Modeling and Analysis of Real Time Fixed Priority Scheduling using UML 2.0

RUMPA HAZRA[1]
SHOUVIK DEY[2]
ANANYA KANJILAL[3]
SWAPAN BHATTACHARYA[4]

[1]Heritage Institute of Technology, Kolkata, India
[2]Cognizant Technology Solutions, Kolkata, India
[3]B.P. Poddar Institute of Management and Technology, Kolkata, India
[4]National Institute of Technology, Surathkal, Karnataka, India
[1](send2rumpa)@gmail.com
[2]send2shouvik@gmail.com
[3]ag_k@rediffmail.com
[4]bswapan2000@yahoo.co.in

Abstract. Real Time Systems (RTS) interact with their environments using time constrained input/output signals. A functional misbehavior or a deviation from the specified time constraints may have catastrophic consequences. Hence, ensuring the correctness of such systems is extremely important and necessary. The increasing complexities of now-a-days ubiquitous real time systems require using an adequate modeling language. Unified Modeling Language (UML), a widely used visual object oriented modeling language, has proved to be effective and suitable for real time systems. The paper discusses the ability of UML and its profile to determine the schedulability of a fixed priority real time system. This paper puts stresses on the occurrence of deadlock in using the Priority Inheritance Protocol and prevention of such using the Priority Ceiling Protocol. Using UML 2.0 Sequence and Timing Diagrams, we model these two protocols and further, we analyze and compare these models.

Keywords: Real Time Systems, UML, Priority Ceiling Protocol, Priority Inheritance Protocol, Deadlock

(Received March 31st, 2013 / Accepted August 8th, 2013)

1 Introduction

Real time systems are now omnipresent in modern societies in several domains such as avionics, control of nuclear power station, multimedia communications, robotics, systems on chip, air-traffic control, process control, and numerous embedded systems etc. Developing a real time embedded system is a sophisticated and complex task.

A real time system is one in which failure can occur in the time domain as well as in the more familiar value domain. These systems can have a mixture of timing constraints, broadly categorised as hard and soft. A hard time constraint requires that a result must be produced within a bounded interval otherwise a serious fault is said to occur. In a soft real time occasional timing faults may be permitted. Examples of soft real time system are video play back system, on line transaction system, telephone switches as well as electronic games.

The real world is inherently concurrent, and a real time system which is linked to the behaviour of the real world must behave in a concurrent manner. Real time systems are therefore usually engineered using a number of concurrently running tasks, with timing constraints placed on them. Because of this concurrency there is contention for resources, requiring scheduling (i.e. how tasks are granted access to a given resource). The processor is an example of a resource which must be scheduled, but other resources (such as network or disk drive bandwidth) may also need to be scheduled.

In RTS, scheduling of tasks with hard deadlines has been an important area of research. An important problem that arises in the context of such real time systems is the effect of blocking. Blocking is occurred due to the need for synchronization of tasks that share common logical or physical resources.

UML which is the de facto standard has become one of the most widely used modeling languages for industrial software systems, essentially because it is a semi-formal notation, relatively easy to use and well supported by tools. It encourages the use of automated tools that facilitate the development process from analysis through coding. This is particularly true for real time embedded systems, whose behavioural aspects can often be described via UML. It is therefore interesting to consider how well UML is adapted to the real time context. One important feature of UML stems from its built-in extensibility mechanisms: stereotypes, tag values and profiles. These allow adapting UML to fit the specificities of particular domains or to support a specific analysis.

The main contribution in this paper is to model, analyze and compare two existing protocols (Priority Inheritance and Priority Ceiling) using UML 2.0 Sequence and Timing diagram. We could not find any such related works where this type of comparative analysis has been done using UML model.

The paper is structured as follows. In section 2 we provide the background of related work on which the models are built. Section 3 describes the actual scope of this work. Section 4 summarises the two real time scheduling protocols considered here. In Section 5, we focus on some of the new built-in features of UML 2.0 that fit the requirements of real time systems. Section 6 gives detail of our proposed work and finally section 7 concludes the paper.

2 Related Works

The vast majority of research work on real time systems is centered on the concept of task. Real time theory does not focus on the problem of generating the task set and assigning non-functional properties to tasks. Generally, task sets are assumed to be given by the designer, using some adhoc software design methodology [1]. The concept of a task is central to both the design and analysis of real time systems.

In particular, formal studies of real time systems frequently represent the time-constrained processing requirements of the system as a set of periodic or sporadic tasks with deadlines [15, 14, 3]. Both preemptive and non preemptive scheduling algorithms have been studied in the literature [12, 13, 14].

Exclusive access to shared resources is typically ensured by having a semaphore [4] guard. In priority inversion [23] higher priority jobs may be blocked by lower-priority tasks. In one of the earlier attempts at tackling blocking in the abstract (as opposed to with respect to a particular environment) Mok [15]proposed that critical sections execute non-preemptively; while this approach restricts blocking to the length of the largest critical section, it has the drawback that even those tasks that do not ever access shared resources are subject to blocking.

Lampson and Redell studied priority inversion and blocking with respect to concurrent programming in the Mesa environment [9, 10], and proposed a number of solutions. These were generalized by Sha, Rajkumar, and Lehoczky [23], and incorporated into the rate-monotonic (RM) scheduling framework [7].

