2002

TR-2002009: A Formal Semantics for UML with Real-Time Constructs

Subash Shankar

A Formal Semantics for UML with Real-Time Constructs

Subash Shankar
Hunter College and the Graduate Center,
City University of New York (CUNY),
695 Park Ave.
New York, NY 10021, USA,
subash.shankar@hunter.cuny.edu

Abstract

This paper describes a formal framework for expressing the semantics of UML augmented with real-time constructs. The formalized aspects of UML consist of concurrent statecharts to represent the dynamic behavior of objects along with interaction diagrams to represent inter-object communication with real-time constraints. The approach is based on a two-dimensional temporal logic to independently capture flow of control as well as flow of time. The paper defines this logic, shows how the semantics can be captured in this logic, and outlines techniques for using these semantics for formal verification. The goal is to provide a simple, intuitive, and validatable semantics that can be used for further formal analysis.

1 Introduction

Formal methods have successfully been applied in many domains including hardware, protocols, and to a lesser degree, software. The goals of formal methods are not limited to analysis and verification, as the process of formalization itself often increases the quality of a design. Additionally, a formal model of linguistic constructs often results in a better understanding of the constructs and sometimes leads to improved language design. However, many formal notations are too complicated or are tightly linked with an underlying verification system, thus making them less applicable than a more simple and intuitive notation.

The Unified Modeling Language (UML) is a semi-formal language that is the defacto standard for expressing aspects of object-oriented software. In particular, UML can be used to model the structure and behavior of object-oriented software diagrammatically. It contains four types of diagrams to model the behavior of objects: statecharts, activity diagrams, and two different forms of interaction diagrams (sequence and collaboration diagrams). Two important

aspects of object behavior are the state transitions that an individual object goes through, and inter-object communication through message passing. Although these can be modeled in several different ways in UML, one direct and intuitive method is to use statecharts for object behavior and sequence diagrams for inter-object communication, and relating the two diagrams on the statechart by using event variables corresponding to messages.

Although the UML has been widely used for modeling traditional object-oriented systems, it does not directly support the modeling of real-time systems. The primary problem is the lack of constructs to express time and time-related properties. The Object Management Group (OMG) has recently produced a draft specification for augmenting UML with support for various system issues including concurrency, schedulability, and time [Obj02]. The specification includes a common model of resources (the "general resource model"), along with a number of derived domain-specific profiles for each of these classes of issues. These profiles are intended to be used independently; for example, a modeler interested in concurrency and real-time issues may include the concurrency and real-time profiles, but exclude the other profiles. The real-time profile includes (among other things) support for timing mechanisms (clocks for measuring elapsed time and timers for timeouts), and several new timing marks for time-related message attributes (*e.g.*, the time a message event is sent, the time an action begins execution in response to a message). UML diagrams may then be annotated with these timing marks along with constraints on these marks, with sequence diagrams being particularly well-matched for representing timing relationships pictorially. However, like the rest of UML, there are no formal semantics for these new real-time constructs.

The main problem in formal modeling, analysis, and verification of UML designs is the lack of a formal semantics. There have been numerous attempts to provide formal semantics for each type of UML diagram, with statecharts receiving particular attention. However, there are numerous semantic ambiguities that complicate the problem; in fact, 21 statechart variants (including some with time constructs predating the real-time profile) are discussed in [vdB94], each corresponding to a different semantics. One approach to formal semantics might select one statechart variant along with a set of profiles, and provide formal semantics for such a combination. Of course, a different semantics is then needed for each such combination, thus leading to combinatorial explosion in the number of semantics. An alternative approach is to show how to extend any semantics with real-time constructs.

This paper takes the latter approach, with the following primary goals:

- Independence of the semantics of time-related constructs from the semantics of the underlying UML diagram
- Simplicity of formalism, for understandability and to allow for independent semantic variation in UML subcomponents
- Relevance and conciseness of the formal model, by avoiding the modeling of language artifacts that are not apparent at the specification level (*e.g.*,

an underlying execution engine that can be used to animate the model)

- Intuitive nature of formalism, to allow for validation of the semantics themselves (against a very informal English model, or against a semi-formal UML specification)

- Closeness of formalism to languages that can be used to express typical system specification criteria, such as performance, liveness, and safety

Temporal logics are the predominant logical formalism for specification of program properties (see, for example, [MP92] for representing liveness and safety properties using linear-time temporal logic, and [CES86] for representing and verifying such properties using model checking). A [linear-time] propositional temporal logic (PTL) formula corresponds essentially to a state machine, and it is thus conceptually simple to model statecharts using PTL by mapping state transitions in statecharts onto time transitions in the temporal logic. For example, a transition from state s_0 to state s_1 can be modeled using a PTL formula whose English reading is: 'if the system is currently is in state s_0, then it will be in state s_1 at the next time'. A similar technique has been used by [Kro87] to model a generic concurrent language using temporal logic. In the UML world, [LQV01] uses such a technique to model UML statecharts using first order temporal logic; however, this technique needs to restrict all state transitions to be associated with times, rather than the general form of instantaneous as well as timed transitions supported by statecharts.

