal{S}$ (default: $true$, assumed to be in $\text{Expr}_\mathcal{S}$)
- $\text{action} \in \text{Act}_\mathcal{S}$ (default: $\text{skip}$, assumed to be in $\text{Act}_\mathcal{S}$)

maps to

\[
M(SM) = (\{s_1, s_2\}, s_0, (s_1, \text{event}, \text{guard}, \text{action}, s_2))
\]
References


