Software Design, Modelling and Analysis in UML

Lecture 17: Reflective Description of Behaviour,
Live Sequence Charts I

2012-01-25

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

Contents & Goals

Last Lecture:
• Constructive description of behaviour completed:
  • Remaining pseudo-states, such as shallow/deep history.

This Lecture:
• Educational Objectives: Capabilities for following tasks/questions.
  • What does this LSC mean?
  • Are this UML model’s state machines consistent with the interactions?
  • Please provide a UML model which is consistent with this LSC.
  • What is: activation, hot/cold condition, pre-chart, etc.?

• Content:
  • Brief: methods/behavioural features.
  • Reflective description of behaviour.
  • LSC concrete and abstract syntax.
  • LSC intuitive semantics.
  • Symbolic Büchi Automata (TBA) and its (accepted) language.
And What About Methods?

- In the current setting, the (local) state of objects is only modified by actions of transitions, which we abstract to transformers.
- In general, there are also methods.
- UML follows an approach to separate
  - the interface declaration from
  - the implementation.
  In C++ lingo: distinguish declaration and definition of method.

- In UML, the former is called behavioural feature and can (roughly) be
  - a call interface \( f(\tau_1, \ldots, \tau_{n_1}) : \tau_1 \)
  - a signal name \( E \)

Note: The signal list can be seen as redundant (can be looked up in the state machine) of the class. But: certainly useful for documentation (or sanity check).
Semantics:

- The **implementation** of a behavioural feature can be provided by:
  - An **operation**.

  In our setting, we simply assume a transformer like $T_f$.

  It is then, e.g. clear how to admit method calls as actions on transitions: function composition of transformers (clear but tedious: non-termination).

  In a setting with Java as action language: operation is a method body.

- The class' **state-machine** ("triggered operation").

  - Calling $F$ with $n_2$ parameters for a stable instance of $C$ creates an auxiliary event $F$ and dispatches it (bypassing the ether).
  - Transition actions may fill in the return value.
  - On completion of the RTC step, the call returns.
  - For a non-stable instance, the caller blocks until stability is reached again.

**Visibility:**

- Extend typing rules to sequences of actions such that a well-typed action sequence only calls visible methods.

**Useful properties:**

- **concurrency**
  - **concurrent** — is thread safe
  - **guarded** — some mechanism ensures/should ensure mutual exclusion
  - **sequential** — is not thread safe, users have to ensure mutual exclusion

- **isQuery** — doesn’t modify the state space (thus thread safe)

- For simplicity, we leave the notion of steps untouched, we construct our semantics around state machines.

Yet we could explain pre/post in OCL (if we wanted to).
You are here.

Course Map

\[ \mathcal{P} = (\mathcal{P}, \mathcal{E}, V, atr), SM \]
\[ M = (\Sigma_{\mathcal{P}}, A_{\mathcal{P}}, \neg_{SM}) \]
\[ \pi = (\sigma_0, \epsilon_0) \xrightarrow{(cons_i, Snd_i)} (\sigma_1, \epsilon_1) \ldots \]
\[ w_n = ((\sigma_i, cons_i, Snd_i))_{i \in \mathbb{N}} \]
\[ G = (N, E, f) \]

Mathematics

UML

Model

Instances
Motivation: Reflective, Dynamic Descriptions of Behaviour

What Can Be Purposes of Behavioural Models?

Example: Pre-Image
(the UML model is supposed to be the blue-print for a software system).

A description of behaviour could serve the following purposes:

- **Require** Behaviour.  
  “System definitely does this”  
  “This sequence of inserting money and requesting and getting water must be possible.”  
  (Otherwise the software for the vending machine is completely broken.)

- **Allow** Behaviour.  
  “System does subset of this”  
  “After inserting money and choosing a drink, the drink is dispensed (if in stock).”  
  (If the implementation insists on taking the money first, that’s a fair choice.)

- **Forbid** Behaviour.  
  “System never does this”  
  “This sequence of getting both, a water and all money back, must not be possible.”  
  (Otherwise the software is broken.)

Note: the latter two are trivially satisfied by doing nothing...
Constructive vs. Reflective Descriptions

[Harel, 1997] proposes to distinguish constructive and reflective descriptions:

- “A language is **constructive** if it contributes to the dynamic semantics of the model. That is, its constructs contain information needed in executing the model or in translating it into executable code.”
  
