Capturing an Application's Temporal Properties with UML for Real-Time

Weiguo He Steve Goddard
Computer Science & Engineering
University of Nebraska - Lincoln
Lincoln, NE 68588-0115
{whe, goddard}@cse.unl.edu

Abstract

The Unified Modeling Language (UML) is commonly used in the development of non-real-time systems and is gaining popularity in the object-oriented real-time community as well. Recently, Rational Software Corporation teamed with ObjecTime, Ltd to develop UML for Real-Time (UML-RT). UML-RT uses the extensibility mechanisms of UML to incorporate concepts from ROOM (Real-time Object-Oriented Modeling language) and Role modeling from ObjecTime. We present a novel method for capturing the temporal parameters of a real-time application in a UML-RT model so that schedulability analysis can be performed.

1. Introduction

Real-time software systems are widely applied in domains such as telecommunications, aerospace, defense, and automatic control applications. High assurance systems engineering of these systems is critical since even a small malfunction has serious consequences. Thus, methods for developing and modeling real-time systems and rigorously verifying behavior before committing to code are of great interest to many researchers [13].

The Unified Modeling Language (UML) is commonly used in the development of non-real-time systems and is gaining popularity in the object-oriented real-time community as well [20]. Recently, Rational Software Corporation teamed with ObjecTime, Ltd to develop UML for Real-Time (UML-RT) [12]. UML-RT uses the extensibility mechanisms of UML to incorporate concepts from ROOM (Real-time Object-Oriented Modeling language) and Role modeling from ObjecTime [21]. It can be used to help capture and understand the structural and behavioral patterns on which all other aspects of the system depend. Moreover, UML-RT is extremely useful in modeling the *reactive* nature of many real-time systems.

Two temporal parameters, release times and deadlines, often distinguish real-time systems from non-real-time systems. In general, a constraint imposed on the timing behavior of a job is called a timing constraint. If the validation that a real-time system always meets the timing constraint is required, we call this system a hard real-time system. Validation has normally meant a demonstration by a provably correct, efficient procedure or by exhaustive simulation and testing. Because of the criticality of real-time systems, efficient validation algorithms and methods as well as scheduling and resource management strategies are generally regarded as essential content in this area [11].

As a successful tool, UML-RT provides a model to analyze complex, event-driven, real-time systems by mirroring the way real-time systems actually work, and ultimately helps construct and generate code for complete, mission critical real-time applications. However, it fails to extract, represent, and analyze temporal parameters, which are the essential attributes of real-time systems. Therefore, it is difficult to validate a system in the early design and development stages for temporal correctness, even though UML models support the validation of functional or behavioral requirements.

We present a novel method for capturing the temporal parameters in an UML-RT model so that schedulability analysis can be performed. Our method uses the extensibility mechanisms of UML. Schedulability analysis of object-oriented systems is itself a complicated topic beyond the scope of this paper. However, we note that once the temporal parameters of the real-time application have been identified using our method, the schedulability analysis work of Saksena and his co-authors can be applied for fixed priority scheduling algorithms [15, 16, 17]. (Their work is briefly compared with ours in section 4.) To the best of our knowledge, schedulability analysis for real-time applications with an object-oriented design and executed with a deadline-driven scheduling algorithm remains an open problem.

The rest of this paper is organized as follows. Section 2 provides an introduction to UML-RT. The method for capturing the temporal properties of the real-time application within the UML-RT model is presented in Section 3. Discussion of related work is deferred to Section 4 (after our method has been presented). Our conclusions are presented in Section 5.

2. UML for Real-Time

We assume the reader has a basic understanding of UML and object-oriented design and development methodologies, but not of UML-RT. Thus, background material on UML-RT is presented to assist the reader in understanding our method for extracting the temporal properties of a real-time application and specifying them as temporal parameters in a UML-RT model. We begin with a brief overview of ROOM and UML for completeness.

ROOM is a modeling language with formal semantics for the purposes of specifying, visualizing, documenting, and automating the construction of real-time systems. ROOM use role-modeling concepts, which capture the structural communication patterns between software components, to support architectural design patterns. ROOM models are composed of actors that communicate with each other by sending messages along protocols. Actors may be hierarchically decomposed, and have behaviors described by ROOM charts. Descriptions of actors, protocols, and behaviors can all be reused through inheritance.

UML is a general-purpose language for visualizing, specifying, constructing, and documenting the artifacts of a software system. The vocabulary of UML encompasses three kinds of building blocks: things, relationships, and diagrams [2]. UML is open-ended, and provides extensibility mechanisms to allow users to extend the language in controlled ways. The mechanisms include *stereotypes*, *tagged values*, and *constraints*. With UML, a software system's use case view, design view, process view, implementation view, deployment view and their interactions can be expressed to capture the architecture of the system.

UML-RT is a combination of concepts in ROOM and UML. It includes constructs for modeling both the structure and behavior of a real-time system [22, 23]. The rest of this section presents these constructs, and uses diagrams from [22].

2.1. Structure Modeling in UML-RT

The three principal constructs for modeling structure are *capsules*, *ports*, and *connectors*.

