iUMLB\_SCXML

# Introduction

iUMLB\_SCXML is a Rodin plug-in that imports/exports state-charts conforming to the SCXML meta-model to/from iUML-B.

<http://www.w3.org/TR/scxml/>

SCXML is a “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 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.

# iUML-B versus SCXML in brief

## Refinement:

Refinement is a central concept in iUML-B, detail is added in refinements by progressive hierarchical nesting. There is no refinement in SCXML.

## Transition firing:

SCXML has equivalent hierarchical state constructs to iUML-B but differs significantly in the transition firing mechanism. In iUML-B transitions fire spontaneously when their guard (including source state) is true. In SCXML, transitions are *triggered* by the occurrence of some other *event,* which may be external or induced by the actions of another transition.

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.

## Transition execution:

When a particular SCXML transition fires it carries out a sequence of actions in particular order. For example a hierarchy of nested source states are exited (performing exit actions) starting from the innermost one and working outwards. Presumably the order of taking actions is significant in SCXML (perhaps some of these actions could write to the same 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.

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 is supported in iUML-B since currently entry/exit actions have not been supported.

## 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 to iUML-B. 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 one in the document.

iUML-B allows different intial states for different incoming transitions. In SCXML this would be done by extending the transition into the substate.

## Entry/Exit Actions:

SCXML includes the concept of entry and exit actions which are executed whenever a transition enters, resp. exits, the containing state.

iUML-B does not current implement entry/exit actions, but it is has been planned to add them. This could be done as part of the work to support the iUMLB\_SCXML plugin, however, due to the extensive semantic differences between iUML-B and SCXML, it may be found that iUML-B entry/exit actions are not useful for supporting SCXML and a different solution is necessary.

## Execution Language:

SCXML has an execution language for writing actions (in entry, exit and transitions). The following constructs are available:-

**raise:** creates an internal trigger event.

**send:** create an external trigger event to be sent to another system (or sends data)

**cancel:** cancels an event that was to be sent

**log:** display message or generate log

**script:** execute a script

**assign:** modify a data value

**if/elseif/else:** conditional execution

**foreach:** iterate over a collection

# Modelling SCXML semantics in iUML-B

Model trigger events as a queue called a trigger queue. (How can we avoid the refinement problem? I.e. the trigger queue is needed from the first refinement but new transitions in refinements need to write to it.)

30/9/15 : Do we need a queue? Maybe it is sufficient to abstract this out of the system and interface with one trigger of each kind at a time. Use non-deterministic selection if more than one kind can be consumed at a time.

To model the big-step/little-step semantics of SCXML we will need to distinguish between external triggers and internal triggers. Probably the best way to do this is to maintain 2 separate queues.

Triggers need to be consumed when they are used to fire a transition.

How can we model the ordering semantics of SCXML?

Model final states as ordinary states with a naming convention e.g. S1Final, where S1 is the parent state. Such states have an entry action to generate a *done* trigger in the internal trigger queue.

Transition elaboration is always one-to-one because SCXML transitions never synchronise. (Except iUMLB initial/final transitions, if used, as these are not transitions in the SCXML sense).

To replicate transition execution action ordering, it would be necessary to define a different event for each action group (e.g. a separate event to perform the exit actions of each state) and then, when a transition is triggered, to initiate the correct sequence of events corresponding to the SCXML ordering of actions.

Regarding execution language, trigger event operations (raise, send, cancel) can be modelled via the trigger data and queues etc. Logging messages can be handled in an abstract way. Assignment is not a problem in Event-B (except for the restriction that a variable can only be assigned once during an event). Executing a script is problematic in Event-B since there is no sequential operator. This can only be done via a sequence of events and explicitly modelling a program counter (or by modelling the script as a state-machine with a transition for each step of the script). Similarly conditional or iterative execution is not directly supported and would have to be explicitly modelled by program counter or statemachine.

# Turnstile Example (ADDED 1/10/15)

A small example is used to investigate issues involved in translation from SCXML to iUML-B. The example was provided by Sandia.

## Turnstile Example in SCXML (ADDED 1/10/15)

Figure 1 shows an example of an SCXML state chart.

