SAND2016-6605C

Reconciling SCXML Statechart Representations and Event-B Lower Level Semantics

Karla Morris[1] and Colin Snook[2]

[1] Sandia National Laboratories, Livermore, California, U.S.A.
knmorri@sandia.gov
[2] University of Southampton, Southampton, United Kingdom
cfs@ecs.soton.ac.uk

Abstract

High consequence systems benefit from verification by formal proof. However, to facilitate efficient automatic proof, notations are often restricted to simplify the verification of state changes. Engineers that are used to richer semantics may find these restrictions difficult to accept. Here we investigate the reconciliation of a Harel style statechart semantics with a declarative, substitution-based notation designed to facilitate proof. We provide a translation of State Chart eXtensible Markup Language (SCXML) statecharts into the Event-B formalism via the iUML-B state-machine notation.

1 Introduction

Formal verification of high consequence systems requires the analysis of formal models that capture the properties and functionality of the system of interest. Proof obligations for systems' properties or requirements can be made more tractable using refinement, where properties are expressed in terms of variables that are introduced at different abstraction levels.

A hierarchical development of a system model uses refinement concepts to link the different levels of abstraction. Each subsequent level increases model complexity by adding details in the form of functionality and implementation method. As the model complexity increases in each refinement level, tractability of the detailed model can be improved by the use of a graphical representation, with rich semantics that can support an infrastructure for formal verification.

The Event-B language [1] provides the logic and refinement theory required to formally analyze a system model. The open-source Rodin tool [2] provides support for Event-B including automatic theorem provers. iUML-B [4] augments the Event-B language with a graphical interface including state-machines.

The goal of this work is to create a unified model representation capable of leveraging the structure and hierarchy that is inherently part of a statechart diagram, which will serve to enable the formal verification of requirements in two stages. First, the translation of the unified representation to Event-B. Second, the analysis of requirements related to the structure of the statechart itself, which is a higher level representation of the model than the Event-B provides.

We base this unified statechart model representation on the State Chart eXtensible Markup Language (SCXML) [6]. This is a general-purpose event-based state machine language that combines concepts from Call Control eXtensible Markup Language (CCXML) and Harel State Tables. Harel State Tables are included in UML. The concrete syntax for SCXML is based on XML. Hence, SCXML is an XML notation for UML style state-machines extended with an action language that is intended for call control features in voice applications.

In § 2 we describe the semantics differences between SCXML and iUML-B and discuss their reconciliation. § 3 presents the extensions made to SCXML for required Event-B features, and § 5 provides some details regarding the development of the translation tool, finally § 5 list some of our future work.

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration (NNSA) under contract DE-AC04-94AL85000.

2 Semantic Differences and their Reconciliation

SCXML and iUML-B have syntactic similarities in their use of a hierarchical state-transition notation with conditional transitions applying actions upon ancillary variables. However, their semantics have significant differences. SCXML, based on Harel statecharts, has so-called 'run-to-completion' semantics, whereas iUML-B follows the 'guarded action' semantics of Event-B. In this section we discuss the implications of this semantic difference with respect to translation between the two notations.

Transition firing: There are three methods of initiating transitions in SCXML:

- 'When' transitions are considered for execution if their source state is active and their *cond* attribute evaluates to true. This is similar to iUML-B transitions, which fire spontaneously when their guard (including source state) is true. In iUML-B if several transitions are simultaneously enabled one of the enabled transitions is non-deterministically chosen for firing whereas SCXML has ordering rules to determine which transition to fire next.
- A transition may be 'triggered' by the occurrence of an external interface event. This could be simulated in iUML-B by generating a flag to represent the trigger and adding a transition guard on the state of the flag. The flag should then be reset by the transition that is triggered in order to 'consume' that trigger event. A special interface event that sets the flag would be generated to represent the external interface receiving a trigger.
- Transitions may also trigger each other within their actions. This could be simulated in iUML-B by a similar mechanism to the trigger events except that the interface event is not needed since the flag is set directly by another transition.

