

UML 2.0 Notation for Modeling Real Time Task Scheduling

Maria Cruz Valiente, Gonzalo Genova, Jesus Carretero, Carlos III University of Madrid

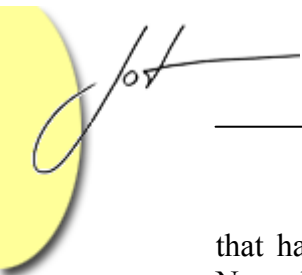
Abstract

UML is the standard visual object modeling language which may be very useful as a system design communication language. However, UML as a real-time modeling language has limitations. It basically provides a lot of syntax, but not enough semantics. The UML Profile for Schedulability, Performance and Time, extends the notation and semantics for the real-time domain. It is supposed to overcome the limitations which are related to and provide suitability for modeling real-time systems. Since we are mainly concerned with the problem of real-time operating system scheduling, the article discusses the ability of UML and its profile to determine the schedulability of a planned piece of software.

1 INTRODUCTION

In real-time systems, the correctness of a computation depends not only on obtaining the right result, but also upon providing the result on time. Because of performance constraints, typically the design of real-time software is addressed jointly by hardware and software components and it is also supported by the features of the operating system. The operating system selected for those applications could be a real-time operating system (RTOS), specifically designed to schedule real-time tasks. In the last few years, real-time processing seems to be the essential part of a operating system, and the scheduler could be considered the most important component of a real-time system. This way, designing this kind of systems may be considered one of the most interesting tasks that a software engineer can perform.

Object-oriented modeling is naturally fitted for capturing various characteristics and requirements of systems. However, the current object-oriented techniques do not provide the tools and proper services in order to design all the characteristics of real-time systems, particularly those concerning constraints (timing constraints, resource constraints...) and non-functional requirements (QoS, security requirements...). The Unified Modeling Language (UML) is an industry standard which provides notation for describing object oriented models. The language represents a collection of the best engineering practices



that have proven successful in the modeling of large and complex systems [UML03]. Nowadays, UML is commonly used as a description language and is more complete than other languages in its support for modeling complex systems, being also suitable for modeling real-time systems [Björkander03], [Douglass98], [Gomaa00]. UML has been also used in a large number of time-critical and resource-critical systems [UMLSPT03]. Despite its real-time capabilities [Douglass98b], [Jigorea00], [Küster01], [Selic98], UML has some limitations as well, because it lacks in notations and semantics to represent several aspects that are of particular concern to real-time system developers [Bichler02], [Lavazza01]. However, it has mechanisms for improving the design phase through its extensibility capabilities: the UML profile mechanism provides a way of specializing the concepts defined in the UML standard. The UML Profile for Schedulability, Performance, and Time was submitted in response to a Request for Proposal (RFP) to define a common, standard way in which timeliness and related properties could be specified. The Schedulability Modeling is a sub-profile, which is intended as a general base for supporting a wide variety of known and future schedulability analysis methods [UMLSPT03]. In order to assess UML validity and limitations for modeling real-time systems, in this article, we provide a concrete periodic task scheduling example in a RTOS. Our goal is to determine how useful UML designing real-time task schedulability is [Cottet02].

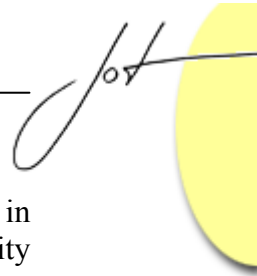
2 REAL-TIME SCHEDULING

Real-time developers are often most concerned with meeting specific time deadlines for tasks. The difficulty in using an operating system for real-time programming is that it must be able to guarantee that the worst-case response time to give control to a process which needs attention is short enough for the process to handle events on time. If the worst-case response time of the process is greater than its deadline, then the scheduling is not feasible. One strategy to ensure efficient response time is to prioritize processes so that more important processes always receive processor attention if they need it.

The scheduler is the part of the operating system that responds to the requests sent by programs, interrupts and gives control of the processor to those processes. It is the component that implements the scheduling algorithms and can also stand alone to act as a centerpiece to a program that requires moderation among many different concurrent tasks. In this working-mode, each task that a program must accomplish is written so that it can be called by the scheduler as necessary. Tasks can be ranked in priority, allowing the scheduler to hand control of the processor to each process in turn.

The major factors that affect real-time system schedulability include [Saiedian04]:

- The event occurrence pattern, whether the event is periodic or aperiodic.
- Each event deadline, whether the deadline is less than, equal or greater than its period.
- Task interaction, whether tasks need to synchronize with each other or not.