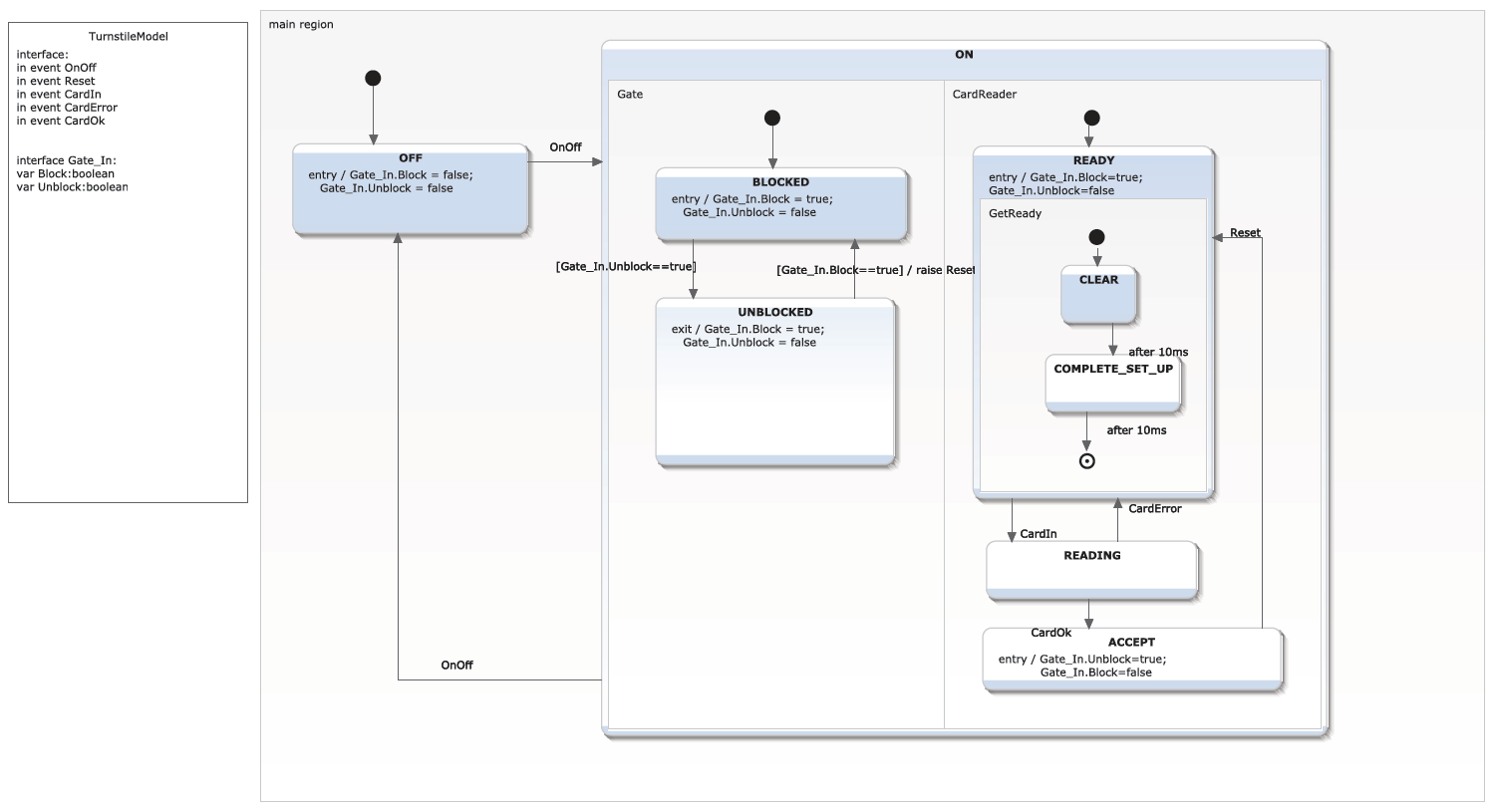


Figure Turnstile Example in SCXML

[Since this Figure was provided, some modifications have been made to this example as follows: The Entry/Exit actions have been changed so that State BLOCKED sets and clears the Gate\_In\_Block flag on entry/exit respectively. Similarly the UNBLOCKED state deals with setting and clearing the Gate\_In\_Unblock flag. The guard on the transition to UNBLOCKED has been changed to CardReader =ACCEPT. The Reset trigger has been moved to the transition to BLOCKED and a new internal trigger, Ready, has been introduced, which is raised by this transition to trigger the transition from ACCEPT to READY.]

While the basic structure of states with nested Statemachines and transitions is the same as that in iUML-B, there are several semantic differences that make translation into iUML-B difficult. We note some features with possible solutions to the problem.

1. Entry and Exit actions. These can be added to iUML-B quite easily however, the lack of sequential composition in Event-B (hence iUML-B) means that the semantics of entry/exit actions will differ in some scenarios. That is, in SCXML the source state’s exit actions are taken before the transition’s actions, which are before the target state’s entry actions. In iUML-B all the actions are taken in parallel, as there is no concept of execution order within an event.
   1. One option would be to allocate each action to a different event and force the events to follow an ordering by introducing an explicit program counter/flag.
   2. A second option is to analyse the sequence of actions and produce a set of parallel Event-B substitutions that is equivalent to the final outcome of the sequence.
   3. Another option is to restrict SCXML so that the actions are parallelisable. 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 would then raise an error if the same variable is assigned in for example, the source states exit actions and the target states entry actions.
   4. One difficulty arises when a transition exits a parent state without specifying a 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.
2. Action sets. Sets of SCXML actions even within one location such as a transition are supposed to be sequential. The same argument as 1 applies even without considering entry/exit actions.
3. Transition triggers. There are two kinds of transitions in SCXML.
   1. ‘When’ transitions can fire spontaneously as soon as their guard becomes true. This is the same as iUML-B transitions and we have no problem translating these.
   2. The other kind of transition is ‘Triggered’ by an interface event. This could be simulated 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.
   3. Transitions may also trigger each other. This could be modelled by a similar mechanism except that the interface event is not needed since the flag is set directly by another transition.
4. Run to completion semantics. SCXML has a run-to-completion (or 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.

## Proposed Translation of Turnstile Example (ADDED 1/10/15)

It is proposed to enhance iUML-B to some extent in order to adopt some features of SCXML where they would be of likely benefit to iUML-B modellers. The following enhancements are proposed:

1. States will own Entry Actions and Exit Actions that will be added to all events that are elaborated by incoming, resp. outgoing, transitions of the state.
2. Exit actions will also be added to all outgoing transitions of parent super-states of the owner state. In some cases this may lead to invalid sets of substitutions. In this case the model can be revised to have separate transitions from each specific nested source state.
3. Statemachines will own a collection of triggers.
   1. Each trigger will generate a BOOL variable. (Note that this is a simplification of SCXML semantics, which permits several triggers of a kind to be queued).
   2. Transitions may reference a trigger. The reference will generate a guard, *<trigger variable> = TRUE* and an action *<trigger variable> := FALSE*.
   3. Transitions may own a collection of ‘Raise’ actions that reference an internal trigger. This will generate an action *<trigger variable> := TRUE.*
   4. Transitions may be designated as external. For external triggers an interface event will be generated with guard *<trigger variable> = FALSE* and action, *<trigger variable> := TRUE*.
4. A partial ‘run-to-completion’ semantics will be introduced by disabling all interface events while any external or internal transition is enabled. (This is not exactly the same as SCXML behaviour since it is possible for an external trigger event to be received but not acted upon immediately if the state machine is not in a position to respond to it. The deferred trigger would remain in the system until other events bring the system into a state where it can be responded to and at that point such responses would interleave with any other current activity.

At this time it is not proposed to adopt or support the following features of SCXML:

1. Sequential execution of actions. Hence, actions within any collection of actions must assign to different variables.
2. Sequential execution of Exit/Transition/Entry sets of actions. Hence, the union of actions within all exit-transition-entry triplets must assign to different variables.
3. Ordering amongst responses to internal triggers and enabled ‘when’ transitions. Hence, the external trigger to be responded to, will be chosen non-deterministically from those that are available.
4. Ordering amongst responses to external triggers. Hence, the external trigger to be responded to, will be chosen non-deterministically from those that are available.

## Turnstile Example in iUML-B (ADDED 1/10/15)

The turnstile SCXML example shown in Figure 2 has been constructed in iUML-B with annotations to illustrate the outcome of following the limited adoption and support described above. Entry/exit actions have been added to the tooling so these are automatically generated in the Event-B (although they are not yet shown on the diagram). Triggers have not yet been added to the tooling so these have been manually added in the generated Event-B.

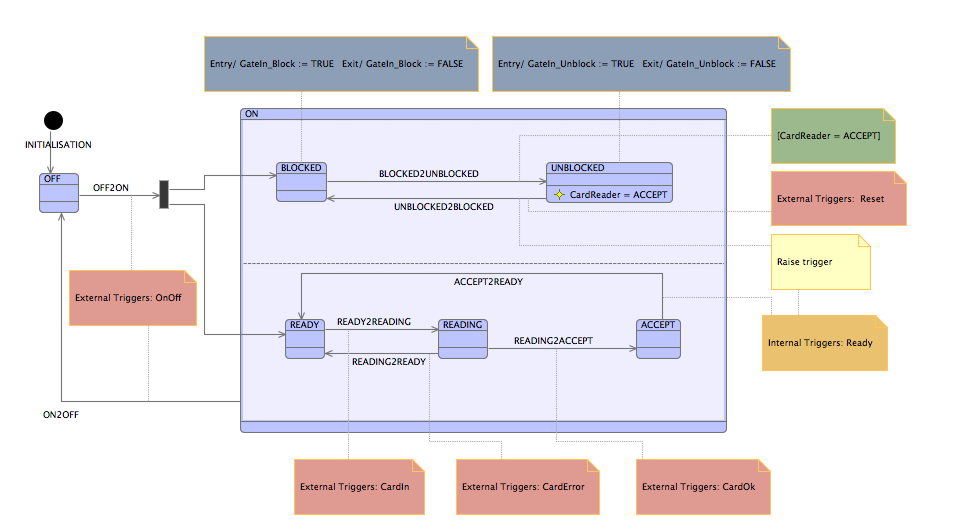
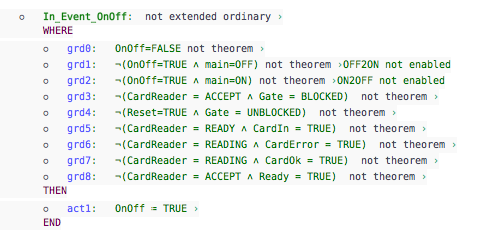