UML (Unified Modeling Language)[17] has become one of the most widely used standards for modeling and designing industrial software systems, essentially because it is a semiformal notation, relatively easy to use and well supported by tools. UML provides a variety of instruments to describe the characteristics of a generic system in corresponding models. However, it is not complete, in the sense that the basic elements of the language cannot cover all potential needs for describing specific systems from any domain. Hence in some cases the definition of domain-specific variants of the UML may be required. The UML however has already been conceived for extensibility, for which purpose it provides a built-in profiling mechanism to extend the language with elements and constructs apt to describe specialized features, though remaining compliant with its standard definition [15].

An extension to UML, called UML-RT [22], has been defined on the basis of ROOM language [21], which is a useful architectural definition language specifically developed for modeling complex real time systems (RTS), and one which is becoming a standard in the industry for RTS development. UML-RT extends UML with stereotyped active objects, called capsules to represent system components, where the internal behavior of a capsule is defined using state machines. The interaction with other capsules takes place by means of protocols that define the sequence of signals exchanged through stereotyped objects called ports and specify the desired behavior over a connector.

UML has been also used in a large number of time-critical and resource-critical systems [20]. Despite its real time capabilities [22, 5, 8, 6] 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[2, 11].

The UML Profile for Schedulability, Performance and Time (SPT-Profile) [20] has been proposed by a working consortium of OMG member companies. A different profile proposed in [1] focuses specifically on the Scheduling sub-profile of the SPT. SPT is still based on version 1.5 of the UML [16], now superseded by the new UML 2.0 superstructure[18]. Indeed, to respond to the changes introduced by the UML 2.0 and also to address several requested improvements to better specify the properties of real time embedded systems, the OMG has now issued a new RFP for UML Profile for Modeling and Analysis of Real time and Embedded Systems (MARTE) [19].

Maria et al. [24] proposed the capabilities of UML for task scheduling in RTS. They identified a task set and showed whether the task set is schedulable or not using UML model. They use the Priority Inheritance Protocol to share critical resources. But Priority Inheritance Protocol does not prevent deadlock. Their work did not highlight this issue. We, in this work, have considered the occurrence of deadlock using Priority Inheritance Protocol and propose a better approach to overcome it.

3 Scope of Work

This paper concentrates on the occurrence of deadlock when Priority Inheritance Protocol is used and prevention of such using the Priority Ceiling Protocol. The main objective of this paper is to compare these two protocols using UML 2.0 Sequence and Timing Diagrams.

The Priority Inheritance Protocol is used for sharing critical resources but it does not prevent deadlock if nested critical sections are used. The shortcomings of the existing Priority Inheritance Protocol are represented using one UML model. Further the Priority Ceiling Protocol is used to overcome these difficulties using an improved model.

We therefore analyze the schedulability of an application with the following characteristics: Task set composed of three dependent periodic tasks T1,T2,T3.

- Tasks T1,T2 and T3 share a critical resource(R1) and tasks T2 and T3 share a critical resource(R2)
- 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.

4 Real Time Scheduling

The scheduler is the part of the operating system that responds to the requests sent by programs. It interrupts and gives control of the processor to those processes. A scheduler provides an algorithm or policy that determines in which order processes get processor for execution according to some pre-defined criteria. In a conventional multitasking operating system, processes interleaved with higher importance (or priority) processes receive preference. Little or no account is taken of deadlines. This is clearly inadequate for real time systems. These systems require scheduling policies that reflect the timeliness constraints of real time processes.

Schedulers produce a schedule for a given set of processes. If a process set can be scheduled to meet given pre-conditions the process set is termed feasible. A typical pre-condition for hard real time periodic processes is that they should always meet their deadlines. An optimal scheduler is able to produce a feasible schedule for all feasible process sets conforming to a given precondition. For a particular process set an optimal schedule is the best possible schedule according to some pre-defined criteria. Typically a scheduler is optimal if it can schedule all process sets that other schedules can.

Schedulers may be preemptive or non-preemptive. The former can arbitrarily suspend a process′s execution and restart it later without affecting the behaviour of that process (except by increasing its elapsed time). Preemption typically occurs when a higher priority process becomes runnable. The effect of preemption is that a process may be suspended involuntarily.

A Non-preemptive scheduler does not suspend a process in this way. This is sometimes used as a mechanism for concurrency control for processes executing inside a resource whose access is controlled by mutual exclusion. Many real time application systems are composed of several independent tasks. Every task in real time systems is commonly executed in a priority-based manner. When a task releases, at that time a unique priority is assigned to it. The highest-priority active task is selected for execution at each instant in time. To ensure efficient response time it is required to prioritize processes so that more important processes always receive processor attention first if they need it. Independent tasks in a RTS execute on a shared computing platform comprised of a preemptable processor and serially reusable non-preemptable resources. Each task requires the processor in order to execute; in addition, some tasks may simultaneously need exclusive access to one or more of the resources during part or all of their execution. Exclusive access to these shared resources is typically ensured only within critical sections.

The notion of priority is commonly used to order access to the processor and other shared resources such as communication channels. In priority scheduling, each task is assigned a priority via some policy. Contention for resources is resolved in favour of the task with the highest priority that is ready to run.

4.1 Priority Inversion

A priority inversion occurs if the highest priority active task cannot execute because some of the resources needed for its execution are held by some other tasks. At that point of time the higher priority task is blocked while the lower-priority tasks execute.

In order to overcome the priority inversion the Priority Inheritance Protocol can be used.

4.2 Priority Inheritance Protocol

Assigned Priority: When a task releases, this priority is assigned to the task. It is a unique priority. Current Priority: It is the priority at which a ready task is scheduled and executed. It may vary with time.

