Modeling Events in Object-Process Methodology and in Statecharts

Iris Reinhartz-Berger, Arnon Sturm, Dov Dori
Technion – Israel Institute of Technology
{ieiris@tx, sturm@tx, dori@ie}.technion.ac.il
Outline

- Motivation
- Methods and Issues Examined
- Event Expression Comparison
- Event Implementation
- Comparison and Summary
Motivation

- Complex systems are often reactive, i.e., they continuously respond to external and internal stimuli (events) and may have time constraints.
- When modeling such systems, the designer should be able to determine the system's behavior, as well as its flow of control.
- System modeling methods must express this action semantics in a way that is both formal and intuitive.
Methods and Issues Examined

• We focus on two behavior oriented approaches:
  - **Statecharts** – Harel’s extension of state machines, used in UML
  - **Object-Process Methodology** – an integrated modeling method that unifies the system function, structure, and behavior within one frame of reference.

• We examine:
  - The **types** of supported events
  - How these event types are **integrated** into complete system specifications
  - What are the potential **implications on** the **code** derived from each method.
### Comparison by Event Types 1/2

<table>
<thead>
<tr>
<th>State Entrance</th>
<th>State Exit</th>
<th>Activity Start</th>
<th>Activity Stop</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Statecharts</strong></td>
<td><strong>Statecharts</strong></td>
<td><strong>OPM</strong></td>
<td><strong>OPM</strong></td>
</tr>
<tr>
<td><img src="#" alt="Diagram" /></td>
<td><img src="#" alt="Diagram" /></td>
<td><img src="#" alt="Diagram" /></td>
<td><img src="#" alt="Diagram" /></td>
</tr>
</tbody>
</table>

- **S’ triggers Triggered Process when it enters s3.**
- **S’ triggers Triggered Process when it exits s3.**
- **Invoking Process triggers Triggered Process when it starts.**
- **Invoking Process triggers Triggered Process when it stops.**
Comparison by Event Types 2/2

**Statecharts**

**Condition Fulfillment**

- s1 \(\rightarrow\) s2
  - \text{true}(c)
  - \text{do/TriggeredProcess}

**Condition Violation**

- s1 \(\rightarrow\) s2
  - \text{false}(c)
  - \text{do/TriggeredProcess}

**External Event**

- s1 \(\rightarrow\) s2
  - Event
  - \text{do/TriggeredProcess}

**OPM**

- C
  - \text{true}
  - \text{false}

- C triggers **Triggered Process** when it enters **true**.

- C triggers **Triggered Process** when it enters **false**.

- Event Generator, which is environmental, triggers **Triggered Process**.
State Change Event

State Change

\[ s_1 \quad s \quad s_2 \]

entered (s1) or entered (s2)

\[ s' \]
do/TriggeredProcess

\[ S \quad s_1 \quad s_2 \]

\[ S \text{ triggers Triggered Process when its state changes.} \]
**Activity Timeout Event**

Invoking Process triggers **Min Constraint Handling** when it lasts less than $T_{min}$ and **Max Constraint Handling** when it lasts more than $T_{max}$. 
State Timeout Event

S triggers **Min Constraint Handling** when $s_1$ lasts less than $T_{\text{min}}$ and **Max Constraint Handling** when $s_1$ lasts more than $T_{\text{max}}$. 
Event Generator, which is environmental, triggers Triggered Process with a reaction time of $T_{\text{min}}$ to $T_{\text{max}}$. If Triggered Process starts within less than $T_{\text{min}}$ then Min Constraint Handling is triggered. If Triggered Process does not start within $T_{\text{max}}$ then Min Constraint Handling is triggered.
Event Implementation

- Similar concepts of Statecharts and OPM.

Event Generator, which is environmental, triggers Triggered Process.

Activating Condition can be true or false.

Triggered Process occurs if Activating Condition is true.

```java
int eventGeneratorWrapper() {
    int res = NOT_CONSUMED;
    if (activatingCondition.getStatus()==true) {
        triggeredProcess.activate();
        res = CONSUMED;
    }
    return res;
}
```
Where Does the Behavior Occur?
- In Statecharts the behavior occurs in the states.
- In OPM the behavior is executed in the processes, which act to change the states of objects.

State-Activity Coupling
- In Statecharts there is a clear coupling between a state of an object and the activity performed within it, so it is easy to detect the system behavior within a single state.
- In OPM, system state may be a combination of several objects’ states, making it more difficult to detect the system behavior within a single state.
- Since in Statecharts each object gets its own “chart” it is not clear how the entire system state is modeled.
Event Source
- In Statecharts it is difficult to follow the event sources.
- In OPM, the event generating source is explicitly specified, enabling the designer to trace the events associated with a specific entity (object, state, or process).

Accessibility, Legibility, Expressivity
- In OPM, OPDs use less text and more graphics, and the semantics is made clear by the corresponding natural OPL sentence, an element that does not exist in Statecharts
Summary

- OPM's visual and textual representations provide a means for expressing various event types in a formal yet intuitive way, which compares favorably with Statecharts.
- OPM supports modeling aspects (other than behavior) which are not modeled in the stand-alone version of Statecharts (e.g., communication between objects, system structure, and architecture).