JOURNAL OF OBJECT TECHNOLOGY

Online at www.jot.fm. Published by ETH Zurich, Chair of Software Engineering. ©JOT, 2002

Vol. 6, No. 9, 2007

Direct Semantics of Extended State Machines

K. Lano, D. Clark

In this paper we present semantic profiles for UML 2 state machines, which are based directly upon the structure of a state machine model, without requiring flattening or other transformations on these models. The approach addresses many of the semantic problems identified for state machines, and provides a basis for semantic analysis and proof of behavioural compatibility between state machines.

1 INTRODUCTION

The state machine notation of UML is widely used and supports dynamic modelling of applications ranging from reactive systems to web applications. However, there remain a number of semantic problems with the notation, particularly in the areas of transition priorities, history states [7] and generalisation [21].

In this paper we consider a substantial subset of UML 2 state machine notation and assign an axiomatic semantics to this. This semantics is assigned directly to the original models, and does not require pre-processing of the models to reduce them to a more basic form. The advantage of this is that the semantics retains the structure and context information of the model, making it easier to relate semantic analysis results to the model.

We apply this semantics to some of the problems identified by other authors, and to support semantic analysis of state machines using the B [1] formalism and tools. We also use the semantics to give a precise set of conditions which ensure behavioural compatibility between two state machine models.

In Section 2 we define the syntax of extended state machines, Sections 3 and 4 define their semantics. Section 5 considers how this semantics can be applied to provide solutions for some of the ambiguities and omissions in UML semantics. Section 6 gives criteria for behavioural compatibility. Section 7 shows how semantic analysis is supported by the semantics. Section 8 gives a comparison with related work.

2 EXTENDED STATE MACHINES

In [13, 15] we defined the syntax and semantics of a subset, UML-RSDS, of UML 2. UML-RSDS state machines have only basic states and no pseudostates. Here we will extend this state machine notation to include additional features of composite states, deferred events, compound transitions (modelled as transitions with sets of sources and targets), history states and final states. Figure 1 shows the subset of the UML 2 behaviour metamodel which we consider here. This omits junction, choice, entryPoint, exitPoint and terminate pseudostates, exit and do actions, and connection point references, and internal and local transitions, but otherwise is identical to the state machine metamodel of [20].

In this paper we will mainly consider protocol state machines. These have transitions with postconditions instead of effects, and states without entry, exit or do activities. We will however consider the semantics of history states for such machines. State invariants will be allowed for both protocol and behavior state machines.

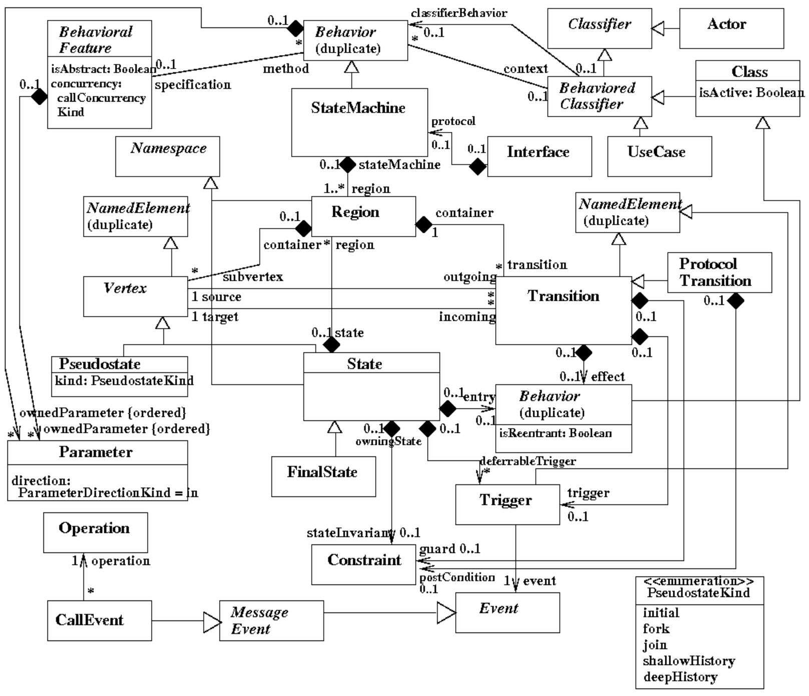

Figure 1: Behaviour metamodel

A *basic* state is a state with $region.size = 0$, other states are *composite* states. A composite state with one region is termed an OR state, and a composite state with more than one region is termed an AND state. The default initial state of an OR state or region s will be denoted $initial_s$.

In addition to the constraints on the metamodel defined in [20], we require:

1. That regions and OR states always have a unique initial pseudostate, and unique default initial state, and at most one final state.

2. That default transitions from history states must have target a top-level substate of the container of the history state. The target state cannot be a pseudostate.

3. Similarly the default transition from an initial state must have target a normal state in the same composite state at the same level as the initial pseudostate.

The notation $s \sqsubseteq s'$ means that $s = s'$ or s is a (recursive) substate of s'.

Normally if one region r of an AND state has a final state, so should all regions of the AND state, otherwise a completion event from the AND state can never be triggered by reaching the final state of r.

