

Modeling Events in Object-Process Methodology and in Statecharts

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

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. One common way for expressing control flows is via Event-Condition-Action (ECA) rules [1]. These rules specify for each action (process) its triggering event and its guarding condition. The action is executed when the triggering event occurs, if and only if the guarding condition is fulfilled at that time. In this paper, we specify how two modeling approaches, Statecharts and Object-Process Methodology (OPM), model the ECA paradigm and compare the expressive power of the respective models. We examine the types of supported events, how these event types are integrated into complete system specifications, and what are the potential implications on the code derived from each one of the specifications.

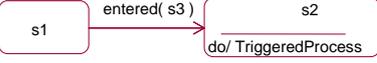
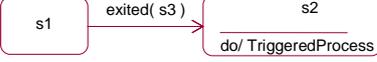
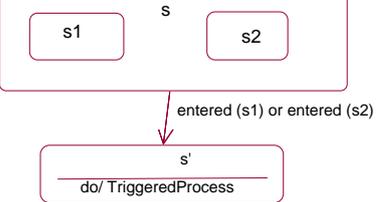
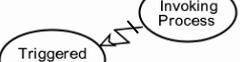
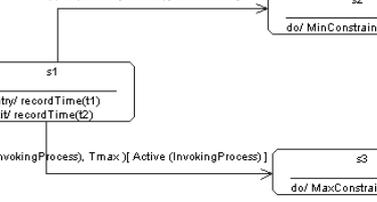
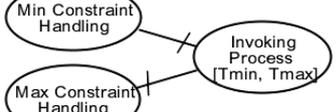
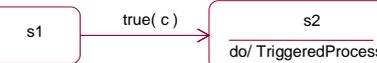
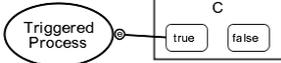
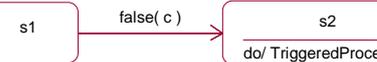
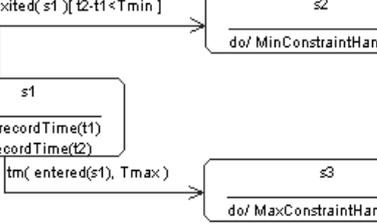
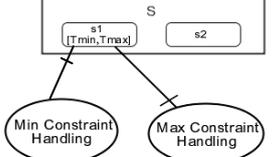
Statecharts [2] is an extension of the conventional formalism of state machines and diagrams. It can be used either as a stand-alone behavioral description or as part of a more general analysis and design method, such as UML [5]. Statecharts is based on states, which specify a situation in which a system (or an object) exists, and transitions, which enable the system to move from one state to another. A transition has the form of an ECA rule, i.e., syntactically formulated as "event [condition] | action".

OPM [3] is an integrated modeling method that unifies the system function, structure, and behavior within one frame of reference. The building blocks of OPM are objects, processes, states, and structural and behavioral links. Each OPM specification consists of a set of graphical representations, called Object-Process Diagrams (OPDs), and a corresponding natural language description, called Object-Process Language (OPL) script. The translation from an OPD set to the corresponding OPL script and vice versa is done automatically, so the designer can interchangeably work on the graphical or textual version of the specification. OPM/T [4], which is an extension of OPM for specifying reactive and real-time systems, has applied the ECA rules to OPM by defining triggering events, guarding conditions, temporal constraints, and timing exceptions. In this paper, OPM refers also to its OPM/T extension.

Table 1 provides a Statecharts model and an OPM model (both an OPD and an OPL sentence) for each one of the common event types. Comparing the models suggested for state entrance, state exit, activity start, activity stop, condition fulfillment, condition violation, and external events, we found no significant differences in the model complexity and accuracy. However, there are some interesting differences that make each method suitable for particular tasks.

- In Statecharts the behavior of the system occurs in the states, and is expressed by text below the line separating the state name and the "do/" command. In OPM, which models structure and behavior in the same model but with different symbols, the behavior is executed in the processes, which act to change the states of objects.

Table 1. Modeling events in Statecharts and OPM

Event Type	Statecharts Model	OPM Model: OPD(top) OPL sentence (bottom)
State Entrance		 <p>S' triggers Triggered Process when it enters s3.</p>
State Exit		 <p>S' triggers Triggered Process when it exits s3.</p>
State Change		 <p>S triggers Triggered Process when its state changes.</p>
Activity Start		 <p>Invoking Process triggers Triggered Process when it starts.</p>
Activity Stop		 <p>Invoking Process triggers Triggered Process when it stops.</p>
Activity Timeout		 <p>Invoking Process triggers Min Constraint Handling when it lasts less than Tmin and Max Constraint Handling when it lasts more than Tmax.</p>
Condition Fulfilment		 <p>C triggers Triggered Process when it becomes true.</p>
Condition Violation		 <p>C triggers Triggered Process when it becomes false.</p>
State Timeout		 <p>S triggers Min Constraint Handling when s1 lasts less than Tmin and Max Constraint Handling when s1 lasts more than Tmax.</p>
External Event		 <p>Event Generator, which is environmental, triggers Triggered Process.</p>

- In Statecharts there is a clear coupling between a state and the activity performed within it, so it is easy to detect the system behavior within a single state. 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).
- 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.
- The state change, activity timeout, and state timeout event types are modeled in OPM more naturally, since they are built into the model. Modeling these events in Statecharts required the definition of composite events, conditions, actions, and synthetic states/transitions. Thus, while modeling the state change event in Statecharts, the redundant state **s** had to be added in order to specify a transition that is enabled whenever one of **s** sub-states is entered. In the timeout events, an additional activity of entrance (**t1**) and exit (**t2**) time recording had to be added in order to be able to compute the time **t2 – t1** spent at state **s1** which is required for the **MinConstraintHandling** activity in case **t2 – t1 < Tmin**. The corresponding maximal time constraint is handled with the build-in timeout mechanism of Statecharts with a complementary condition check.
- OPM supports defining reaction timeout constraints on each one of the event types. A reaction timeout constraint expresses temporal restrictions on the minimal and maximal time that can elapse between the event occurrence and the beginning of the triggered process or activity.

