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

BLA BLA

1 Introduction

The formal verification of high consequence systems requires the analysis of formal models that capture the properties and functionality of the system of interest. Discharging proof obligations for systems' properties or requirements can be made more tracktable depending on the abstraction use to create the model, as properties are expressed in term of variables that are relevant at different abstraction levels.

A herarchical development of a system model makes use of refinement concepts to link the different levels of abstraction. Each subsequent level increases model complexity by adding implementation details to the model in the form of functionality, capabilities and finner requirements. As the model complexity increases in each refinement level tractability of the model can be improve 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 require to formaly analyze a system model. The open-source Rodin tool [2] auments the Event-B language by providing a graphical interphace in the form of iUML-B. The goal of this work is to create a unified model representation capable of leveraging the structure and herarchy that is inherently part of a statechart diagram, which will serve to enable the formal verification of requirements from two different perspective. First, the translation of the unified representation to Event-B. Second, the analysis of requirements related to the structure of the statechart it self, which is a higher level representation of the model.

We base this unified statechart model representaiton on SCXML. This is a general-purpose event-based state machine language that combines concepts from CCXML and Harel State Tables. Harel State Tables are included in UML. The concrete syntax for SCXML[1] 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.

An example of SCXML syntax is shown in Figure 1. A graphical representation of this example is shown in Figure 3. We use this example to illustrate points throughout the paper.

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. Figures 3 and 4

[1] http://www.w3.org/TR/scxml/

illustrate this correspondence between diagrammatic elements. However, their semantics have significant differences. SCXML is based on Harel state-charts with so called, 'run to completion' semantics, whereas iUML-B is designed upon the 'guarded action' semantics of Event-B. In this section we discuss the implications of this semantic difference with respect to translation between the 2 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 transitions 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 guard on the trigger flag to the transitions that are triggered by it. The flag should then be reset by whichever transition is triggered by it 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 modelled 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 use the value of a variable that is being written, the value is the value before the transition started being executed.

SCXML transitions can be designated 'internal' which prevents exiting and re-entering its source state in some cases. In SCXML, target state can be omitted which results in a transition that does not change state (this is different from a transition that exits a state and then re-enters the same state). Neither of these features are supported in iUML-B. A transition must have a target state and if it is the same as the source state the transition performs any exit and entry actions of its source/target state when it fires.

Events: The meaning of event is very 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. that the parent state is exiting. Hence any transitions that target a final state are part of a transition that leaves the parent state. For a 'root' state-machine, the final state means that the state-machine has been left completely and no state is active.

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. If no initial state is specified, the default is the first state in the document. 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, resp. 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 a 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.

Another difficulty arises when a transition exits a parent state without specifying any particular nested sub-state. Strictly, only the exit actions of the currently active sub-state should be executed. However, this would be difficult in iUML-B due to the lack of any conditional execution. How do we deal with this problem?

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 state-chart. We provide an extension to SCXML (Section 3) so that the target refinement level of an 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. The SCXML schema allows extension elements and attributes belonging

```
<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>
```

(a)

```
<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>
```

(b)

Figure 1: SCXML model representation: (a) invariant declaration (b) state and related transitions

```
variables
  GateIn_Block

invariants
  @typeof_Block GateIn_Block \in BOOL
  @GateCondition gate = BLOCK => GateIn_Block = TRUE

events
  event INITIALISATION  extends INITIALIZATION
    then
      @init_Block GateIn_Block = false
  end

  event BLOCKED_OUT_TRANS
    where
      @isin_BLOCKED gate = BLOCKED
      @check_CardAccept OnIn_CardAccept=TRUE
    then
      @enter_UNBLOCKED gate := UNBLOCKED
      @set_CardAccept OnIn_CardAccept := FALSE
      @set_Block GateIn_Block := FALSE
  end
```

Figure 2: Event-B translation

Table 1: New Elements

Element in iumlb:	Meaning	Legal Attributes in iumlb:
invariant	generates an invariant in Event-B or iUML-B	name, derived, refinement, predicate, comment
guard	generates a guard in Event-B or iUML-B	name, derived, refinement, predicate, comment

to a different namespace to be added. The tooling (both XML and EMF) 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* (Table 1) as well as a number of new attributes which are shown in Table 2. 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 and may also need 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 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.

Examples of iumlb extensions can be found in Figure 1.

4 Translation Tool

The iUML-B tooling is based on the Eclipse Modelling Framework (EMF). It is therefore beneficial to load the SCXML model into EMF so that our existing model to model transformation technology can be used to define the new SCXML to iUML-B translation. An EMF meta-model for SCXML is available from the Sirius [3] project. It supports SCXML functionality as well as providing generic model loading facilities for new namespace extensions such as those we introduce in 3.