There are several complications that arise when attempting to use PTL for modeling general time-constrained inter-object communication. For example, if some statechart transitions are instantaneous and another transition is dependent on an event from another object that requires 5 time units to transmit, it is not possible to directly capture the transmission time. The fundamental problem is that there are two distinct notions of flow: the flow of micro-time as objects make [typically] instantaneous state transitions, and the flow of macro-time as objects communicate with each other. The presence of these two distinct time modalities is further complicated by UML constructs that rely on both modalities (for example, when an otherwise instantaneous transition must wait for an incoming event from another object).

A recent trend in logics is the use of polymodal logics, which combine multiple modalities to provide logics capable of expressing properties without resorting to a more complicated first order or higher order logic. Several such logics and combination techniques have been proposed (see for example [FG92, FG96, KW91, BR97]). Two particularly common applications are logics combining time and belief modalities to model multi-agent systems where beliefs change over time, and logics combining transaction-time and valid-time modalities to model the evolution of temporal databases. This paper shows a new application of polymodal logics, by introducing a two-dimensional temporal logic that independently captures micro-time (referred to as state), as well as macro-time (referred to as time). The logic is a polymodal logic that is essentially the independent fusion of two PTLs.

The paper is organized as follows. Section 2 outlines the UML features that we intend to formalize. Section 3 introduces the formal syntax and semantics of the two-dimensional logic, and Section 4 then shows how to capture UML semantics in this logic. There are several useful applications of the resulting semantics, and Section 5 discusses the particularly useful verification application. Finally, Section 6 summarizes the paper and outlines future research and conclusions.

2 The UML Variant

As mentioned earlier, this paper uses statecharts to model the states that an object transitions through, and sequence diagrams to model inter-object messages. Our primary goal in this paper is to show how the formal semantics of such a model can be extended to cover some real-time constructs similar to those in the UML real-time profile. In particular, our goal is not to provide yet another statechart variant with a new semantics. Thus, we first stipulate one simple statechart variant in this section.

A statechart contains a number of states, along with transitions between these states. This paper assumes that the statechart has been flattened to eliminate hierarchy, and also avoids discussion of the numerous semantic issues resulting from ambiguities in statechart semantics (the reader is referred to [vdB94] for discussion of these issues). Suppose there is a transition, t, from state s_1 to s_2, which is labeled by $ev[c]/a$. Then, if the object is in state s_1 and the trigger event ev is received while the guard condition c is satisfied, the object emits event a and transitions to state s_2. The guard condition is evaluated only once when the trigger occurs, and the transition is not taken if it is false on that evaluation - ev needs to be retriggered for the guard to be evaluated again. Harel's original definition of statecharts ([Har87]) and most statechart variants assume the *perfect synchrony hypothesis*, which essentially states that the transition occurs instantaneously. This hypothesis is intended to capture the relatively fast execution of a program so that it reacts to events fast enough to make all necessary state transitions before the next event occurs. We denote such transitions as state transitions (to distinguish them from time transitions).

Sequence diagrams are used to represent inter-object events. We do not distinguish events from messages. Each triggering event, ev_i, and corresponding action is associated with four timing marks:

- sendTime ($send_i$): the time event i was sent by the sender
- receiveTime (rec_i): the time the event message was received by the receiver
- startTime ($start_i$): the time the execution of the corresponding action commenced
- endTime (end_i): the time the execution of the corresponding action ended

These are similar to the timing marks provided by the UML real-time profile. Time is assumed to be discrete here.

The perfect synchrony hypothesis ensures that transitions occur instantaneously (unless otherwise stated), while constraints based on timing marks may be used for transitions that require time. Although [plain] UML allows events to correspond to the passing of time, the association of time with state transitions is often too simple a model for real-time applications, and constraints on timing marks are needed. For example, it might be desirable to model the constraint that the difference between a message's sendTime and its receiveTime is less than 5, and this can be done by adding a timing constraint to the corresponding sequence diagram. Thus, there are three types of transitions from a timing perspective: instantaneous, timed where the time is a function of the transition, and timed where the time is derived from sequence diagram annotations for the triggering event. This paper denotes the micro-time instantaneous transitions as state transitions, and the macro-time timed transitions as time transitions.

3 The Two-Dimensional Temporal Logic $\mathcal{L}_2$

This section defines the syntax and semantics of the two-dimensional temporal logic, $\mathcal{L}_2$, which is essentially an independent fusion of two traditional linear-time propositional temporal logics (PTLs).