Capsules are the fundamental elements in UML-RT modeling. They are a specialization of classes that have the same properties as classes. As such, capsules are a grouping of objects, based on common characteristics. Capsules are distinguished from ordinary classes by their communication mechanisms. Each capsule has at least one port through which it communicates with other objects, and ports are the only means of interaction with the external world for capsules. A capsule may contain collaborating sub-capsules, as shown in Figure 1. A capsule may have at most one state machine that is in charge of sending and receiving signals through ports, and controlling certain constructs of the internal structure. A state machine depicts the behavioral characters of capsules and is described in Section 2.2. Normally, ports, connectors, and sub-capsule are created and destroyed at the same time as the containing capsule. However, some sub-capsules can be created and destroyed dynamically by the state machine of the capsule. A capsule may also contain plug-in roles, which are actually placeholders for sub-capsules that are filled in dynamically.

Ports are objects that act as interfaces on the boundary of a capsule. (Thus, ports are objects within a capsule.) There are two kinds of ports: *relay ports* and *end ports*. Relay ports are connected to sub-capsules to simply pass all signals through, while end ports are connected to the capsules' state machine as the ultimate sources and destinations of signals sent by capsules. Each port of a capsule plays a particular role in the collaboration between the capsule and its interacting object. To define the legal flow of signals between two interacted capsules, the concept of a *protocol* is introduced. (Protocols are described in Section 2.2.) Normally every port is associated with a protocol in a connection. In UML-RT, ports can be illustrated in different ways depending on the purpose of a diagram, as shown in Figures 2, 3, and 4.

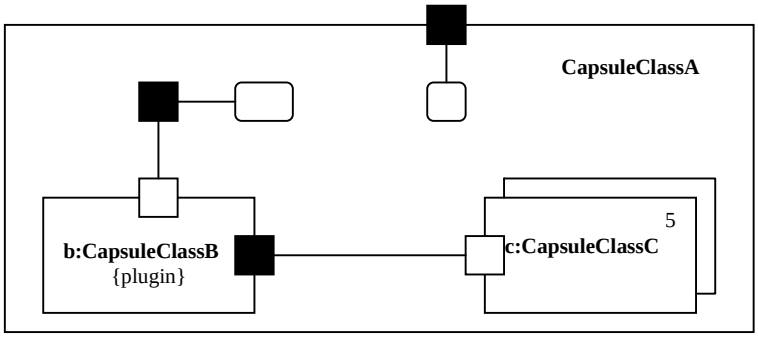

Figure 1: Collaboration diagram (internal view of a capsule).

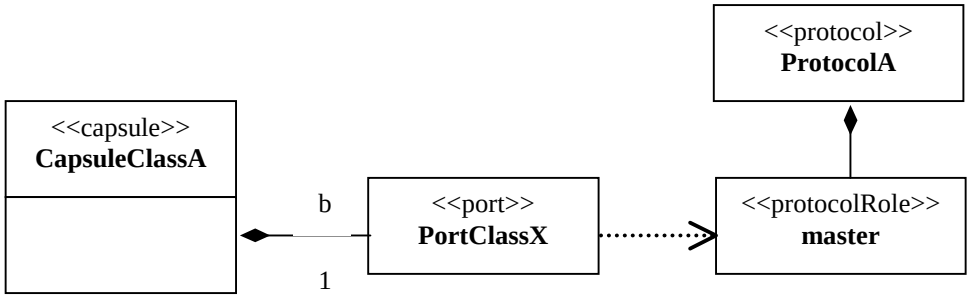

Figure 2: An example of a single port named b belonging to capsule class CapsuleClassA. The port realizes the master role of the protocol defined by protocol class ProtocolA.

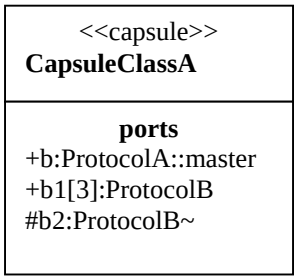

Figure 3: Port notation-class diagram.

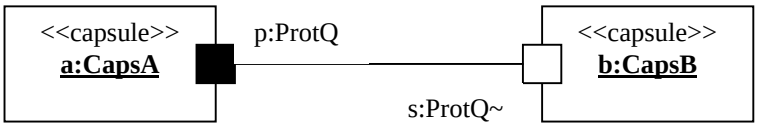

Figure 4: Port notation – collaboration diagram. Black-filled squares represent ports with base roles. White-filled squares represent ports with conjugate roles.

Connectors are actually information channels that support the defined signal-based protocols in a model. A connector can only connect ports that play complementary roles in the protocol associated with the connector. The difference between a connector and a protocol is that a protocol is an abstract specification of desired behaviors while a connector is a physical object that transmits signals between ports. In UML, a connector is represented as an association (as shown in Figures 1 and 4). The relationship between a connector and its associated protocol is implicit through the connected ports.

2.2. Behavior Modeling in UML-RT

The three basic constructs for modeling behavior of a system are *protocols*, *state machines*, and *time service*.

Protocols define a contractual agreement between the communicating participants. They are usually associated with connectors and leave the roles defined by them to be realized by ports. In the specification of a protocol, incoming and outgoing signal sets are normally listed to describe behavior that occurs across a connector. See Figure 5.

As a specialization, binary protocols involve just two participants. The roles played by different participating ports under this protocol are called base role and conjugate role respectively. Only the base role is specified in the protocol, whereas the conjugate role is derived by directly inverting the incoming and outgoing signal sets.