3 SEMANTICS FOR UML STATE MACHINES

We give first the semantics for simple state machines as used in [15], and then extend this to the full metamodel of Figure 1. The semantics of protocol and behavior state machines for a class C are incorporated into theories representing the semantics of C. This enables semantic checks of the consistency of the state machine models compared to the class diagram model.

The semantics is expressed in terms of temporal logic theories using the notation of Real-time Logic (RTL) and Real-time Action Logic (RAL) [10]. The reason for using this general framework is that related notations of UML, such as interactions, require explicit treatment of the times of events.

Each UML class and model is represented as a temporal logic theory, which has semantic elements representing structural and behavioral features of the class or model, and axioms defining their properties. A generic instance of C is represented as a theory $\mathcal{I}_C$, the class itself by a theory Γ_C, and models M by a theory Γ_M composed from the theories of the classes of M.

The following temporal logic notations are used to define the semantics:

1. The times $\leftarrow(op(p), i)$, $\rightarrow(op(p), i)$, $\uparrow(op(p), i)$, $\downarrow(op(p), i)$ of sending, request arrival, activation and termination of an operation execution $(op(p), i)$. These are enumerated by the index $i : \mathbb{N}_1$ in order of the reception times $\rightarrow(op(p), i)$.

2. Formulae $P \circledcirc t$, denoting that formula P holds at time t, and expressions $e \circledast t$ denoting the value of expression e at time t.

As an example of this logic, we define the meaning of UML time triggers. A relative time trigger *after* T on a transition away from a state s represents that the transition should be taken if the state has remained occupied for at least T time units continuously:

$$\mathit{duration}(\varphi_s) \circledast \mathit{now} \geq T$$

where φ_s expresses that s is occupied, and

$$\begin{aligned} \mathit{duration}(P) \circledast t \ &= \\ &\mathit{max}(\{0\} \cup \{x : \mathit{TIME} \mid \forall\, y : \mathit{TIME} \cdot t - x \leq y \wedge y \leq t \ \Rightarrow \ P \circledcirc y\}) \end{aligned}$$

Protocol State Machines

The semantics of a protocol state machine SC of a class C (for example, as in Figure 2) is expressed as follows in the instance theory $\mathcal{I}_C$:

1. The set of states (normal states, including final states) is represented as a new enumerated type $State_{SC}$, and a new attribute c_state of this type is added to $\mathcal{I}_C$. The axiom

   $$c\_state \in State_{SC}$$

   holds. Local attributes of the state machine are represented as attributes of $\mathcal{I}_C$.

2. We specify the initialisation $c\_state := initial_{SC}$ of this attribute to the default initial state of SC. This initialisation is invoked by the C constructor operation $init_C$ ("When an instance of a behaviored classifier is created, its classifier behavior is invoked", page 420 of [20]).

3. Each transition tr from a state src to a state trg, triggered by $m(x)$, with guard G and postcondition $Post$, is represented as an additional pre/post specification of m (page 521 of [20]):

   $$(c\_state = src \;\; \wedge \;\; G)$$

   is added as an additional disjunct of the precondition of $m(x)$, and

   $$(c\_state = src \;\; \wedge \;\; G)@pre \;\; \Rightarrow \;\; (c\_state = trg \;\; \wedge \;\; Post)$$

   as an additional conjunct of the postcondition.

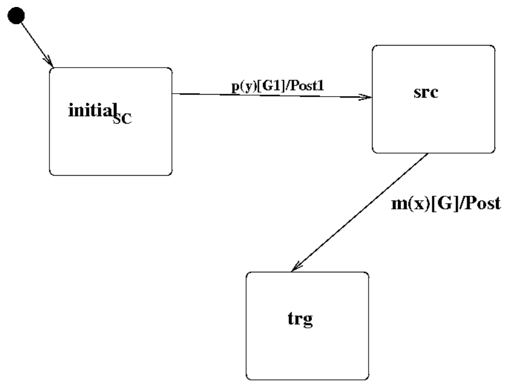

Figure 2: Protocol state machine

Only operations with at least one transition in the state machine have such derived constraints – other operations are assumed not to change the state (page 521 of [20]).

4. State invariants Inv_s have the semantics:

$$(StateInv): \\ c\_state = s \;\Rightarrow\; Inv_s$$

The semantics of a flat behavioral state machine can also be defined in the instance theory of its associated class, using composite actions [15].

4 SEMANTICS FOR EXTENDED STATE MACHINES

We extend the semantics of flat protocol state machines to state machines with OR and AND composite states, compound transitions and history and final states. The additional complexity introduces the possibility of alternative semantics, especially in the areas of transition priority and history state behaviour.

For each OR state s in the state machine, we define a state attribute $state_s : State_s$ where $State_s$ represents the set of normal states (including final states) directly contained in s. Regions of an AND state are also represented by a type and an attribute in the same manner (and so must be named). Each such OR state/region has a default initial state $initial_s$ and each $state_s$ is initialised to this value. If a final state is present, it is denoted by $final_s$. If s has a history pseudo-state as a direct substate, then an attribute $last_s : State_s \cup \{unset\}$ is also introduced, to record the last active top-level substate of s. This is initialised to the value $unset$ to indicate that no state of s has previously been occupied. When a final state of s is entered, $last_s$ is reinitialised to $unset$.