3.1 Syntax

The language of PTL consists of a countable number of quantifier-free predicates[1], $\mathcal{A}$, the operators of traditional propositional logic: $\top$, $\bot$, $\neg$ (not), $\wedge$ (and), $\vee$ (or), $\rightarrow$ (implication), and $\leftrightarrow$ (iff), and the temporal operators: $\circ$ (next),$\Box$ (henceforth, or always), $\Diamond$ (eventually), and $\mathcal{U}$ (strong until). The $\circ$, $\Box$, and $\Diamond$ operators are unary future-only operators, while the other temporal operators are dyadic future-only operators. As usual, the $\mathcal{U}$ operator is the strong version, which asserts that the second operand does indeed hold at the current or some future time. Only the $\mathcal{U}$ and $\circ$ operators are actual parts of the logic, as other temporal operators are in reality merely abbreviations:

- $\Diamond P \equiv \top \mathcal{U} P$
- $\Box P \equiv \neg \Diamond \neg P$

However, we define the logic to include the complete set of operators for simplicity. For notational simplicity, we also impose bounds on temporal operators by affixing an interval as a superscript; for example, $\Box^{[2,10]}$ is read as 'all times between 2 and 10 time units (inclusive) from now', and $\Diamond^{[0,3]}$ is read 'within 3 time units (inclusive) from now'. These are purely syntactic sugar since equivalent formulas may be expressed as conjunctions or disjunctions of plain PTL

[1] It is technically incorrect to refer to this logic as propositional since predicates may include function and relation symbols. However, these symbols may be considered to be uninterpreted for the purposes of this paper.

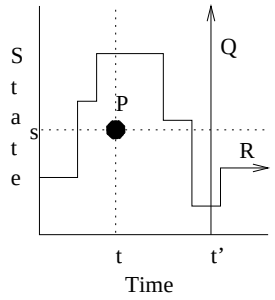

Figure 1: Two-dimensional space for $\mathcal{L}_2$

formulas. Similarly, iterated unary operators are represented using superscripts - for example, $\circ^n$ indicates n $\circ$ symbols.

In PTL, time is a sequence of positive integers, with propositions taking on values that may be different at each timepoint. In the two-dimensional logic $\mathcal{L}_2$, propositions take on values that may be different at each (time, state) pair. For example, Figure 1 shows the grid over which predicates range, along with 3 predicates on this grid. States in this logic are local to timepoints - that is, each timepoint contains its own full complement of states. $\mathcal{L}_2$ contains a full set of temporal operators for states as well as timepoints: $\overline{\circ}$, $\overline{\Box}$, $\overline{\Diamond}$, $\overline{\mathcal{U}}$, $\widehat{\circ}$, $\widehat{\Box}$, $\widehat{\Diamond}$, and $\widehat{\mathcal{U}}$. Time and state operators are distinguished by having either carets or bars, respectively.

3.2 Semantics

As suggested by Figure 1, a model for $\mathcal{L}_2$ is given by a 5-tuple $\langle\ T,\ t_0,\ S,\ s_0,\ \eta\ \rangle$ where:

- T is a [countably infinite] sequence of timepoints with initial time t_0
- S is a sequence of [countably infinite] states with initial state s_0
- η is a set of mappings $\eta_{i,j}$ where each $\eta_{i,j}$ is a valuation function assigning Boolean values to the elements of $\mathcal{A}$ at time i and state j.

Where obvious from context, state indices may be used instead of the state itself, and likewise for time (*e.g.*, states 2 and s_2 are treated synonymously).

Figure 2 gives the model-theoretic semantics of $\mathcal{L}_2$ (including abbreviations). It is similar to PTL semantics; note, in particular, that the operators are reflexive, and both the $\overline{\mathcal{U}}$ and $\widehat{\mathcal{U}}$ operators are the strong versions, which assert that their second operands eventually become true.

Satisfiability and validity in $\mathcal{L}_2$ are defined using an anchored notion rather than a floating notion (that is, all formulas are expressed with respect to time and state 0): A $\mathcal{L}_2$ formula F is said to be *satisfiable* if there is some model $\mathcal{M}$ such that $\mathcal{M}, t_0, s_0 \models F$. Similarly, a $\mathcal{L}_2$ formula F is said to be *valid* if $\mathcal{M}, t_0, s_0 \models F$ for every model $\mathcal{M}$. For example, Figure 1 shows a model

