Software Design, Modelling and Analysis in UML

Lecture 20: Live Sequence Charts

2015-02-03

Prof. Dr. Andreas Podelski, Dr. Bernd Westphal

Albert-Ludwigs-Universität Freiburg, Germany

Contents & Goals

Last Lecture:

- Hierarchical State Machines completed.
- Behavioural feature (aka. methods).

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:
  - Reflective description of behaviour.
  - LSC concrete and abstract syntax.
  - LSC semantics.
You are here.

Course Map

UML

\( \mathcal{F} = (T, \epsilon, V, atr), SM \)

\( M = ( \Sigma, A, \rightarrow_{SM} ) \)

\( \pi = (\sigma_0, \varepsilon_0) \xrightarrow{(\text{cons}, S_{\text{init}})} (\sigma_1, \varepsilon_1) \cdots \xrightarrow{w_\pi} \omega \)

G = (N, E, f)

CD, SM

\( \varphi \in \text{OCL} \)

expr

CD, SD

\( \mathcal{F}, SD \)

\( B = (Q_{SD}, q_0, A, \rightarrow_{SD}, F_{SD}) \)

\( \mathcal{U} \)

Mathematics

OD

UML

\( \mathcal{G} = (N, E, f) \)
Recall: 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!
Recall: What is a Requirement?

Recall:
- The **semantics** of the **UML model** $M = (C, D, S, M, O)$ 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; something which is either satisfied or not satisfied by a computation

$$\pi = (\sigma_0, \varepsilon_0) \xrightarrow{(cons_0, Snd_0)} (\sigma_1, \varepsilon_1) \xrightarrow{(cons_1, Snd_1)} \cdots \in [M],$$

denoted by $\pi \models \vartheta$ and $\pi \not\models \vartheta$, resp.

Simplest case: OCL constraint.

---

**Live Sequence Charts — Concrete Syntax**
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 LSC 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 may reply with `done` to the environment.
- Instance Lines:

- Messages: (asynchronous or synchronous/instantaneous)
Conditions and Local Invariants: \( (expr_1, expr_2, expr_3 \in \text{Expr}_S) \)

Intuitive Semantics: A Partial Order on Simclasses

(i) Strictly After:

(ii) Simultaneously: (simultaneous region)

(iii) Explicitly Unordered: (co-region)
Partial Order Requirements

- 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 LSC 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 may reply with ‘done’ to the environment.

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="chart1.png" alt="chart" /></td>
<td><img src="location1.png" alt="location" /></td>
<td><img src="message1.png" alt="message" /></td>
</tr>
<tr>
<td>cold:</td>
<td><img src="chart2.png" alt="chart" /></td>
<td><img src="location2.png" alt="location" /></td>
<td><img src="message2.png" alt="message" /></td>
</tr>
<tr>
<td>always vs. at least once</td>
<td>must vs. may progress</td>
<td>mustn’t vs. may get lost</td>
<td>necessary vs. legal exit</td>
</tr>
</tbody>
</table>
Example: Modes

- 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 LSC 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 may reply with ’done’ to the environment.

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_S), activation mode (AM ∈ {init, inv}), and pre-chart.
**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 \( AC \in \text{Expr}_\varphi \), activation mode \( AM \in \{ \text{init}, \text{inv} \} \), and pre-chart.

**Intuition**: (universal case)
- given a computation \( \pi \), **whenever** \( expr \) holds in a configuration \( (\sigma_i, \varepsilon_i) \) of \( \xi \)
  - 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 LSC 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** may reply with ‘done’ to the environment.
Restricted Syntax
Restricted Abstract Syntax

Cuts

A set $C \subseteq \mathcal{L}$ is called cut if:
- 1. downward closed wrt. $\leq$
- 2. closed wrt. $\sim$
- 3. at least one loc. per instance $\mathcal{L}_i$ (if more than one, then unordered)
Towards Automata
You are here.
Language of a Model

Model

\[ M = (\Sigma, A_{ SM}) \]

Instances

\[ \pi = \sigma_0, \epsilon_0 \]

\[ w^i = (\sigma_i, cons_i, Snd_i) \in N \]

\[ w^u = (\sigma_i, cons_i, Snd_i) \in N \]

\[ B = (Q_{ SD}, q_0, A_{ SD}, F_{ SD}) \]

\[ G = (N, E, f) \]

\[ \varphi \in \text{OCL} \]

\[ \mathcal{F} = (\mathcal{F}, E, V, atr), SM \]

\[ \mathcal{F}, SD \]

\[ \mathcal{G} = (N, E, f) \]

\[ \pi \in \text{UML} \]