If the history state is a deep history, then a $last_{ss}$ variable is added for each OR substate and subregion ss of s.

The top level states of the state machine itself are also represented by an attribute $state : State$.

For each state x in the state machine diagram, a predicate φ_x can be defined, which expresses that x is part of the current state configuration of the state machine (Table 1).

State s	*State predicate* φ_s
Top-level state	$state = s$
Immediate substate of OR state/region r	$\varphi_r \wedge state_r = s$

Table 1: State predicate

A predicate $InitialState_s$ expresses that the (recursive) initial state of s is occupied (Table 2).

Using these predicates, the state-changing behaviour of transitions can be expressed as pre and post conditions. The *enabling condition* $enc(tr, s)$ of a transition

State s	$InitialState_s$
OR state, $initial_s$ basic	$state_s = initial_s$
OR state, $initial_s$ composite	$state_s = initial_s \wedge InitialState_{initial_s}$
AND state	conjunction of $InitialState_r$ for each region r of s

Table 2: Initial state predicate

tr

$$s \longrightarrow_{op[G]/Post} t$$

from a state s is $\varphi_s \wedge G$, conjoined with $\neg\,(\varphi_{ss} \wedge G')$ for each different transition

$$tr' : ss \longrightarrow_{op[G']/Post'} tt$$

triggered by op on a state ss, $ss \neq s$, $ss \sqsubseteq s$.

This expresses that tr is only enabled on s if higher-priority transitions for the same trigger operation/event are not enabled.

The complete enabling condition of tr is the conjunction of the enabling conditions from each explicit source of tr (tr may be a compound transition with multiple source states $sources(tr)$):

$$enc(tr) \;=\; \wedge_{s \in sources(tr)}\; enc(tr, s)$$

The precondition derived from a transition tr triggered by op is then $enc(tr)$.

The enabling condition is a critical semantic aspect which can be defined in different ways to produce different semantic profiles for state machines. We could use an alternative definition $enc'(tr)$, which is the conjunction of $enc(tr, s)$ for each *explicit and implicit* source s of tr. Implicit sources are those AND state regions which contain no explicit source of tr but will be exited when it takes place.

For the postcondition, there are several cases. A predicate $Target_{tr}$ expresses what state(s) are directly entered because of the transition tr (Table 3).

In the third and fourth cases, tr_0 is a transition identical to tr except that it is targeted at the default history state of p. In the third case tr_i is a transition identical to tr except that its target is s_i, the substate of p equal to $last_p$. The difference between shallow and deep history is that in the former, composite substates of the last active state will be entered at their initial state, whilst with deep history they are entered at their last active state, defined by the predicate $LastState_p$ (Table 4). In the final case, tf is a transition composed from tr followed by any completion-triggered transitions triggered by reaching t, from p or (recursively) from superstates of p.

For transitions with multiple targets, the conjunction of the $Target$ predicate for each target is taken.

State t	$Target_{tr}$
composite state	$InitialState_t$
basic non-history state	φ_t
shallow history state in OR-state/region p with direct substates $\{s_1, ..., s_m\}$	$(last_p@pre = unset \;\Rightarrow\; Target_{tr_0}) \;\wedge$ $(last_p@pre = s_1 \;\Rightarrow\; Target_{tr_1}) \;\wedge$ $... \;\wedge$ $(last_p@pre = s_m \;\Rightarrow\; Target_{tr_m})$
deep history state in OR-state/region p	$(last_p@pre = unset \;\Rightarrow\; Target_{tr_0}) \;\wedge$ $(last_p@pre \neq unset \;\Rightarrow\; LastState_p)$
final state $final_p$	$Target_{tf}$

Table 3: Target state predicate

State s	$LastState_s$
OR state, $last_s$ basic	$state_s = last_s$
OR state, $last_s$ composite	$state_s = last_s \wedge LastState_{last_s}$
AND state	conjunction of $LastState_r$ for each region r of s

Table 4: Last state predicate

In addition to the postcondition describing the direct target, the transition may also cause other states to be reinitialised. After taking account of the effect of history and final states, for each AND composite state x, if transition tr causes x to be entered, then all the regions of x which do not contain an actual target of tr must be reinitialised. This additional effect (which may apply to several AND compositions) is expressed by a predicate $ReInit_{tr}$.

The complete postcondition of tr is the conjunction of its explicit postcondition, its target state predicate(s), and $ReInit_{tr}$.

Axioms 1 and 2 of Section 3 can therefore be restated using the $state_r$ variables for each OR state and region r. Axiom 3 applies with the pre and postconditions derived from each transition as described above. Axiom 4 holds in the form

$$\varphi_s \;\Rightarrow\; Inv_s$$

for each state s.

Other elements of UML state machine notation can also be given a semantics:

1. If event e is deferred in state s, which also has a set of explicit transitions $tr_i : s_i \rightarrow_{e[G_i]/Post_i} t_i$ for e, where $s_i \sqsubseteq s$, then this means that e cannot be consumed unless one of these transitions is enabled:

   $$\forall\, i : \mathbb{N}_1 \cdot (\varphi_s \;\Rightarrow\; enc(tr_1) \vee ... \vee enc(tr_n)) \circledcirc \uparrow(e, i)$$

   This is in accordance with the semantics of [20], whereby substates which accept an event override superstates which defer it.