Due to the necessity of considering the schedulability analysis as early as possible in software development, its integration with the industry UML standard (schedulability analysis can be integrated with object-oriented design) and its schedulability extension (UML Profile for Schedulability, Performance, and Time) is a very important issue, because it may allow to detect unfeasible real-time architectures at an early stage of development, preventing costly design mistakes, accelerating development schedules and allowing developers to verify the real-time performance of the design throughout the software lifecycle [Martins03].

Real-time task model

A real-time application is specified by means of a set of tasks. Real-time tasks are the basic executable entities that are scheduled; they may be periodic or aperiodic, and have hard (late data are bad data) or soft (late data may still be good data) real-time constraints. A task model is defined by its main timing parameters [Cottet02], [Liu73] (see Figure 1). The quality of scheduling depends on the exactness of these parameters, so their determination is an important aspect of real-time design.

- r , task release time, i.e. the triggering time of the task execution request.
- C , task worst-case computation time, when the processor is fully allocated to it.
- D , task relative deadline, i.e. the maximum acceptable delay for its processing.
- T , task period (valid only for periodic tasks).
- When the task has hard real-time constraints, the relative deadline allows computation of the absolute deadline $d = r + D$. Transgression of the absolute deadline causes a timing fault.

Also, when tasks are allowed to access shared resources, their access needs to be controlled in order to maintain data consistency.

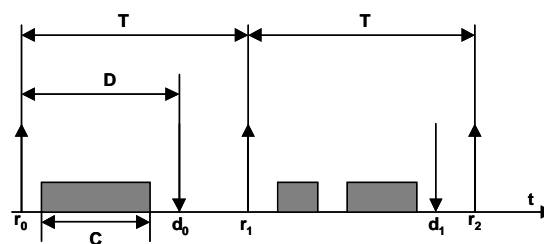
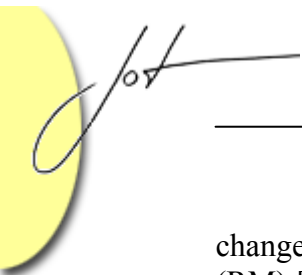


Figure 1. Real-time task model

Fixed-priority algorithm: Rate Monotonic

A scheduling algorithm is a set of rules that determine the task to be executed at a particular moment [Liu73]. In schedulability analysis, thread priorities are of primary concern. Basic on-line algorithms are designed with a simple rule that assigns priorities according to the temporal parameters shown in the previous subsection. If the considered parameter has a fixed value, i.e., request rate or deadline, the algorithm is static because the priority is fixed. The priorities are assigned to tasks before execution and do not



change over time. A basic algorithm with fixed-priority assignment is Rate Monotonic (RM) [Liu73]: given a set of periodic tasks, assigning the priorities according to the RM algorithm means that tasks with shorter periods (higher request rates) get higher priorities. RM scheduling is preemptive, i.e., a task can be preempted by a task with higher priority. Research has shown that this algorithm solves many of the issues in real-time systems, including predictable real-time industrial computing systems [Saiedian04].

Priority inversion

In most real-time systems, the tasks are not independent, they synchronize with one another in a mutually exclusive manner to share a resource. In preemptive scheduling, that is driven by fixed priority and where critical resources are protected by a mutual exclusion mechanism, the priority inversion phenomenon can occur [Cottet02].

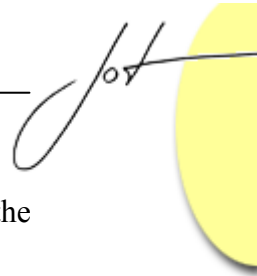
Priority inversion is the phenomenon that causes a high priority task to be delayed indefinitely while awaiting a resource that has been held by a low priority task. The low priority task is often unable to run due to the presence of an unrelated medium priority task. In the situation that results, one finds that the high priority task is effectively denied the CPU because of the lower medium priority task.

In order to prevent the priority inversion phenomenon, the priority inheritance protocol can be applied. The basic idea is to dynamically change the priority of some tasks. So a task τ_i , which is using a critical resource inside a critical section, gets the priority of any task τ_j waiting for this resource whenever the priority of task τ_j is higher than that of task τ_i .

3 REAL-TIME UML

UML is a language for expressing the constructs and the relationships of complex systems. UML helps in describing and designing software systems, particularly software systems built using the object-oriented style [Fowler03]. Although UML is mainly intended for general-purpose modeling, it may be also used for real-time systems modeling [Douglass04]. The OMG has adopted a standard way to capture timeliness, performance, and schedulability properties of real-time systems: UML Profile for Schedulability, Performance, and Time. This profile has increased the interest on using UML and object-oriented technology to model and implement real-time systems [Saiedian04]. The profile does not invent any new techniques, but offers the possibility to exchange timeliness properties between UML modeling tools and schedulability analysis tools. Since the major tools available in the market place are all based on the RM analysis, and the team who developed the profile had experience and expertise predominantly with RM analysis, the specification is basically biased towards this algorithm.

