Comparing UML, SCXML and the Apache implementation of SCXML engine

Nicola Migliorini

European Southern Observatory
Garching bei München, Germany

nmiglior@eso.org

Abstract

That of statechart diagrams is a powerful formalism able to represent the behaviour of complex systems. State Chart XML is defined in a working draft from W3C. The Apache Foundation is supporting SCXML providing a working implementation and a standalone engine able to process an SCXML document. UML has adopted the statecharts formalism but as UML is designed for very general purpose many differences are present between OMG definition and the Apache implementation of the SCXML engine. With this premise it is easy to understand that mapping a model from UML to SCXML is not trivial. This document presents a mapping of UML state machine to SCXML. It is not a complete mapping but it will focus on the features defined in the Generic State Machine Engine Software Requirement Specification [1].

1 Introduction

Model Driven Development focuses on developing software components using abstractions of software systems, i.e. models. It is meant to simplify the process of designing the components of the system and the relations between different modules. Models are then converted into code. The conversion can be made manually or automatically, using tools designed for this.

UML is a general purpose modeling language widely used in software development. UML aims to be a standard modeling language which can model concurrent and distributed systems. It is a de facto industry standard, and is evolving under the auspices of the Object Management Group (OMG). Version 2.3 [3] is the reference for this document. Focusing on state machines, the most used formalism is UML statechart, a formalism described in the UML standard and initially defined by D. Harel. UML statechart diagram is an object-based variant of Harel statechart. With statechart diagrams you can describe the behaviour of a system with a quite simple and easy to understand

graphic formalism. Statecharts add to the classical state diagrams a formalism to describe multiple cross-functional state diagrams within a state machine without loosing readability. The formalism offers the possibility to model superstates, pseudostates and activity as a part of a state. In addition UML statechart imports the concepts of hierarchically nested states and orthogonal regions, as defined by Harel, and extends the notion of actions.

State Chart XML (SCXML) is currently a working draft published by the World Wide Consortium [4] which provides rules to describe statecharts model in a XML dialect. SCXML provides a general purpose execution environment based on Harel's Statecharts. Events, transitions and policies to interpret the behaviour of state machine are used to describe complex state machines with features such as sub-states, parallel states, synchronization and concurrency. SCXML is not yet a standard but it is a work in progress. The latest working draft is dated May 2010 [5].

The Apache Foundation is supporting SCXML through the Apache Commons project. Commons SCXML [6] is an implementation of a Java SCXML engine capable of executing a state machine defined in a SCXML document. The latest implementation of Commons SCXML is v.0.9 and is dated December 2008.

Working to develop a tool to translate a UML model to a SCXML document brought to my attention some differences mainly due to the continuous changes in the standards and in the working draft that are not always quickly implemented in the Apache engine. In addition the distance between the release date of the implementation and the latest working draft increases the number of differences. When the working draft from W3C will be ready to become a standard an update of the engine from the Apache Foundation will probably smooth all these details.

2 State Machines, Statecharts and SCXML

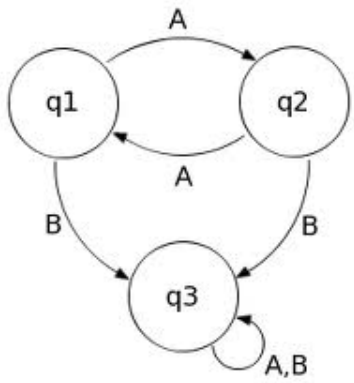

Figure 1: A state machine

A finite state machine (Fig. 1) is a behaviour model representing a reactive system. It consists of a set of states and transitions between them. Every transition is triggered

by an event. Events are the inputs of the systems. In other words events can lead to a different state in which one or several actions can be performed. Actions can be associated with states or with transitions and represent the system's output. All this is represented using simple and easy to read diagrams. These diagrams are similar to a flow graph

