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.
\[
\mathcal{I} = (\mathcal{T}, \mathcal{C}, V, \text{atr}), \quad SM
\]

\[
M = (\Sigma_{\mathcal{I}}, A_{\mathcal{I}}, \rightarrow_{SM})
\]

\[
\pi = (\sigma_0, \varepsilon_0) \xrightarrow{(\text{cons}_0, \text{Snd}_0)} (\sigma_1, \varepsilon_1) \cdots
\]

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

\[
\mathcal{O} = (\mathcal{O}_D)
\]

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

\[
\mathcal{D}, \mathcal{S} = (T, C, V, \text{atr})
\]

\[
\mathcal{S} = (\Sigma, A_{\mathcal{I}}, \rightarrow_{SM})
\]

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

\[
\pi = (\sigma_i, \text{cons}_i, \text{Snd}_i)_{i \in \mathbb{N}}
\]
Motivation: Reflective, Dynamic Descriptions of Behaviour
[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** $M = (CD, \mathcal{I}M, \mathcal{O}D)$ is the transition system $(S, \rightarrow, S_0)$ constructed according to discard/dispatch/commence-rules.
- The **computations of** $M$, denoted by $\llbracket M \rrbracket$, 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

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

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

Simplest case: OCL constraint.
Live Sequence Charts — Concrete Syntax
Example: Live Sequence Charts

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

Environment : LightsCtrl
CrossingCtrl : BarrierCtrl

秘书长

$\langle \text{secreq} \rangle$

$\langle \text{lights}_\text{on} \rangle$

$\langle \text{lights}_\text{ok} \rangle$

$\langle \text{barrier}_\text{down} \rangle$

$\langle \text{barrier}_\text{ok} \rangle$

$\langle \text{done} \rangle$

$\langle \text{cold} \rangle$

$\langle \text{hot} \rangle$

$\langle \text{sync} \rangle$

$sync\text{huous/instantaneous message}$

instance

object name

class name

simultaneous region

conditions

head

local invariant

main chart

coregion

Re-chart
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:**

```
Environment : C
```

- LSC: \( L \)
- AC: actcond
- AM: invariant I: strict
• **Messages:**  (asynchronous or synchronous/instantaneous)
• **Conditions and Local Invariants:** \((expr_1, expr_2, expr_3 \in Expr)\)
(i) **Strictly After:**

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

(iii) **Explicitly Unordered:** (co-region)
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><strong>hot:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><img src="chart_hot.png" alt="chart" /></td>
<td>![location_hot.png]</td>
<td>![message_hot.png]</td>
<td>![condition_hot.png]</td>
</tr>
<tr>
<td><strong>cold:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>![chart_cold.png]</td>
<td>![location_cold.png]</td>
<td>![message_cold.png]</td>
<td>![condition_cold.png]</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

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

**LSCs**: Activation condition \((AC \in \text{Expr } \mathcal{F})\), activation mode \((AM \in \{\text{init, inv}\})\), and pre-chart.
One major defect of MSCs and SDs: they don’t say when the scenario has to/may be observed.

**LSCs:** Activation condition ($AC \in Expr_\mathcal{S}$), activation mode ($AM \in \{\text{init, inv}\}$), 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
  - whose $k$ is not further restricted,
  
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

\[ x : \quad y : \quad z : \]

\[ E \quad F \quad G \]
Restricted Abstract Syntax

\[(I, (\mathcal{L}, \leq), \sim, \mathcal{I}, \text{Msg})\]

\[
\begin{align*}
&x : \\
y : \\
z :
\end{align*}
\]

\[E, F, G, \preceq, \sim, S, \text{Msg}\]
A set $C \subseteq \mathcal{L}$ is called a cut iff
- downward closed \( \subseteq \)
- closed \( \sim \)
- at least one loc. (for instance line (if more than one, then unordered)
The firedets expression is shown in the diagram:

- $x, y, z$ are variables.
- $l_{1,0}, l_{1,1}, l_{1,2}$, etc., are specific values.
- $E, F, G$ are events.