  A constructive description tells **how** things are computed (which can then be desired or undesired).

- “Other languages are **reflective** or **assertive**, and can be used by the system modeler to capture parts of the thinking that go into building the model – **behavior included** –, to derive and present views of the model, statically or during execution, or to set constraints on behavior in preparation for verification.”
  
  A reflective description tells **what** shall or shall not be computed.

**Note:** No sharp boundaries!

Constructive UML

UML provides two visual formalisms for constructive description of behaviours:

- **Activity Diagrams**
- **State-Machine Diagrams**

We (exemplary) focus on State-Machines because

- somehow “practice proven” (in different flavours),
- prevalent in embedded systems community,
- indicated useful by [Dobing and Parsons, 2006] survey, and
- Activity Diagram’s intuition changed from transition-system-like to petri-net-like...
Recall: What is a Requirement?

Recall:
- The semantics of the UML model \( M = (\\mathcal{P}, \mathcal{M}, \mathcal{D}) \) is the transition system \((S, \rightarrow, S_0)\) constructed according to discard/dispatch/commence-rules.
- The computations of \( M \), denoted by \([M]\), are the computations of \((S, \rightarrow, S_0)\).

Now:
A reflective description tells what shall or shall not be computed.

More formally: a requirement \( \vartheta \) is a property of computations, sth. which is either satisfied or not satisfied by a computation

\[
\pi = (\sigma_0, \varepsilon_0) \xrightarrow{(\text{cons}_0, \text{Snd}_0)} (\sigma_1, \varepsilon_1) \xrightarrow{(\text{cons}_1, \text{Snd}_1)} \cdots \in [M],
\]
denoted \( \pi \models \vartheta \) and \( \pi \not\models \vartheta \).

OCL as Reflective Description of Certain Properties

- invariants:

\[
\forall \pi \in [M] \forall i \in \mathbb{N} : \pi_i \models \vartheta,
\]

- non-reachability of configurations:

\[
\#\pi \in [M] \# i \in \mathbb{N} : \pi_i \models \vartheta
\]

\[\iff \forall \pi \in [M] \forall i \in \mathbb{N} : \pi_i \models \neg \vartheta\]

- reachability of configurations:

\[
\exists \pi \in [M] \exists i \in \mathbb{N} : \pi_i \models \vartheta
\]

\[\iff \neg (\forall \pi \in [M] \forall i \in \mathbb{N} : \pi_i \models \neg \vartheta)\]

where
- \( \vartheta \) is an OCL expression or an object diagram and
- “\(\models\)” is the corresponding OCL satisfaction or the “is represented by object diagram” relation.
**In General Not OCL: Temporal Properties**

**Dynamic** (by example)
- **reactive behaviour**
  - “for each \( C \) instance, each reception of \( E \) is finally answered by \( F \)"
  \[ \forall \pi \in [M] : \pi \models \vartheta \]

- **non-reachability** of system configuration sequences
  - “there mustn’t be a system run where \( C \) first receives \( E \) and then sends \( F \)”
  \[ \not\exists \pi \in [M] : \pi \models \vartheta \]

- **reachability** of system configuration sequences
  - “there must be a system run where \( C \) first receives \( E \) and then sends \( F \)”
  \[ \exists \pi \in [M] : \pi \models \vartheta \]

**But:** what is “\( \models \)” and what is “\( \vartheta \)”?

---

**Interactions: Problem and Plan**

---

4. Define the **language** \( L(M) \) of a **model** \( M \) — basically its computations. Each computation \( \pi \in [M] \) corresponds to a **word** \( w_\pi \).