4.2.1 Rules of the Priority Inheritance Protocol

1. Scheduling rule:- A ready task is scheduled preemptably in a priority driven manner according to current priority.
2. Allocation rule:- When a task T requests a resource R,
   a) If R is free it is allocated to T and is held by T until T releases R.
   b) If R is not free then the task is blocked.
3. Priority Inheritance rule:- When requesting task T becomes blocked, the task T_l which blocks T inherits the current priority Π(t) of T. When T_l releases R its priority reverts to Π(t′), where t′ is the time it acquired the resource R.

   The Priority Inheritance Protocol does not prevent deadlock, which is explained in the next section using an example. The Priority Ceiling Protocol can be used to overcome deadlock.

4.3 Basic Priority Ceiling Protocol

The Priority Ceiling Protocol extends the Priority Inheritance Protocol to prevent deadlocks. This protocol makes two key assumptions:

i) The assigned priority of all tasks is fixed.

ii) The resources required by all tasks are known a priori before the execution of any task begins.

The priority ceiling of a critical resource R is the highest priority of all the tasks that use the resource R. Current priority ceiling of a system $\Pi'(t)$at any time $'t'$ is equal to the highest priority ceiling of all the resources used at that time. If all the resources are free, then $\Pi'(t)=\Omega$ where Ω is a non existing priority level that is lower than the lowest priority of all tasks.

4.3.1 Rules of the Basic Priority Ceiling Protocol

1. Scheduling rule:- At the release time t, the current priority Π(t) of every task is equal to the assigned priority. The task remains at that priority level except by rule 3.
2. Allocation rule:- When a task T request R, one of the following conditions occur:
   a) If R is not free then T becomes blocked
   b) If R is free then one of the following conditions occur:
      i) T′s priority is higher than the current priority ceiling $\Pi'(t)$, R is allocated to T.
      ii) If T′s priority Π(t) is not higher than the priority ceiling $\Pi'(t)$,R is allocated to T only if T is the task holding the resource whose priority ceiling is $\Pi'(t)$.Otherwise T′s request is denied.
3. Priority Inheritance rule:- When T becomes blocked, the task T_l which blocks T inherits the current priority Π(t) of T. T_l executes at its inherited priority until the time it releases every resource whose priority ceiling is equal to higher than Π(t). At that time priority of T_l returns to its priority $\Pi'(t')$ at the time t′ when it was granted the resource.

5 UML 2.0 for Real Time System

The Unified Modelling Language (UML) is a graphical modeling language for visualizing, specifying, constructing and documenting the artifacts of software systems.UML is widely used to express the general-purpose software design models.

UML as a real time modeling language has some limitations. It basically provides a lot of syntax, but not enough semantics. The UML profile for real time modeling, formally called the UML profile for Schedulability, Performance and Time (UML/SPT), was adopted by the OMG in 2002 [20].

Core of the SPT profile is the general resource modeling framework, itself consisting of three sub-profiles dealing respectively with resource modeling, concurrency and time-specific concepts. Then, based on this common framework, more specific sub profiles are defined. It is supposed to overcome the limitations which are related to and provide suitability for modeling real time systems.

The SPT profile does not invent any new techniques, but offers the possibility to exchange timeliness properties between UML modeling tools and schedulability analysis tools. The profile defines a number of stereotypes, tagged values and constraints, the user can add more features depend on requirements.

UML 2.0 provides some concepts, including active objects, concurrent composite states and concurrent operations. In order to express timing constraints, UML 2.0 provides two data types: Time and TimeExpression. These timing statements can be used either in state diagrams or in Sequence diagrams. Moreover, UML 2.0 introduces a new diagram called Timing Diagram to allow reasoning about time to visualize conditions or state changes over time. UML can better model the real time software systems through its extended features for real time systems. In this work we model our work using some of these features.

6 PROPOSED WORK

In order to describe the task set (mentioned in section 3) and the critical sections of tasks T1, T2 and T3, new parameters are added that specify the three components of any critical section.

- Ca: task duration before entering the critical section
- Cb: task duration within the critical section
- Cc: task duration after the critical section

Then, the computation time becomes C= Ca + Cb + Cc

In [24] the Priority Inheritance Protocol is used for sharing a critical resource but this protocol does not prevent deadlock. In this paper the occurrence of deadlock is first highlighted by considering the following task set which is described by the classical parameters given in Table 1. Further, deadlock avoidance is discussed using the Priority Ceiling Protocol by considering the same task set.

Table 1: A Task Set Sharing Critical Resources

$Task$	R_i	C_i	C_a	C_b	C_c	$priority$
T_1	4	2	1	1	0	1
T_2	2	5	1	4	0	2
T_3	0	5	1	4	0	3

R_i represents release time of task T_i, C_i represents computation time of task T_i and P_i represents priority of task T_i.

6.1 Drawback of Priority Inheritance Protocol

Deadlock occurrence is illustrated in Figures 1 and 2 using a UML 2.0 sequence diagram and a UML 2.0 timing diagram, respectively.

6.1.1 Description of UML 2.0 Sequence Diagram

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, "Priority Inheritance Protocol". Each lifeline in the diagram represents an individual participant in the scenario.

s1: Scheduler. A scheduler (in our domain, a processor) is responsible for processing the acquisition requests from the clients of a 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 the access is possible. The scheduler determines a schedule that allocates a set of scheduling tasks to its set of execution engines.

r1, r2: 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 task. 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 tasks (in our model, ′Priority Inheritance′).