$$
\begin{array}{lll}
\mathcal{M}, t, s \models \top & & \\
\mathcal{M}, t, s \not\models \bot & & \\
\mathcal{M}, t, s \models P & \text{iff} & \eta_{t,s}(P) = t \text{ for } P \in \mathcal{A} \\
\mathcal{M}, t, s \models \neg A & \text{iff} & \mathcal{M}, t, s \not\models A \\
\mathcal{M}, t, s \models A \wedge B & \text{iff} & \mathcal{M}, t, s \models A \text{ and } \mathcal{M}, t, s \models B \\
\mathcal{M}, t, s \models A \vee B & \text{iff} & \mathcal{M}, t, s \models A \text{ or } \mathcal{M}, t, s \models B \\
\mathcal{M}, t, s \models A \rightarrow B & \text{iff} & \mathcal{M}, t, s \models \neg A \vee B \\
\mathcal{M}, t, s \models A \leftrightarrow B & \text{iff} & \mathcal{M}, t, s \models (A \rightarrow B) \wedge (B \rightarrow A) \\
\mathcal{M}, t, s \models \widehat{\circ} A & \text{iff} & \mathcal{M}, t+1, s \models A \\
\mathcal{M}, t, s \models \widehat{\Box} A & \text{iff} & \mathcal{M}, t', s \models A \text{ for every } t' \geq t \\
\mathcal{M}, t, s \models \widehat{\Diamond} A & \text{iff} & \mathcal{M}, t', s \models A \text{ for some } t' \geq t \\
\mathcal{M}, t, s \models A \widehat{\mathcal{U}} B & \text{iff} & \mathcal{M}, t', s \models B \text{ for some } t' \geq t \text{ and} \\
 & & \mathcal{M}, t'', s \models A \text{ for every } t'' \text{ such that } t \leq t'' < t' \\
\mathcal{M}, t, s \models \overline{\circ} A & \text{iff} & \mathcal{M}, t, s+1 \models A \\
\mathcal{M}, t, s \models \overline{\Box} A & \text{iff} & \mathcal{M}, t, s' \models A \text{ for every } s' \geq s \\
\mathcal{M}, t, s \models \overline{\Diamond} A & \text{iff} & \mathcal{M}, t, s' \models A \text{ for some } s' \geq s \\
\mathcal{M}, t, s \models A \overline{\mathcal{U}} B & \text{iff} & \mathcal{M}, t, s' \models B \text{ for some } s' \geq s \text{ and} \\
 & & \mathcal{M}, t, s'' \models A \text{ for every } s'' \text{ such that } s \leq s'' < s'
\end{array}
$$

Figure 2: $\mathcal{L}_2$ Semantics

satisfying the following formulas: $\widehat{\circ}^t \overline{\circ}^s P$, $\widehat{\circ}^{t'} \overline{\Box} Q$, and $\widehat{\Box} \overline{\Diamond} R$. An example of a valid formula is: $\widehat{\Box}(\overline{\Box} P \rightarrow \overline{\Diamond} P)$.

4 Translating the UML Model into $\mathcal{L}_2$

It is conceptually simple to translate statecharts into PTL. We start with an imprecise but descriptive example. The transition, t, from state s_1 to s_2 and labeled by $ev_i[c_j]/ev_k$, can be translated to:

$$\Box((at_1 \wedge occur_i \wedge c_j) \rightarrow (occur_k \wedge \circ at_2))$$

where at_i holds iff the system is currently in state s_i, and the $occur_i$ and $occur_k$ variables correspond to the occurrence of the transition events ev_i and ev_k respectively. As with most axioms, the above axiom states that if a set of preconditions is met, then the appropriate events are triggered and the transition is taken on the next cycle. As mentioned earlier, the problem occurs when attempting to capture mixed modalities where there is a time component to the events. Since the state transitions in the UML model are mapped onto time transitions in the temporal logic, it is not possible to represent both state- and time- transitions.

Suppose the event ev_k requires 2 time units to be sent. Then, by going to a

Variables	Domain	English Reading
$at_{i,j}$	boolean	State machine i is in state j
$occur_i$	boolean	Event i has occurred
$send_i$	integer	The time that message i was sent
rec_i	integer	The time that message i was received
$start_i$	integer	The time that the action for message i started
end_i	integer	The time that the action for message i finishes

Figure 3: Variables in TRL's Formal Semantics

two-dimensional temporal logic, the above transition can be translated to:

$$\hat{\Box}(\bar{\Diamond}(at_1 \wedge occur_i \wedge c_j) \rightarrow (\hat{\circ}^2 occur_k))$$

$$\hat{\Box}\bar{\Box}((at_1 \wedge occur_i \wedge c_j) \rightarrow \bar{\circ} at_2)$$

Note that the axiom states that the $occur_k$ variable becomes true in state 0 2 time units into the future, rather than the current state. In contrast, the transition to state s_2 occurs on the following state in the current timepoint. Although these particular axioms are imprecise and included solely for explanatory reasons, they form the two fundamental structural forms in the translations below.