2. Define the **language** \( L(I) \) of an **interaction** \( I \) — via Büchi automata.

1. Then (conceptually) \( \pi \models \vartheta \) if and only if \( w_\pi \in L(I) \).
Definition. Let $S = (S, C, V, atr)$ be a signature and $D$ a structure of $S$. A **word** over $S$ and $D$ is an infinite sequence

$$(\sigma_i, cons_i, Snd_i)_{i \in \mathbb{N}_0} \in (\Sigma^D \times 2^{D(C)} \times \text{Evs}(D, C)) \times (\Sigma^D \times 2^{D(C)} \times \text{Evs}(D, C)) \times \mathbb{N}.$$

---

**The Language of a Model**

**Recall:** A UML model $M = (C, \mathcal{M}, C, D)$ and a structure $D$ denotes a set $[M]$ of (initial and consecutive) **computations** of the form

$$(\sigma_0, \varepsilon_0) \xrightarrow{a_0} (\sigma_1, \varepsilon_1) \xrightarrow{a_1} (\sigma_2, \varepsilon_2) \xrightarrow{a_2} \ldots$$

where

$$a_i = (cons_i, Snd_i, u_i) \in (\Sigma^D \times 2^{D(C)} \times \text{Evs}(D, C)) \times (\Sigma^D \times 2^{D(C)} \times \text{Evs}(D, C)) \times \mathbb{N}.$$ 

For the connection between models and interactions, we **disregard** the configuration of the ether and who made the step, and define as follows:

**Definition.** Let $M = (C, \mathcal{M}, C, D)$ be a UML model and $D$ a structure. Then

$$\mathcal{L}(M) := \{ (\sigma_i, cons_i, Snd_i)_{i \in \mathbb{N}_0} \in (\Sigma^D \times \tilde{A})^\omega \mid \exists (\varepsilon_i, u_i)_{i \in \mathbb{N}_0} : (\sigma_0, \varepsilon_0) \xrightarrow{(cons_0, Snd_0)} u_0 (\sigma_1, \varepsilon_1) \cdots \in [M] \}$$

is the **language** of $M$. 
Model Consistency wrt. Interaction

- We assume that the set of interactions $\mathcal{I}$ is partitioned into two (possibly empty) sets of universal and existential interactions, i.e.

\[ \mathcal{I} = \mathcal{I}_\forall \cup \mathcal{I}_\exists. \]

**Definition.** A model

\[ M = (\mathcal{C}_D, \mathcal{C}_M, \mathcal{C}_D, \mathcal{I}) \]

is called consistent (more precise: the constructive description of behaviour is consistent with the reflective one) if and only if

\[ \forall I \in \mathcal{I}_\forall : \mathcal{L}(M) \subseteq \mathcal{L}(I) \]

and

\[ \forall I \in \mathcal{I}_\exists : \mathcal{L}(M) \cap \mathcal{L}(I) \neq \emptyset. \]

Interactions: Plan

- In the following, we consider Sequence Diagrams as interaction $I$,
- more precisely: Live Sequence Charts [Damm and Harel, 2001].
- We define the language $\mathcal{L}(I)$ of an LSC — via Büchi automata.
- Then (conceptually) $\pi \models \emptyset$ if and only if $w_\pi \in \mathcal{L}(I)$.

Why LSC, relation LSCs/UML SDs, other kinds of interactions: later.
Example
**Building Blocks**

- **Instance Lines:**
  
  ![Environment and C Diagram]

- **Messages:** (asynchronous or synchronous/instantaneous)
  
  ![Message Diagram]

- **Conditions and Local Invariants:** \( (expr_1, expr_2, expr_3 \in Expr_{\geq}) \)
  
  ![Conditions and Invariants Diagram]

---

**Intuitive Semantics: A Partial Order on Simclasses**

(i) **Strictly After:**

(ii) **Simultaneously:** (simultaneous region)

(iii) **Explicitly Unordered:** (co-region)

**Intuition:** A computation path violates an LSC if the occurrence of some events doesn’t adhere to partial order obtained as the transitive closure of (i) to (iii).
**Example: Partial Order Requirements**

LSC: L
AC: actcond
AM: invariant I: strict

Environment: LightsCtrl
Operational [1, 3]: CrossingCtrl
[1, 5]: BarrierCtrl
t(10) lights_on

CrossingCtrl
LightsCtrl
BarriercCtrl

**LSC Specialty: Modes**

With LSCs,
- whole charts,
- locations, and
- elements

have a mode — one of **hot** or **cold** (graphically indicated by outline).