2. Internal transitions $\rightarrow_{e[G]/Post}$ of state s are expressed as pre and post-conditions of e with the form $\varphi_s \wedge G$ and $(\varphi_s \wedge G)@pre \;\Rightarrow\; \varphi_s \wedge Post$

The semantic definition for behavioral state machines is similar, except that we need to define the sequence of actions executed by a transition, in addition to the target state predicate.

5 SOLUTIONS FOR SEMANTIC PROBLEMS

Many of the semantic problems identified in [7] remain in the UML 2 state machine notation definitions of [20]. In particular, the definitions of transition priority have not been improved and remain ambiguous: "The priority of joined transitions is based on the priority of the transition with the most transitively nested source state" (page 547 of [20]). Page 548 of [20] gives an algorithm for calculating the fired set of transitions when an event occurs, this algorithm involves starting from "innermost nested simple states", which does not resolve cases such as Figure 3, reproduced from [7]. We assume that all the transitions are triggered by the same event and have *true* guards.

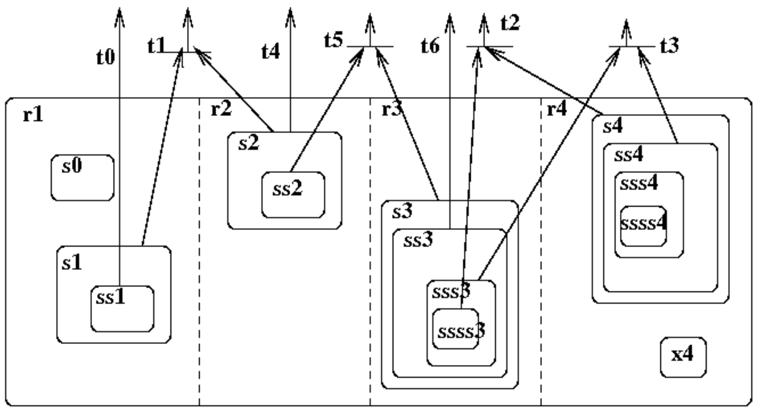

Figure 3: Priority examples

Our semantics defines solutions to the problems of ambiguity and imprecision of UML identified in [7] for transition priority and history states. For priority the semantics implies that a transition t has priority over another t' in state s if t is enabled in this state and t' is not.

For history states, the semantics means that the history of an OR state/region is always unset by entering a final state (page 536 of [20]), and that this history is independent of the history of any other state in the model.

Transition priorities

We define that transition t has priority over transition t' (for the same trigger) in a state s if

$$\varphi_s \;\Rightarrow\; enc(t) \wedge \neg\, enc(t')$$

In the example of Figure 3 we have:

$$\begin{aligned}
enc(t_0) &\equiv state_{r1} = s1 \wedge state_{s1} = ss1\\
enc(t_1) &\equiv state_{r1} = s1 \wedge state_{s1} \neq ss1 \wedge state_{r2} = s2 \wedge state_{s2} \neq ss2\\
enc(t_2) &\equiv state_{r3} = s3 \wedge state_{s3} = ss3 \wedge state_{ss3} = sss3 \;\wedge\\
&\quad state_{sss3} = ssss3 \wedge state_{r4} = s4 \wedge state_{s4} \neq ss4\\
enc(t_3) &\equiv state_{r3} = s3 \wedge state_{s3} = ss3 \;\wedge\\
&\quad state_{ss3} = sss3 \wedge state_{sss3} \neq ssss3 \wedge state_{r4} = s4 \wedge state_{s4} = ss4\\
enc(t_4) &\equiv state_{r2} = s2 \wedge state_{s2} \neq ss2\\
enc(t_5) &\equiv state_{r2} = s2 \wedge state_{s2} = ss2 \wedge state_{r3} = s3 \wedge state_{s3} \neq ss3\\
enc(t_6) &\equiv state_{r3} = s3 \wedge state_{s3} = ss3 \wedge state_{ss3} \neq sss3
\end{aligned}$$

When no clear highest priority transition exists from a particular state combination, such as $s0$, $ss2$, $ssss3$ and $ss4$, then no transition is enabled.

Non-determinism still remains possible in UML state machines, in the cases:

1. Two transitions with the same priority can be enabled at the same time from the same state (conflicting transitions). For example, t_0 and t_4 and t_6 are all enabled in the state $ss1$, $s2$, $ss3$ (not $sss3$) and $x4$.
2. The order of entry actions to orthogonal regions, exit actions from orthogonal regions, and actions on transitions executed in orthogonal regions as part of the same event reaction are undefined (page 547, [20]).
3. The choice of enabled transitions exiting a choice point is not defined (page 555, [20]).

The first indicates a potential inconsistency in a state machine model and should be eliminated: it can be checked statically, since the enabling conditions (omitting transition guards) consist only of equalities and inequalities over finite sets. The second can be modelled using a parallel execution operator || [15], which permits either concurrent or interleaving implementations. The third indicates an ambiguous model, which should be made unambiguous by refining the conditions concerned.