A **state machine** is a behavior that specifies the sequences of states an object goes through during its lifetime in response to events, together with its responses to those events. It is a means of modeling the dynamic aspects of a system. State machines are mostly concerned with the flow of control from activity to activity, and the states of the objects and the transitions among those states (a simple state machine is shown in Figure 6).

Time service provides a facility to transfer timing requirements from capsules, or state machines into signal-based events to trigger certain actions. Time service can be accessed through standard ports called *service access points*. The protocol associated is called TimeServiceSAP (a system-defined protocol role). When a task needs to initiate a time-related activity, it asks the time service to send a special timeout message at the desired time instant. The facility to help realize this service is *timer*, which includes both one shot and periodic timers. For example, the RTTimespec class is used to hold relative times. To ask the time service to send out a timeout message every 10 seconds, we can set up a periodic timer with the following function call: timer.informEvery(RTTimespec(10, 0)).

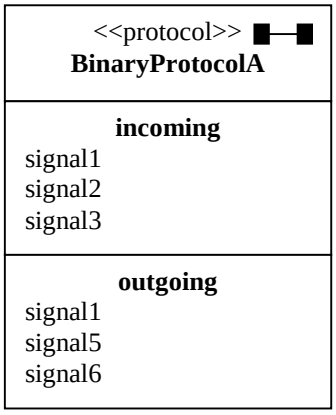

Figure 5: Class diagram - protocol.

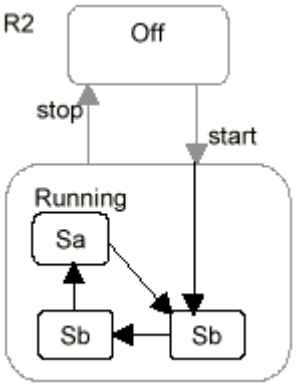

Figure 6: A simple state machine.

3. Capturing Temporal Properties in UML-RT

Our long-term goal is to automate the evaluation of the temporal correctness of real-time applications developed using UML-RT. Evaluating the temporal correctness of an application is called schedulability analysis in the real-time literature. The traditional methods of schedulability analysis identify tasks in the system whose jobs are executed with a regular pattern. Task attributes—such as the execution pattern (rate-based, periodic, sporadic, or aperiodic), start time, maximum execution time, response time bound, and precedence constraints—are then defined and a schedulability condition is evaluated. A positive result from the schedulability condition implies temporal correctness (i.e., all response time bounds will be met).

While some work has been done on determining the schedulability of object-oriented real-time systems (e.g., [1, 4, 5, 10, 15, 16, and 17]), to the best of our knowledge, only Saksena and his co-authors [15, 16, 17] have developed a systematic approach for capturing the temporal attributes of an application from the UML-RT model. Their method uses an extended sequence diagram to identify the tasks that are executed as a result of an external event. Our method differs primarily in that we use the protocol signals associated with each port of a capsule to identify the initial task set. This approach recognizes that all work performed in a reactive system is done so in response to an event (which is triggered by a signal in UML-RT). Our method has three steps: (1) create the initial real-time task set from the actions performed in response to events handled by objects in the system; (2) refine the task set by identifying precedence constraints in the causal set of actions using a sequence diagram; and (3) represent the resulting real-time task set and associated temporal attributes using the extensibility mechanism of UML.

3.1. Creating the Initial Task Set

Active objects of a real-time system execute in response to events generated by other objects (in capsules), the expiration of timers, or the external environment. These events are called triggering events. A sequence of jobs executed in response to a recurring (triggering) event constitutes a task in real-time scheduling theory. Therefore, the task set is created by identifying all active objects and the work performed in response to triggering events. All such events are represented by one of the following three types of signals in a UML-RT model.

The first type is the signal defined in protocols that are realized by ports via connectors between different capsules. The majority of signals in a UML-RT model belong to this category. **The second type** is the signal transmitted between the time service and capsules. When a capsule needs to initiate a timer in time service, it sends a signal to timer service for the application. Upon the expiration of the timer, a signal is sent back from time service to the capsule to trigger an event. The protocol applied for this purpose is system pre-defined Timing protocol. **The third type** of signals is found between capsules and their environment, including the system itself and user operations. In UML-RT, events generated by the external environment (including human interaction) are represented using the actor communicates-association scheme of ROOM. Under this scheme, use cases and actors interact by sending signals to one another. There can be only one communicates-association between a use case and an actor. The complete network of such associations constructs a picture of the communication between the system and its environment.

Thus, we build our initial task set by associating a task with the actions performed by active objects in response to every signal in a UML-RT model. A task's execution pattern (e.g., rate-based, periodic, sporadic, or aperiodic) and frequency is determined by the signal

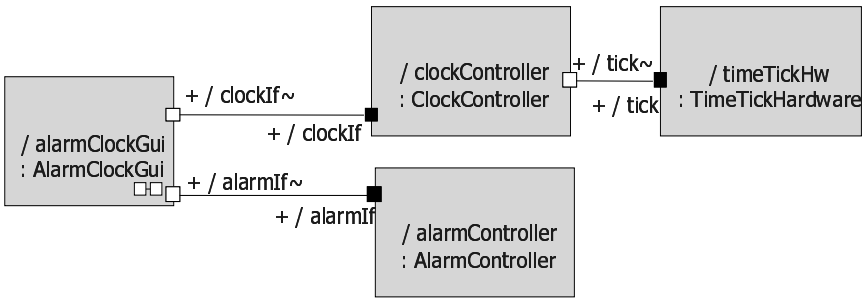

Figure 7: Structure Diagram – AlarmClockDesign.

triggering the task (active object). If a signal is defined in a binary protocol, it represents one task. Otherwise, if a signal is defined in a multiple protocol, the number of capsules interacting via the signal determines the number of tasks required to model the object's execution behavior. The multiple tasks derived from the same signal are unique from a scheduling theory perspective since they have different temporal attributes. The temporal parameters of tasks are specified as attributes of the triggering signal by extending UML-RT slightly (as described in Section 3.3).