Run-to-completion semantics: SCXML has a run-to-completion (aka big-step/little-step) semantics. This means that an external trigger is only consumed when no transition can be taken without doing so. This is quite cumbersome to implement in iUML-B since it requires constructing the conjunction of the negated guards of all the transitions that are internally triggered (including 'when' transitions) and adding this to all externally triggered transitions.

Composition of execution actions: When a particular SCXML transition fires it carries out a sequence of actions in a well-defined predictable order. For example, a hierarchy of nested source states are exited (performing their exit actions sequentially) starting from the innermost one and working outwards. The order of execution is significant when some of these actions write to, or use the value of, a previously written variable. In Event-B, all actions of a transition are executed simultaneously in parallel by the elaborated event. It is not possible (i.e. not well-formed) for two of these actions to write to the same variable. If any actions read the value of a variable, the value is the value before the transition started being executed.

Events: The meaning of event is different between iUML-B and SCXML. In iUML-B, transitions are sub-parts of events. In order for an event to be enabled for firing, all of its sub-parts (transitions) must be simultaneously enabled. This means that two different transitions with the same event can only fire at the same time and hence will never fire if they are sourced from different states of the same parent state-machine. In SCXML, events are triggers that enable transitions to fire. If two different transitions from different source states are both triggered by the same event, one may fire without the other if one source state is not active.

Final States: The concept of a final state differs between iUML-B and SCXML. In SCXML a state machine (or parent state) may reside in a final state indicating that it is done and waiting for another transition to exit the parent state. In iUML-B a final state is not a proper state of the parent state-machine. It is merely a notation for indicating that the state-machine is becoming non-active, i.e. the parent state is exiting.

Initial States: Initial states are similar in both notations. The transition from the initial state forms part of the actions to enter the parent state. However, the correspondence between incoming transitions to the parent state and initial transitions is more explicit in iUML-B. SCXML has another way to specify an initial state using an attribute of the state. In this case there is no way to add extra transition actions. iUML-B allows different initial states for different incoming transitions. In SCXML this would be done by extending the transition into the substate which, in iUML-B is also an optional alternative to the multiple initial states method.

Entry/Exit Actions: SCXML and iUML-B both include the concept of entry and exit actions which are executed whenever a transition enters or exits the containing state. However, their use in iUML-B is restricted by the lack of sequential composition in Event-B. For example, if an exit action of a state, **S**, assigns to a variable, **V**, then no transitions from **S** are allowed to assign to **V** either directly or via entry actions of their target state. We restrict the SCXML models that can be translated so that executing the actions in parallel is equivalent to executing them in sequence. Effectively this means that the same variable cannot be assigned more than once in any set of actions that will be taken when a transition fires. The Event-B static checker will raise an error if this restriction is violated.

Refinement: Refinement is a central concept of Event-B where detail is built up in stages facilitating validation of abstract concepts before introducing complexity. In iUML-B state-machines, refinement is achieved by adding nested state-machines to existing states. There is no refinement in SCXML. The entire system is introduced in one hierarchical statechart. We provide an extension to SCXML (section 3) so that the target refinement level of a translated SCXML element can be specified.

3 Extending SCXML

To facilitate Event-B formal verification, extensions to the SCXML modelling notation are necessary so that additional modelling features required by Event-B can be integrated with the SCXML model. An example statechart model is shown in figure 1, and a section of the corresponding SCXML syntax is shown in figure 3. The iUML-B representation of the model is shown in figure 2. We use this example to illustrate different points throughout the manuscript. The SCXML schema allows extension elements and attributes belonging to a different namespace to be added. The SCXML tooling provides fallback mechanisms so that these extensions are supported without the need for syntactic definition. We define a new namespace, *iumlb* and add two new elements, *iumlb:invariant* and *iumlb:guard* as well as a number of new attributes which are shown in Table 1. Invariants are not supported in SCXML but are needed to describe verifiable properties of a model. SCXML transitions only have a single *cond* attribute whereas we need to introduce conjuncts of a transition condition at various refinement steps. We also need to be able to designate some invariants or guards as theorems that can be derived from the preceding conjuncts. New attributes are introduced to support the predicate (string) and