T1, T2, T3: 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

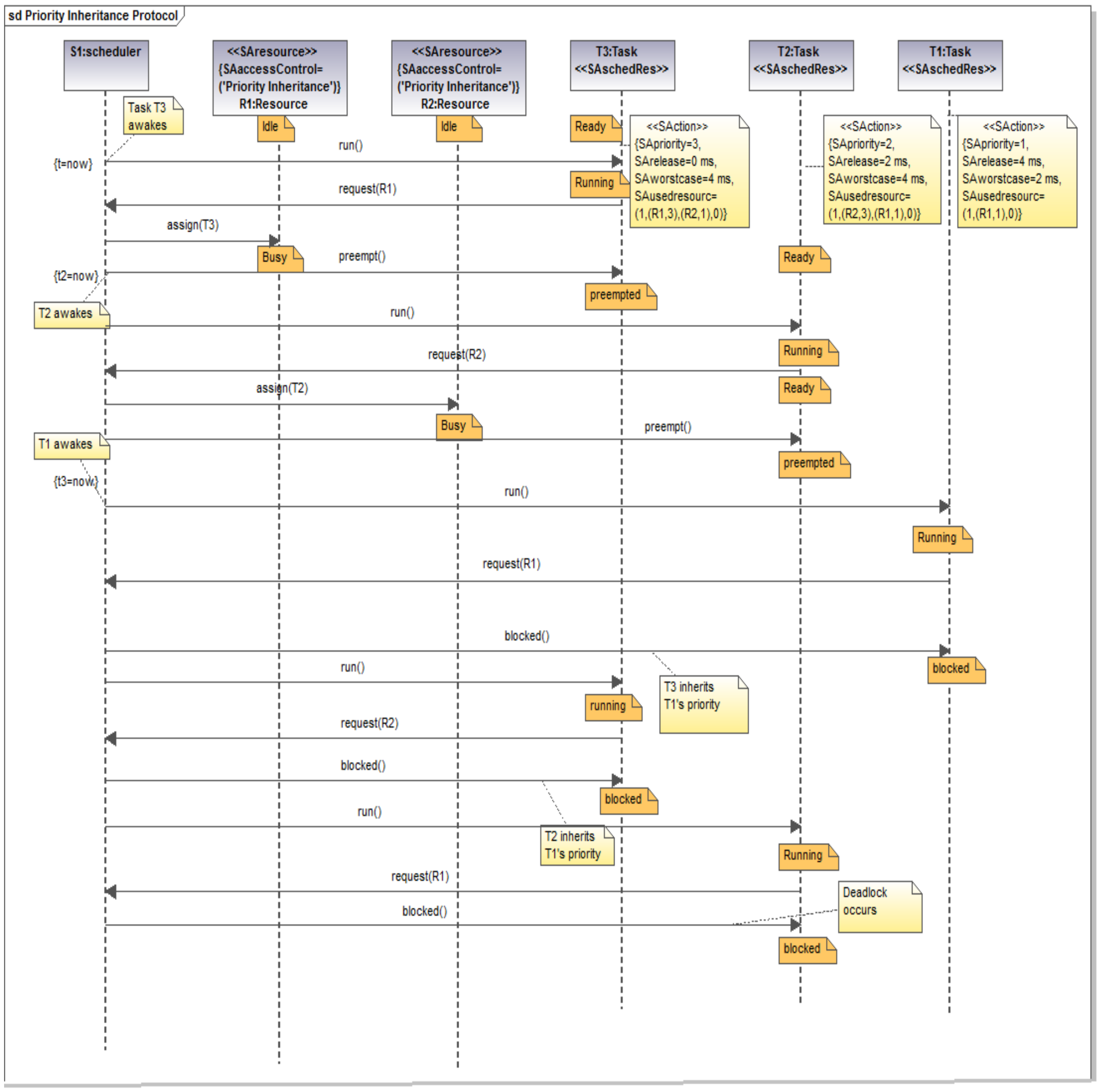
sd Priority Inheritance Protocol
S1:scheduler
<<SAresource>> {SAaccessControl= ('Priority Inheritance')} R1:Resource
<<SAresource>> {SAaccessControl= ('Priority Inheritance')} R2:Resource
T3:Task <<SASchedRes>>
T2:Task <<SASchedRes>>
T1:Task <<SASchedRes>>
Task T3 awakes
Idle
Idle
Ready
<<SAction>> {SApriority=3, SArelease=0 ms, SAworstcase=4 ms, SAusedresource= (1,(R1,3),(R2,1),0)}
<<SAction>> {SApriority=2, SArelease=2 ms, SAworstcase=4 ms, SAusedresourc= (1,(R2,3),(R1,1),0)}
<<SAction>> {SApriority=1, SArelease=4 ms, SAworstcase=2 ms, SAusedresourc= (1,(R1,1),0)}
{t=now}
run()
Running
request(R1)
assign(T3)
Busy
preempt()
Ready
{t2=now}
preempted
T2 awakes
run()
Running
request(R2)
Ready
assign(T2)
Busy
T1 awakes
preempt()
preempted
{t3=now}
run()
Running
request(R1)
blocked()
blocked
run()
T3 inherits T1's priority
running
request(R2)
blocked()
blocked
run()
T2 inherits T1's priority
Running
request(R1)
Deadlock occurs
blocked()
blocked

Figure 1: Sequence diagram showing deadlock occurrence using the example task set given in Table 1