Example. We illustrate our method of identifying real-time tasks using a small part of a simple real-time AlarmClock application, which was presented and modeled in the Rational Rose RealTime 6.0 tutorial [14]. We have used some of the diagrams, including protocols, provided in the software and the tutorial, and modified them as needed for our presentation. In the AlarmClock system, the basic components are the alarm controller, clock controller, time tick hardware, and GUI. The GUI includes a GUI proxy, smart buttons, and a button controller. All of these components are abstracted as capsules in the model. Protocols are defined to describe the messages exchanged between different objects. Moreover, necessary state diagrams, collaboration diagrams, and structure diagrams are drawn to deeply analyze the system.

For the purpose of simplifying the presentation, we focus on only a part of the system. Without losing generality, the sub-system we analyze is the clock control system, which interacts with most of the other components. To make the implementation as simple as possible, we have limited the number of capsules in the model. So signals sent to the GUI proxy and smart buttons are both treated as passing to AlarmClockGUI without differentiation. The structure diagram of the system is shown in Figure 7.

The central component in this sub-system is the ClockController capsule, which encapsulates the current time, and responds to time setting button events originating from the GUI. As shown in Figure 7, it accepts signals from capsule TimeTickHardware via port tick to set time and sends signals to capsule AlarmClockGui via port clockIf to force the GUI to visually show time. There are several protocols that define the incoming and outgoing signals transmitted between these capsules, time service, and the environment, to fulfill the functionality, as shown in Figure 8.

The TimeTickInterface protocol connects the TimeTickHardware capsule with other capsules that need to know about the passage of time. The ClockInterface protocol includes messages exchanged between the AlarmClockGui and the ClockController. The Timing protocol allows capsules to communicate with time service. The GUIComm protocol includes messages exchanged between the Real-Time AlarmClock model and the external world.

In the clock control sub-system, there are two categories of functionality: showing clock time and setting clock time. ***From the protocols shown in Figure 8, a task set of 8 tasks can be obtained. Figure 9 graphically depicts the resulting task set.***

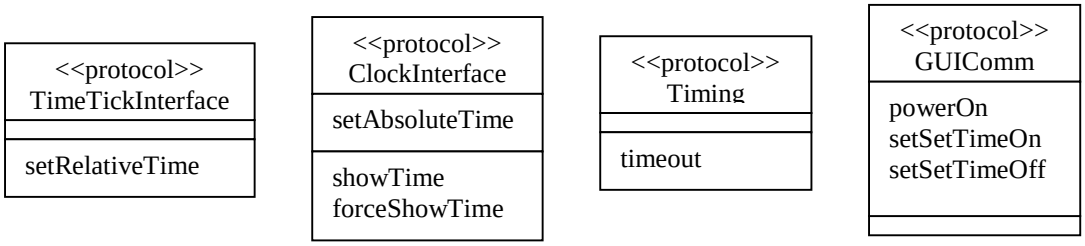

Figure 8: Protocols used in the clock control sub-system.