The UML Profile for Schedulability, Performance, and Time uses stereotypes, tagged values, and constraints with specific names. Although the UML Profile for Schedulability, Performance, and Time predefines a number of stereotypes, tagged values



and constraints, the user is free to add more, whenever it clarifies or simplifies the analysis and design models.

The stereotypes may extend one or more classes through extensions as part of the profile, and enables the use of domain specific terminology or notation. Stereotypes may clarify the model to domain experts trying to construct or understand the system. However, stereotypes are more useful when they contain special values that apply to that kind of stereotype: tagged values.

A tagged value is a property name-value pair and provides the definition of the kind of information added to the stereotype.

A constraint is a user-defined rule or rule of correctness. We use constraints to express restrictions that make sense in our particular domain.

UML dynamic modeling

UML dynamic modeling addresses the dynamic aspects of the system. The dynamic model, also referred to as behavioral model, describes how objects interact with each other. For our purpose, UML allows to model the traces of interactions among many objects working together and provide the important information required for schedulability analysis, which is captured in sequence or timing diagrams. We can use sequence diagrams and timing diagrams to describe scenarios that have timing requirements.

Sequence diagrams emphasize message sequence, so the next message in time is the message following the current one on the diagram. Time goes down the page, but usually linearity is not implied; that is, further down implies later in time, but the same distance at different places in the diagram does not imply the same amount of time. Also, messages on sequence diagrams are only partially ordered, so in many cases the relative timing between messages is not specified. Sequence diagrams support concurrent semantics, which means that the relative ordering of some messages is not specified; we would expect that, if one event (such as sending or receiving a message) is marked below another one in the diagram, then the second one comes after the first one in absolute time, but this is true only if both events belong to the same lifeline, not if they belong to different lifelines; therefore, when messages begin on different lifelines, we cannot know which one is first in time; and, if messages terminate on different lifelines, we cannot assume that they are received in the same sequence as they are depicted, even when they are sent in that particular sequence.

Timing diagrams were not included in UML 1.x., even though electrical engineers have used timing diagrams for a long time in the design of electronic state machines. A timing diagram is a simple representation with time along the horizontal axis and object state or attribute value along the vertical axis. However, timing diagrams are different from statecharts, even though we can see the change in state of the different lifelines, because statecharts specify all the reactive behavior of the classifier it is specifying, but not a particular scenario. In UML 2.0, they are used to show interactions when a primary purpose of the diagram is to reason about time. Since timing diagrams are shown against

a common clock (depicted with the time ruler on the diagram), messages on timing diagram are fully ordered.

Run-to-completion semantics

UML defines a run-to-completion semantics for event consumption, which imposes that no other event is dequeued before the processing of the previous event is fully completed. Events are consumed one by one. That means that the object will not accept new incoming events until the previous one has been fully consumed..

4 CASE STUDY: TASK SET SCHEDULING SHARING A CRITICAL RESOURCE

The main objective of this article is to check the capabilities of UML for task scheduling in a RTOS. We have to determine if the proposed task set is schedulable or not using UML models. We therefore analyze the schedulability of an example application with the following characteristics:

- Task set composed of three dependent periodic tasks τ_1 , τ_2 and τ_3 .
- Tasks τ_1 and τ_3 share a critical resource.
- A task in a critical section can be preempted by a higher priority task which does not need the same resource.
- Deadlines are equal to periods.
- Preemptive fixed priority scheduling algorithm considered: Rate Monotonic (RM). We use descending priority values to indicate higher priority.

In order to describe in more detail this task set and the critical sections of task τ_1 and τ_3 , we add new parameters that specify the three components of C_i :

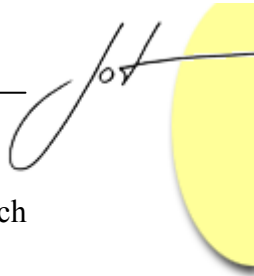
- C_i^α : task duration before entering the critical section.
- C_i^β : critical section duration.
- C_i^γ : task duration after the critical section.