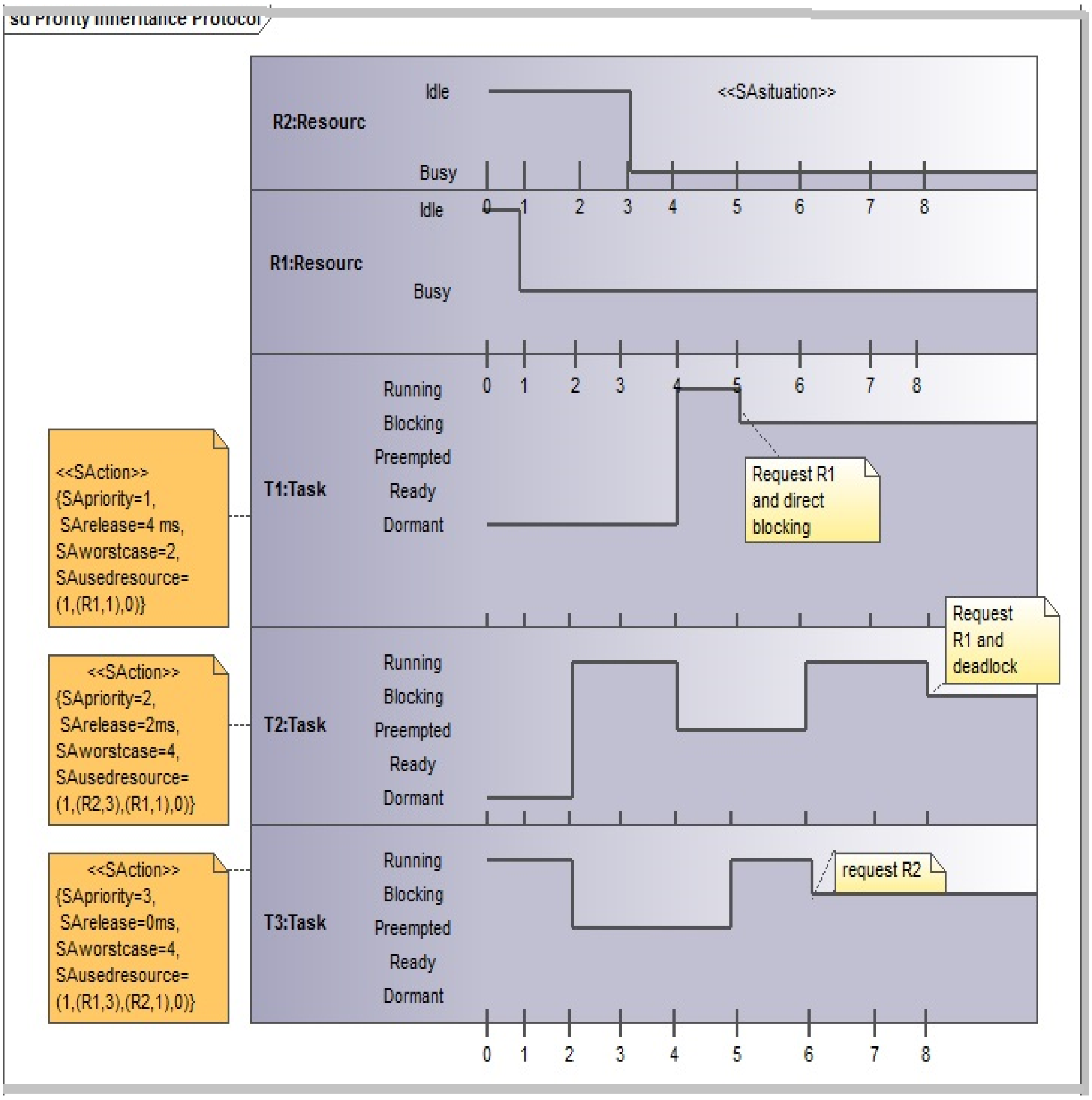
sd Priority Inheritance Protocol
R2:Resourc
Idle
Busy
<<SAsituation>>
R1:Resourc
Idle
Busy
T1:Task
T2:Task
T3:Task
Running
Blocking
Preempted
Ready
Dormant
0 1 2 3 4 5 6 7 8
<<SAction>> {SApriority=1, SArelease=4 ms, SAworstcase=2, SAusedresource=(1,(R1,1),0)}
<<SAction>> {SApriority=2, SArelease=2ms, SAworstcase=4, SAusedresource=(1,(R2,3),(R1,1),0)}
<<SAction>> {SApriority=3, SArelease=0ms, SAworstcase=4, SAusedresource=(1,(R1,3),(R2,1),0)}
Request R1 and direct blocking
Request R1 and deadlock
request R2

Figure 2: Timing diagram showing deadlock occurrence using the example task set given in Table 1

a single scenario concurrently with other concurrent units. In the general resource modeling of the UML Profile for Schedulability, Performance and Time, an action is defined as a kind of scenario. Therefore, the stereotype <<SAaction>> of this profile (schedulability modeling) is used to characterize the behaviour of each task in the proposed model.

The new metaclass in UML 2.0, TimeObservationAction, is used to know when a task awakes. A time observation triggers 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".

Another new metaclass in UML 2.0, StateInvariant, is used to show the different states associated with each lifeline as restrictions. A state invariant is a constraint on the state of a lifeline. If the constraint is true, the trace is a valid trace.

Finally, notes are used to display the textual information.