Figure - Turnstile Example in iUML-B

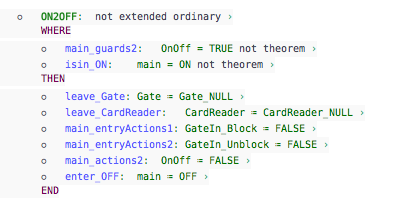
Entry actions on state OFF are omitted since the transition ON2OFF collects the exit actions of both BLOCKED and UNBLOCKED.

In iUML-B the transitions are labelled with the event that executes that transition and NOT the event that triggers it. To avoid any confusion with triggering events a naming convention has been adopted for transition events: *<Source State Name>2<Target State Name>*.

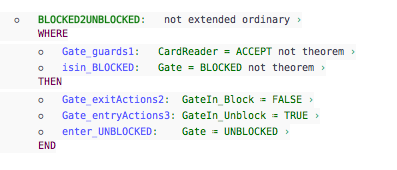
Boolean variables are added for each trigger event. These are named according to the SCXML event names. Interface events to set them are added with the naming convention: *In\_Event\_<trigger name>.* These events are guarded by the negation of each transition event guard.For example:



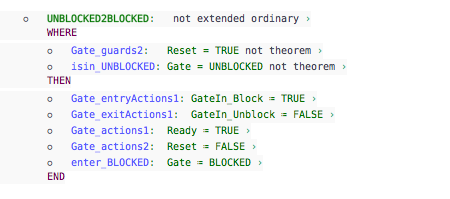
An example of a triggered transition event is shown below. It is guarded by the trigger and by the source state. The actions, in the order shown, are to leave the two nested Statemachines, perform the two entry actions of the target state, reset the trigger and change the state-machine state to the target state. (Notice that the exit actions of states nested in the source state are not generated because this transition exits the parent state without specifying a particular nested source state).



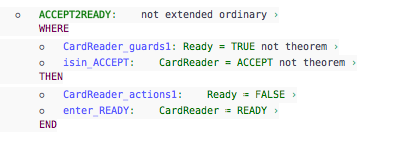
An example of a when transition is shown below. It simply doesn’t have a trigger. It is enabled as soon as the *CardReader = ACCEPT* state is entered and hence must be taken before any new Interface events will be enabled. (Notice it also involves exit and entry actions of its source and target state).



The following transition event, which responds to the Reset trigger, raises an internal trigger *Ready*

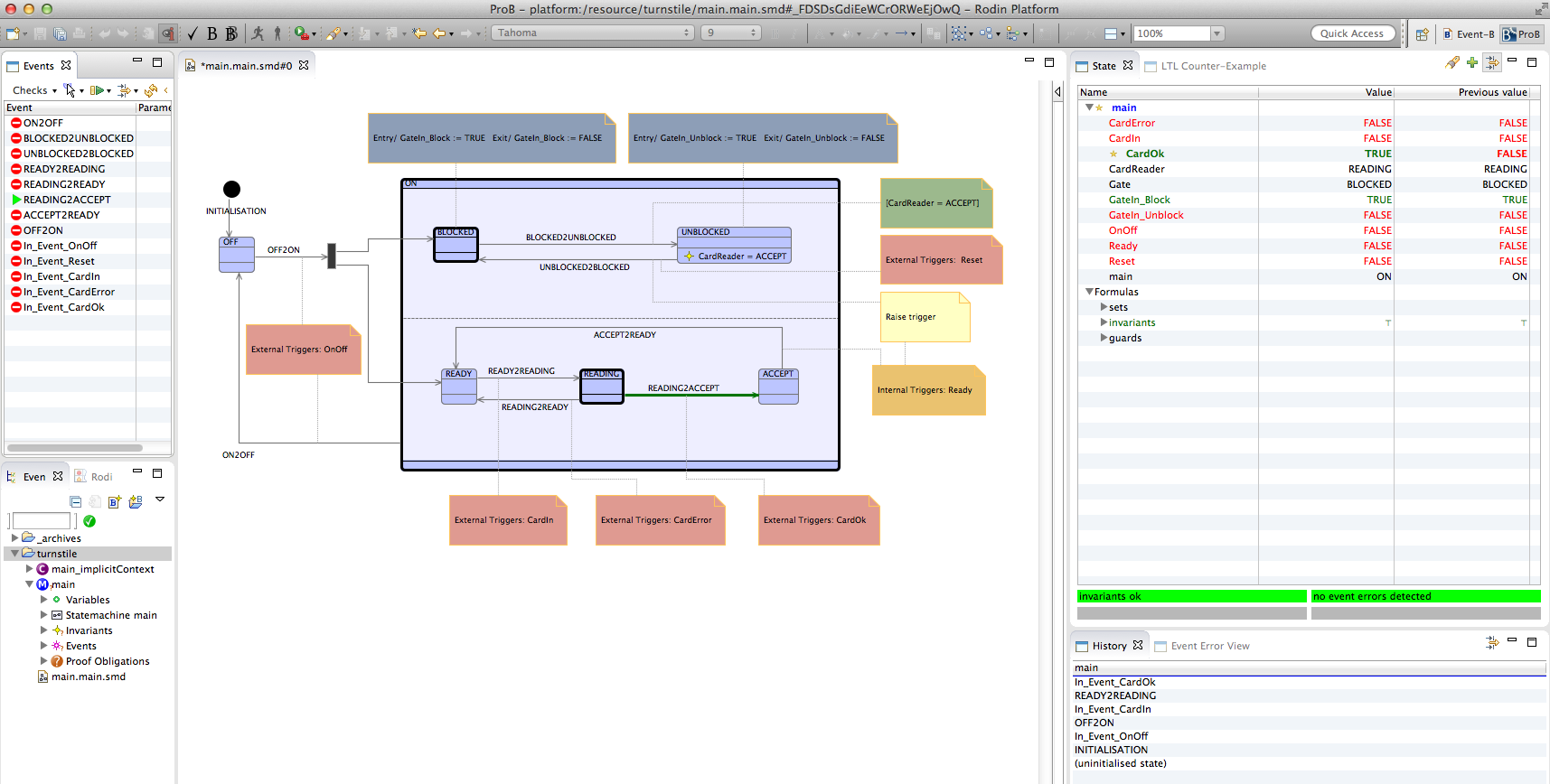


This internal trigger immediately enables another transition event, which must be taken before any interface events become enabled.