Before defining the semantics, we first define the set of variables that are used in the semantics. Figure 4 lists these variables, along with their English readings. All boolean variables vary over both state and time. We also denote by $n_{\mathcal{S}}$ the number of statecharts in the model.

There are several commonly used formulas that are simpler to define as macros. The *step* macro defines transitioning from state s_j to s_k in statechart i:

$$step(i, j, k) \quad \equiv \quad at_{i,j} \bar{\mathcal{U}} at_{i,k}$$

Note that the transition need not necessarily occur on the next state; however, since the $\bar{\mathcal{U}}$ operator is strong, the transition will eventually occur (at the current timepoint).

The *enterState* macro defines the first timepoint at which a state is reached - that is, enterState(i,j) holds at time-state pair $(t, 0)$ if statechart i transitions into state s_j at some state in time t. This occurs either when transitioning to s_j at some non-negative state, or transitioning to s_j at state s_0 of some timepoint where the statechart was not in s_j at the end of the previous timepoint:

$$\begin{aligned} \hat{\Box}(\hat{\circ} enterState(i, j) \quad &\leftrightarrow \quad (\hat{\circ}(\neg at_{i,j} \wedge \bar{\Diamond} at_{i,j}) \vee (\bar{\Diamond}\bar{\Box}\neg at_{i,j} \wedge \hat{\circ} at_{i,j})) \\ enterState(i, j) \quad &\leftrightarrow \quad \bar{\Diamond} at_{i,j} \end{aligned}$$

These two axioms capture the meaning of *enterState* for the positive-time and zero-time cases.

The translation of UML into $\mathcal{L}_2$ is decomposed into two parts: global and local formulas. Whereas global formulas correspond to properties of all state machines, local formulas correspond to individual transitions in the model.

Global Formulas

There are three classes of global formulas: initial state, final state, and control flow. The initial state formula specifies that all statecharts start in their initial state:

$$\bigwedge_{i=1}^{n_{\mathcal{S}}} at_{i,0} \tag{1}$$

where the initial state of each statechart is labeled as state 0. Recall that all formulas implicitly refer to time and state 0.

Similarly, the final state global formula ensures that once a statechart transitions to its final state (denoted s_f), it does not restart.

$$\bigwedge_{i=1}^{n_{\mathcal{S}}} \hat{\Box}\bar{\Box}(at_{i,f} \rightarrow \bar{\circ} at_{i,f}) \tag{2}$$

$$\bigwedge_{i=1}^{n_{\mathcal{S}}} \hat{\Box}(\bar{\Diamond} at_{i,f} \rightarrow \hat{\circ} at_{i,f}) \tag{3}$$

Control flow global formulas ensure that illegal control flows do not occur. A statechart can not be in two states at once:

$$\hat{\Box}\bar{\Box}(at_{i,j} \rightarrow \neg at_{i,k}) \tag{4}$$

for any i such that $1 \leq i \leq n_{\mathcal{S}}$ and for all j, k, such that $j \neq k$.

If a statechart is in a state s_i and waiting for a trigger, it does not change states except when the trigger occurs. The actual state transition is modeled through the local formulas for the transition, while the global formula ensures that the transition does not occur on time changes.

$$\hat{\Box}(\bar{\Diamond}\bar{\Box} at_{i,j} \rightarrow \hat{\circ} at_{i,j}) \tag{5}$$

for each i such that $1 \leq i \leq n_{\mathcal{S}}$ and for all j where the transition to state s_j of statechart i is non-timed.

Local Formulas

For each transition in the model, there are two components to its formalization: the actual state transition, and the actions performed on the transition. This section first formalizes the actual transitions for three types of transitions: triggerless, time-triggered (*i.e.*, timeout), and regular transitions with triggers, guards, and actions. In all three cases, the statechart is assumed to be deterministic, either by having only one outgoing transition from each state or by having mutually exclusive guard conditions on outgoing transitions (the determinism assumption is relaxed later). Then, the actions (for all three types) are formalized.

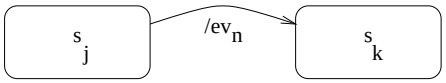

Figure 4: Example triggerless transition

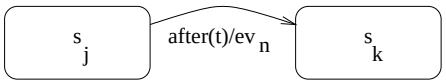

Figure 5: Example Time-Triggered Transition

Consider the triggerless transition of Figure 4. The formalization of this transition is simple:

$$\hat{\Box}\bar{\Box}(at_{i,j} \rightarrow step(i,j,k)) \tag{6}$$

Since multiple statecharts may be running in parallel, the formalization does not specify the exact state in which the transition occurs, and no restrictions are imposed on statechart execution (for example, concurrent statecharts may execute synchronously, be arbitrarily interleaved, or in any other manner). This assumption is discussed below in more detail.