If we take the stronger definition of enabling, enc' from Section 4, then many cases of transitions which conflict under the enc definition no longer conflict. t_0 for example has the implicit source $r2$, which is disabled if t_4 is enabled. However, this definition is further from the visual representation, since it requires the determination of the (possibly non-obvious) implicit sources of transitions.

History states

The UML documents also do not clearly define the meaning of history states. Page 523 of [20] states "*deepHistory* represents the most recent active configuration of the composite state that directly contains this pseudostate (eg, the state configuration that was active when the composite state was last exited)". This suggests a semantics where 'most recent active' means 'the state from which the composite state was last exited'. But page 528 of [20] says instead that deep history can be defined even if no exit from the composite state has taken place. We assume that 'most recent active' means 'most recently active before the transition to the history state', regardless of whether the transition came from inside or outside of the composite state.

Our semantics also resolves the problems of history states described by [7]. Figure 4 illustrates the problems identified in [7].

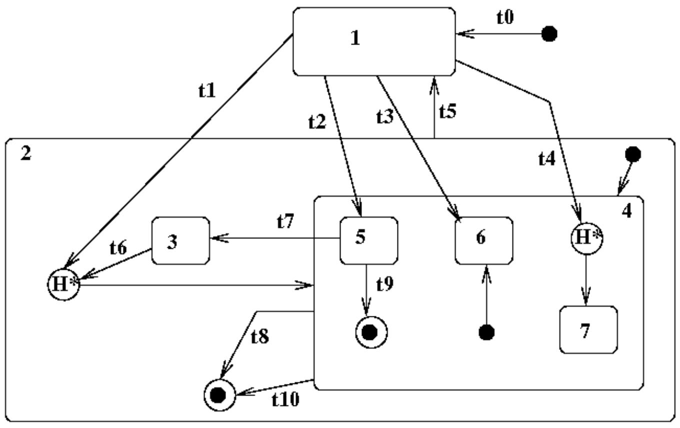

Figure 4: Example of history states

These problems are:

1. It is not clear if default transitions from history states must go to normal non-final states (not pseudostates or final states). We have enforced this restriction.

2. The notion of 'last active' state is ambiguous, it is not clear if this can include final states of composite states (cf. example 3 below). We enforce that final states cannot be considered as being last active states, instead, entry to a final state resets the record of the last active state in the composite state. The reason for this decision is that we consider final states only have a meaning as a signal that the containing state has completed its activity.

3. Do history states of nested states affect deep history entry to these states? In our semantics they do not, only the last active states of these states determine the target of a deep history entry to an already visited state (cf. example 1 below).

4. Does the reset of the last active state in a composite state p on entry to p's final state also reset the records of last active state in its substates? In our semantics it does not, the reset only applies to p (cf. example 2 below).

Tables 5, 6 and 7 illustrate the effect of our semantics in some scenarios of Figure 4. We assume that all transitions have different triggers, and that only t_8 has a completion event trigger.

Transition	*new state*	*new value of* $last_2$	*new value of* $last_4$
t_0	1	unset	unset
t_3	2, 4, 6	4	6
t_5	1	4	6
t_2	2, 4, 5	4	5
t_7	2, 3	3	5
t_6	2, 3	3	5

Table 5: History example 1

In Table 5 the most recently occupied substate of state 2, prior to the transition $t6$ to 2's history state, is the actual destination of this transition, in this case it is the state 3.

Transition	*new state*	*new value of* $last_2$	*new value of* $last_4$
t_0	1	unset	unset
t_2	2, 4, 5	4	5
t_{10}	2, $final_2$	unset	5
t_5	1	unset	5
t_4	2, 4, 5	4	5

Table 6: History example 2

In Table 6 the history of state 2 is unset by the transition t_{10} to its final state, but the history of state 4 is not unset, since it never enters its final state in this example.

Transition	*new state*	*new value of* $last_2$	*new value of* $last_4$
t_0	1	unset	unset
t_2	2, 4, 5	4	5
t_9	2, $final_2$	unset	unset
t_5	1	unset	unset
t_1	2, 4, 6	4	6

Table 7: History example 3

In Table 7 the transition $t9$ leads to the final state of state 4 and triggers $t8$ which also unsets state 2's history, thus subsequent entry to the history state of 2 uses the default history transition of this state.

6 BEHAVIOURAL COMPATIBILITY

Behavioural compatibility is the requirement that the specified behaviour of a superclass object should not be violated by a subclass object.

In the UML 2 documents various transformations on state machines are proposed for extending a superclass state machine to a subclass machine (page 548 of [20]): splitting a source state, adding states to a composite state, etc. The transformations aim to preserve the capability of clients to invoke particular sequences of operations, however the behaviour of these same sequences of operations may be unexpected to the client who only knows the superclass specification.

In particular, transitions should only have their target state replaced by a more specific state (corresponding to a strengthened postcondition) rather than an arbitrary state (as in [20]), and it is valid for transitions to be entirely replaced by a set of more specific transitions for the same trigger.