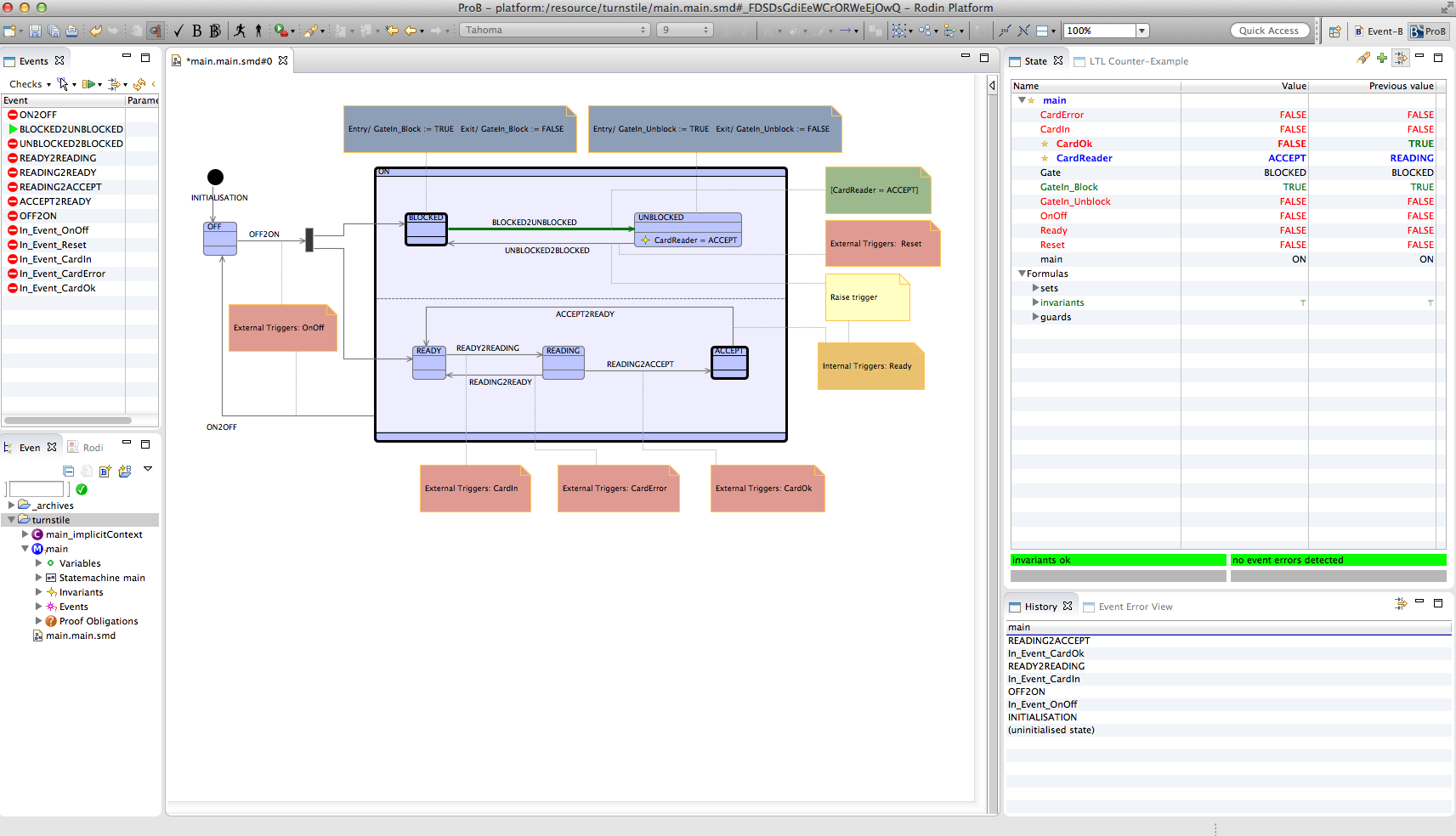


## Animation of iUML-B Turnstile Example (ADDED 1/10/15)

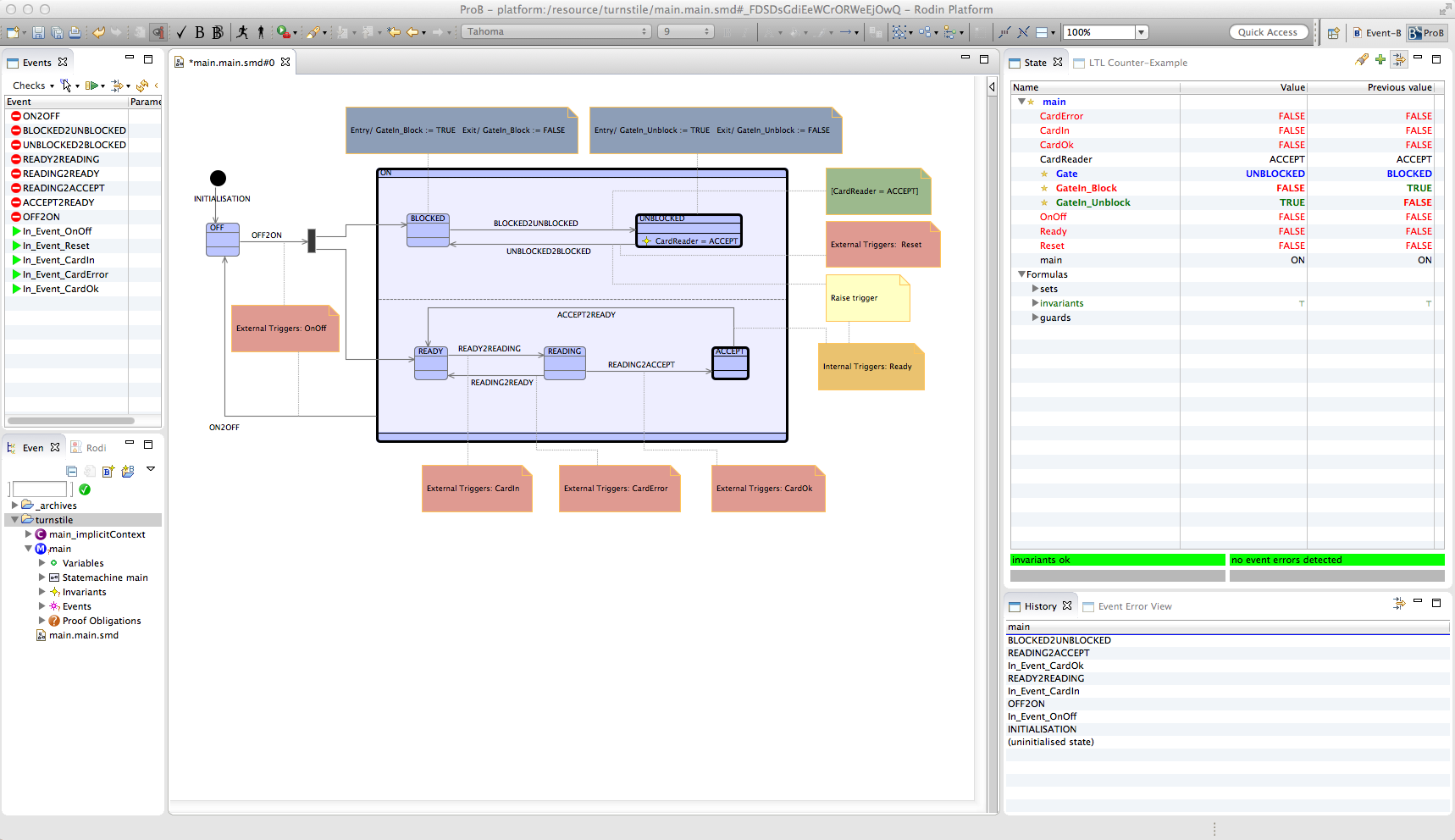
The Animation illustrates the behaviour of the model. In the first screenshot the CardReader has reached the READING state and an external trigger CardIn has been received, which enables the READING2ACCEPT transition. Since a statemachine transition is enabled, no Interface events are enabled.



