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
Recapitulation: Consider the following association:

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

- **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

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

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

Basically the same rule as before (similar for other multiplicities):

\[
\begin{align*}
\text{role}(w) & : \tau_C \rightarrow \tau_D & \mu = 0..1 \text{ or } \mu = 1, \\
\text{role}(\text{expr}_1(w)) : \tau_C \rightarrow \tau_D & \mu = 0..1 \text{ or } \mu = 1, \text{ expr}_1(w) : \tau_C, \\
& w : \tau_{C1}, \text{ and } C_1 = C \text{ or } \xi = + \\
\langle r : \ldots \langle \text{role} : D, \mu, \xi, \ldots \rangle, \ldots \langle \text{role'} : C, \ldots \rangle, \ldots \rangle & \in V
\end{align*}
\]

Navigability

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
- ‘\times’: ...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.

**Multiplicities as Constraints**

**Recall:** The multiplicity of an association end is a term of the form:

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

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

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

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

where \( N_i \leq N_{i+1} \) for \( 1 \leq i \leq 2k \), \( N_1, \ldots, N_{2k-1} \in \mathbb{N}, \quad N_{2k} \in \mathbb{N} \cup \{ * \} \).
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 \( \mu_{\text{COCL}}(\text{role}) := \text{context } C \text{ inv} : \)

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

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

for each \( \mu \neq 0.1, \mu \neq 1 \),

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

And define

\[ \mu_{\text{COCL}}(\text{role}) := \text{context } C \text{ inv} : \text{not(oclIsUndefined(\text{role}))} \]

for each \( \mu = 1 \).

Note: in \( n \)-ary associations with \( n > 2 \), there is redundancy.

Multiplicities as Constraints Example

\[ \mu_{\text{COCL}}(\text{role}) = \text{context } C \text{ inv} : \]

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

Note: 0..1 is equivalent to 0.0..1..1

CD:

\[ \begin{array}{c|c|c}
\text{role}_1 & C & (x) \\
0..1 & v : \text{Int} & 4, 17 \\
\hline
\text{role}_2 & 3..\ast & \\
\end{array} \]

Inv(CD) =

- \{ context \ C \text{ inv} : \text{role}_2 \rightarrow \text{size}() = 4 \text{ or } \text{role}_2 \rightarrow \text{size}() = 17 \} \cup \{ context \ C \text{ inv} : \text{role}_3 \rightarrow \text{size}() = 4 \text{ or } \text{role}_3 \rightarrow \text{size}() = 17 \} \cup \{ context \ C \text{ inv} : 3 \leq \text{role}_4 \rightarrow \text{size}() \} \} \]
Why Multiplicities as Constraints?

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

- $\mu = 0..1$, $\mu = 1$:
  many programming language have direct correspondences (the first corresponds to type pointer, the second to type reference) — therefore treated specially.
- $\mu = *$:
  could be represented by a set data-structure type without fixed bounds — no problem with our approach, we have $\mu_{OCL} = true$ 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...

Multiplicities Never as Types...?

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:

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

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

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

\[ \text{basic (classes and attributes)} \]
\[ \text{distinguish extended (visibility)} \]

From now on: \( \text{Inv}(\mathcal{D}) = \{\text{constraints occurring in notes}\} \cup \{\mu_{\text{OCL}}(\text{role})\} \)

\( \langle r : \ldots, \langle \text{role} : D, \mu, \ldots\rangle, \ldots, \langle \text{role} : C, \ldots\rangle, \ldots \rangle \in V \) or

\( \langle r : \ldots, \langle \text{role} : C, \ldots\rangle, \ldots, \langle \text{role} : D, \mu, \ldots\rangle, \ldots \rangle \in V, \)

\( \text{role} \neq \text{role}', \mu \notin \{0..1\} \).

Properties

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

\[ \text{does not allow} \]

\[ \text{Class} \]

\[ \text{Role} \]

\[ \text{Constraint} \]

\[ \text{Relationship} \]
Properties

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 at most one ( r )-link to a single other object</td>
<td>current setting</td>
</tr>
<tr>
<td>bag</td>
<td>one object may have multiple ( 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 sequence 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 ( \text{role}(\text{expr}) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>unique</td>
<td>( \tau_D \rightarrow \text{Set}(\tau_C) )</td>
</tr>
<tr>
<td>bag</td>
<td>( \tau_D \rightarrow \text{Bag}(\tau_C) )</td>
</tr>
<tr>
<td>ordered, sequence</td>
<td>( \tau_D \rightarrow \text{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) \[ C \rightarrow_{\text{role}}^{0.1} D \]

  (ii) \[ C \rightarrow_{\text{role}}^{0.\ast} D \]

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

- Form (i) introduces \( \text{role} : C_{0.1} \), and form (ii) introduces \( \text{role} : C_{\ast} \) in \( V \).
- In both cases, \( \text{role} \in \text{atr}(C) \).
- We drop \( \lambda \) and go back to our nice \( \sigma \) with \( \sigma(u)(\text{role}) \subseteq \mathcal{D}(D) \).
Where Shall We Put OCL Constraints?