Then, we have the computation time $C_i = C_i^\alpha + C_i^\beta + C_i^\gamma$, so the task set is described by the classical parameters given in Table 1

Task	r_i	C_i	D_i	T_i	C_i^α	C_i^β	C_i^γ	Prior.
τ_1	0	2	6	6	1	1	0	1
τ_2	0	2	8	8	2	0	0	2
τ_3	0	4	12	12	0	4	0	3

Table 1. Case study: a task set sharing a critical resource

According to the RM algorithm, where tasks with shorter periods get higher priorities, task τ_1 receives the highest priority and task τ_3 the lowest one.



For simplicity, we have been considered UML run-to-completion semantics for each event consumption.

The schedule obtained under the RM algorithm is illustrated in Figures 2 and 3 using a UML 2.0 sequence diagram and a UML 2.0 timing diagram, respectively.

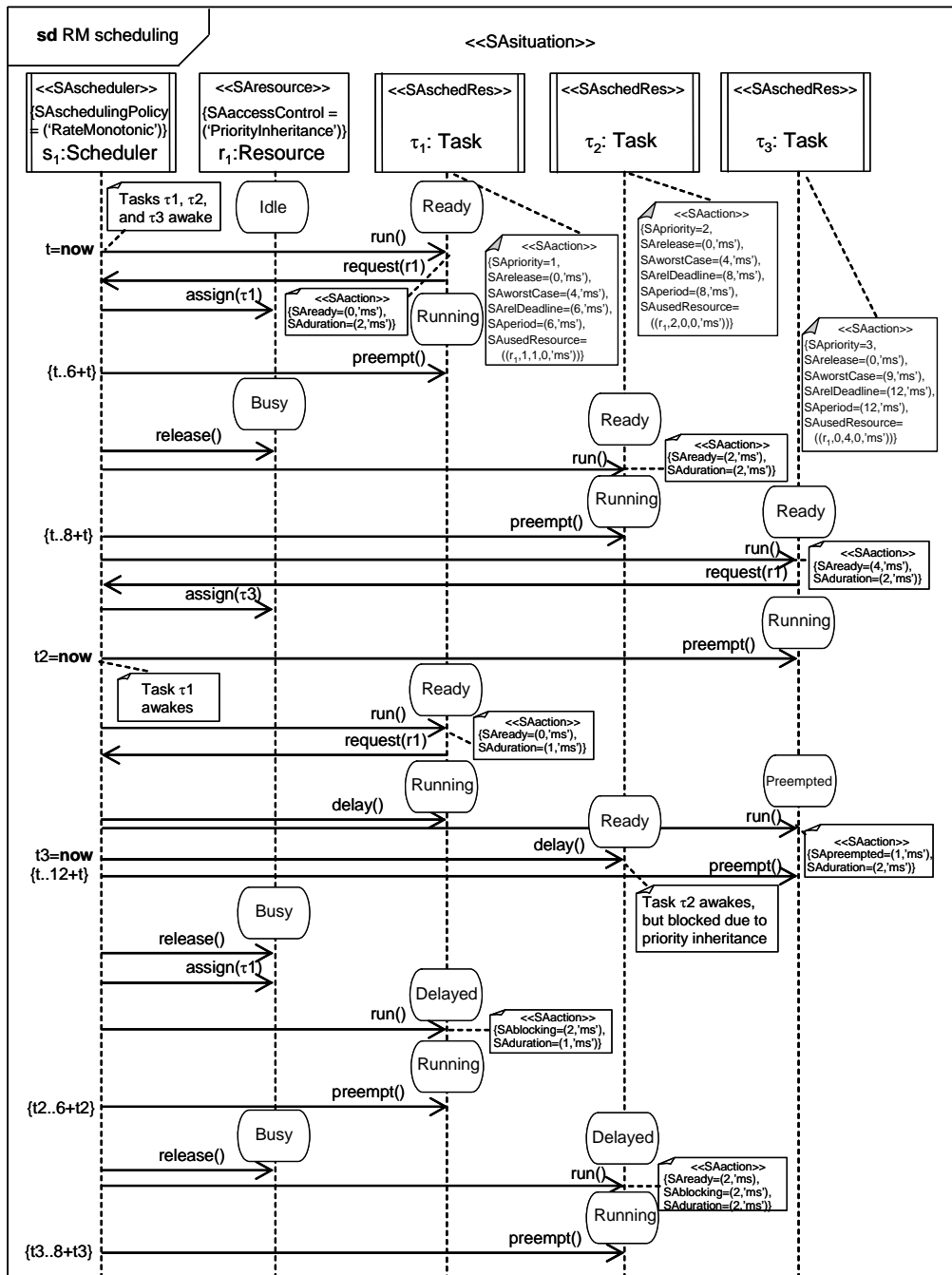


Figure 2. Sequence diagram showing the example task set accordingly to the RM scheduling

Analyzing the sequence diagram, we can stereotype this diagram as <<SAsituation>> to use it in the context of schedulability analysis, representing a real-time situation.