Formally:

1. $F \neq \emptyset$
2. $C \setminus C' = F$
3. For all events $e'$ in $F$, the sendings are in $C$
4. Direct successor:
   \[ \forall e \in F \exists e' \exists e' : e, e' \leq e' \Rightarrow e \leq e' \]
Alphabet — Progress Transitions

\begin{align*}
E_{x,y} & \quad l_{1,0} \\
E & \quad l_{1,x} \\
F & \quad l_{2,1} \\
G & \quad l_{2,2} \\
& \quad l_{3,1} \\
F_{y,z} \land \neg G_{y,x} & \quad l_{1,2} \\
& \quad l_{2,3} \\
F_{y,z} \land \neg G_{y,x} & \quad l_{3,0} \\
& \quad l_{3,0} \\
& \quad l_{3,0} \\
\end{align*}
Loops

\begin{align*}
&\text{loops } l_{1,0}, l_{1,1}, l_{2,0}, l_{2,1}, l_{3,0}, l_{3,1}, l_{1,2}, l_{2,3} \\
&\text{states } q_1, q_2, q_3, q_4, q_5, q_6, q_7 \\
&\text{transitions} \\
&- E\!_x, y \\
&- E\?_x, y \\
&- F\!_y, x \\
&- F\?_y, x \\
&F\?_y, z \land \neg G\?!_y, x \\
&\neg G\?!_y, x \\
&\neg G\?!_y, x \\
&G\?!_y, x \land \neg F\?!_y, z \\
&\neg (F\?_y, z \lor G\?!_y, x) \\
&F\?_y, z \land G\?!_y, x \\
&\neg F\?!_y, z \\
&G\?!_y, x \\
&F\?_y, z \\
&\neg F\?!_y, z \\
&\text{true}
\end{align*}
Language

\[ E \succneq y \]

\[ \neg \exists l_1 \in \{0, 1, 2\} \neg \exists l_2 \in \{0, 1, 2\} \neg \exists l_3 \in \{0, 1\} \]

\[ (E \land \neg F) \lor (\neg E \land F) \lor G \land \neg G \land \neg F \]

\[ \neg (F \lor G) \]

\[ \text{true} \]
You are here.
\[ \mathcal{S} = (T, C, V, atr), SM \]

\[ M = (\Sigma^{\mathcal{S}}, A_{\mathcal{S}}, \rightarrow_{SM}) \]

\[ \pi = (\sigma_0, \varepsilon_0) \xrightarrow{(cons_0, Snd_0)} u_0 \xrightarrow{(\sigma_1, \varepsilon_1)} \cdots \]

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

\[ \varphi \in OCL \]

\[ C_D, SM \]

\[ C_D, SD \]

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

\[ \phi \in OCL \]

\[ \pi = (\sigma_i, cons_i, Snd_i)_{i \in \mathbb{N}} \]

\[ \mathcal{O}_D \]

\[ UML \]

\[ Mathematics \]
Language of a Model
Definition. Let $\mathcal{S} = (T, C, V, atr, E)$ be a signature and $D$ a structure of $\mathcal{S}$. A **word** over $\mathcal{S}$ and $D$ is an infinite sequence

$$(\sigma_i, cons_i, Snd_i)_{i \in \mathbb{N}_0} \in \left( \sum_\mathcal{D} \times 2^\mathcal{D}(C) \times \text{Evs}(E, D) \times \mathcal{D}(C) \times 2^\mathcal{D}(E) \times \text{Evs}(E, D) \times \mathcal{D}(C) \right)^\omega.$$
Recall: A UML model \( \mathcal{M} = (C_D, I_M, O_D) \) and a structure \( D \) denotes a set \( \llbracket \mathcal{M} \rrbracket \) of (initial and consecutive) computations of the form

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

\[
a_i = (cons_i, Snd_i, u_i) \in 2^{D(E)} \times \text{Evs}(E, D) \times D(C) \times 2^{D(C)} \times \text{Evs}(E, D) \times D(C) \times D(C).
\]

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 \( \mathcal{M} = (C_D, I_M, O_D) \) be a UML model and \( D \) a structure. Then

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

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

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

- Let $I = (T, C, V, atr, E)$ be a signature and $X$ a set of logical variables,

- The signal and attribute expressions $Expr_I(E, X)$ are defined by the grammar:

$$\psi ::= \text{true} \mid \text{expr} \mid E_{x,y} \mid E_{x,y} \mid \neg \psi \mid \psi_1 \lor \psi_2 \mid E_{x,y}$$

where $\text{expr} : \text{Bool} \in Expr_I$, $E \in E$, $x, y \in X$. 
Let \((\sigma, \text{cons}, \text{Snd}) \in \Sigma_\mathcal{F} \times \tilde{A}\) be a triple consisting of \text{system state}, \text{consume set}, \text{and send set}.