6.1.2 Observation from Sequence Diagram

The sequence diagram in Figure 1 shows that deadlock occurs. T1 is blocked by T3. T3 is waiting for a resource that is held by T2. T2 is waiting for a resource that is held by T3. As a result all of the three tasks are in the blocking state.

6.1.3 Description of UML 2.0 Timing Diagram

The timing diagram can be stereotyped as <<SAsituation>> to use it in the context of schedulability analysis, representing a real time situation.

The notations of the rectangular frame and the five sided box are same as in the previous sequence diagram, but now we have different elements in the model. Five lifelines are generated one each for the two resources (r1, r2) and the three tasks (T1, T2 and T3) respectively. In this case, scheduler (s1) can be ignored, because it is not necessary to understand the scheduling. Since the changes in states of different lifelines can be represented over linear time, there is no need to show the message passing.

The task states used in the timing diagram are explained in Table 2. There are two simple states for the resource lifeline : idle and busy. Using the timing diagrams it can be seen how the states get changed over time for each lifeline. Therefore, it is not required 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, it is not required to use notes indicating when a task awakes (when the state of a task changes to "Ready") [24].

6.1.4 Result and discussion

Using Timing diagram it can be explained how deadlock occurs in Priority Inheritance Protocol.

At time 0, T3 is released and executes at its assigned priority 3. At time 1, resource R1 is assigned to T3.

At time 2, T2 is released. It pre-empts T3 (as priority of T2 is greater than priority of T3) and starts to execute.

At time 3, T2 requests resource R2. R2, being free, is assigned to T2. The task T2 continues to execute.

At time 4, T1 is released and it pre-empts T2 (as priority of T1 is greater than priority of T2).

At time 5, T1 requests R1 but R1 is already assigned to T3. So T1 is now directly blocked by T3 though priority of T1 is greater than priority of T3. According to rule 3, T3 inherits T1′s priority (i.e. 1) and T3 continue execution.

At time 6, T3 request R2 but R2 is already assigned to T2. So T3 is blocked by T2 though current priority of T3 (presently priority of T3 is 1 which it inherits from T1) is greater than priority of T2. According to rule 3, T2 inherits T3′s priority (i.e. 1) and T2 continue execution.

At time 8, T2 request for R1 but R1 is already assigned to T3. So T2 is blocked by T3. As T3 is already blocked by T2, deadlock occurs.

Table 2: Task states

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

6.2 Sequence diagram and Timing diagram

Timing diagrams and Sequence diagrams are the two kinds of interaction diagram more adequate to model task scheduling. UML allows modeling the traces of interactions among many objects working together and providing the important information required for schedulability analysis, which is captured in Sequence or Timing diagrams. 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. In this work both Sequence diagram and Timing diagram are used because Sequence diagram or Timing diagram alone does not depict the scenario completely.

6.2.1 Advantage of Sequence diagram over Timing diagram

UML Sequence diagrams are used to model the flow of messages, events and actions between the objects or components of a system.

Sequence diagrams are often used to design the interactions between components of a system that need to work together to accomplish a task.

It focuses on when the individual objects interact with each other during execution. It is particularly useful for modeling usage scenarios such as the logic of methods and the logic of services.

Sequence diagrams emphasize message sequence, so the in time of the next message is the message following the current one on the diagram.

Timing diagram does not represent these.

6.2.2 Advantage of Timing diagram over Sequence diagram

A Timing diagram is a simple representation with time along the horizontal axis and objects state or attribute value along the vertical axis.

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. Messages on Sequence diagrams are only partially ordered, so in many cases the relative timing between two messages is not specified.

When messages on Sequence diagrams begin or finish on different lifelines, it is not possible to compare which one starts or terminates first.

Time goes down the page on Sequence diagrams, 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. However, each diagram provides different points of view to the same scenario and both could be very useful.

6.3 Deadlock Avoidance

Deadlock avoidance is illustrated in Figures 3 and 4 using a UML 2.0 Sequence diagram and a UML 2.0 Timing diagram, respectively. Priority Ceiling Protocol is used to prevent deadlock.

6.3.1 Observation from Sequence Diagram

From the Sequence diagram it can be easily seen that deadlock can be prevented. All the three tasks complete their executions.

6.3.2 Result and discussion

The Timing diagram shows how deadlock can be prevented using Priority Ceiling Protocol.

T3 is released at time 0. Ceiling of the system at time 1 is Ω. When T3 requests R1, it is allocated to T3 according to (i) in part (b) of rule 2. After the allocation of R1, the ceiling of the system is raised to 1, the priority ceiling of R1.

At time 2, T2 is released and it pre-empts T3 (as priority of T2 is greater than priority of T3). At time 3, T2 requests resource R2. R2 is free; however because the ceiling $\Pi'(3)(=1)$ of the system is higher than priority of T2, T2′s request is denied according to (ii) in part (b) of rule 2. T2 is blocked and T3 inherits T2′s priority.

At time 4, T1 is released and it pre-empts T3 (as priority of T1 is greater than priority of T3).

At time 5, T1 requests resource R1 and becomes directly blocked by T3 and T3 inherits T1′s priority. At time 5, T3 requests for resource R2. R2 is free and it is allocated to T3 because T3 holding the resource R1 whose priority ceiling is equal to $\Pi'(t)(=1)$.

At time 6, T3 releases R2 and at time 7, T3 releases R1. So T3 executes at its inherited priority $\Pi(t)$ (=1) until the time when it releases every resource whose priority ceiling is equal to higher than $\Pi(t)$(i.e. its inherited priority). T3 completes its execution at time 7.