In UML 2.0, the notation for an interaction in a sequence diagram is a solid-outline rectangle (a rectangular frame). The five sided box at the upper left hand corner names the sequence diagram: keyword *sd* followed by the interaction name, “RM Scheduling”¹.

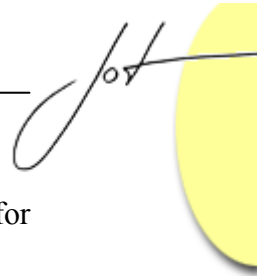
Each lifeline in the diagram represents an individual participant in the scenario:

- s_1 :Scheduler. The stereotype <<SAscheduler>> of the UML Profile for Schedulability, Performance, and Time (schedulability modeling) represents a scheduler, in this context, a kind of resource broker² for a execution engine (in our domain, a processor) that is responsible for processing acquisition requests from clients of the service and, based on the appropriate access control policy for that service, it dispenses access to the service. If a service instance is busy, then the reply may remain pending until access is possible. Shortly, the scheduler determines a schedule that allocates a set of scheduling jobs to its set of execution engines. The tag “SAschedulingPolicy” represents the set of rules for assigning processor time to a set of scheduling jobs (in our model, ‘RateMonotonic’). The class Scheduler has been designated as active where its instance (s_1) has its own thread of control and can initiate control activity. The active objects are shown with the new notation in UML 2.0: class box with an additional vertical bar on either side, instead of the thick border.
- r_1 :Resource. The stereotype <<SAresource>> of the UML Profile for Schedulability, Performance, and Time (schedulability modeling) represents a kind of protected resource (e.g., a semaphore) that is accessed during the execution of a scheduling job. It may be shared by multiple concurrent actions and must be protected by a locking mechanism. The tag “SAaccessControl” represents the access control policy for handling requests from scheduling jobs (in our model, ‘PriorityInheritance’).
- τ_1, τ_2, τ_3 : Task. The stereotype <<SAschedRes>> of the UML Profile for Schedulability, Performance, and Time (schedulability modeling) represents a unit of concurrent execution (in our domain, a task), which is capable of executing a single scenario concurrently with other concurrent units. The class Task has been designated as active where each of its instances (τ_1, τ_2 and τ_3) has its own thread of control and can initiate control activity.

In the general resource modeling of the UML Profile for Schedulability, Performance, and Time, an action is defined as a kind of scenario. Therefore, we use the stereotype <<SAaction>> of this profile (schedulability modeling) to characterize the behavior of each task in our model (we had to add some tags that there were not present in the <<SAaction>> stereotype of the schedulability model):

¹ The notation within this rectangular frame comes in several forms depending on the specific diagram: sequence diagram, communication diagram (which was known as collaboration diagram in UML 1.x), interaction overview diagram (new in UML 2.0) and timing diagram (new in UML 2.0)

² A broker is an instance of a run-time entity responsible for controlling access to the exclusive services of a resource.



-
- **SABlocking**, representing the length of time that the task is blocked waiting for the resource.
 - **SAduration** (we have added this tag), representing the total duration of the action (assignment of processor to task).
 - **SAPeriod** (we have added this tag, only for periodic tasks), representing the task period.
 - **SAPreempted** representing the length of time that the task is preempted, when runnable, because the processor is allocated to a higher priority task which is ready to run.
 - **SAPriority**, representing the priority of the task from the scheduling perspective.
 - **SAReady**, representing the length of time since the task awakes (beginning of a period).
 - **SARelDeadline**, representing the desired time by which the action should be complete.
 - **SArelease**, representing the triggering time of the task execution request.
 - **SAusedResource**, representing the list of resources that the task uses, and time specifications (this is our interpretation, because the profile does not explain how we can represent it): ((resourceName1,durationBCS,durationCS,durationACS), (resourceName2,durationBCS,durationCS,durationACS),..., (resourceNameN,durationBCS,durationCS,durationACS)), where durationBCS is the task duration before entering the critical section, durationCS is the critical section duration, and durationACS is the task duration after the critical section.
 - **SAworstCase**, representing the task worst-case computation time, i.e., the longest period of time required for an action to complete its execution, including all the overheads (blocking, delay, preemption, etc.).

We use the new metaclass in UML 2.0, **TimeObservationAction**, to know when a task awakes. A time observation action is an action that, when executed, returns the current value of time in the context in which it is executing. It is depicted with the keyword “now”. A time observation action lets us determine the absolute deadline that each task has to meet, shown in the diagram with time constraints.