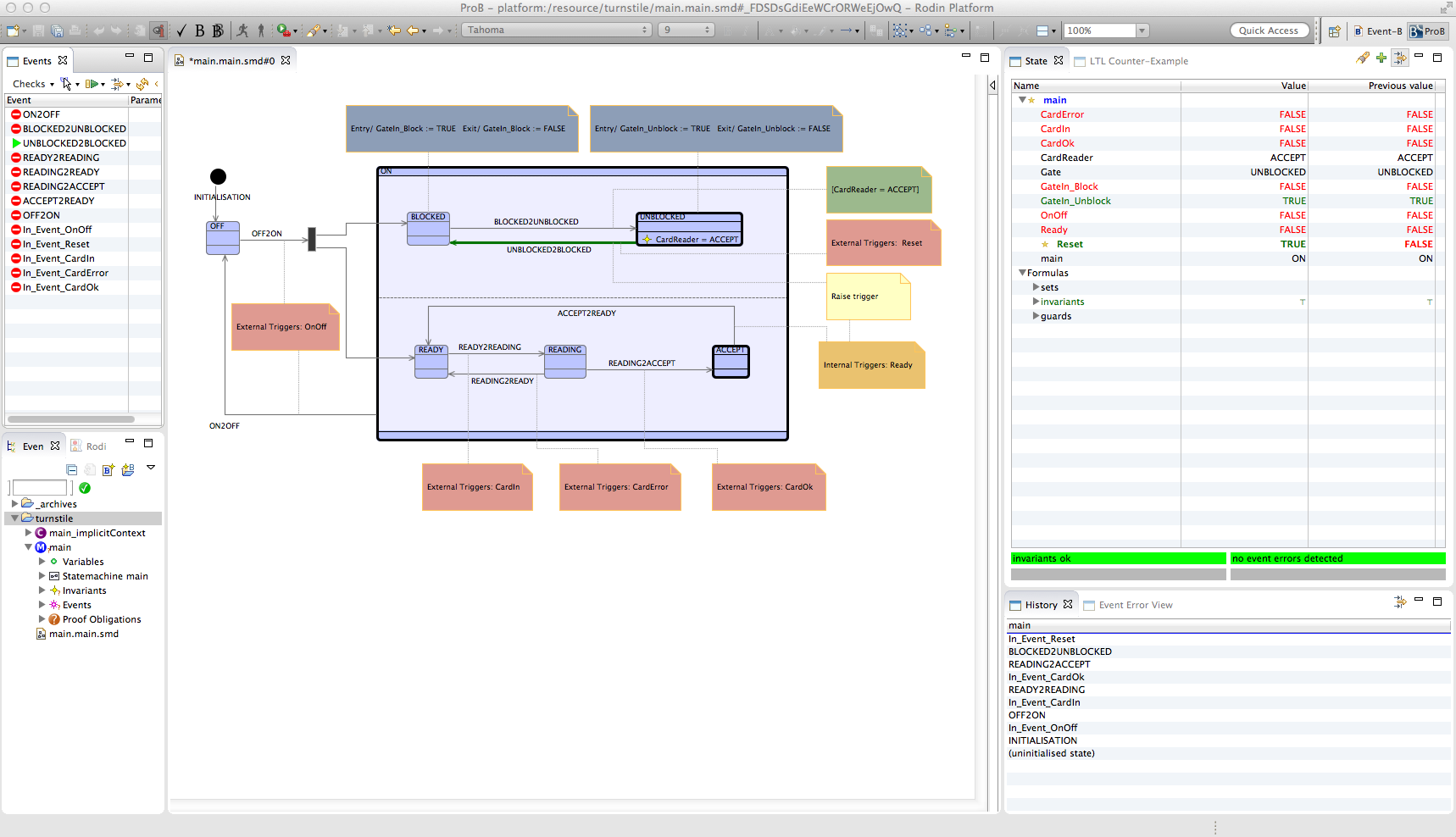
When the READING2ACCEPT transition fires, it enables BLOCKED2UNBLOCKED.