Hierarchical nested state charts are translated to similar corresponding state-machine structures in iUML-B. There are two alternative styles of Event-B representation for iUML-B state-machines. Currently the state-variables style is adopted because it is simpler to translate from the SCXML model. However, the alternative state-enumeration style has benefits in user readability and may be supported in future. This would require conventions regarding the name of the state-machine to be adopted and used by the modeller in order to construct guards that refer to the current value of the state-machine. SCXML features, such as initial states, entry/exit actions, and transition actions have corresponding similar features in iUML-B and their translation is relatively straightforward. Since SCXML 'final' states, unlike iUML-B final states, are more akin to real states, their translation is less straightforward. An iUML-B state, final state and transition from the former to the latter are added to the corresponding iUML-B state-machine. The transition elaborates all Event-B events that are elaborated by transitions that exit the parent iUML-B state.

Table 2: New Attributes

iumlb Element	Meaning	scxml Legal Parent
iumlb:label	string used as the name of an Event-B event elaborated by the generated i-UML-B	scxml:transition
iumlb: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
iumlb:comment	string used as a comment on the generated iUML-B element	iumlb:invariant, iumlb:guard, (could be added to more)
iumlb:type	string used as the membership set for the Event-B variable generated from the parent data element	scxml:data
iumlb:name	string used for the name or label of a generated iUML-B element	iumlb:invariant, iumlb:guard
iumlb:predicate	string used for the predicate of a guard or invariant	iumlb:invariant, iumlb:guard
iumlb:derived	boolean indicating that the guard is a theorem (default to false)	iumlb:invariant, iumlb:guard

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 *iscxml:state* elements, the refinement level refers to any state machines generated from the children nested in that state 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 generated 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, clearly, 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 *iscxml:datamodel* elements, model ancillary variables in the way 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. The *iscxml:id* attribute of the *iscxml: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. Some syntax conversion is performed to convert the predicate from SCXML format into the Event-B mathematical language. The variable is introduced at the same refinement level as the parent element that contains it.

5 Case Study

Find a system that we can model and some how describe the benefits (e.g. model simplification) of using a specific syntax over the other.

- What model behavior can you capture with each semantic?
- What properties of the model are easier to simulate?
- Where do we introduce unnecessary complexity?

6 Conclusion

Just to have a reference [**?**]

7 Future Work

8 Acknowledgments

References

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

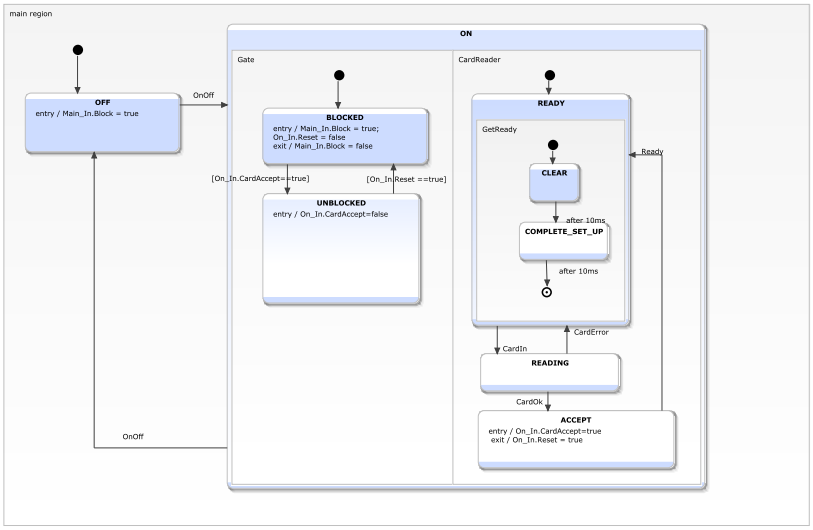

Figure 3: SCXML Statemachine diagram

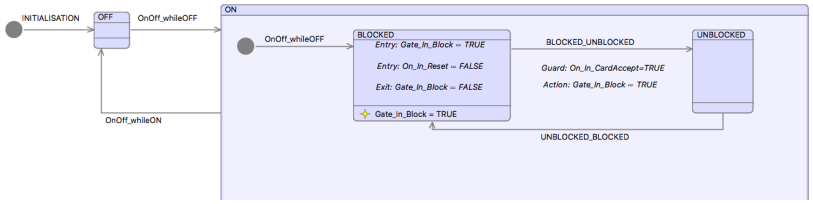

Figure 4: Part of State-machine diagram in iUML-B

[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.