Mathematics

\[ \pi = (\sigma_0, \epsilon_0) \]

\[ w^u = (\sigma_i, cons_i, Snd_i) \]

\[ G = (N, E, f) \]

\[ \varphi \in \text{OCL} \]

\[ \mathcal{F} = (\mathcal{F}, E, V, atr), SM \]

\[ \mathcal{F}, SD \]

\[ \mathcal{G} = (N, E, f) \]
**Words over Signature**

**Definition.** Let \( \mathcal{S} = ( \mathcal{F}, \mathcal{C}, V, \mathcal{at}, \mathcal{E}) \) be a signature and \( \mathcal{D} \) a structure of \( \mathcal{S} \). A word over \( \mathcal{S} \) and \( \mathcal{D} \) is an infinite sequence

\[
(\sigma_i, \text{cons}_i, \text{Snd}_i)_{i \in \mathbb{N}_0} \in \left( \Sigma_{\mathcal{S}} \times 2^{\mathcal{D}(\mathcal{E})} \times \text{Evs}(\mathcal{E}, \mathcal{D}) \times 2^{\mathcal{D}(\mathcal{E})} \times \text{Evs}(\mathcal{E}, \mathcal{D}) \times 2^{\mathcal{D}(\mathcal{C})} \right)^\omega.
\]

---

**The Language of a Model**

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

\[
(\sigma_0, \varepsilon_0) \xrightarrow{a_0} (\sigma_1, \varepsilon_1) \xrightarrow{a_1} (\sigma_2, \varepsilon_2) \xrightarrow{a_2} \ldots \text{ where}
\]

\[
a_i = (\text{cons}_i, \text{Snd}_i, u_i) \in \left( \mathcal{D}(\mathcal{E}) \times \text{Evs}(\mathcal{E}, \mathcal{D}) \times 2^{\mathcal{D}(\mathcal{C})} \times \mathcal{D}(\mathcal{C}) \right) \times \mathcal{D}(\mathcal{C}).
\]

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 \( \mathcal{M} = ( \mathcal{C}, \mathcal{D}, \mathcal{A}, \mathcal{O} ) \) be a UML model and \( \mathcal{D} \) a structure. Then

\[
\mathcal{L}(\mathcal{M}) := \{ (\sigma_i, \text{cons}_i, \text{Snd}_i)_{i \in \mathbb{N}_0} \in (\Sigma_{\mathcal{S}} \times \tilde{A})^\omega | \exists (\varepsilon_i, u_i)_{i \in \mathbb{N}_0} : (\sigma_0, \varepsilon_0) \xrightarrow{\text{cons}_0, \text{Snd}_0} (\sigma_1, \varepsilon_1) \cdots \in \llbracket \mathcal{M} \rrbracket \}
\]

is the **language** of \( \mathcal{M} \).
Example: The Language of a Model

\[ L(M) := \{(\sigma_i, \text{cons}_i, \text{Snd}_i)_{i \in \mathbb{N}_0} \in (\Sigma^E \times \tilde{A})^\omega \mid \exists (\varepsilon_i, u_i)_{i \in \mathbb{N}_0} : (\sigma_0, \varepsilon_0) \xrightarrow{\text{cons}_0, \text{Snd}_0} u_0 \rightarrow (\sigma_1, \varepsilon_1) \cdots \in \llbracket M \rrbracket \} \]

Signal and Attribute Expressions

- Let \( \mathcal{T} = (\mathcal{I}, \mathcal{C}, V, \mathit{atr}, \mathcal{E}) \) be a signature and \( X \) a set of logical variables,

- The signal and attribute expressions \( \operatorname{Expr}_{\mathcal{T}}(\mathcal{E}, X) \) are defined by the grammar:

  \[
  \psi ::= \text{true} \mid \text{expr} \mid E^1_{x,y} \mid E^2_{x,y} \mid \neg \psi \mid \psi_1 \lor \psi_2 \mid E_{x,y}^{l/l}
  \]

  where \( \text{expr} : \text{Bool} \in \operatorname{Expr}_{\mathcal{E}}, E \in \mathcal{E}, x, y \in X \).
• Let \((\sigma, \text{cons}, \text{Snd}) \in \Sigma_D \times \bar{A}\) be a triple consisting of system state, consume set, and send set.
• Let \(\beta : X \rightarrow \mathcal{D}(C)\) be a valuation of the logical variables.

