**General Differences Between SCXML and UML**

**Bran Selic**, January 16, 2015

The general environment of SCXML state machines is completely different from that of UML state machines. In SCXML, a state machine is an independent self-contained entity that can interact with other, or *external*,entities located in a Web-based environment (e.g., external entities are accessed via URIs). External entities can be other SCXML state machines or any other kind of Web-based application. This interaction can be either synchronous or asynchronous. In other words, unlike UML, in SCXML there is no explicit structural context defined in which a state machine is defined; i.e., the (single) state machine *is* the top-level concept.

The data model, event model, and action language of SCXML are all specific to it and not equivalent to those of UML, although most action language (*executable content*) elements can be mapped to standard programming language equivalents.

**SCXML State Machines vs. UML State Machines**

It is fairly clear that SCXML was inspired in great part by UML. There is a significant conceptual, semantic, and even terminological overlap. However, there are also some important differences.

**Syntax**

SCXML only supports three kinds of pseudostates: initial, deep history, and shallow history. This means that it does not support exit and entry points, fork and join points, or terminate pseudostates. Also, there is no support for submachine states or state machine redefinition. Finally, protocol state machines are also not supported.

An SCXML document defines a single state machine as a set of states and a set of associated data (i.e., its “extended” state). States own their outgoing transitions and can be either atomic or compound. The latter own other states (substates). A special kind of state is a parallel state, which is the conceptual equivalent of a UML region. Any state, whether atomic or compound, can have entry and exit actions. Transitions can have zero or more triggering events and an optional guard condition. States of any kind can optionally own local sets of data, something that has no equivalent in UML. Finally, the UML concept of “do” behaviors associated with states is not supported in SCXML.

Like UML, SCXML supports completion event, but, in contrast to UML, these events are named and used to explicitly trigger completion transitions. However, their effect is the same, although it appears that SCXML gives the modeler more control over the triggering because individual completion events can be differentiated. For example, it is possible to define a trigger that refers explicitly to the completion of a particular state. This cannot be done in UML since completions event triggers are implicit.

**Semantics**

From an initial cursory analysis, it seems that the semantics of those SCXML concepts that have UML equivalents are compatible with UML semantics. If we set aside the major differences in the structural context mentioned above, it is fair to say that the semantics of SCXML state machines are a subset of those found in UML.

**Summary of differences**

1. No structural context

2. SCXML specific action language

3. Event model is different

4. Does not support all pseudostates (join, fork, terminate, exit/entry points not supported)

5. No support for submachines

6. States can own local extended data variables

7. No support for “do” behaviors

8. More refined model of completion events