The second type of transition is the time-triggered transition illustrated in Figure 5. In this case, statechart i needs to stay in state s_j for the remaining states in the current time point and all states in the next $t-1$ timepoints. This is formalized by:

$$\hat{\Box}\bar{\Box}(at_{i,j} \rightarrow \bar{\Box} at_{i,j}) \tag{7}$$

$$\hat{\Box}(enterState(i,j) \rightarrow ((\bigwedge_{p=1}^{t-1} \hat{\circ}^p \bar{\Box} at_{i,j}) \wedge \hat{\circ}^t at_{i,k})) \tag{8}$$

Recall that the $enterState(i,j)$ formula was defined earlier to essentially capture the first timepoint at which state s_j is reached.

Now, consider the full transition of Figure 6. The transition may occur only if the trigger ev_m occurs and c holds, at which point the the transition may be immediately taken:

$$\hat{\Box}\bar{\Box}((at_{i,j} \wedge occur_m \wedge c) \rightarrow step(i,j,k)) \tag{9}$$

In addition to taking the transition (for all three types of transitions), the statechart must also perform an operation and emit event ev_n, possibly subject

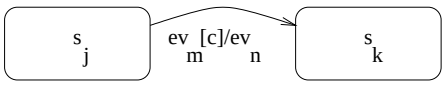

Figure 6: Example (Regular) Transition

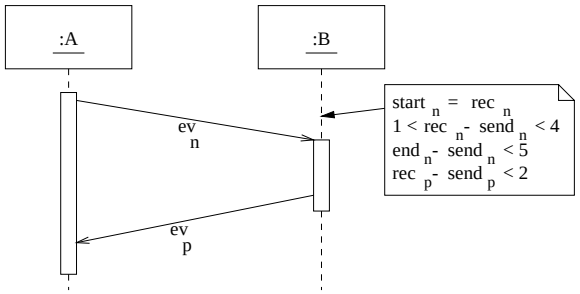

Figure 7: Example Sequence Diagram with Real-Time Constraints

to real-time constraints. UML allows for various types of actions, including return values, create, and destroy actions; we assume here that an action is an event message to an existing object, and other types of actions need to be first converted to this simplified action notion. We also assume that all actions are associated with real-time constraints.

Section 2 listed four timing marks associated with each message: sendTime, receiveTime, startTime, and endTime. Suppose event ev_n and its corresponding action have the following values for these timing marks: $send_n$, rec_n, $start_n$, and end_n, and suppose the action results in the generation of ev_p. Then the generation of events is formalized by:

$$\Box(enterState(i,k) \leftrightarrow (\hat{\circ}^{start_n - send_n} occur_n \wedge \hat{\circ}^{rec_p - send_n} occur_p)) \qquad (10)$$

Although this axiom only applies to events associated with real-time constraints, it is obvious how to generate axioms for instantaneous events.

Note that a statechart may impose a different constraint between ev_n and ev_p, and it may thus be desirable to prove consistency between these two diagrams. If the model does not specify specific values for the timing marks and uses constraints instead, (10) is modified. For example, if the model includes the constraints represented on the sequence diagram of Figure 7, then these constraints are incorporated into the formalization by modifying (10) to:

$$\Box(enterState(i,k) \leftrightarrow (\hat{\Diamond}^{[2,3]} occur_n \wedge \hat{\Diamond}^{[2,5]} occur_p)) \qquad (10)$$

In this case, a desirable property may be that the specified end time, end_n, follows $start_n$, and this property may be expressed as another formula that needs to be proven from the axioms representing the formalization.

Semantic Variations

As mentioned earlier, the main goal of this paper is to show how real-time constructs can be formalized, rather than to provide a formalization of a particular statechart variant. Thus, it is desirable to show how other statechart variants can be modeled. First, we relax the determinism assumption. Suppose state s_j

has multiple outgoing transitions that are not mutually exclusive, and the statechart must non-deterministically choose one of these. If none of the transitions is time-triggered, this is trivial to formalize, since we simply replace the right side of the implications with the disjunction of the translations for each of the transitions. In cases where exactly one outgoing transition is time-triggered (*i.e.*, it acts as a timeout), additional preconditions stating that the other transitions have not been taken are added to (7) and (8). The case where more than one outgoing transition is time-triggered results in a new set of ambiguities which is beyond the scope of this paper.

