Have:

- **Means to model the structure of the system.**
  - Class diagrams graphically, concisely describe sets of system states.
  - OCL expressions logically state constraints/invariants on system states.

Want:

- **Means to model behaviour of the system.**
  - Means to describe how system states evolve over time, that is, to describe sets of sequences $\sigma_0, \sigma_1, \ldots \in \Sigma^\omega$ of system states.

### What Can Be Purposes of Behavioural Models?

**Example**: Pre-Image (the UML model is supposed to be the blueprint for a software system). A description of behaviour could serve the following purposes:

- **Require Behaviour.**
  - "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.**
  - "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.**
  - "This sequence of getting both, a water and all money back, must not be possible."
  - (Otherwise the software is broken.)
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.
  
  "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.
  
  "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.
  
  "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 Behaviour in 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 (between UML 1.x and 2.x) from transition-system-like to petri-net-like...

• Example state machines:
Brief History:

- Rooted in Moore/Mealy machines, Transition Systems, etc.
- Manifest in tool Statemate Harel et al. (1990) (simulation, code-generation); nowadays also in Matlab/Simulink, etc.
- From UML 1.x on:
  - State Machines (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.
Roadmap: Chronologically

– 10 – 2016-12-01 – Sstmover –

17

Syntax:

(i) UML State Machine Diagrams.

(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.

(xii) Later: Hierarchical state machines.

UML ModelInstances

\[ NSW \text{ E} \quad CD, \quad SM \text{ S} \]

\[ SM \text{ M} = (\Sigma DS, A_\text{S}, \rightarrow SM) \]

\[ \phi \in \text{OCL expr} \]

\[ CD, SD \text{ S}, SD \text{ B} = (Q SD, q_0, A_\text{S}, \rightarrow SD, F SD) \]

\[ \pi = (\sigma_0, \varepsilon_0) \]

\[ \pi = (((\sigma_0, \text{cons}_0, \text{Snd}_0)) \rightarrow \pi) \rightarrow u \]

\[ \pi = (((\sigma_i, \text{cons}_i, \text{Snd}_i)) \in \mathbb{N}) \]

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

Mathematics
We will see later that this choice does no harm semantically.

If not explicitly given, then this one:

If things are missing?

Maps to each state machine a UML model consists of a set

$SM \in C$ of state machines, $S := \{SM \in C \mid \forall \text{action } \in \text{event } \in \text{guard } \in A \} \in C$. State-Machines belong to Classes $C$ associated with a class

$SM \in \text{event} \in \text{guard} \in \text{action}$, $s \in \text{state machine}$. Each UML state machine diagram

$\exists C \cdot \text{action} \in \text{event } \in \text{guard } \in \text{action}$ with

$\in A$. The following holds true where $s$ maps to $S$.
the object-oriented model of the previous example. 


The causality model also (i.e., it is a semantic variation point). 


T owards UML State Machines Semantics: 


Communication actions. 


Objects respond to 


The causality model is quite straightforward: 


The dispatching method by which a particular behavior is associated with a given 


We will see later that this choice does no harm semantically. 


For simplicity, we even assume a bijection, i.e. we assume that each class 


Each instance of class 


Intuition 1 


Intuition 2 


0 


If not explicitly given, then this one: 


Each state machine 


In the following, we assume that 


State-Machines belong to Classes 


Rhapsody Demo II 


6.2.3 The Basic Causality Model 


6.2.3 The Basic Causality Model
Event occurrences are detected, dispatched, and then processed by the state machine, one at a time.

The semantics of event occurrence processing is based on the run-to-completion assumption, interpreted as run-to-completion processing. Run-to-completion processing means that an event can only be taken from the pool and dispatched if the processing of the previous event is fully completed.

The processing of a single event occurrence by a state machine is known as a run-to-completion step. Before commencing on a run-to-completion step, a state machine is in a stable state configuration with all entry/exit/internal-activities (but not necessarily do-activities) completed.
The order of dequeuing is
run-to-completion step
•
Before commencing on a
run-to-completion step
•
The processing of a single event occurrence by a state machine is known as a
run-to-completion assumption

The semantics of event occurrence processing is based on the
run-to-completion step
•
Thus, an event occurrence will never be processed 
[...]

Event occurrences are detected, dispatched, and then processed by the state
machine, one at a time.

The same conditions apply after the
run-to-completion step
•

Run-to-completion processing

is the
run-to-completion step
•
[\text{[IOW,]}]
The
run-to-completion step
\[\text{[IOW,]}\]
The
run-to-completion step
\[\text{[IOW,]}\]

2011b,
OMG
\[2011b,\)
574)
Behavior can be modeled using UML State Machines. UML State Machines are inspired by Harel’s Statecharts. State Machines belong to Classes. State machine behavior follows the Basic Causality Model of UML, in particular:

- Objects process events.
- Objects can be stable or not.
- Events are processed in a run-to-completion step, processing only starts when being stable.

References