To ensure semantic behavioural compatibility, we can define three syntactic conditions on the protocol state machine C of a subclass CC of a class AC and the protocol state machine A of AC [11, 12]:

1. *Refinement*: For every state s of C, there is a state $\sigma(s)$ of A, and for every transition tr of C triggered by an operation of AC there is a transition $\sigma(tr)$ of A such that:

   (a) $\sigma(s)$ is initial in A if s is initial in C.

   (b) $\sigma(tr) : \sigma(s) \rightarrow \sigma(t)$ in A if $tr : s \rightarrow t$ in C.

   (c) tr and $\sigma(tr)$ have the same trigger.

   (d) $Post_{tr} \;\Rightarrow\; Post_{\sigma(tr)}$

   This means that any behaviour of C must satisfy the specifications of behaviour of A. σ is termed an *abstraction morphism*.

2. *Adequacy*: For each state s of A there is at least one state s' of C such that $\sigma(s') = s$. The disjunction of the state invariants of all such s' is equivalent to the state invariant of s. If s is initial in A, so is one of the s' in C.

   For each transition $tr : s \rightarrow t$ in A there are transitions $tr' : s' \rightarrow t'$ of C such that $\sigma(tr') = tr$, for every state s' such that $\sigma(s') = s$. The disjunction of guards of the tr' with source a particular such s' should be equivalent to the guard of tr.

   This means that behaviour defined in the superclass must also be defined in the subclass.

3. *Locality*: If a new operation op of CC has a transition $tr : cs \rightarrow ct$ in C for which $\sigma(cs) \neq \sigma(ct)$, or which modifies any data feature of AC, then op must (in terms of its effect on the data of AC and state of A) be expressible as a procedural combination of operations of AC.

These conditions can be formally deduced from the requirement that the semantics of CC together with C, as a theory, is a theory extension of the theory of AC and A [11]. Refinement and adequacy also correspond to the usual definition of refinement in state-based system specification [1]. The locality condition is a consequence of the frame axiom [8], and corresponds to Liskov's composition requirement for new operations introduced in subclasses [18], with respect to observational equivalence under the query operations of the superclass: it ensures that clients of the subclass do not see unexpected behaviour – new transitions and pathways between states not present in the superclass specification.

This definition can also be used for extended state machines, with the additional restriction that σ must respect state nesting: if $s \sqsubseteq_C s'$ then $\sigma(s) \sqsubseteq_A \sigma(s')$.

In addition, if a transition tr of C has multiple sources s_1, ..., s_n, then $\sigma(tr)$ must have the set $\sigma(s_1)$, ..., $\sigma(s_n)$ as its set of sources. Similarly for target states.

If s' is initial in an OR state or region s in C, then either $\sigma(s') = \sigma(s)$, or $\sigma(s')$ is initial in $\sigma(s)$.

For extended state machines the abstract and concrete states are both defined as state configurations, in terms of tuples of OR and region state variables. The condition of adequacy means that for each abstract state (configuration) s, every concrete state (configuration) s' which abstracts to s must have the same possible behaviour as s with regard to the operations of the generalised model.

Using the above conditions, we can verify that many generalisation transformations on state machines are semantically correct:

1. A postcondition of a transition can be strengthened.

2. A transition can be split into several cases from the same source state, with guards logically partitioning the original guard, and with targets equal to the original target [5].

3. A new region can be added to a concurrent composite state, provided this region has no trigger in common with transitions which exit the existing composite state. It is valid to refer to the state of this region in the transition guards of other regions – provided that all possible states of the new region are alternatives in the guards of the refined transitions from any particular state (Figure 5).

Other generalisation transformations can be proved correct by direct reference to the semantics. For example, introducing a deferral of an event leads to a stronger semantic theory for a state machine, and hence a semantic subtyping. Introducing a history pseudostate into an OR-state/region without a history pseudostate is also a refinement, as is interchanging H and $H*$ pseudostates in a region/OR-state which contains only basic substates.

Once a transformation has been proved correct, it can be used as required to produce or verify correct refinements, without requiring repetition of the proof.

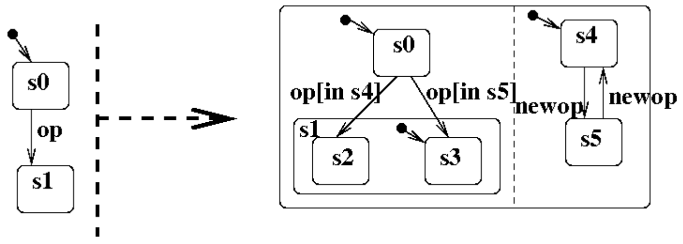

Figure 5: Adding a region

7 SEMANTIC ANALYSIS

Semantic analysis of UML models is necessary to ensure their internal consistency and their correctness with respect to requirements. Instead of writing new analysis tools, the UML-RSDS tools utilise translations to the formal notations B and SMV to make use of the existing industrial-strength tools for these notations. The translations are based directly on the semantics given above, so that the translations preserve the meaning of the UML models and enable immediate interpretation of analysis results in terms of the original models.