Two options:

(i) Notes.

(ii) Particular dedicated places.

(i) Notes:

A UML note is a picture of the form

```
[ text ]
```

_text_ can principally be everything, in particular comments and constraints.

Sometimes, content is explicitly classified for clarity:

```
OCL: expr
```
OCL in Notes: Conventions

stands for

\[ C \]
\[ \ldots \]
\[ \ldots \]

Where Shall We Put OCL Constraints?

(ii) **Particular dedicated places** in class diagrams: (behav. feature: later)

\[
\xi v : \tau \{p_1, \ldots, p_n\} \{ \text{expr} \}
\]
\[
\xi f(v_1 : \tau_1, \ldots, v_n : \tau_n) : \tau \{ p_1, \ldots, p_n \} \{ \text{pre} : \text{expr}_1 \}
\]
\[
\text{post} : \text{expr}_2
\]

For simplicity, we view the above as an abbreviation for

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

\[ \text{context } f \text{ pre} : \text{expr}_1 \text{ post} : \text{expr}_2 \]
Invariants of a Class Diagram

- Let $CD$ be a class diagram.
- As we (now) are able to recognise OCL constraints when we see them, we can define $Inv(CD)$ as the set $\{\varphi_1, \ldots, \varphi_n\}$ of OCL constraints occurring in notes in $CD$ — after unfolding all abbreviations (cf. next slides).
- As usual: $Inv(CD) := \bigcup_{CD \in CD} Inv(CD)$.
- Principally clear: $Inv(\cdot)$ for any kind of diagram.

Invariant in Class Diagram Example

If $CD$ consists of only $CD$ with the single class $C$, then
- $Inv(CD) = Inv(CD) = \ldots$
**Semantics of a Class Diagram**

**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

\[
[\mathcal{CD}] := (\mathcal{S}(\mathcal{CD}), \text{Inv}(\mathcal{CD})).
\]

Given a structure \( \mathcal{S} \) of \( \mathcal{S} \) (and thus of \( \mathcal{CD} \)), the class diagrams describe the system states \( \Sigma_{\mathcal{S}} \), of which some may satisfy \( \text{Inv}(\mathcal{CD}) \).

**In pictures:**

\[
\mathcal{CD} = \{ CD_1, \ldots, CD_n \}
\]

- signature \( \mathcal{S}(\mathcal{CD}) \)
- invariants \( \text{Inv}(\mathcal{CD}) \)
- distinguish (basic, classes and attributes)
- extended (visibility)

**Pragmatics**

**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.
Constraints vs. Types

Find the 10 differences:

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

\[
\begin{array}{|c|}
\hline
\mathcal{D}(T) = \{3\} \\
\hline
\end{array}
\]

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

\[
\cup \{n \in \mathbb{N} \mid n > 17\}
\]

- \(x = 4\) 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 Charts)* *(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]
**Signature With Signals**

**Definition.** A tuple

$$\mathcal{I} = (\mathcal{T}, \mathcal{C}, V, atr, \epsilon), \quad \epsilon \subseteq \mathcal{C} \text{ a set of signals},$$

is called **signature (with signals)** if and only if

$$(\mathcal{T}, \mathcal{C}, V, atr)$$

is a signature (as before).

**Note:** Thus conceptually, a signal is a class and can have attributes of plain type and associations.
**Definition.**

A **core state machine** over signature $\mathcal{S} = (\mathcal{T}, C, V, atr, \delta)$ 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

$$ \rightarrow \subseteq S \times (\mathcal{E} \cup \{\bot\}) \times Expr_\mathcal{S} \times Act_\mathcal{S} \times S $$

is a labelled transition relation.

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

```
\[
\text{annot} ::= \left( \text{event}\right) \left( \left\{ \text{event} \right\} ^* \right) \left( \left\{ \text{guard} \right\} \right) \left( \left\{ \text{action} \right\} \right)
\]
```

with

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

maps to

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

References


