Contents & Goals

Last Lecture:
- Construction 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 \( f(\tau_1, \ldots, \tau_n) : \tau \) 
  - a signal list \( \langle \langle \text{signal} \rangle \rangle \)
- 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).

Behavioural Features

Semantics:
- The implementation of a behavioural feature can be provided by:
  - As operation.
    - The class’ state-machine (‘triggered operation’):
      - Calling \( f \) with parameters for a \( \text{stable} \) instance of \( C \)
      - Create an auxiliary event \( E \) and dispatch it (leaving the other).
      - Transition actions may fill in the return value.
      - On completion of the RTC step, the call returns.
    - For a non-stable instance, the call blocks until stability is reached again.
  - The class’ state-machine (‘triggered operation’):
    - Calling \( f \) with parameters for a \( \text{stable} \) instance of \( C \)
    - Create an auxiliary event \( E \) and dispatch it (leaving the other).
    - Transition actions may fill in the return value.
    - On completion of the RTC step, the call returns.

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

- Useful properties:
  - concurrency
  - guarded — some mechanisms must/have to 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).
Motivation: Reflective, Dynamic Descriptions of Behaviour

What Can Be Purposes of Behavioural Models?

Example: Pre-Image

(Im UML model is supposed to be the blue-print for a software system).

A description of behaviour could serve the following purposes:

- Requires 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.)
- Allows Behaviour: "System does subset of this"
  "After inserting money and choosing a drink, the drink is dispensed (if it is stock)."
  (If the implementation insists on taking the money first, that's a fair choice.)
- Forbids 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 behaviour:

- 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?

- The semantics of the UML model $M = (V, W, F, P)$ is the transition system $\langle V, \rightarrow \rangle$ constructed according to Ecological Disposition (corresponding rules).
- The computations of $M$, denoted by $\mathcal{A}M$, are the computations of $\langle V, \rightarrow \rangle$.

Now:

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

More formally: a requirement $\phi$ is a property of computations, e.g., which is either satisfied or not satisfied by a computation.

Let $\pi = (x_1, x_2, \ldots) \in \frac{\mathcal{A}M}{2\times}\frac{\mathcal{A}M}{2\times}\frac{\mathcal{A}M}{2\times}$ denote $\pi \models \phi$ if and only if $x_1, x_2, \ldots \in \mathcal{A}M$.

Recall: OCL as Reflective Description of Certain Properties

- Invariants: $\forall x \in \mathcal{A}M \forall y \in \mathcal{A}M : x = y \rightarrow \phi$.
- Non-reachability of configurations: $\exists y \in \mathcal{A}M \forall x \in \mathcal{A}M : x = y \rightarrow \neg \phi$.
- Reachability of configurations: $\exists y \in \mathcal{A}M \forall x \in \mathcal{A}M : x = y \rightarrow \phi$.

where

- $\phi$ is an OCL expression or an object diagram and
- $\phi$ is the corresponding OCL satisfaction or the "is represented by object diagram" relation.

OCL as Reflective Description of Certain Properties

In General Not OCL: Temporal Properties

Dynamic (by example)

- reach behavior
  - For each $C$ instance, each reception of $E$ is finally answered by $F$.

Non-reachability of system configuration sequences

- There mustn’t be a system run where $C$ first receives $E$ and then sends $F$.

Reachability of system configuration sequences

- There must be a system run where $C$ first receives $E$ and then sends $F$.

But: what is $\langle x_{\theta}\rangle$ and what is $\{x_{\phi}\}$?

The Language of a Model

Recall: A UML model $M = (V, W, F, P)$ and a structure $P$ denotes a set $\mathcal{A}M$ of (initial and consecutive) computations of the form:

$\mathcal{A}M : \left\{ (x_1, x_2, \ldots) : \left( \begin{array}{c} x_1, x_2, \ldots \in \mathcal{A}M \end{array} \right) \right\}$

For the connection between models and interactions, we disregard the configuration of the actor and who made the stop, and define as follows:

Definition. Let $\mathcal{M} = (V, W, F, P)$ be a UML model and $P$ a structure. Then:

$\mathcal{C}M = \left\{ (x_1, x_2, \ldots) : \left( \begin{array}{c} x_1, x_2, \ldots \in \mathcal{A}M \end{array} \right) \right\}$

is the language of $\mathcal{M}$.
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 $\mathcal{M} = (\mathcal{B}_V, \mathcal{B}_W, \mathcal{C}_B, \mathcal{C}_S)$ is called consistent (more precisely: the constructive description of behaviour is consistent with the reflective one) if and only if

- $\forall I \in \mathcal{I}_\forall \forall : L(\mathcal{M}) \subseteq L(I)$
- $\forall I \in \mathcal{I}_\exists \exists : L(\mathcal{M}) \cap L(I) \neq \emptyset.$

In the following, we consider Sequence Diagrams as interactions $I$, more precisely: Live Sequence Charts [Damman and Harel, 2001].

We define the language $L(I)$ of an LSC — via Büchi automata. Then (conceptually) $\pi \models \vartheta$ if and only if $\pi \in L(I)$.

Why LSC, relation LSCs/UMLSDs, other kinds of interactions: later.