All formalizations of the transitions allowed for arbitrary interleaving of concurrent statecharts. However, if the statechart variant requires that transitions occur immediately (as is often the case), (6) and (9) still hold if the *step* macro is redefined to:

$$step(i,j,k) \quad \equiv \quad \overline{\circ}at_{i,k}$$

Alternatively, it may be desirable to keep the original definition of *step*, and prove that the resulting model is deterministic regardless of interleaving by specifying a determinism property in $\mathcal{L}_2$. Several other semantic variations of statecharts, including ambiguities related to self-triggers, may also be realized by modifying the *step* macro accordingly.

Our model also assumed that all inter-object events are associated with real-time constraints. If this is not the case, (10) needs to be modified to deal with events that occur at the same time-point but a different state than at which the state transition was taken. There are several ambiguities in statechart semantics that deal with when exactly such events occur; however, it is simple to express appropriate axioms once these ambiguities are resolved and a particular statechart variant is selected.

Our model of real-time only allowed certain forms of time constraints. We restricted the form of constraints to make the resulting axioms decidable (under certain restrictions to be discussed later). Although these constraints are normally sufficient, more general form of constraints can be supported by introducing theories of arithmetic as needed to represent and solve such constraints. We are currently implementing the axioms of this paper and such constraints using the PVS theorem prover. Another extension of real-time modeling involves continuous time. It is conceptually simple to vary $\mathcal{L}_2$ to support continuous time over the time axis, and generate a continuous-time semantics; however, the resulting logic is difficult to work with for applications such as verification.

The key point of our approach is that real-time issues are isolated from other issues. For example, many semantic variations of state charts can be formalized without affecting the real-time axioms (10). Given local formulas for each transition in the statechart, the resulting collection of global and local formulas may then be used for further analysis. The next section outlines how these axioms may be used for one such application: formal verification. We believe that these axioms are simple and intuitive enough to also support a number of other applications.

5 Verification

Before discussing how to verify real-time UML models, we first need to discuss what to verify.

Properties

The classical verification properties of safety ('P never occurs', where P is an undesirable property) and liveness ('P eventually occurs', where P is a desirable property) are of course still useful. However, there are two special cases to consider depending on whether instantaneous transitions are to be considered. In applications where both instantaneous and timed transitions are important, safety and liveness properties are expressed as $\hat{\Box}\bar{\Box}\neg P$ and $\hat{\Diamond}\bar{\Diamond}P$ respectively. In some applications, we are only concerned with observable behavior, and instantaneous transitions are not observable. In such cases, the properties only specify what holds in state 0 of each timepoint. Thus, safety and liveness properties for these applications are expressed as $\hat{\Box}\neg P$ and $\hat{\Diamond}P$ respectively.

Another set of real-time properties deal with performance. These properties typically state that a certain event or action occurs within t timepoints after another event. Such properties are simple to model in the same way as for safety and liveness properties, as long as care is taken to define exactly which model artifacts are observable.

A third class of properties deal with the validity of the model itself. For example, we may wish to show consistency properties between a statechart and a sequence diagram (*e.g.*, the timing constraints on the sequence diagram must be realizable with the corresponding statechart). As another example, we may wish to show that a certain apparently non-deterministic UML model is in reality deterministic. Several such examples were briefly mentioned in Section 4.

Reasoning

The logic $\mathcal{L}_2$ is undecidable, since [Har83] shows that a simpler logic is Σ_1^1-hard. However, there are several useful decidable restrictions of $\mathcal{L}_2$. In particular, if the number of transitions in each timepoint is bounded, the logic then becomes decidable. A sufficient condition on statecharts to ensure boundedness is that there are no cycles in instantaneous transitions, and this is a reasonable restriction (in fact, many statecharts variants make a stronger restriction to address well-known causality issues). If such a boundedness constraint is imposed, it is possible to fold the two-dimensional model onto a one-dimensional time-line extending infinitely into the future, as shown in Figure 8. In prior work ([Sha98]), we have shown a scheme for automatically translating $\mathcal{L}_2$ formulas into the decidable logic PTL using this folding.

We are currently implementing two tools to support verification. First, we are implementing a decision procedure that translates $\mathcal{L}_2$ formulas to PTL (under the boundedness constraint mentioned above) and then applies traditional

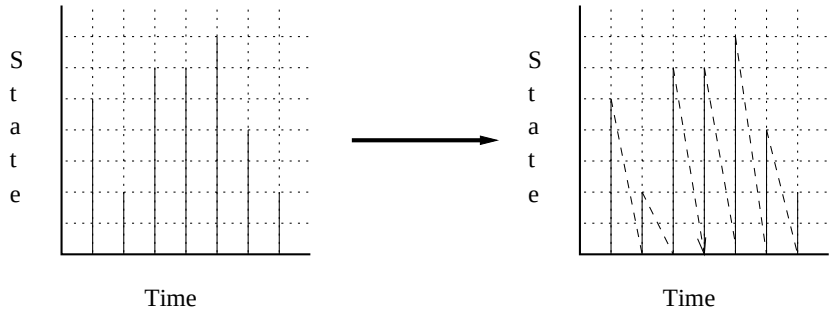

Figure 8: Mapping the $\mathcal{L}_2$ grid onto a PTL line

tableau-based decision procedures for PTL. Second, we are implementing strategies for directly reasoning about $\mathcal{L}_2$ statements using the PVS theorem prover.