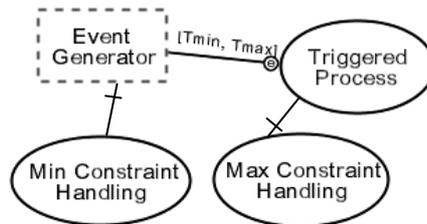


Figure 1. Adding reaction timeout constraints to the external event model

For example, Figure 1 specifies the external event model with reaction timeout constraints. **Triggered Process** should normally begin its execution between **Tmin** and **Tmax** after the **Event Generator** has created an event (e.g., an external stimulus). If this process is about to begin before **Tmin**, then the **Min Constraint Handling** process is activated. Similarly, if **Triggered Process** does not begin after **Tmax**, then **Max Constraint Handling** process is activated. Modeling this constraint in Statecharts is not straightforward.

As the examples in Table 1 demonstrate, OPL reads as natural language and thus it enhances the readability of the graphical models, making it easy for humans who are not familiar with OPM graphical notations and their semantics to interpret the semantics correctly. At the same time, OPL provides a solid infrastructure for automated code generation, which follows the common implementation of ECA rules, when dealing with events. In particular, it separates the ECA components into two groups: the triggering elements and the executing ones. The triggering elements are the event initiator (object, state, or process) and the event specification (denoted by an event link). The executing element is a wrapping process, which executes the original activity if and only if the preconditions are fulfilled. These preconditions can be either for a normal process execution or for the

execution of a time exception handling. We plan to provide OPM with an implementation generator which will follow the Statecharts code conversion rules. Figure 2 shows an example to an OPD that models the general structure of an ECA rule, its OPL counterpart, and the corresponding pseudo code of the wrapping process.

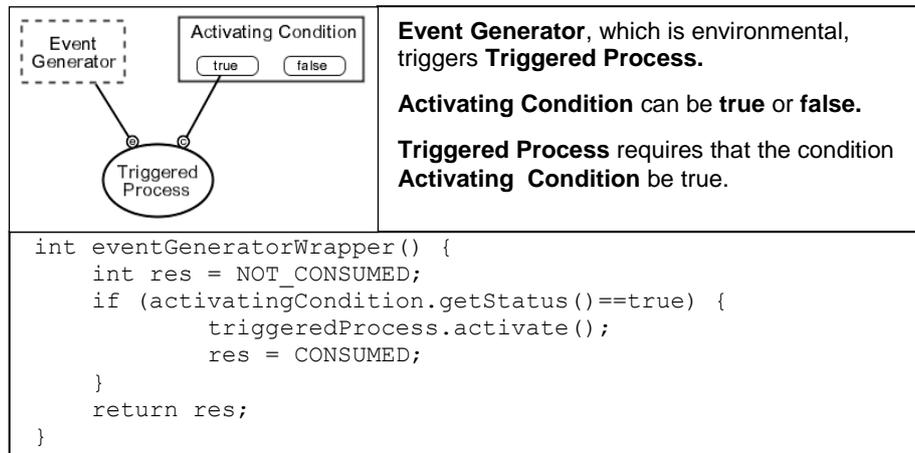


Figure 2. An implementation generation example

In 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. The textual representation of OPM models (i.e., OPL) can be verified by the system customers and compared against the requirements they expressed. At the same time, the OPL script explicitly specifies event implementation concepts and is therefore amenable to automated code generation. In other words, OPL bridges the requirement specification and the implementation stages of the system lifecycle development. As a future work we plan to write an OPL compiler that will generate executable code. It will be able to generate code for the various event types presented in this paper, as well as supporting other modeling aspects that can be expressed in OPM but not in the stand-alone version of Statecharts (e.g., communication between objects, system structure, and architecture).

References

- [1] S. Chakravarthy, SNOOP: An expressive Event Specification Language for Active Databases, Technical Report UF-CIS-TR-93-007, 1993, pp. 1-25.
- [2] D. Harel, Statecharts: A Visual Formalism for Complex Systems, Science of Computer Programming, 8(3), pp. 231-274, 1987.
- [3] D. Dori, Object-Process Methodology - A Holistic Systems Paradigm, Springer Verlag, Heidelberg, New York, 2002.
- [4] M. Peleg and D. Dori, Extending the Object-Process Methodology to Handle Real-Time Systems. Journal of Object-Oriented Programming, 11, 8, pp. 53-58, 1999.
- [5] Unified Modeling Language Specification – version 1.3, 1999,
<http://www.rational.com/media/uml/resources/documentation/ad99-06-08-ps.zip>.