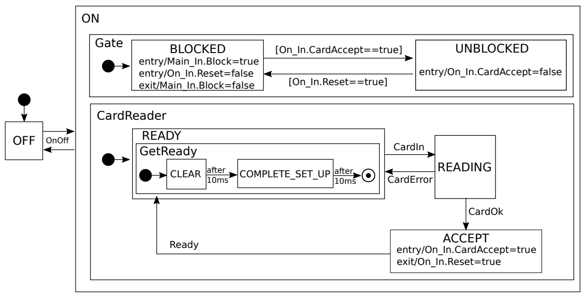

Figure 1: SCXML Statechart diagram

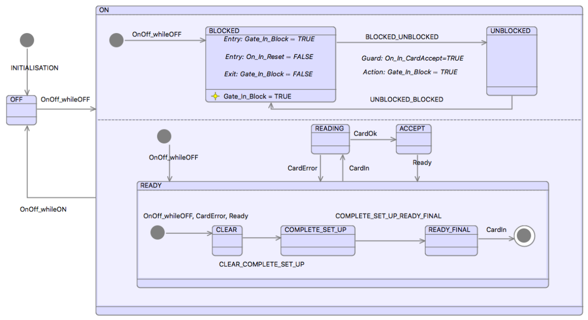

Figure 2: State-machine diagram in iUML-B at refinement level 3 (partially annotated with guards and actions)

```
<iumlb:invariant iumlb:refinement="1" predicate="TRUE = TRUE" name="inv_top_level"/>
<datamodel iumlb:refinement="2">
  <data expr="false" id="Gate_In.Block" iumlb:type="BOOL"/>
</datamodel>
<!-- Other model details -->
<state id="BLOCKED">
  <transition cond="[On_In.CardAccept==true]" target="UNBLOCKED">
    <iumlb:guard name="gd1" predicate="On_In.CardAccept==true" refinement="2"/>
    <assign expr="true" location="Gate_In.Block" iumlb:refinement="3"/>
  </transition>
  <onentry>
    <assign expr="true" location="Gate_In.Block"/>
    <assign expr="false" location="On_In.Reset"/>
  </onentry>
  <onexit>
    <assign expr="false" location="Gate_In.Block"/>
  </onexit>
  <iumlb:invariant predicate="Gate_In.Block == TRUE" name="GateCondition"/>
</state>
```

Figure 3: Part of SCXML model corresponding to figure 1 (including iumlb extension elements explained in section 3)

the derived (boolean) theorem property of invariants and guards. The concept of refinement is not supported in SCXML. We introduce a new integer valued attribute, *iumlb:refinement*, which may be attached to any element of either namespace in order to specify the refinement level of that element.

Figure 3 shows a state, *BLOCKED*, containing a transition that owns an *iumlb:guard*. The guard reflects the *cond* attribute of the transition and is introduced at refinement level 2. The state also owns an *iumlb:invariant* with the predicate *Gate_In.Block == TRUE*.

4 Translation Tool

The iUML-B tooling is based on the Eclipse Modelling Framework (EMF) [5]. It is therefore beneficial to load the SCXML model into EMF so that our existing model transformation

Table 1: SCXML Extension Attributes

Attribute name:	Meaning	Allowed Parents
label	string used as the name of an Event-B event elaborated by the generated i-UML-B	scxml:transition
refinement	non-negative integer representing the refinement level at which the parent element should be introduced	scxml:scxml, scxml:datamodel, scxml:data, scxml:state, scxml:parallel, scxml:transition, scxml:onEntry, scxml:onExit, scxml:assign, iumlb:invariant, iumlb:guard
type	string used as the membership set for the Event-B variable generated from the parent data element	scxml:data
name	string used for the name or label of a generated iUML-B element	iumlb:invariant, iumlb:guard
predicate	string used for the predicate of a guard or invariant	iumlb:invariant, iumlb:guard
derived	boolean indicating that the guard is a theorem (default to false)	iumlb:invariant, iumlb:guard

technology can be used to implement the SCXML to iUML-B translation. An EMF metamodel for SCXML is available from the Sirius [3] project. It supports SCXML functionality, and provides generic model loading capabilities for new namespace extensions such as those we describe in section 3.