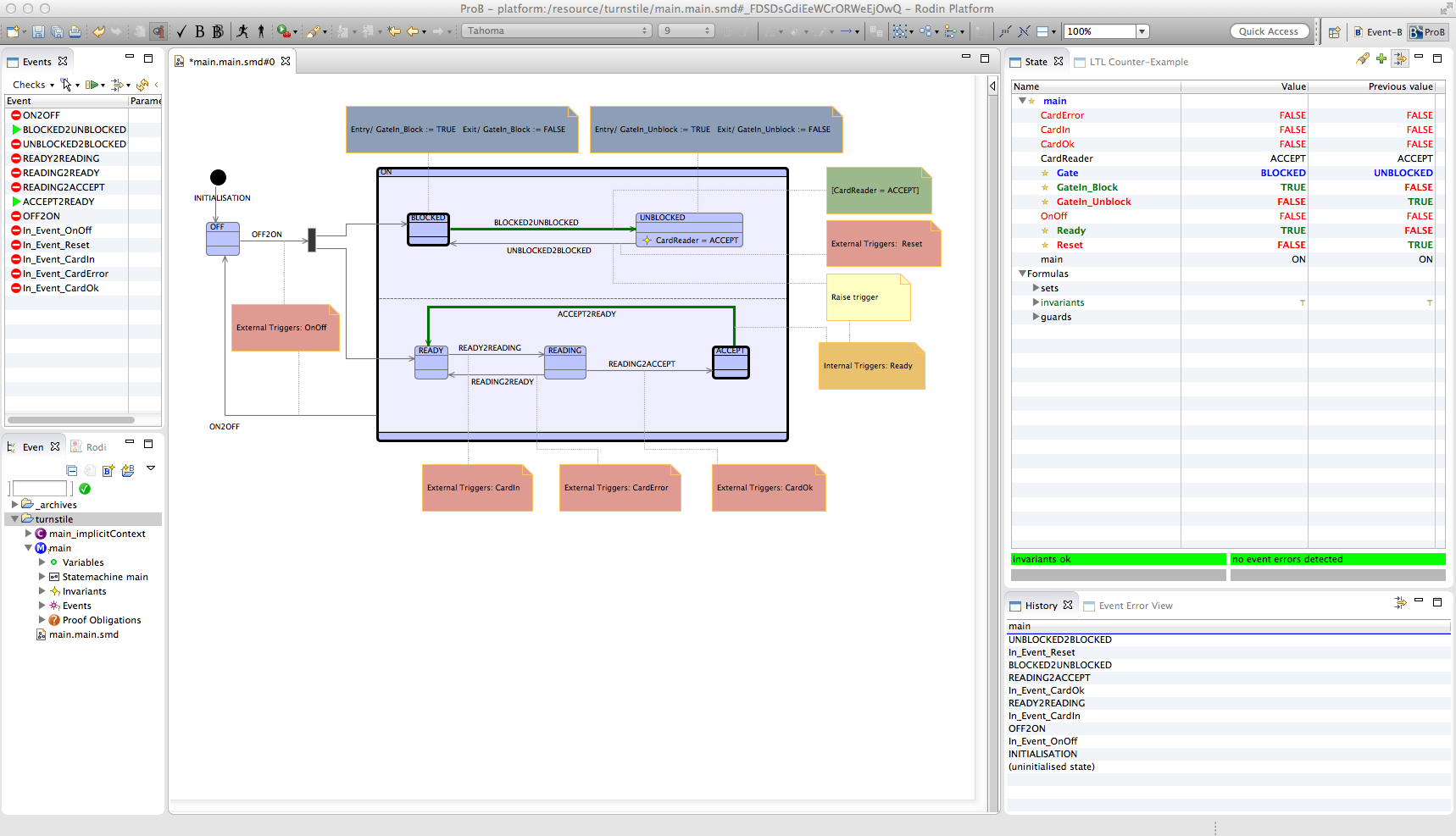
Once BLOCKED2UNBLOCKED fires, no transitions are enabled because the system is in a state requiring an external trigger. Hence the Interface events are enabled.



If a Reset trigger is received, UNBLOCKED2BLOCKED is enabled.



After UNBLOCKED2BLOCKED is fired, it raises an internal trigger that enables ACCEPT2READY. (Note that BLOCKED2UNBLOCKED is enabled since CardReader is still in the ACCEPT state. Taking this transition could lead to an invariant violation. Probably the best solution for this is to change BLOCKED1UNBLOCKED to be internally triggered rather than just guarded).



The interface triggers allow several transitions to be able to fire from the same trigger. Hence abstracting away from triggers is a significant restriction. For example triggers provide a way to respond in alternative ways depending on the current state of the system (something that state-chart people always find an unacceptable loss in UML-B).

SCXML notation does not explicitly model the interface triggers and queue giving the impression that they are not significant. However, the SCXML semantics is given by the processing engine (imagined or real). Hence the meaning of a trigger name attached to a transition is that an interface event is required to receive and queue a trigger flag and the transition has behaviour to consume this trigger.

Since Event-B has a completely different (but also implicit) semantic processing engine, it is necessary to generate Event-B model to explicitly add the SCXML interface trigger behaviour. Otherwise the translation does not provide an equivalent model.

# SCXML Tooling

## SCXML in EMF

An EMF meta-model for SCXML is available from the Sirius project. It is used by the Sirius project as a test sample so has not been released to users but is available in the github source code repository and will be copied and built for release with the iUML-B-SCXML tools. It supports SCXML functionality including data modelling and action/expression language.

Running this code in a debug environment enables us to produce the following SCXML model using the EMF sample editor. The serialised XMI is shown to the left.

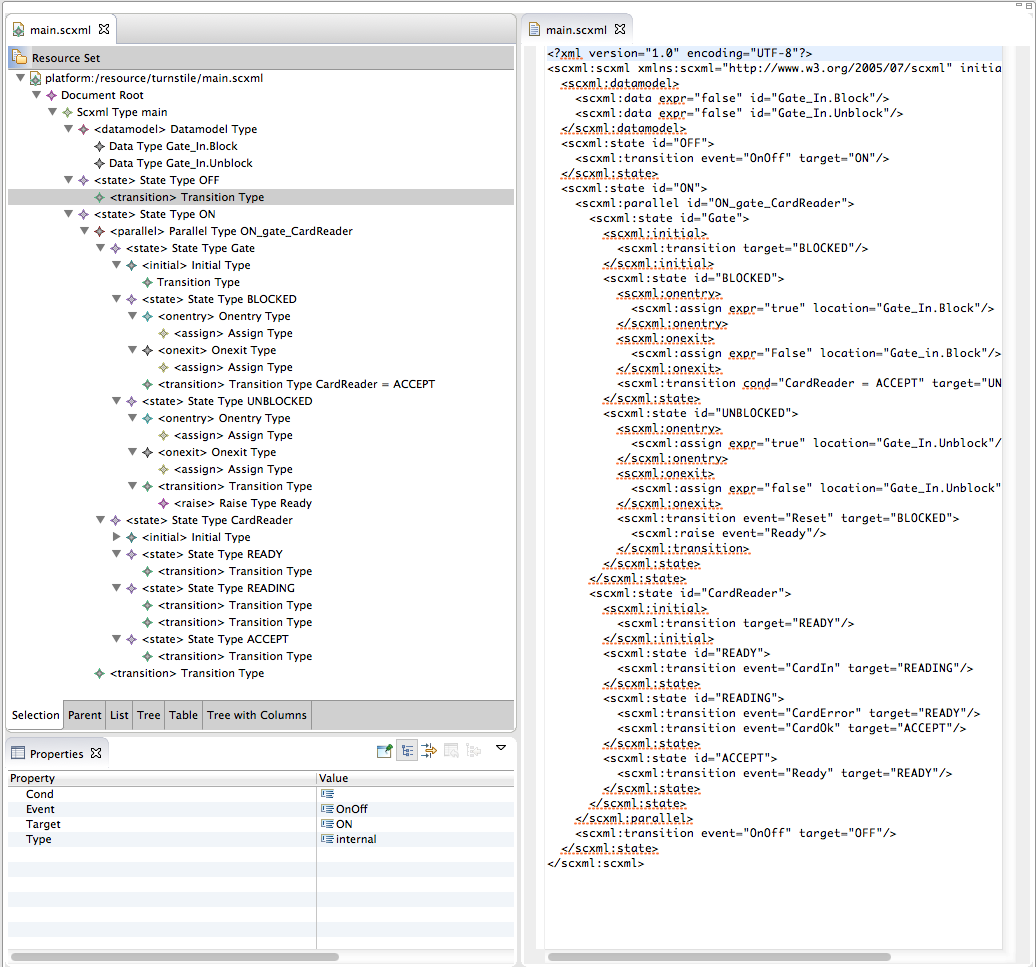


Figure -SCXML model that has been created in EMF and opened in a plain text editor

The serialisation shown above is the default produced for a newly created model. It contains the prefix tag ‘scxml:’ on each element, which is not standard scxml syntax. However, the serialisation is flexible and is able to load/update models without this prefix being introduced.