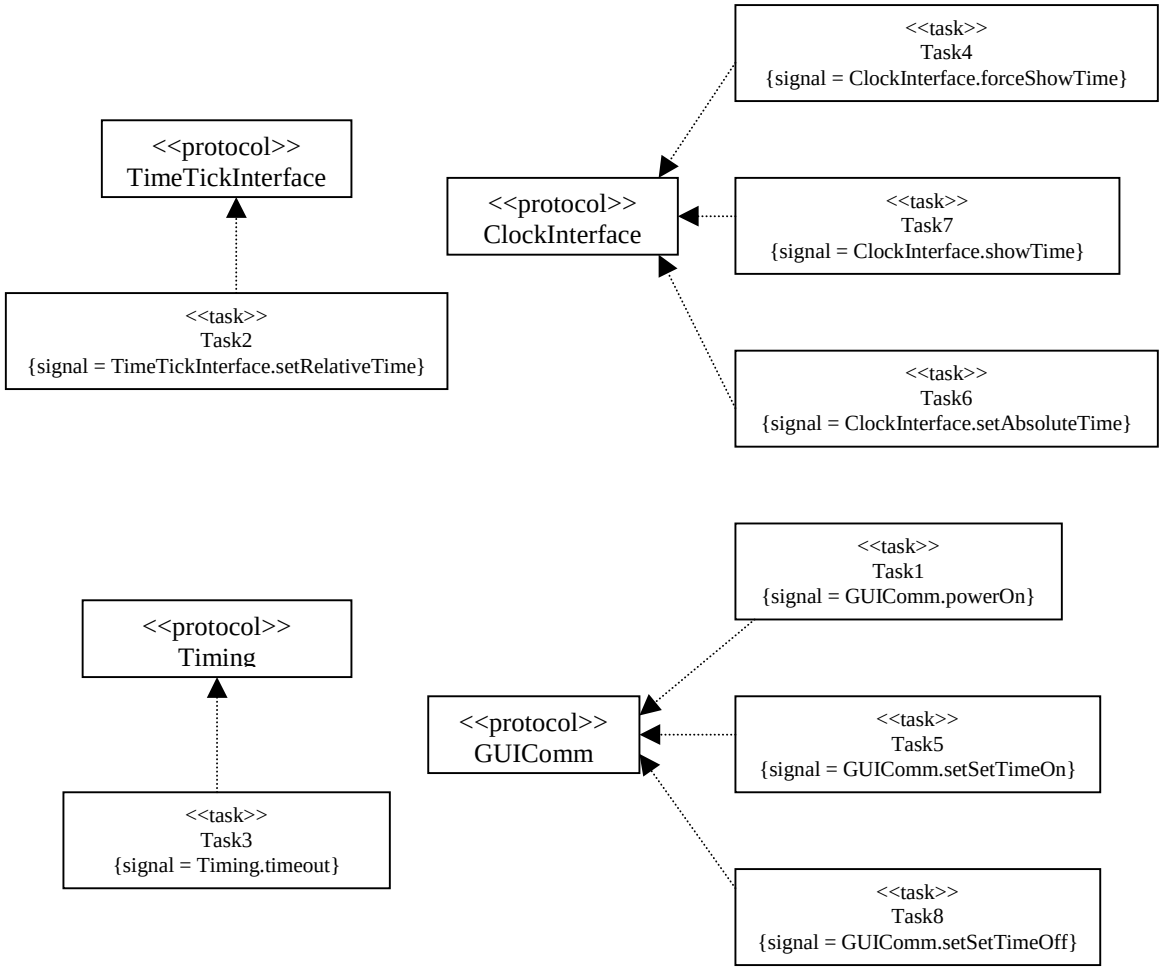

Figure 9: Generation of a task set based on signals defined in protocols.

3.2. Refining the Task Set with Precedence Constraints

In real-time systems, tasks are not always independent and may be restricted to execute in some order. This is especially true when considering a chain of tasks connected by connectors. We call the tasks related to the same signal *base task* and *conjugate task* respectively, in accordance with the concepts of *base role* and *conjugate role* in UML-RT. A conjugate task should not be executed until the arrival of a signal generated by its base task. Therefore, precedence constraints exist between two such tasks. Accordingly, these constraints should be taken into account during schedulability analysis.

A sequence diagram shows an interaction in a system [2]. It depicts the set of objects in the interaction and their relationships, including the messages that may be transmitted between them. It emphasizes the time ordering of messages, from which precedence constraints between events are implied. *We adopt this approach to graphically show the existence of precedence constraints in the task set.* (We note that Saksena an his co-authors also used sequence diagrams to create the task set *and* to identify precedence constraints [15, 16, 17]. Our use differs in that we use them to refine the existing task set, not to create it.)

In Figure 10, we present the sequence diagrams for *showing clock time* and *setting clock time* respectively. Four tasks are required to show clock time correctly. The first task T_1 (Apply Timer) is associated with capsule TimeTickHardware and is triggered by signal powerOn. It is a sporadic task that sets a periodic timer to count ticks. The second task T_2 (Ticking) is triggered by a signal timeout in capsule TimeTickHardware. It is a periodic task that records the passage of a tick. The third task T_3 (Tick Accumulation) in ClockController, is triggered by the periodic signal setRelativeTime generated by task T_2. It keeps track of the passage of ticks and notifies AlarmClockGUI to update time once a second. In capsule AlarmClockGui, task T_4

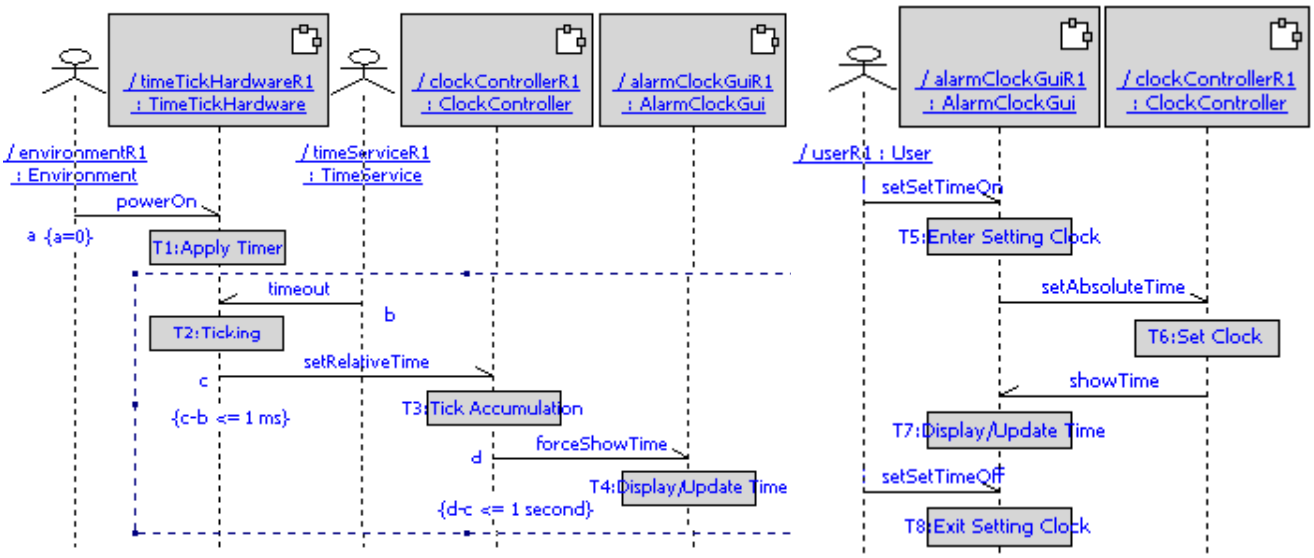

Figure 10: Sequence Diagram for showing and setting clock time.