The UML-RSDS tools express the semantics of a state machine for a class by extending the class diagram with additional enumerated type definitions for the $State_s$ types of Section 4, new attributes of the class for each $state_s$ variable, and new pre and postconditions of the operations.

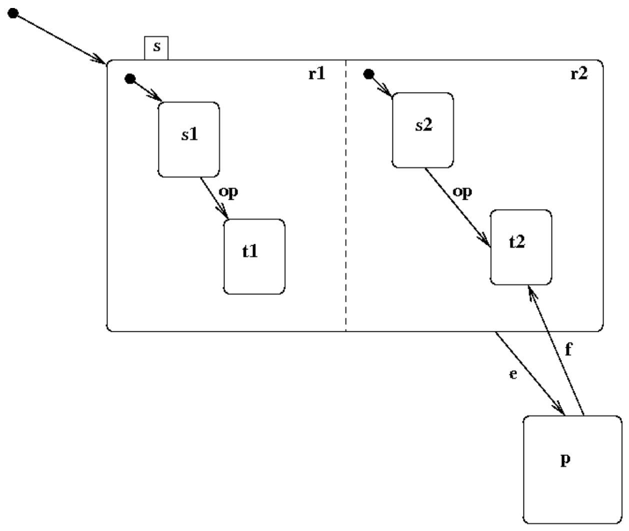

Figure 6: Translation example

For example, the state machine of Figure 6 for a class C is interpreted as the following definition of op:

```
op()
```

```
pre: state_C = s & state_r1 = s1 or state_C = s & state_r2 = s2
post: ((state_C = s & state_r1 = s1)@pre  =>  state_C = s & state_r1 = t1) &
      ((state_C = s & state_r2 = s2)@pre  =>  state_C = s & state_r2 = t2)
```

In the translation to B, each class is interpreted in a B module, called a *machine* [13]. The structure of these modules corresponds closely to that of the class. For this example, a machine of the following form would be produced for class C, expressing the instance theory $\mathcal{I}_C$:

```
MACHINE C SEES SystemTypes
VARIABLES state_C, state_r1, state_r2
INVARIANT state_C : StateC & state_r1 : State_r1 &
    state_r2 : State_r2
INITIALISATION state_C := s || state_r1 := s1 || state_r2 := s2
OPERATIONS
  op =
    PRE state_C = s & state_r1 = s1 or
        state_C = s & state_r2 = s2
    THEN
        IF state_C = s & state_r1 = s1 THEN state_r1 := t1 END ||
        IF state_C = s & state_r2 = s2 THEN state_r2 := t2 END
    END

END
```

Proofs of consistency and validation properties can then be carried out using a toolkit for B such as B4free [3] or the B Toolkit [4].

Translation to SMV allows temporal properties of specifications to be verified [2].

An additional facility in the UML-RSDS toolset supports automated application of model transformations, such as the behaviour refinements described in Section 6. Also a search facility is provided, which attempts to construct an abstraction morphism between two state machines.

8 RELATED WORK

Many approaches to defining the semantics of UML state machines use flattening to reduce a state machine with composite states and features such as history states to simple finite state machines in which there are only non-composite states and simple (single source, single target) transitions without pseudostates [6, 19]. The problem with this approach is that the structure of the original model will be lost and the number of states and transitions to be considered increase significantly. For example, the simple parallel machine of Figure 6 expands to the flattened model of

Figure 7. In this version the meaning of the model is less clear, and its semantic representation is more complex.

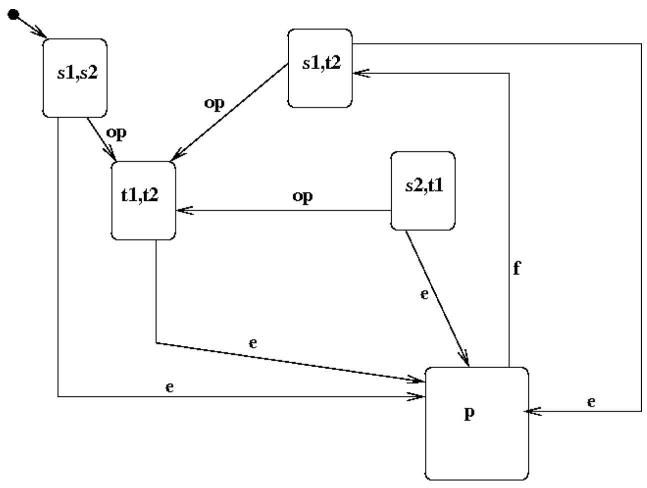

Figure 7: Flattened parallelism example

As far as possible, our semantics represents the meaning of state machines in notations which are close to UML class diagram and OCL notations. The semantics may therefore be more accessible to UML users than semantics which use external formalisms such as Petri Nets ([19]) or term algebras ([17]). An axiomatic semantics is also well-suited for use with logic-based semantic analysis tools such as B. Compared to [17] we do not represent sync states, however we can express the semantics of time-triggered transitions using the RAL formalism [15], extending [17].

The approach of [16] is close to ours, but translates directly into B from statecharts, instead of utilising an underlying axiomatic semantics. Elements of UML state machine notation such as time triggers, which require a temporal logic semantics, are not handled by this approach.