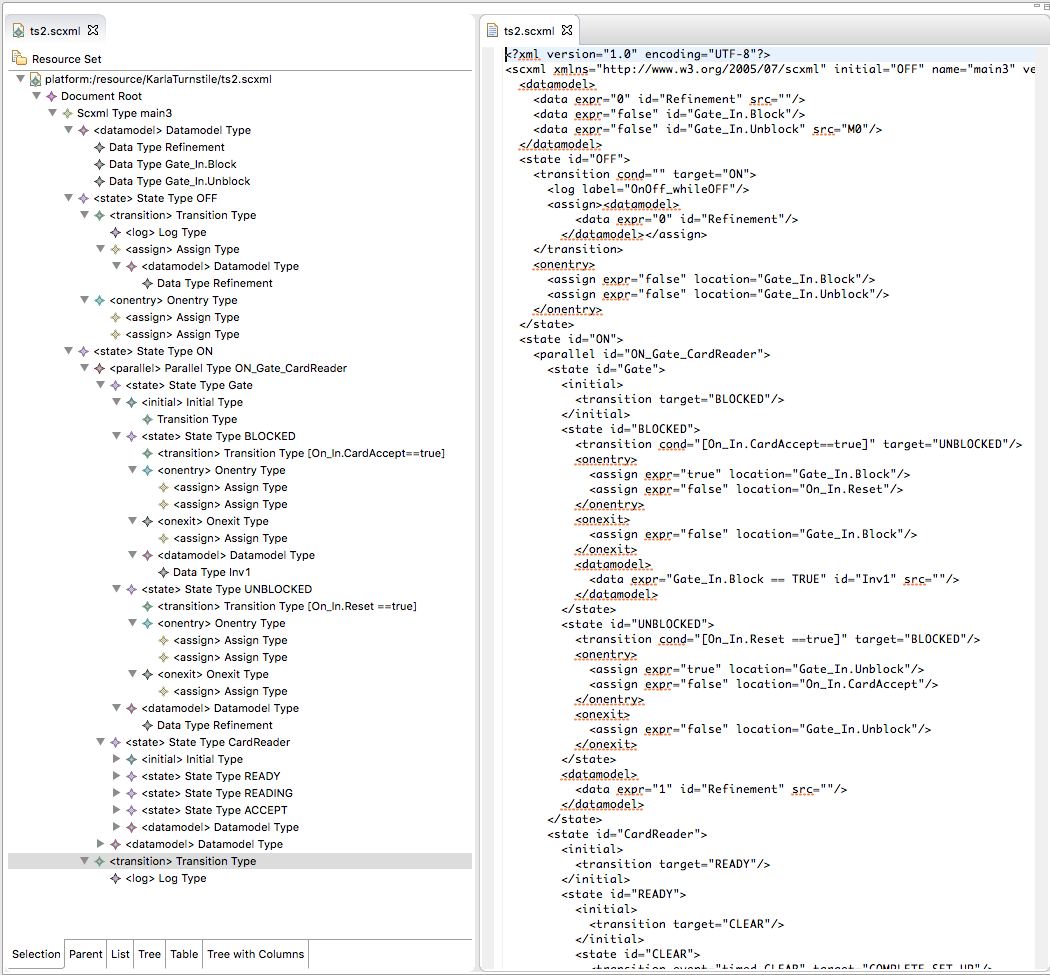


Figure - Example SCXML opened in EMF editor and plain text editor

## SCXML to iUML-B Translation tool

A tool to automatically translate SCXML models into iUML-B has been produced. The tool supports a limited subset of SCXML features. Additional conventions are adopted to express modelling features that are desired in the produced iUML-B such as refinement levels. SCXML models intended for translation must be produced in accordance with this restricted and enhanced SCXML functionality.

SCXML has limited support for extensibility making it difficult to annotate models with extra information to control the translation. This made it difficult to find a way to indicate the refinement

Hierarchical nested state charts are translated to similar corresponding state-machine structures in iUML-B according to the rules described in 5.2.1. 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.

### Extensions to SCXML

The following syntax extensions are added to SCXML models to support modelling features needed in iUML-B/Event-B. These extensions are prefixed with ‘*iumlb:*’ in order to distinguish them from the scxml XML parser. (So that they are ignored by SCXML simulation tools). They are loaded by EMF as generic feature maps (‘*Any*’ for contained elements and ‘*AnyAttribute*’ for attributes).

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

| **iumlb Attribute** | **Meaning** | **Legal Parents** |
| --- | --- | --- |
| iumlb:label | string used as the name of an Event-B event elaborated by the generated i-UML-B transition | scxml:transition |
| iumlb:refinement | non-negative integer representing the refinement level at which the parent element should be introduced.  *<we may need to make this a range>* | 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 but that’s all for now>* |
| 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 |

#### Notes on refinement levels:

For scxml:State the refinement level refers to any state machines generated from the children nested in the state (i.e. generated iUML-B states must be all based on the refinement level of the containing statemachine).

For iumlb:invariants the generated invariant is only generated at the specified refinement level, not in subsequent refinements.