Then

- \((\sigma, \text{cons}, \text{Snd}) \models_{\beta} \text{true}\)
- \((\sigma, \text{cons}, \text{Snd}) \models_{\beta} \neg \psi\) if and only if not \((\sigma, \text{cons}, \text{Snd}) \models_{\beta} \psi\)
- \((\sigma, \text{cons}, \text{Snd}) \models_{\beta} \psi_1 \lor \psi_2\) if and only if
  \((\sigma, \text{cons}, \text{Snd}) \models_{\beta} \psi_1\) or \((\sigma, \text{cons}, \text{Snd}) \models_{\beta} \psi_2\)
- \((\sigma, \text{cons}, \text{Snd}) \models_{\beta} \text{expr}\) if and only if \(I[\text{expr}](\sigma, \beta) = 1\)
- \((\sigma, \text{cons}, \text{Snd}) \models_{\beta} E_{x,y}\) if and only if \(\exists \vec{d} \bullet (\beta(x), (E, \vec{d}), \beta(y)) \in \text{Snd}\)
- \((\sigma, \text{cons}, \text{Snd}) \models_{\beta} E'_{x,y}\) if and only if \(\exists \vec{d} \bullet (\beta(x), (E, \vec{d}), \beta(y)) \in \text{cons}\)

Observation: semantics of models keeps track of sender and receiver at sending and consumption time. We disregard the event identity.

Alternative: keep track of event identities.

---

**TBA over Signature**

**Definition.** A TBA

\[ B = (\text{Expr}_B(X), X, Q_{\text{init}}, \rightarrow, Q_F) \]

where \(\text{Expr}_B(X)\) is the set of signal and attribute expressions 
\(\text{Expr}_\mathcal{S}(\mathcal{S}, X)\) over signature \(\mathcal{S}\) is called **TBA over \(\mathcal{S}\)**.

- Any word over \(\mathcal{S}\) and \(\mathcal{D}\) is then a word for \(B\).
  (By the satisfaction relation defined on the previous slide; \(\mathcal{D}(X) = \mathcal{D}(\mathcal{C})\).)
- Thus a TBA over \(\mathcal{S}\) accepts words of models with signature \(\mathcal{S}\).
  (By the previous definition of TBA.)
TBA over Signature Example

\[(\sigma, \text{cons}, \text{Snd}) \models_{\beta} \text{expr} \iff I[\text{expr}](\sigma, \beta) = 1;\]

\[(\sigma, \text{cons}, \text{Snd}) \models_{\beta} E_{x,y} \iff (\beta(x), (E, \bar d), \beta(y)) \in \text{Snd}\]
Activation Condition

Universal vs. Existential Charts
Prechart

Conditions
Conditions

Back to UML: Interactions
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

\[
\mathcal{M} = (\mathcal{D}, \mathcal{M}, \mathcal{O}, \mathcal{I})
\]

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

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

and

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

Interactions as Reflective Description

- In UML, reflective (temporal) descriptions are subsumed by interactions.
- A UML model \( \mathcal{M} = (\mathcal{D}, \mathcal{M}, \mathcal{O}, \mathcal{I}) \) has a set of interactions \( \mathcal{I} \).
- An interaction \( I \in \mathcal{I} \) can be (OMG claim: equivalently) diagrammed as
  - sequence diagram,
  - timing diagram, or
  - communication diagram (formerly known as collaboration diagram).
Interactions as Reflective Description

- In UML, reflective (temporal) descriptions are subsumed by interactions.
- A UML model \( M = (\mathcal{C}, \mathcal{D}, \mathcal{S}, \mathcal{O}, \mathcal{I}) \) has a set of interactions \( \mathcal{I} \).
- An interaction \( I \in \mathcal{I} \) can be (OMG claim: equivalently) diagrammed as
  - sequence diagram, timing diagram, or
  - communication diagram (formerly known as collaboration diagram).

Why Sequence Diagrams?

Most Prominent: Sequence Diagrams — with long history:
- Message Sequence Charts, standardized by the ITU in different versions, often accused to lack a formal semantics.
- Sequence Diagrams of UML 1.x

Most severe drawbacks of these formalisms:
- unclear interpretation: example scenario or invariant?
- unclear activation: what triggers the requirement?
- unclear progress requirement: must all messages be observed?
- conditions merely comments
- no means to express forbidden scenarios
Thus: *Live Sequence Charts*

- **SDs of UML 2.x** address *some* issues, yet the standard exhibits unclarities and even contradictions [Harel and Maoz, 2007, Störrle, 2003]
- For the lecture, we consider *Live Sequence Charts* (LSCs) [Damm and Harel, 2001, Klose, 2003, Harel and Marelly, 2003], who have a common fragment with UML 2.x SDs [Harel and Maoz, 2007]
- **Modelling guideline:** stick to that fragment.

*References*