(Display/Update Time) updates the display of time periodically upon the arrival of signal forceShowTime from task *T3*. These four tasks are all released at the moment when the overall system is initiated (power on) with the described precedence constraints. (Periodic tasks are contained within the dotted rectangle. Tasks outside the dotted rectangle are sporadic.) The other functionality–setting clock time–requires four sporadic tasks. Task *T5* (Enter Setting Clock) performs the steps required to enter the setting-clock-time mode. Task *T6* (Set Clock) sets desired time. Task *T7* (Display/Update Time) updates the display of time set by task *T6*. Task *T8* (Exit Setting Clock) performs the steps required to exit the setting-clock-time mode. These tasks are associated with signals setSetTimeOn, setAbsoluteTime, showTime, and setSetTimeOff respectively.

Thus, from the sequence diagrams, tasks *T2* and *T3* are base task and conjugate task; they have precedence constraints because *T3* can not be executed until *T2* has finished its execution and sent the signal setRelativeTime. Similarly, tasks *T3* and *T4* have precedence constraints between them. Task *T4* will not begin its execution until tasks *T3* has recorded the accumulation of ticks for one second. Similarly, precedence constraints exist between tasks *T5*/*T6*, and tasks *T6*/*T7*.

3.3. Representing Task Attributes in UML-RT

We now show how to represent the resulting real-time task set and associated temporal attributes using the extensibility mechanism of UML. The traditional ways of extending UML are *stereotypes*, *tagged values*, and *constraints*. A stereotype allows users to create new kinds of problem-specific building blocks. It represents the meaning and usage of the newly created element. A tagged value provides new information in terms of pseudo attributes in the element's specification. Tags are normally enclosed in braces in the form of {tag = value}, but tags with long text values may be placed in a separate compartment at the bottom of the classifier's icon. A constraint is a way of adding new rules or modifying existing ones. To realize the extension in our model, we create an abstract data class that encapsulates the seven-tuple **(T, S, R, E, D, Pred, Succ**), as shown in Figure 11, where

- **T** – The task type: a combination of rate-based/periodic/sporadic/aperiodic and preemptable/non-preemptable. (We believe that most active objects are best modeled with the rate-based execution model [9], but also support modeling object execution using more traditional real-time task execution models.)
- **S** – The absolute start time (first release time) of the task.
- **R** – The execution rate of the task specified as (*x*, *y*): *x* executions in *y* time units. (A periodic task with period *y* is specified as (1, *y*).)
- **E** – The maximum expected execution time of the task.
- **D** – The relative response time (deadline) for the task.
- **Pred** – The predecessor task (or the environment) that generates the signal to be processed.
- **Succ** – The successor task (or the environment) that receives the signal generated by this task.

In Figure 11, the abstract class is modeled with the stereotype and tagged value mechanism. An instance of the class, which is discussed next, is shown as well. With this extension, all events in the system generated either by capsules or the environment (to trigger activities or state transitions) can be captured.

Returning to our AlarmClock example, in which we previously identified eight tasks and their precedence constraints (if any), the task set is represented using the tuple notation as:

π = {T_1: (sporadic/nonpreemptable, 0, (1, infinity), e_1, 1ms, environment, TimeTickHardware)

T_2: (periodic/preemptable, 0, (1, 1ms), e_2, 1ms, time service, T_3)

T_3: (periodic/preemptable, 0, (1, 1s), e_3, 1s, T_2, T_4)

T_4: (periodic/preemptable, 0, (1, 1s), e_4, 1s, T_3, AlarmClockGui)

T_5: (sporadic/preemptable, 0, (x_5,y_5), e_5, d_5, environment, T_6)

T_6: (sporadic/preemptable, 0, (x_5,y_5), e_6, d_6, T_5, T_7)

T_7: (sporadic/preemptable, 0, (x_5,y_5), e_7, d_7, T_6, AlarmClockGui)

T_8: (sporadic/preemptable, 0, (x_5,y_5), e_8, d_8, environment, AlarmClockGui)}

The graphical representation of task T_2 is shown as an instance of the task class in Figure 11. For space considerations, we have omitted the graphical representation of the other tasks. Notice that some of the temporal attributes in our example are specified with variables. These variables represent attributes that we were unable to identify directly from the model, and require user inputs. However, our method clearly identifies these attributes and makes it clear which variables are yet to be defined during any phase of the design. (Note that the execution rates for tasks T_6, T_7, and T_8 are the same as the execution rate of task T_5. This is because of the precedence relations between tasks T_5, T_6, and T_7, and the fact that we expect task T_8 (Exit Setting Clock) to be executed just as often as task T_5 (Enter Setting Clock).)