### Supported SCXML features

| **SCXML** | **iUML-B** | **Notes** |
| --- | --- | --- |
| *Scxml* | A refinement chain of Event-B machines each containing an initialisation event and a root level iUML-B state-machine | The depth of the refinement chain is found by searching the scxml for the maximum refinement annotation |
| *iumlb:invariant* that is contained directly in the top level *scxml* | An invariant is added to the Event-B machine that has been produced from the containing scxml. | Added only at the refinement level defined in the iumlb:invariant (default 0) |
| *State* that is contained in an *scxml* or another *state* (i.e. not contained in a *parallel*) | An iUML-B state is added to the iUML-B state-machine that has been produced from the containing scxml or state | The iUML-B state is also added as a refined iUML-B state in all of the refinements of the parent iUML-B state-machine |
| *iumlb:invariant* that is contained in a *state* that generates an iUML-B state (i.e. not contained in a *parallel*). | An invariant is added to the iUML-B state that has been produced from the containing scxml. | Added only at the refinement level defined in the iumlb:invariant (defaults to first level at which containing iUML-B state is introduced) |
| *State* that is contained in a *parallel* element (which is, in turn, contained in another parent state or scxml from which an iUML-B state has been generated). | An iUML-B state-machine is added to the iUML-B state that has been produced from the *state* that contains the containing *parallel*. | The nested iUML-B state-machine is added starting from the refinement level that is annotated on the source *state* and continuing throughout subsequent refinements. |
| *State* that contains *state*s | A nested iUML-B state-machine, with an iUML-B initial state, is added to the iUML-B state that has been produced from the source state, if any, or from its containing state if it did not produce an iUML-B state. | The nested iUML-B state-machine is added starting from the refinement level that is annotated on the source state and continuing throughout subsequent refinements. |
| *Initial* Attribute of an *scxml* element | An iUML-B initial state and a transition from it to the iUML-B state indicated in the initial attribute is added to the iUML-B state-machine produced from the parent scxml | The iUML-B initial state and iUML-B transition are added at all refinement levels. The iUML-B transitions are set to elaborate the Event-B INITIALISATION event for that refinement level. |
| *Final* | An iUML-B state, an iUML-B final state and a transition from the former to the latter are added to the iUML-B state-machine that has been produced from the containing scxml or state. The iUML-B transition elaborates all Event-B events that are elaborated by iUML-B transitions that exit the parent iUML-B state . | The iUML-B state, iUML-B final state and iUML-B transition are also added as corresponding refined elements in all of the refinements of the parent iUML-B state-machine |
| *Transition* | An iUML-B transition is added to the iUML-B state-machine that has been produced from the containing scxml or state. The iUML-B transition’s source and target match those that have been produced from the corresponding scxml states given by the scxml transition’s source (i.e. containing) state and target state (which is named in the transitions *Target* attribute). (For initial states, the iUML-B equivalent is produced for each iUML-B state-machine without reference to the scxml initial element, but can be found by naming convention). The transition elaborates Event-B events according to the rules given in section 5.2.1.1. | The iUML-B transition and elaborated Event-B events are also added as corresponding refined elements in all of the refinements of the parent iUML-B state-machine |
| *Target* attribute of a *transition* element | Used to determine the transitions target state as described above. |  |
| *~~Cond~~* ~~attribute of a~~ *~~Transition~~* ~~element~~ | ~~An iUML-B guard, reflecting the predicate string found in the~~ *~~cond~~* ~~attribute, is added to the corresponding iUML-B transition.~~ | ~~Added at all refinement levels containing the transition.~~ |
| *iumlb:guard* element contained in a *Transition* element | An iUML-B guard, reflecting the predicate string found in the *iumlb:guard* attribute, is added to the corresponding iUML-B transition. | Added from the refinement level defined for the *iumlb:guard*. |
| *iumlb:label* attribute of a *Transition* element. | The *iumlb:label* attribute is used to name an Event-B event that the corresponding iUML-B transition should elaborate. See 5.2.1.1 | Added at all refinement levels containing the transition. |
| *Assign* element contained in a *Transition* element | An iUML-B Action is added to the iUML-B transition corresponding to the containing *transition* | Added at all refinement levels containing the transition. |
| *Assign* element contained in a *OnExit* element | An iUML-B exit action is added to the iUML-B state produced from the *state* that contains the containing *OnExit.* | The iUML-B exit action is also added to all of the refinements of the iUML-B state. |
| *Assign* element contained in a *OnEntry* element | An iUML-B entry action is added to the iUML-B state produced from the *state* that contains the containing *OnEntry.* | The iUML-B entry action is also added to all of the refinements of the iUML-B state. |
| *Data* | Data elements are interpreted in several different ways as explain in section 5.2.1.2. They may either be used as annotations to control the refinement levels, used to generate variable declarations, or used to generate invariants. |  |

#### Rules for constructing events to be elaborated by transitions

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

The event names are obtained (cumulatively) by the following methods:

1. the transition has iumlb:label attributes,

~~b) the transition has log labels,~~

c) the transition's source is an initial state (see below), or,

d) if none of the above provide any labels, a default 'source\_target' format is used.

Note that 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.

If the transition is in an initial state at the outer state chart level the name is INITIALISATION.

If the transition is in 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.

#### Rules for interpreting Data elements

*Data* elements (which are collated in *Datamodel* elements) are used ~~in a number of ways either~~ to model ancillary variables as part of the model in the way that SCXML intends them to be used ~~or in order to introduce invariants, which are not otherwise supported by SCXML, into the model or as annotations to aid the translation~~.

1. Deleted: use iumlb:refinement instead *~~Data~~* ~~elements with the~~ *~~id~~*~~= “Refinement” are interpreted as an annotation to indicate the refinement level at which to first introduce the iUML-B element that is generated from the SCXML element that contains the~~ *~~Datamodel~~* ~~element containing the~~ *~~Data~~* ~~element. The~~ *~~expr~~* ~~attribute must contain a non-negative integer (0 represents the most abstract machine, 1 the first refinement and so on). If a Scxml element does not contain such a refinement annotation, it is assumed that its refinement level is the same as it’s parent’s refinement level.~~
2. Deleted: use iumlb:invariant instead. ~~Data elements whose~~ *~~expr~~* ~~attribute parses as a predicate are interpreted as an invariant. (Some syntax conversion is performed to convert the predicate from SCXML format into Event-B mathematical language). The~~ *~~id~~* ~~attribute is interpreted as the label for the generated invariant. The invariant is introduced at the same refinement level as the parent element that contains it. [N.B. may need to introduce invariant later?]~~
3. Data elements ~~whose~~ *~~expr~~* ~~attribute parses as a value~~ are interpreted as a variable. The type of the variable is given in an iumlb:type attribute. ~~deduced from the value. Currently BOOL and INT are supported.~~ The *id* attribute is interpreted as the name of the variable. 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 Event-B mathematical language). The variable is introduced at the same refinement level as the parent element that contains it.

### SCXML features which are not supported

* The Initial attribute is only supported within scxml elements, it is not supported within state. An initial state should be added explicitly with an appropriate transition targeting the required initial state.

### Conventions and restrictions

* The translation assumes that each SCXML state-chart (nested or otherwise) has exactly one initial state.
* Care must be taken when using (entry, exit and transition) actions since the semantics of SCXML and iUML-B differ:
  + It must not be possible to write to the same variable in two actions that could be executed as part of the same transition firing. If violated this will be raised as an Event-B static checker error.
  + If a variable is written by an action and then used (read) in another action that is executed as part of the same transition firing, the behaviour of the SCXML model will be different from that of the iUML-B since the SCXML actions are executed sequentially whereas the iUML-B actions are executed in parallel. This will not be reported to the user as a warning or error.

### Limitations

The following limitations of the current approach to modelling in scxml and translating to iUML-B are noted.

TBD