At that time 7, T1 and T2 are ready. But T1 has higher priority (i.e.,1) it resumes.

At time 8, T1 completes its execution and T2 resumes.

7 Conclusions

The behavior of real time software systems do not depend only on the values of input and output signals, but also on their time of occurrences. Ensuring the correctness of such systems within the specified time constraints is a difficult and complex task. Therefore, complexity of real time systems is continuously increasing which makes their design very challenging. Unified Modeling Language (UML), the standard visual object-oriented modeling language, is suitable to deal with this complexity.

In the last few years, real time processing seems to be the essential part of an operating system, and the scheduling of real time systems is an important area of research in today′s life. In this paper, we consider fixed

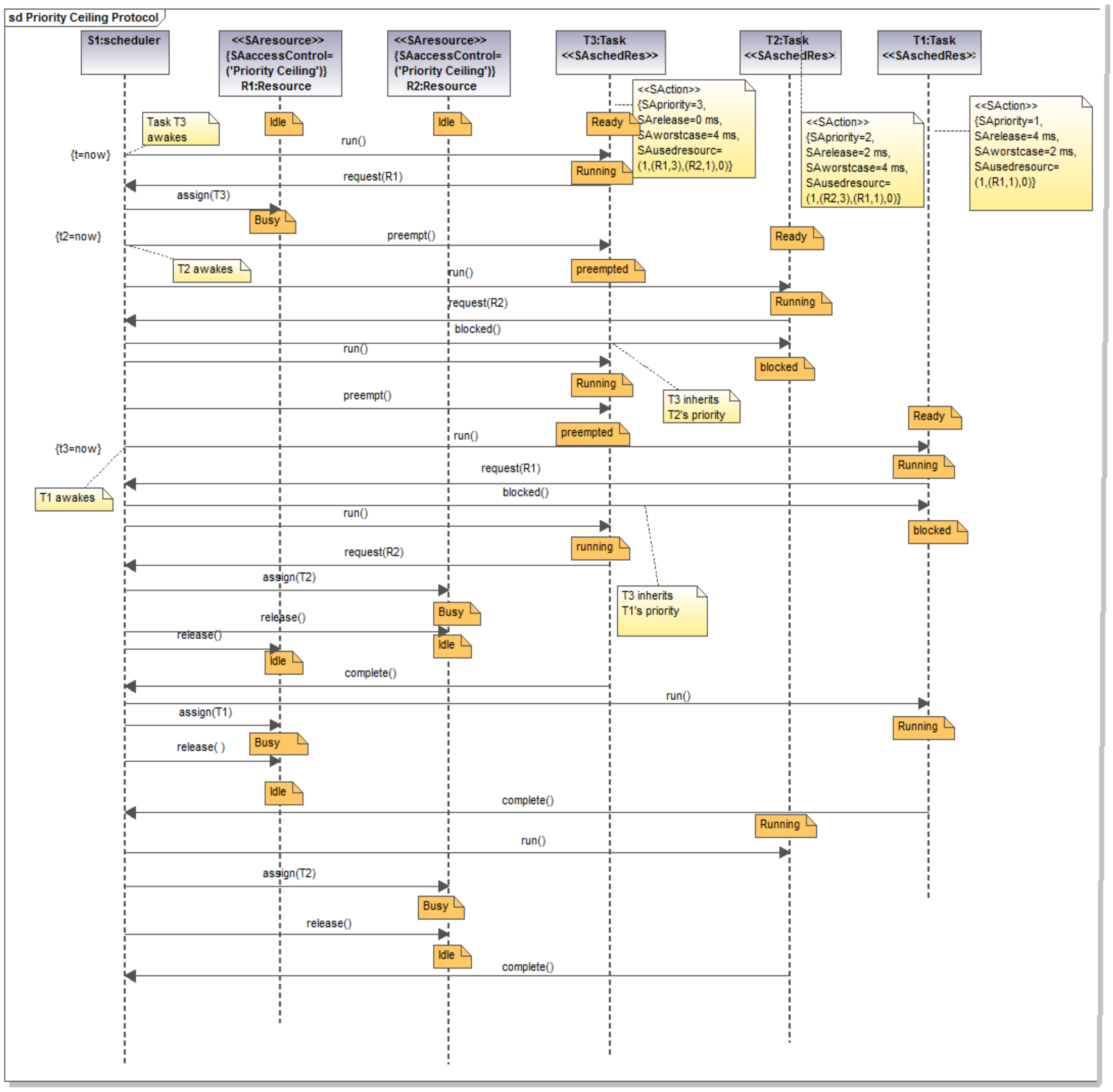
sd Priority Ceiling Protocol
S1:scheduler
<<SAresource>> {SAaccessControl= ('Priority Ceiling')} R1:Resource
<<SAresource>> {SAaccessControl= ('Priority Ceiling')} R2:Resource
T3:Task <<SAschedRes>>
T2:Task <<SAschedRes>>
T1:Task <<SAschedRes>>
<<SAction>> {SApriority=3, SArelease=0 ms, SAworstcase=4 ms, SAusedresourc= (1,(R1,3),(R2,1),0)}
<<SAction>> {SApriority=2, SArelease=2 ms, SAworstcase=4 ms, SAusedresourc= (1,(R2,3),(R1,1),0)}
<<SAction>> {SApriority=1, SArelease=4 ms, SAworstcase=2 ms, SAusedresourc= (1,(R1,1),0)}
Task T3 awakes
{t=now}
Idle
Idle
Ready
run()
Running
request(R1)
assign(T3)
Busy
{t2=now}
preempt()
Ready
T2 awakes
preempted
run()
Running
request(R2)
blocked()
run()
blocked
Running
T3 inherits T2's priority
preempt()
Ready
preempted
run()
{t3=now}
Running
request(R1)
T1 awakes
blocked()
run()
blocked
running
request(R2)
assign(T2)
T3 inherits T1's priority
Busy
release()
release()
Idle
Idle
complete()
run()
assign(T1)
Running
Busy
release( )
Idle
complete()
Running
run()
assign(T2)
Busy
release()
Idle
complete()