Using this method, a task set for schedulability analysis based on the UML-RT model of the system can be successfully developed. Thus, the schedulability of the application can be evaluated for a given scheduling algorithm as soon as the temporal parameters are fully specified. If the task set is not schedulable, some temporal parameters will need to be adjusted to acceptable ranges, or more seriously the system may need to be re-designed. Thus, the earlier we discover such problems in the design process, the better. By doing scheduling analysis, confidence that the system being developed will work correctly with all temporal parameters being met is gained before the final stage of product development. Our method strictly follows all of the concepts in UML-RT. Moreover, the newly introduced abstract data class is realizable with minor extensions and promotes temporal properties to first-class attributes of the application.

4. Related Work

State machines in UML are based on *statecharts* presented by Harel in [6] to deal with the specification and design of complex discrete-event systems such as multi-computer real-time systems, and communication protocols. His approach recognizes the states/events pair for describing the behavioral aspects of complex systems. Statecharts extend conventional state diagrams with three additional elements: levels of states, inter-level transitions, and communication between concurrent components. Harel's work turns the method of state diagrams into a highly integrated and structured language. Although Harel also advocated associating parameters with states, and verify the description of a

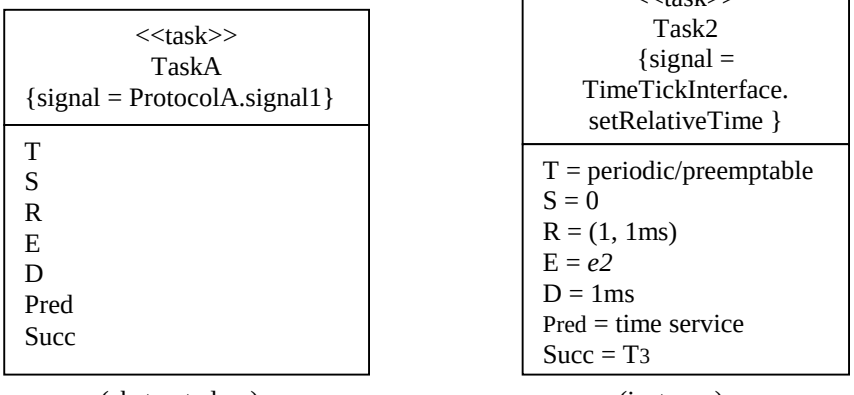

Figure 11:Class diagram-task

system against a Temporal Logic (TL) [18] specification of the system, we are not aware of any extensions of UML that do so.

Another specification language for real-time systems is *modechart*, which was first proposed by Jahanian and Mok [7, 8] and later extended by Brockmeyer *et. al*. [3]. A mode is defined as a partition of the overall states of a system that represents a modular specification of large state machines. Similar to statecharts, modes are recognized in different levels hierarchically and connected by mode transitions. At the same level, modes are either serial or parallel. Modechart is described semantically with real time logic (RTL) [8] to verify absolute timing constraints. RTL was invented with the purpose of specifying timing properties, in both relative and absolute form, of events in real-time systems. Certain actions are supposed to happen upon the occurrence of some events. An action normally has two events associated with it, one represents the initialization of the action, and the other is its completion.

These methods certainly have their usage and advantages when specifying requirements. Moreover, they can be used to validate real-time systems against the temporal parameters captured by them. However, we conjecture practitioners have been slow to adopt these methods because of their complexity. It would also seem that these methods rely primarily on the behavioral state/event pairs of real-time systems without providing adequate support modeling other aspects of the system. Thus, we have focused our research in this area on UML-RT, which is likely to be widely accepted by the object-oriented real-time community.

This work was greatly influenced by [19] in which Selic proposed an extension to ROOM models to support schedulability analysis. By recognizing that digital computers actually transform continuous-time signals into series of events that occur at regular time intervals, the concept of a periodic timer was introduced for ROOM models. A periodic timer was defined as a six-tuple **Tp**: (T, D, C, A, P, X). T is the period of the timer. D is the deadline. C is the execution time. A is the destination to which the timeout event will be sent. P is the priority of the timeout event, and X is an optional data item accompanied with the timeout message. With this extension, Selic claimed it would be possible to perform scheduling analysis from ROOM models, but no details were given. Selic only considered periodic tasks triggered by a timer. Other events initiated by an object, or by the system environment, were not addressed.

This work is most closely related to research conducted by Saksena and his co-authors [15, 16, 17]. Though their research concerning schedulability analysis of object-oriented real-time systems was originally based on ROOM, it directly applies to UML-RT, as shown in [17]. While there has been other work done on determining the schedulability of object-oriented real-time systems (e.g., [1, 4, 5, 10]), to the best of our knowledge, only Saksena and his co-authors have developed a systematic approach for capturing the temporal attributes of an application from the UML-RT model. Their method uses an extended sequence diagram to identify a causal set of actions (called a transaction) that are executed as a result of an external event. As stated previously, our method differs primarily in that we use the protocol signals associated with each port of a capsule to identify the initial task set. Once the task set and associated temporal parameters have been identified using our method, the schedulability analysis work of Saksena and Karvelas in [17] can be applied for fixed priority scheduling algorithms. To the best of our knowledge, schedulability analysis for real-time applications with an object-oriented design and executed with a deadline-driven scheduling algorithm remains an open problem.