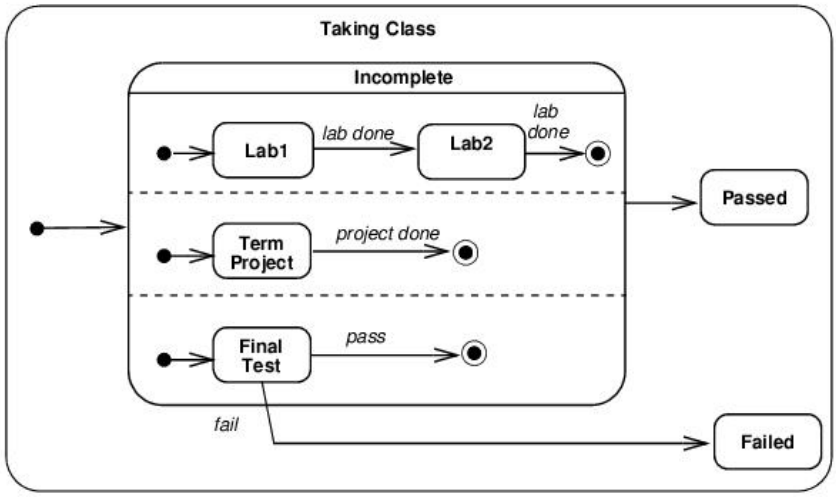

Figure 2: A simple statechart

Statecharts (Fig. 2) were initially defined by D. Harel in his paper *"Statecharts: a visual formalism for complex systems"*. Essentially he introduced in the finite state formalism concepts like hierarchy, orthogonality and broadcast communications. The idea was then adopted from OMG and statecharts are now part of UML standard.

With SCXML you can describe complex state machines using an XML based markup language. In figure 3 there is the classical stopwatch example taken from the Apache Commons [6] website.

On the example the structure of the model is easily recognizable: there are states, with a unique name and inside every state the transitions. Each transition is triggered in response to an event. Following a transition the machine change its state to the targeted one. Models representing simple state machines are easy to read and easy to handle.

Without an engine capable to parse and execute SCXML models these files would be useless. The Apache foundation and the community have developed a java based execution environment. The SCXML distribution provides a java standalone class to test SCXML models from a command line. I set up the environment following the user guide from the Apache website and tested it with the examples provided. Events can be sent and interpreted and also variables can be set. The interface is rather basic but clear enough for testing purpose.

```
<?xml version="1.0" encoding="UTF-8"?>
<scxml xmlns="http://www.w3.org/2005/07/scxml" version="1.0" ↩
   initial="ready">
    <state id="ready">
        <transition event="watch.start" target="running"/>
    </state>
    <state id="running">
        <transition event="watch.split" target="paused"/>
        <transition event="watch.stop" target="stopped"/>
    </state>
    <state id="paused">
        <transition event="watch.unsplit" target="running"/>
        <transition event="watch.stop" target="stopped"/>
    </state>
    <state id="stopped">
        <transition event="watch.reset" target="ready"/>
    </state>
</scxml>
```

Figure 3: The classical watch example

3 Comparison

I started my work from the examples used in Harel's paper. For each example in the article I tried to draw a UML diagram first, and then the corresponding code for a SCXML model strictly following the W3C's draft. After that I started testing the obtained model with the Apache SCXML engine. For each model I had to make small adjustments to fit the syntax expected from the engine. In addition, I adopted some workarounds to obtain desired features not directly supported with the Apache implementation.

– Aggiungere grafici per gli esempi più complessi –

3.1 Simple state

UML	**SCXML**	**Apache**
State	<state>	<state> tag with a unique id attribute

Table 1: State

- A UML state is a condition or situation during the life of an object during which it satisfies some condition, performs some activity, or waits for some event. An object remains in a state for a finite amount of time.
- In SCXML <state> holds the representation of a generic state.

- A working SCXML model for the Apache implementation must have an unique id for each state.