9 FUTURE WORK

For behavior state machines, there are unclear semantics concerning entry actions and do activities, in particular. We can use the idea that entry actions are designed to establish state invariants (the state is stable when these invariants hold true – page 546 of [20]), and therefore they occur just before entry of the state, meaning that a transformation which moves the entry actions to the end of the actions of each incoming transition of the state is an equivalence. Do activities *op* which are continuously executed while in a state *s* also may be designed as refinements of state invariants. They satisfy an axiom of the form

$$\varphi_s \;\Rightarrow\; \#active(op) = 1$$

where $\#active(op)$ counts the number of active executions of *op* at a time.

A do-once activity of state s, which generates a completion event on its termination, can be modelled as an initial transition to a final state (within s if s is basic, or in a new region of s otherwise).

10 CONCLUSION

We have defined an axiomatic semantics for a large part of UML 2 state machine notation, using the informal OMG Superstructure definition of the semantics as the basis. Our semantics resolves ambiguities and incompleteness in the informal semantics, in the areas of transition priority and history behaviour. It is used as the basis for semantic analysis using proof and model-checking tools, in the UML-RSDS toolset [14], and these have been applied to large reactive system case studies [9].

The axiomatic semantics approach has the advantage of expressing UML semantics in a high-level manner, in a formalism which is similar to, but independent of, UML.

REFERENCES

[1] J-R Abrial. *The B Book: Assigning Programs to Meanings*. Cambridge University Press, 1996.

[2] K. Androutsopoulos, *Verification of Reactive System Specifications using Model Checking*, PhD thesis, King's College, 2004.

[3] B4free, http://www.b4free.com, 2007.

[4] B-Core UK Ltd., The BToolkit, 2006.

[5] S. Cook, J. Daniels, *Designing Object Systems: Object-oriented Modelling with Syntropy*, Prentice Hall, 1994.

[6] W Damm, B Josko, A Pnueli, and A Votintseva. A discrete-time UML semantics for concurrency and communication in safety-critical applications. *Science of Computer Programming*, (55):81–115, 2005.

[7] H. Fecher, J. Schonborn, M. Kyas, W-P de Roever, *29 New Unclarities in the Semantics of UML 2.0 State Machines*. In *Formal methods and software engineering ICFEM 2005*, volume 3785 of *LNCS*, pages 52–65. Springer, 2005.

[8] J Fiadeiro and T Maibaum. Sometimes "tomorrow" is "sometime". In *Temporal Logic*, volume 827 of *Lecture Notes in Artificial Intelligence*, pages 48–66. Springer-Verlag, 1994.

[9] P. Kan, *Specification of Reactive Systems using RSDS*, PhD thesis, King's College London, 2006.

[10] K. Lano, *Logical Specification of Reactive and Real-Time Systems*, *Journal of Logic and Computation*, Vol. 8, No. 5, pp 679–711, 1998.

[11] K Lano, D Clark, K Androutsopolous, and P Kan. Invariant-based synthesis of fault-tolerant systems. In *FTRTFT*. Springer-Verlag, 2000.

[12] K Lano, D Clark, and K Androutsopolous. From implicit specifications to explicit designs in reactive system development. In *IFM '02*, 2002.

[13] K. Lano, D. Clark, K. Androutsopoulos, *UML to B: Formal Verification of Object-oriented Models*, IFM 2004.

[14] K. Lano, *Constraint-Driven Development*, to appear in Information and Software Technology, 2007.

[15] K. Lano, *A Compositional Semantics of UML-RSDS*, submitted to SoSyM, 2006.

[16] D. Le, E. Sekerinski, S. West, *Statechart Verification with iState*, FM 06, Canada, 2006.

[17] J. Lilius, I. Paltor, *The Semantics of UML State Machines*, Turku Centre for Computer Science, TUCS technical report 273, 1999.

[18] B Liskov and J Wing. Specifications and their use in defining subtypes. In *ZUM '95 Proceedings*, volume 967 of *LNCS*. Springer-Verlag, 1995.

[19] J. Merseguer, J. Campos, S. Bernardi, S. Donatelli, *A compositional semantics for UML state machines aimed at performance evaluation.* In M. Silva, A. Giua, and J.M. Colom, editors, Proc. of the 6 Int. Workshop on Discrete Event Systems (WODES 2002), 2002.

[20] OMG. UML superstructure, version 2.0. OMG document formal/05-07-04, 2005.

[21] A Simons. A theory of regression testing for behaviourally compatible object types. In *3rd Conf. UK Software Testing Research (5-6 September)*, pages 103–121, 2005.

ABOUT THE AUTHORS

Dr Kevin Lano Kevin Lano is a Reader in Software Engineering in the Department of Computer Science at King's College London. He has been involved in the combination of object-oriented and formal methods for many years, in the EROS and pUML groups, and in many UK and European projects. He can be contacted at Kevin.Lano@kcl.ac.uk

Dr David Clark David Clark is a lecturer in the Department of Computer Science at King's College London. He is a member of the Software Systems and Modelling Team within the department's Software Engineering Research Group. He may be contacted at David.J.Clark@kcl.ac.uk. See also http://www.dcs.kcl.ac.uk/staff/david/