We use also the new metaclass in UML 2.0, **StateInvariant**, to show the different states associated to each lifeline as restrictions. A state invariant is a constraint on the state of a lifeline, and this constraint is evaluated during runtime. If the constraint is true the trace is a valid trace. It is a possible way to show the state value of a lifeline in the sequence diagram. For instance, the state of the resource r_1 has to be “Busy” if we want to release it.

Finally, we use notes to display the textual information that we want to emphasize. It is assumed that the task set is schedulable because there is not any element indicating a timing fault (i.e., the tasks do not miss their deadlines), but it is difficult to check by simple inspection of the diagram.

Figure 3 illustrates the timing diagram for the proposed scheduling. We have represented a longer period of time than in sequence diagram spanning over several scheduling cycles.

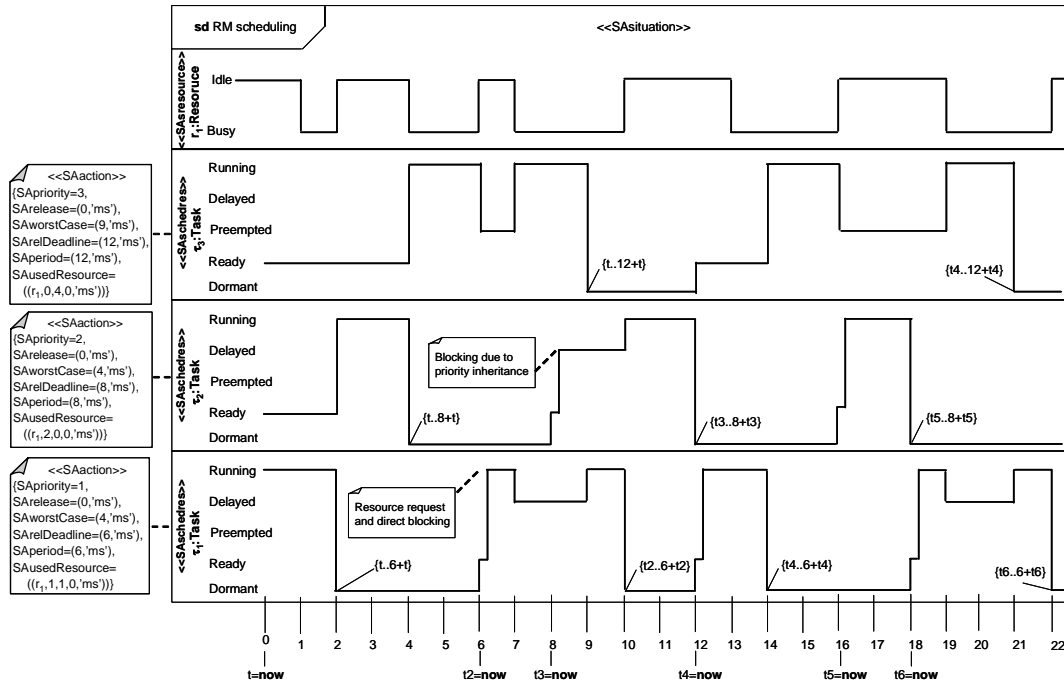
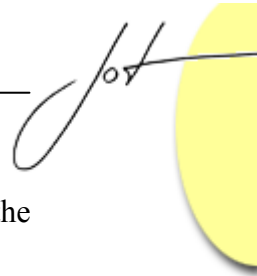


Figure 3. Timing diagram showing the example task set accordingly to the RM scheduling

Again, we stereotype the diagram <<SAsituation>> to use it in our context of schedulability analysis. The notation of the rectangular frame and the five sided box is the same as in the previous sequence diagram, but now we have different elements in the model. We generate the diagram with four lifelines: the resource (r₁) and the three tasks (τ₁, τ₂, τ₃). In this case, we ignore the scheduler (s₁), because it is not necessary for understanding the scheduling. Since we can show the change in state of the different lifelines over linear time, then we can understand what is happening and there is no need to show the message passing.

Task states used in the timing diagram are explained in Table 2. The resource lifeline has two simple states, idle or busy. The timing diagrams let us see how the states change over time for each lifeline and it is very useful in scheduling. Therefore, we do not need to use the metaclass StateInvariant as a restriction in lifelines to know the state value at a particular time.