6 Further Research and Conclusions

As listed in Section 1, the primary goal of our approach was to devise a simple and intuitive formalism that independently captures the semantics of micro-time statechart transitions and macro-time constructs similar to those in the emerging real-time profile. We believe that $\mathcal{L}_2$ is well suited towards these goals, as shown by the relative conciseness, simplicity, and independence of concerns in the axioms of Section 4. Simplicity and intuition are, of course, in the eye of the beholder; however, we have provided a similar formalization of VHDL in prior work ([SS97]), and been able to validate the correctness of the semantics against the Language Reference Manual (chapter 4 of [Sha98]).

There are a number of statecharts semantics in the literature, with many of them providing a detailed operational semantics that essentially translates statecharts into state transition machines (*i.e.*, a Kripke model). If these state machines are then integrated with real-time constructs from sequence diagrams, it is simple to generate the semantics of the resulting model using the two-dimensional approach of this paper. Thus, we believe that our independence goal is also met.

To the best of our knowledge, the closest related work is by [LQV01], which models UML statecharts using a first order temporal logic. However, since that approach must associate a time with every transition, it is forced to disallow instantaneous transitions. Conversely, the use of a first order temporal logic allows for directly modeling continuous time, though the price for this greater expressive power is undecidability. In the non-UML world, the work of [Kro87], which provides a semantics of a generic concurrent language using temporal logic, is also closely related to our approach. However, the approach there maps control flow through the concurrent program onto time flow in the temporal logic, thus not allowing for an independent time flow. An alternative approach is to use timed automata ([Alu99]). We believe that the use of a two-dimensional

logic, though not as general, better satisfies the goals of real-time UML.

We are currently pursuing further research along several lines. First, as mentioned in Section 5, we are implementing proof procedures for $\mathcal{L}_2$, and plan on using the resulting systems to prove properties of real-time UML diagrams. Second, we are extending our semantics to cover other real-time constructs in the real-time profile, such as the attributes of timing mechanisms (*e.g.*, clock skew, drift). Finally, we plan on applying our proof procedures on several real-time applications expressed using UML.

References

[Alu99] Rajeev Alur. Timed automata. In *International Conference on Computer-Aided Verification (CAV)*, pages 8–22, 1999.

[BR97] P. Blackburn and M. De Rijke. Why combine logics? *Studia Logica*, 59:5–27, 1997.

[CES86] E. M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 8(2):244–263, April 1986.

[FG92] M. Finger and D.M. Gabbay. Adding a temporal dimension to a logic. *Journal of Logic Language and Information*, 1:203–233, 1992.

[FG96] Marcelo Finger and Dov Gabbay. Combining temporal logic systems. *Notre Dame Journal of Formal Logic, Special Issue on Combining Logics*, 37(2):204–232, Spring 1996.

[Har83] David Harel. Recurring dominoes: Making the highly undecidable highly understandable. In *Conference on Foundations of Computation Theory*, pages 177–194, 1983.

[Har87] David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.

[Kro87] F. Kroeger. *Temporal Logic of Programs.* Sprinter Verlag, 1987.

[KW91] Marcus Kracht and Frank Wolter. Properties of independently axiomatizable bimodal logics. *Journal of Symbolic Logic*, 56(4):1469–1485, December 1991.

[LQV01] Luigi Lavazza, Gabriele Quaroni, and Matteo Venturelli. Combining UML and formal notations for modelling real-time systems. In *Joint European Software Engineering Conference (ESEC) and International Symposium on the Foundations of Software Engineering (FSE)*, pages 196–206, 2001.

[MP92] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Sprinter Verlag, 1992.

[Obj02] Object Management Group. *UML Profile for Schedulability, Performance, and Time Specification, Draft Adopted Specification*, January 2002.

[Sha98] Subash Shankar. *Formal Verification of VHDL Designs Using Temporal Logics*. PhD thesis, University of Minnesota, 1998.

[SS97] S. Shankar and J. Slagle. A polymodal semantics for VHDL. In *Advances in Hardware Design and Verification (CHARME)*, pages 88–105. Chapman & Hall, 1997.

[vdB94] Michael von der Beeck. A comparison of statecharts variants. In *Formal Techniques in Real Time and Fault Tolerant Systems (FTRTFT)*, pages 128–148, 1994.