Let \(\beta : X \rightarrow \mathcal{D}(\mathcal{C})\) 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 \(I[\text{expr}](\sigma, \beta) = 1\)
- \((\sigma, \text{cons}, \text{Snd}) \models_\beta \text{E}_{x,y}^1\) if and only if \(\exists \vec{d} \bullet (\beta(x), (E, \vec{d}), \beta(y)) \in \text{Snd}\)
- \((\sigma, \text{cons}, \text{Snd}) \models_\beta \text{E}_{x,y}^2\) if and only if \(\exists \vec{d} \bullet (\beta(x), (E, \vec{d}), \beta(y)) \in \text{cons}\)

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

\text{Alternative}: keep track of event identities.
**TBA over Signature**

**Definition.** A TBA

\[ \mathcal{B} = (\operatorname{Expr}_\mathcal{B}(X), X, Q, q_{\text{ini}}, \rightarrow, Q_F) \]

where \( \operatorname{Expr}_\mathcal{B}(X) \) is the set of **signal and attribute expressions** \( \operatorname{Expr}_\mathcal{I}(\mathcal{E}, X) \) over signature \( \mathcal{I} \) is called **TBA over \( \mathcal{I} \).**

- Any word over \( \mathcal{I} \) and \( \mathcal{D} \) is then a word for \( \mathcal{B} \).
  (By the satisfaction relation defined on the previous slide; \( \mathcal{D}(X) = \mathcal{D}(\mathcal{C}) \).)

- Thus a TBA over \( \mathcal{I} \) accepts words of models with signature \( \mathcal{I} \).
  (By the previous definition of TBA.)
TBA over Signature Example

\[
(\sigma, \text{cons}, \text{Snd}) \models_\beta \text{expr} \iff I[\text{expr}]((\sigma, \beta)) = 1;
(\sigma, \text{cons}, \text{Snd}) \models_\beta E_{x,y} \iff (\beta(x), (E, d), \beta(y)) \in \text{Snd}
\]
Activation, Chart Mode
Activation Condition

AC: x \cdot z = s_1

\begin{align*}
x : & \\
y : & \\
z : & \\
\end{align*}

E

F

\begin{align*}
G
\end{align*}
Universal vs. Existential Charts

\[
x : \quad y : \quad z :
\]

\[
E \quad F \quad G
\]
Prechart
Conditions
Conditions

\[ q_1 \overset{E_{x,y}^!}{\rightarrow} \neg E_{x,y}^! \]
\[ q_2 \overset{E_{x,y}^?}{\rightarrow} \neg E_{x,y}^? \]
\[ q_3 \overset{E_{x,y}^?}{\rightarrow} \neg F_{y,x}^! \]
\[ q_4 \overset{F_{y,x}^!}{\rightarrow} \neg(F_{y,z}^? \lor G_{y,x}^?!) \]
\[ q_5 \overset{F_{y,z}^? \land \neg G_{y,x}^?!}{\rightarrow} \neg G_{y,x}^?! \]
\[ q_6 \overset{G_{y,x}^?!}{\rightarrow} \neg F_{y,z}^? \]
\[ q_7 \overset{F_{y,z}^?}{\rightarrow} \neg G_{y,x}^?! \]
\[ q_7 \overset{true}{\rightarrow} \]

\[ \text{UNIVERSAL:} \]
\[ \mathcal{M} \models \text{LSC} \iff \]
\[ \forall \tau = (s_i, c_0, s_i, t, s_{i+1}) \land B \cdot \]
\[ s_i F_{s_i} \text{ event} \]
\[ \Rightarrow (s_{i+1}, c_0, s_{i+1}, t, s_{i+2}, c_{i+2}, s_{i+2}) \text{ is accepted by } \]
\[ \text{bwt(LSC)} \]
Back to UML: Interactions
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

$$\mathcal{M} = (C_D, S_M, O_D, \mathcal{I})$$

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

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

and

$$\forall \mathcal{I} \in \mathcal{I}_\exists : \mathcal{L}(\mathcal{M}) \cap \mathcal{L}(\mathcal{I}) \neq \emptyset.$$
In UML, reflective (temporal) descriptions are subsumed by interactions.

A UML model $M = (CD, SM, OD, I)$ has a set of interactions $I$.

An interaction $I \in I$ can be (OMG claim: equivalently) diagrammed as

- sequence diagram, timing diagram, or
- communication diagram (formerly known as collaboration diagram).
In UML, reflective (temporal) descriptions are subsumed by **interactions**.

A UML model \( M = (C,D,SM,OM,I) \) has a set of interactions \( I \).

An interaction \( I \in I \) 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