The time axis is linear so it clarifies absolute timing of events, state changes and relative timing between the different lifelines. Therefore, we do not need to use notes indicating when a task awakes (when the state of a task changes to "Ready"), and we do not need the stereotype <<SAaction>> representing the length of time in different actions



as in the sequence diagram, because we can use the time axis to know it (we only use the stereotype <<SAaction>> to characterize the task set parameters).

We use notes to emphasize some aspects of the schedule, for instance, when task τ_2 is blocked due to priority inheritance.

Once more, we use the metaclass TimeObservationAction to control when a task awakes, and we use this time value to set the constraints (task deadlines). Now, since we have the time along the horizontal axis, we can check the scheduling and see that the schedule is feasible or not.

State	Description
Dormant	The task is set up.
Ready	The task awakes.
Preempted	When running, the task is preempted.
Delayed	The task is waiting for a signal or a resource.
Running	Assignment of processor to task.

Table 2. Task states

For our purpose, the timing diagram view is probably more adequate than the sequence diagram view, but the last one offers another point of view which is quite interesting as well. It is therefore better to use the two diagrams in order to have both views. However, even after having modeled and understood the schedule with the two diagrams, we can not assure that the schedule is feasible, i.e., the two diagrams themselves are not sufficient for deciding if the task set is schedulable or not: we do not know if we will have enough resources to perform the schedule (available processor, memory space...), and we should apply the mathematical feasibility model [Cottet02] to the diagrams. Since, in our case study, the tasks are not independent, then we have to consider a task set sharing resources. Thus, we have to calculate the worst-case response time for each task by including the blocking delay (assessment of a task response time): the utilization factor of the processor has to be low enough to let the task set to be scheduled, whatever the blocking time due to the shared resources

5 CONCLUSIONS

UML is intended to be a well-defined language which can be productively used with a wide range of different systems. UML 2.0 represents the biggest change that has happened yet to the UML. The new version is an incremental improvement to the UML 1.x standards, improving the clarity of the UML for capturing architectures and improving its scalability. UML and the profiles provide a mechanism to model specific features of some domains, and although the user is free to add notation to UML diagrams

through the profiles, it would be very important that those contributions be part of the standard in order to enable the exchange of models between the different modeling tools. In this way, any developer could take advantage of it.

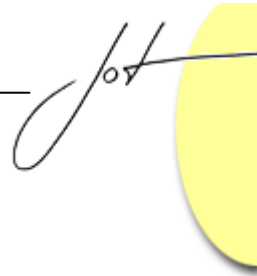
With respect to real-time systems, UML 2.0 has introduced options to manage the complexity of interactions and, relating to time, the new type of diagram, timing diagram, provides an expressive way to detail change in value or state of the different classifiers and attributes. UML and the UML profile for Schedulability, Performance, and Time allow models where information related to time aspects may be annotated. Thus, software engineers may be able to specify and design their real-time systems with UML models which could be the formal description of the system, so that the different models could be analyzed in order to assure that they accomplish all the properties needed in the system.

Timing diagrams and sequence diagrams are the two kinds of interaction diagram more adequate to model task scheduling. Thus, we recommend to use both diagrams when analyzing schedulability. A timing diagram is similar to a sequence diagram in that they both show scenarios of collaborations, but they are not the same at all. Although timing diagrams do not show any information beyond that available in annotated sequence diagrams, the absolute timing of events, state changes and the relative timing among the lifelines is clearer and more readable than on sequence diagrams, even when explicit timing constraints are added.

Since deadlines are of critical importance, as opposed to message sequence, we think that perhaps it is therefore better to use timing diagrams instead of sequence diagrams. However, each diagram provides different points of view to the same scenario and both could be very useful.

However, although the resulting models are quite complete and facilitate communication and understanding, the diagrams themselves are not sufficient to indicate if we have a feasible schedule. We would need to apply the mathematical feasibility model to the UML diagrams.

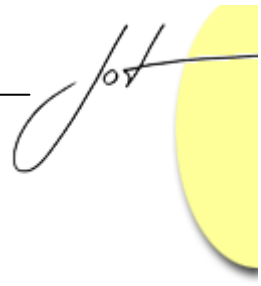
Also, as we mentioned before, since the UML profile for Schedulability, Performance, and Time is clearly biased towards rate monotonic analysis, it would be interesting to extend the specification to comprise cyclic scheduling. Static cyclic scheduling is easier to implement and will be sometimes more efficient than fixed priority scheduling, for instance, for control related tasks with strict requirements on small timing variations: reaching the same performance level with fixed priority scheduling may require higher sampling rates or more sophisticated control algorithms. Event handling could be another example where static cyclic scheduling should be recommended [Lonn99], [Drapper99].