3.2 Initial pseudostate

UML	SCXML	Apache
Initial PseudoState	initial as an attribute or <initial> tag	initial attribute or <initial> tag (mandatory for complex states)

Table 2: Initial pseudostate

- In UML diagrams the initial state is a pseudostate. It has an arrow that point to the initial state. The used symbol is a small solid filled circle. With the same symbol you can specify the default substate of a complex state.
- In SCXML there are two ways to indicate the first state of a state machine:
  - initial as an attribute of the <scxml> tag. The value must be the **unique ID** of the first state (if initial is not defined the default Initial State is the first defined state)
  - For complex states <initial> as child of a <state> tag. <initial> must enclose a conditionless <transition> with a descendent of the parent <state> as target. In complex states also the initial attribute can be used. This attribute is shorthand for an <initial> child with an unconditional transition. Again if an initial sub-state is not defined the first defined sub-state is taken as initial.
- With the Apache implementation an initial state **must** be defined as attribute of <scxml>. Also in complex state a default initial substate **must** be defined. If initial is defined as attribute the processed model will be modified to obtain an equivalent <initial> tag.

3.3 Final pseudostate

UML	SCXML	Apache
Final PseudoState	<final> tag	<final> tag with id attribute

Table 3: Final pseudostate

- In UML the Final State is a pseudostate. When it is reached the region is completed. When all the regions in the state machine are completed the entire state machine is completed. A Final Pseudostate has no exit transition. The symbol used is a circle surrounding a small solid circle.
- In SCXML <final> element is used to indicate the Final State of a compound state (as child of <state>) or of the entire state machine (as child of <scxml> element).
- To get a model working with the Apache engine <final> must have an id attribute with a valid id name as value.

3.4 Entry and exit actions

UML	**SCXML**	**Apache**
Entry action	<onentry> tag	<onentry> tag
Exit action	<onexit> tag	<onexit> tag

Table 4: Entry an exit actions

- In a number of modeling situations, there is the need to dispatch the same action whenever you enter a state, no matter which transition led you there. Similarly, leaving a state, you'll want to dispatch the same action no matter which transition led you away. For doing this UML Entry and Exit Actions can be used.
- In SCXML Entry and Exit actions consists of actions performed as a part of the correspondent <onentry> or <onexit> element.
- No particular differences with the Apache implementation.

3.5 Transition

UML	**SCXML**	**Apache**
Transition	<transition> tag	<transition> tag

Table 5: Transition

- In UML a transition is a relationship between two states indicating that an object in the first state will enter the second state when a specified event occurs and specified conditions are satisfied. On such a change of state, the transition is said to fire. It is expressed in the form:

trigger[guard]/activity

Trigger is the event that fires a transition . The guard is a boolean condition. The activity is some behavior executed during the transition. All parts are optional. Every transition must have at least one source and one target. A "dangling" transition is not allowed.

- In SCXML <transition> element is child of a <state> element and has event (valid values are SCXML events), condition (a boolean expression) and target (a state id) attributes. All attributes are optional. A transition without a target offers an event handler without the side-effect of leaving the recent state (i.e. an internal transition).

3.6 Internal Transition

UML	**SCXML**	**Apache**
Transition	<transition> tag	<transition> tag

Table 6: Internal Transition

- Events can be handled without leaving a State. These internal transitions don't fire the state's entry and exit actions as state is not really left. Note that that's different from transitions whose target is the source state (self transition). In this case the state is left and re-entered and Entry and Exit Actions are triggered.
- In SCXML and Apache an internal transition is mapped as a targetless transition. Such transition acts as an event handler.

3.7 Superstates and substates

UML	**SCXML**	**Apache**
Superstate	<state> or <parallel> A default substate must be defined	<state> or <parallel>
Substate	<state> tag	<state> tag

Table 7: Superstates and substates

A substate is a state that's nested inside another one. A Superstate that has substates is called a composite state. A composite state may contain either concurrent (orthogonal) or sequential (disjoint) substates.