5. Conclusions

We have presented a method to capture temporal parameters of a real-time application by extending UML-RT. The method has three steps: (1) create the initial real-time task set from the actions performed in response to events handled by active objects in the system; (2) refine the task set by identifying precedence constraints in the causal set of actions using a sequence diagram; and (3) represent the resulting real-time task set and associated temporal attributes using the extensibility mechanism of UML. The third step of our method creates an abstract class in the UML-RT model to store the temporal parameters and other relevant information.

Our method has three benefits. First, it enhances the description and modeling of a real-time system by capturing more temporal parameters and making them first-class attributes in the model. Second, it enriches (UML and) UML-RT and can easily be integrated into tool sets that support such modeling. Third, and perhaps the most significant, it helps ensure that a real-time system modeled with UML-RT is schedulable (so no deadlines will be missed) and such analysis can begin early in the design process.

We are currently working to automate our method to generate a task set for schedulability analysis automatically from a UML-RT model. We are also evaluating our method with more complicated real-time systems.

References

[1] M. Awad, J. Kuusela, and J. Ziegler. Object-Oriented Technology for Real-Time Systems: Practical Approach using OMT and Fusion, Prentice Hall, 1996.

[2] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling language User Guide*, Addison Wesley Longman, Inc., 4th Printing, 1999.

[3] M. Brockmeyer, F. Jahanian, C. Heitmeyer and B. Labaw, "Debugging and Testing Real-Time Specifications: A Flexible, Extensible Simulation Environment for the Modechart Toolset," *Real-Time Technology and Applications Symposium Workshop* (RTAS). June 1997.

[4] A. Burns and A. J. Wellings. "HRT-HOOD: A Design Method for Hard Real-Time," *Real-Time Systems*, 6(1):73–114, 1994.

[5] H. Gomaa. Software Design Methods for Concurrent and Real-Time Systems. Addison-Wesley Publishing Company, 1993.

[6] D. Harel, "Statecharts: A Visual Formalism for Complex Systems," *Science of Computer Programming* 8 (1987), pp. 231-274.

[7] F. Jahanian and A. Mok, "Modechart: A Specification Language for Real-Time Systems," *IEEE Transactions on Software Engineering*, vol. 20, no. 12, Dec. 1994.

[8] F. Jahanian and A. K. Mok, "Safety Analysis of Timing Properties in Real-Time Systems," *IEEE Trans. Software Engineering*, vol. SE-12, Pages 890-904, Sept. 1986.

[9] K. Jeffay and S. Goddard, "A Theory of Rate-Based Execution," *Proceedings of the 20th IEEE Real-Time Systems Symposium*, Dec. 1999, pp. 304-314.

[10] L. Kabous and W. Nebel. "Modeling Hard Real-Time Systems with UML the OOHarts Approach," *Proceedings, International Conference on Unified Modeling Language (UML'99)*, 1999.

[11] J. W. S. Liu, *Real-Time Systems*, Prentice Hall, 2000.

[12] A. Lyons, "UML for Real-Time Overview," ObjecTime Ltd., http://www.objectime.com/, Apr 1998.

[13] W. E. McUmber and B. H.C. Cheng, "UML-Based Analysis of Embedded Systems Using a Mapping to VHDL," *4th IEEE International Symposium on High-Assurance Systems Engineering*, November 17-19, 1999, pp. 56-63.

[14] Rational Software Corporation, Rational Rose RealTime 6.0 http://www.rational.com/products/rosert/, April 2000.

[15] M. Saksena, P. Freedman, P. Rodziewicz, "Guidelines for Automated Implementation of Executable Object Oriented Models for Embedded Control Systems," *Proc. of the Real-Time Systems Symposium*, Dec. 1997, pp. 240-251.

[16] M. Saksena, A. Ptak, P. Freedman, P. Rodziewicz, "Schedulability Analysis for Automated Implmentations of Real-Time Object-Oriented Models," *Proc. of the Real-Time Systems Symposium*, Dec. 1998, pp. 92-102.

[17] M. Saksena and P. Karvelas, "Designing for Schedulability: Integrating Schedulability Analysis with Object-Oriented Design," to appear in *Proc. of the 12th Euromicro Conference on Real-Time Systems*, June 2000.

[18] R. L. Schwartz and P. M. Melliar-Smith, "Temporal Logic Specification of Distributed Systems," *Proc. 2nd IEEE International Conference on Distributed Computer Systems*, 1981, 446-454.

[19] B. Selic, "Periodic Tasks in ROOM," *Workshop on OO Real-Time Systems*, ACM OOPSLA '95, Austin Texas, Oct. 15-19, 1995.

[20] B. Selic, "Turning Clockwise: Using UML in the Real-Time Domain," *Communications of The ACM*, Vol. 42, No. 10, October 1999, pp. 46-54.

[21] B. Selic, G. Gullekson, and P.T. Ward, *Real-Time Object-Oriented Modeling*, John Wiley and Sons. 1994.

[22] B. Selic and J. Rumbaugh, "Using UML for Modeling Complex Real-Time Systems," ObjecTime Ltd., http://www.objectime.com/, March 11, 1998.

[23] J. Shahani and S. Garone, "The Rose Family Grows," International Data Corporation, http://www.idc.com, August 1999.