REFERENCES

- [Bichler02] L. Bichler, A. Radermacher, A. Schür John Smith: “Evaluating UML Extensions for Modeling Real-Time Systems”, *In Proceedings of the 7th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS 2002)*. Los Alamitos, CA, USA, IEEE Computer Society 2002, pp. 271-8 Conference Paper.
- [Björkander03] M. Björkander, C. Kobryn: “Architecting Systems with UML 2.0”. *IEEE Computer Society, IEEE Software*. 2003.
- [Cottet02] F. Cottet, J. Delacroix, C. Kaiser, Z. Mammeri: “Scheduling in real-time systems”. John Wiley & Sons. 2002.
- [Douglass98] B. P. Douglass: “Real-Time UML. Developing Efficient Objects for Embedded Systems”. Addison-Wesley. 1998.
- [Douglass98b] B. P. Douglass: “Designing Real-Time Systems with UML”. *Embedded Systems Programming*. 1998.
- [Douglass04] B. P. Douglass: “Real Time UML. Advances in The UML For Real-Time Systems”. Addison-Wesley. Third Edition. 2004.
- [Drapper99] D. Draper, A. K. Jónsson, D. P. Clements, D. Joslin: “Cyclic Scheduling”. *In Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*. Morgan Kaufmann Publishers Inc. 1999, pp. 1016-1021. San Francisco CA, USA.
- [Fowler03] M. Fowler: “UML Distilled. A Brief Guide to the Standard Object Modeling Language”. Addison-Wesley. Third Edition. 2003.
- [Gomaa00] H. Gomaa: “Designing Concurrent, Distributed, and Real-Time Applications with UML”. Addison-Wesley. 2000.
- [Jigorea00] R. Jigorea, S. Manolache, P. Eles, Z. Peng: “Modelling of Real-Time Embedded Systems in an Object-Oriented Design Environment with UML”. 2nd ARTES Graduate Student Conference. Chalmers University of Technology, Göteborg (Sweden). March 2000.
- [Küster01] J. M. Küster, J. Stroop: “Consistent Design of Embedded Real-Time Systems with UML-RT”. *4th IEEE International Symposium on Object Oriented Real-Time Distributing Computing ISORC 2001*. Los Alamitos, CA, USA, IEEE Computer Society 2001, pp. 31-40 Conference Paper.
- [Lavazza01] L. Lavazza, G. Quaroni, M. Venturelli: “Combining UML and Formal Notations for Modelling Real-Time Systems”. *Software Engineering Notes*, vol. 26, no. 5, pp. 196-206, September 2001.

- [Liu73] C. L. Liu, J. W. Layland: "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment". *Journal of the ACM*, vol. 20, no. 1, pp. 46-61, January 1973.
- [Lonn99] H. Lonn, J. Axelsson: "A Comparison of Fixed-Priority and Static Cyclic Scheduling for Distributed Automotive Control Applications". *In Proceedings of the 11th Euromicro Conference on Real-Time Systems (ECRT99)*. York, England. 1999.
- [Martins03] P. Martins: "Integrating Real-Time UML Models with Schedulability Analysis". White Paper in Tri Pacific Software Products 2003.
- [Saiedian04] H. Saiedian, S. Raguraman: "Using UML-Based Rate Monotonic Analysis to Predict Schedulability". *IEEE Computer Society*, vol. 37, no. 10, pp. 56-63, October 2004.
- [Selic98] B. Selic, J. Rumbaugh: "Using UML for Modeling Complex Real-Time Systems". White Paper, published by ObjecTime Limited, 340 March Rd., Kamata, Astario, Canada. 1998.
- [UML03] UML 2.0 Superstructure Specification. August 2003, ptc/03-08-02. Unified Modeling Language Infrastructure Specification, September 2003, ptc/03-09-15. <http://www.omg.org/>.
- [UMLSPT03] UMLTM Profile for Schedulability, Performance, and Time Specification. January 2005, version 1.1, formal/05-01-02. <http://www.omg.org/>.



About the authors



Maria Cruz Valiente is a PhD researcher and teaching assistant in the Department of Informatics at the Carlos III University of Madrid. Her research in Information Engineering Group is focused on domain analysis and modeling, and modeling methods in software engineering.



Gonzalo Genova is an Associate Professor of Software Engineering in the Department of Informatics at the Carlos III University of Madrid. His main research subjects, within the Information Engineering Group and in close cooperation with The Reuse Company, is modeling and modeling languages in software engineering, as well as requirements engineering.



Dr. Jesus Carretero is a professor in the Department of Informatics at the Carlos III University of Madrid. His research interests include parallel and distributed systems and real time systems.