Hierarchical nested statecharts are translated to similar corresponding state-machine structures in iUML-B. Figures 1 and 2 illustrate this correspondence between diagrammatic elements.

4.1 Refinement Levels

An *iumlb:refinement* attribute is used to indicate the first refinement level at which an element should be introduced in the generated iUML-B/Event-B model. In general, elements with no refinement attribute adopt that of their parent. However, for *scxml:state* elements, the refinement level refers to the complete state machine(s) generated from any nested child states irrespective of whether those children specify a different refinement level. This is because generated iUML-B states cannot be added to an existing state-machine in later refinements. For *iumlb:invariants* the corresponding Event-B invariant is only generated at the specified refinement level, not in subsequent refinements. This is because Event-B invariants are visible through all subsequent refinements.

Note that our approach to refinement in SCXML largely restricts us to superposition refinement where entirely new details are introduced at each refinement level. It may be possible to support 'ranges' in the refinement attribute, enabling a model element to be replaced by some other element in a true data refinement. We plan to investigate this in future work, although several coexisting alternative representations of the same concept may be problematic for the

SCXML semantics.

4.2 Constructing events elaborated by transitions

The Event-B events that are elaborated by an iUML-B transition are named as follows:

1. If the transition has *iumlb:label* attributes, events are generated and named according to the label attributes.
2. If the transition's source is an initial state at the outer state chart level the transition elaborates the special Event-B INITIALISATION event.
3. If the transition's source is an initial state of a nested state chart the names of all the events that are associated with incoming transitions to the parent state are used.
4. If none of the above provide any labels, a default 'source_target' format is used.

Trigger events are deliberately not used for transition events because we want to keep them as a separate concept from transition firing in line with SCXML semantics.

4.3 Data elements

Data elements collated, in *scxml:datamodel* elements, model the ancillary variables in the usual SCXML style. Data elements are translated to Event-B variables of type given in an *iumlb:type* attribute translated into an Event-B subset invariant. An example of this SCXML representation of an Event-B invariant can be seen on line 3 in figure 3. The *scxml:id* attribute of the *scxml:data* element is interpreted as the name of the variable and the value is used as the right hand side of an assignment action to initialise the variable.

5 Future Work

We have imported SCXML models into Event-B to bring the strong verification methods of Event-B to SCXML models. However, the significant semantic differences between SCXML and Event-B meant that we were forced to restrict the kinds of SCXML models we can support. In particular the run-to-completion semantics of SCXML are difficult to support in Event-B without an explosion of artefacts to model the necessary sequentiality. In future work we will investigate the impact of the restrictions we have imposed on SCXML and whether additional support can be provided to lift or ease such restrictions. We will implement a return translation from iUML-B to SCXML to support round trip model development.

References

[1] J-R. Abrial. *Modeling in Event-B: System and Software Engineering.* Cambridge University Press, 2010.

[2] J-R. Abrial, M. Butler, S. Hallerstede, T.S. Hoang, F. Mehta, and L. Voisin. Rodin: An open toolset for modelling and reasoning in Event-B. *Software Tools for Technology Transfer*, 12(6):447–466, November 2010.

[3] Eclipse Foundation. Sirius project website. https://eclipse.org/sirius/overview.html, 2016.

[4] C. Snook. iUML-B statemachines. In *Proceedings of the Rodin Workshop 2014*, Toulouse, France, 2014. http://eprints.soton.ac.uk/365301/.

[5] D. Steinberg, F. Budinsky, and E. Merks. *EMF: Eclipse Modeling Framework.* Eclipse (Addison-Wesley). Addison-Wesley, 2009.

[6] W3C. SCXML specification website. http://www.w3.org/TR/scxml/, 2015.