**Example**

**Building Blocks**

- **Instance Lines:**
  - $\text{LightsCtrl}$, $\text{BarrierCtrl}$, $\text{CrossingCtrl}$

- **Messages:** (asynchronous or synchronous/instantaneous)
  - $\text{lights}$, $\text{on}$, $\text{barrier}$, $\text{down}$, $\text{lights}$, $\text{ok}$,
  - $\neg MvUp$, $\text{done}$

- **Conditions and Invariants:** $\{ \text{expr}_1, \text{expr}_2, \text{expr}_3 \in \mathcal{E} \}$

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

- **Strictly After:**

- **Simultaneously:** (simultaneous region)

- **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 transition closure of (i) to (iii).
Example: Partial Order Requirements

LSC: L
AC: actcond
AM: invariantI:strict

Environment: LightsCtrl
Operational [1, 3]: CrossingCtrl
CrossingCtrl t(10) t:
BarrierCtrl [1, 5] se
creq
lights on
barrier
down
lights ok
barrier ok
¬ MvUp
done

CrossingCtrl
LightsCtrl
BarrierCtrl

LSC Specialty: Modes
With LSCs,
• whole charts,
• locations, and
• elements
have a mode — one of hot or cold (graphically indicated by outline).

Example: What Is Required?
LSC: L
AC: actcond
AM: invariantI:strict

Environment: LightsCtrl
Operational [1, 3]: CrossingCtrl
CrossingCtrl t(10) t:
BarrierCtrl [1, 5] se
creq
lights on
barrier
down
lights ok
barrier ok
¬ MvUp
done

CrossingCtrl
LightsCtrl
BarrierCtrl

• Whenever the CrossingCtrl has consumed a scenario 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 shall 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, during this time dispatch time not included.
• It shall keep an active 'Hold'.
• 'lights ok' and 'barrier ok' may occur in any order.
• After having consumed both, CrossingCtrl replies with 'done' to the environment.

Example: Modes
LSC: L
AC: actcond
AM: invariantI:strict

Example: 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_C), activation mode (AM ∈ {init, inv}), and pre-chart.

Intuition: (universal case)
• given a computation π, whenever edge holds in a configuration (σ, ε) of:
  • which is initial, i.e. δ = 0, or
  • which is not further restricted,
  • and if the pre-chart is obtained from δ, by δ = ν,
then the main-chart has to follow from δ + ν = 1, δ.

Example: What Is Required?
LSC: L
AC: actcond
AM: invariantI:strict

Environment: LightsCtrl
Operational [1, 3]: CrossingCtrl
CrossingCtrl t(10) t:
BarrierCtrl [1, 5] se
creq
lights on
barrier
down
lights ok
barrier ok
¬ MvUp
done

CrossingCtrl
LightsCtrl
BarrierCtrl

• Whenever the CrossingCtrl has consumed a scenario 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 shall 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, during this time dispatch time not included.
• It shall keep an active 'Hold'.
• 'lights ok' and 'barrier ok' may occur in any order.
• After having consumed both, CrossingCtrl replies with 'done' to the environment.
LSC Body: Abstract Syntax

Let \( \Theta = \{ \text{hot, cold}\} \). An LSC body is a tuple

\[ (I, \Theta, \Sigma, \rightarrow, \text{Mag}, \text{Cond}, \text{LocInv}) \]

where

- \( I \) is a finite set of instance lines,
- \( (I, \subseteq) \) is a finite, non-empty, partially ordered set of locations,
- each \( l \in I \) is associated with a temperature \( \Theta(l) \) \( \in I \) and an instance line \( i \in I \),
- \( \rightarrow \subseteq I \times I \) is an equivalence relation on locations, the simulation relation,
- \( \Sigma = (\Sigma, \bot, \rightarrow) \) is a signature,
- \( \text{Mag} \subseteq \Sigma \times \Sigma \) is a set of asynchronous messages
  with \((l, h) \in \text{Mag}\) if only if \( l \neq h \).
- \( \text{LocInv} \subseteq \Sigma \times \Sigma \) a set of synchronous messages
  with \((L, \text{expr}, \theta) \in \text{LocInv}\) if only if \( L = \theta - 1 \) for all \( L \in I \),
- \( \text{LocInv} \subseteq \Sigma \times \Sigma \) a set of local invariants
  \( \exists \text{LocInv} \cdot \theta \) and \( \text{LocInv} \) with \( (l, \bullet, \theta) \in \text{LocInv}\) is a set of local invariants.

Well-Formedness

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

- For each location \( l \in I \), \( l \) is the location of
  - a condition, i.e.
    \[ \exists (l, \text{expr}(\theta), \text{Cond} : l \in I) \]
  - a local invariant, i.e.
    \[ \exists (l, \text{loc}, \text{expr}(\theta), \text{LocInv} : l \in I) \]

- For each location \( l \) equivalent to \( l \) which is the location of
  - a message, i.e.
    \[ \exists (l, \text{msg}, \text{loc}) \in \text{Mag} : l \in I, l \]
  - as instance head, i.e. \( l \) is minimal with respect to \( l \).

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

References