<table>
<thead>
<tr>
<th>chart</th>
<th>location</th>
<th>message</th>
<th>condition/local inv.</th>
</tr>
</thead>
<tbody>
<tr>
<td>hot:</td>
<td><img src="hot.png" alt="Diagram" /></td>
<td><img src="message.png" alt="Diagram" /></td>
<td><img src="condition.png" alt="Diagram" /></td>
</tr>
<tr>
<td>cold:</td>
<td><img src="cold.png" alt="Diagram" /></td>
<td><img src="message.png" alt="Diagram" /></td>
<td><img src="condition.png" alt="Diagram" /></td>
</tr>
</tbody>
</table>

always vs. at least once
must vs. may progress
mustn’t vs. may get lost
necessary vs. legal exit
Example: Modes

**LSC Specialty: Activation**

One **major defect** of MSCs and SDs: they don’t say when the scenario has to/may be observed.

**LSCs**: Activation condition (AC ∈ Expr), activation mode (AM ∈ \{init, inv\}), and pre-chart.

**Intuition**: (universal case)
- given a computation π, whenever expr holds in a configuration (σ_i, ε_i) of ξ
  - which is initial, i.e. k = 0, or \(AM = \text{initial}\)
  - whose k is not further restricted, \(AM = \text{invariant}\)
- and if the pre-chart is observed from k to \(k + n\),
  - then the main-chart has to follow from \(k + n + 1\).
Example: What Is Required?

- Whenever the CrossingCtrl has consumed a 'secreq' event
- then it shall finally send 'lights_on' and 'barrier_down' to LightsCtrl and BarrierCtrl,
- if LightsCtrl is not 'operational' when receiving that event, the rest of this scenario doesn't apply; maybe there's another sequence diagram for that case.
- if LightsCtrl is operational when receiving that event, it shall reply with 'lights_ok' within 1–3 time units,
- the BarrierCtrl shall reply with 'barrier_ok' within 1–5 time units, during this time (dispatch time not included) it shall not be in state 'MvUp',
- 'lights_ok' and 'barrier_ok' may occur in any order.
- After having consumed both, CrossingCtrl replies with 'done' to the environment.

Live Sequence Charts — Abstract Syntax
Let $\Theta = \{\text{hot, cold}\}$. An **LSC body** is a tuple

$$(I, (\mathcal{L}, \preceq), \sim, \mathcal{R}, \text{Msg}, \text{Cond}, \text{LocInv})$$

where

- $I$ is a finite set of instance lines,
- $(\mathcal{L}, \preceq)$ is a finite, non-empty, partially ordered set of locations, each $l \in \mathcal{L}$ is associated with a temperature $\theta(l) \in \Theta$ and an instance line $i_l \in I$,
- $\sim \subseteq \mathcal{L} \times \mathcal{L}$ is an equivalence relation on locations, the simultaneity relation,
- $\mathcal{R} = (\mathcal{P}, \emptyset, V, \text{attr})$ is a signature,
- $\text{Msg} \subseteq \mathcal{L} \times \mathcal{B} \times \mathcal{L}$ is a set of asynchronous messages with $(l, b, l') \in \text{Msg}$ only if $l \sim l'$,

**Note:** instantaneous messages — could be linked to method/operation calls.

- $\text{Cond} \subseteq (2^{\mathcal{P}} \setminus \emptyset) \times \mathcal{P} \times \Theta$ is a set of conditions with $(L, \text{expr}, \theta) \in \text{Cond}$ only if $l \sim l'$ for all $l, l' \in L$,
- $\text{LocInv} \subseteq \mathcal{L} \times \{\circ, \bullet\} \times \mathcal{P} \times \Theta \times \mathcal{L} \times \{\circ, \bullet\}$ is a set of local invariants,

### Well-Formedness

**Bondedness/no floating conditions:** (could be relaxed a little if we wanted to)

- For each location $l \in \mathcal{L}$, if $l$ is the location of
  - a **condition**, i.e. $\exists (L, \text{expr}, \theta) \in \text{Cond} : l \in L$,
  - a **local invariant**, i.e. $\exists (l_1, i_1, \text{expr}, \theta, l_2, i_2) \in \text{LocInv} : l \in \{l_1, l_2\}$, or
  - then there is a location $l'$ equivalent to $l$ which is the location of
    - a **message**, i.e. $\exists (l_1, b, l_2) \in \text{Msg} : l \in \{l_1, l_2\}$, or
    - an **instance head**, i.e. $l'$ is minimal wrt. $\preceq$.

**Note:** if messages in a chart are cyclic, then there doesn’t exist a partial order (so such charts don’t even have an abstract syntax).
References
References