- In UML a substate is rendered just as a state inside another state. Substates may be nested to any level.
- In SCXML composite substates are defined with the <state> element inside a parent <state> element. The default inital state is identified with the <initial> element. If not present the defaut state is the first defined state. Orthogonal states are defined through the <parallel> element. Each orthogonal substate is identified with a <state> element.
- In Apache a default substate MUST be defined using <initial> for both orthogonal and composite states. In addition with <parallel> each orthogonal region of the state represents an auxiliary state wrapping the substates. For this reason each region should be identified with a name when modeling it in MagicDraw . In the current version of Apache Commons SCXML (v0.9) <parallel> can't contain an <history> , <onentry> and <onexit> as childs. A <transition> element as child of <parallel> is ignored. A possible solution is to use a <state> wrapper element around the <parallel> to hold the deep history element, the Entry and Exit actions and the transition. Another limitation imposed by Apache Commons SCXML is about <invoke> element in composite states. A composite state should contain either one <parallel>, one <invoke> or any number of <state> children. That is <invoke> cannot be a child of a complex state unless the state has no substates (i.e. is a simple state)

  — Aggiungere grafici per gli esempi piu complicati —

3.8 History pseudostate

UML	**SCXML**	**Apache**
History pseudostate	<history> with a <transition>	<state> or <parallel> A default substate must be defined
Substate	<state> tag	<state> tag

Table 8: History pseudostate

- In UML diagrams a history pseudostate can remember the last active configuration of the object before leaving a composite state. There are two kind of history: shallow history can remember only the history of the immediate nested states. Deep history can remember the configurations of all the nested states, at any depth. A default state must be defined, to enter when there is no history for the state.
- In SCXML <history> is a child of <state>, must have a unique id and a <transition> element as a child. This transition is conditionless and represents the default history state taken when there is no history for the parent state. If the history is

shallow the target must be an immediate substate, otherwise can be any other descendant state.

–Aggiungere esempi di utilizzo per history –

3.9 Activities

UML	**SCXML**	**Apache**
Activities	<invoke> tag	<invoke> tag

Table 9: Activities

- In UML an object, while in a state, can do some work. DoActivities are activities that can take a finite amount of time and can be interrupted by some events.
- In SCXML Do Activities are called with <invoke>.This element causes an instance of an external service. with <param> element data can be passed to the service.
- The Apache engine can also handle DoActivities via the <invoke> element. An external state machine can be fired and the developper can let this external element deal with different implementations of activities.

4 Summary

Translating a model defined with UML statechart to an Apache working SCXML model is not trivial. The developer must be careful with small implementation details and in some cases special workarounds must be taken. SCXML provide an execution environment that can support all the features defined in the GSME Requirements. These features are:

- Initial and Final pseudo-states
- Composite states
- Orthogonal states
- Guards and IsIn
- Entry and Exit actions
- Actions on transitions
- Shallow and Deep history
- DoActivities

All these features are interpreted correctly by the Apache engine.

It was very interesting to see how things change from the specification to a real and working implementation of the engine. Most of the problems are due to the latest modifications of the working draft and to the old implementation of the Apache engine. In addition, the fact that the W3C specifications are still in the form of a working draft keeps away the developers from investing too much time in making changes that may be not definitive. Certainly when the working draft will take the form of a recomandation a new impulse will hit also the development process of the Apache engine. Hopefully this will smooth a lot of differences and make easier the adoption of this interesting framework.

References

[1] L. Andolfato and G. Chiozzi (2010), *Generic State Machine Engine Software Requirements Specification.*

[2] Harel, David (1987), *Statecharts: a visual formalism for complex systems.*

[3] Object Management Group (2010), *http://www.omg.org/spec/UML/2.3/.*

[4] World Wide Web Consortium (2010), *http://www.w3c.org/.*

[5] World Wide Web Consortium (2010), *http://www.w3.org/TR/scxml/.*

[6] Apache Foundation (2010), *http://commons.apache.org/scxml.*