Figure 3: Sequence diagram showing deadlock avoidance using Table 1

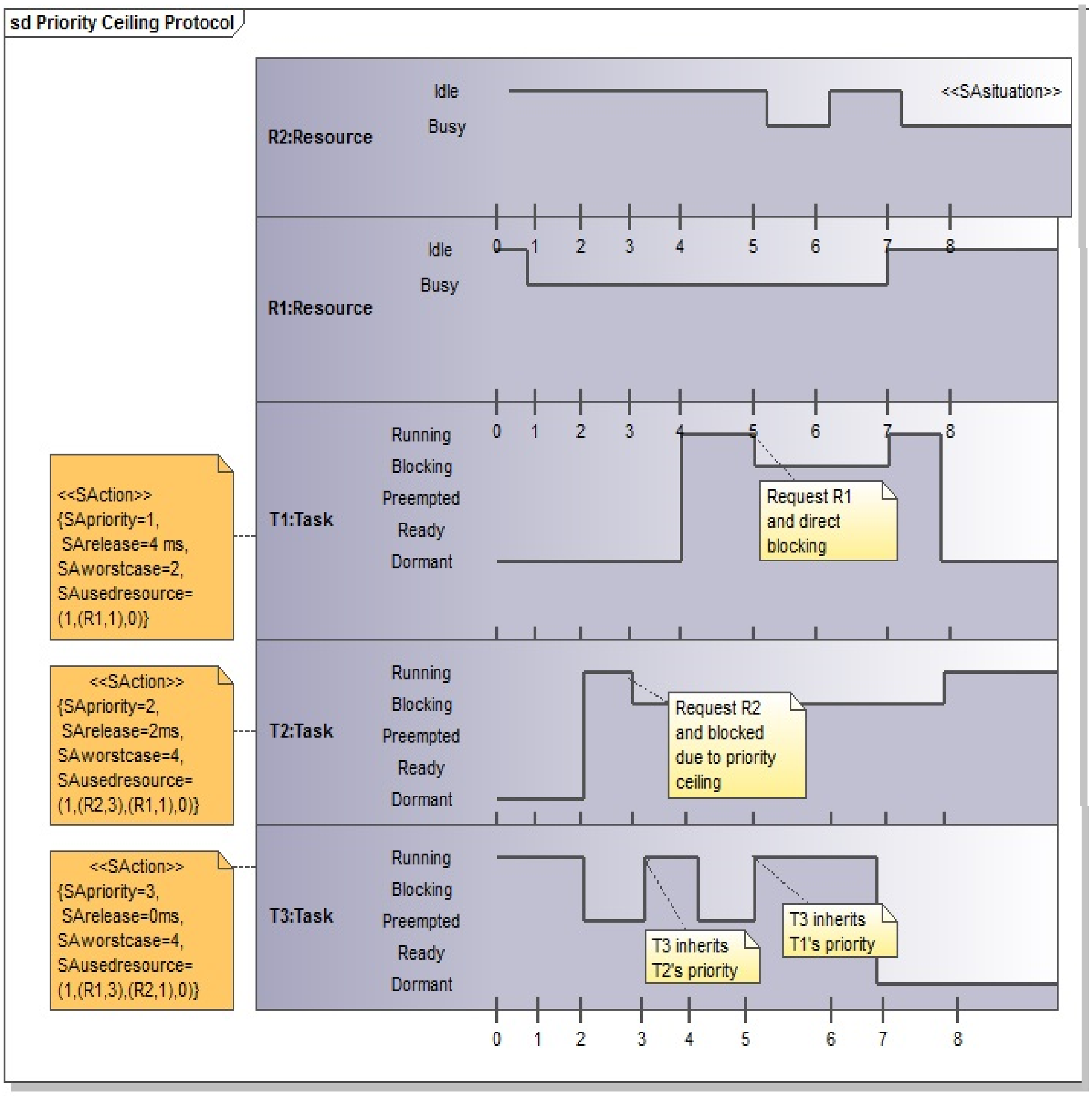
sd Priority Ceiling Protocol
R2:Resource
Idle
Busy
<<SAsituation>>
R1:Resource
Idle
Busy
T1:Task
Running
Blocking
Preempted
Ready
Dormant
Request R1 and direct blocking
T2:Task
Running
Blocking
Preempted
Ready
Dormant
Request R2 and blocked due to priority ceiling
T3:Task
Running
Blocking
Preempted
Ready
Dormant
T3 inherits T2's priority
T3 inherits T1's priority
0 1 2 3 4 5 6 7 8
<<SAction>> {SApriority=1, SArelease=4 ms, SAworstcase=2, SAusedresource=(1,(R1,1),0)}
<<SAction>> {SApriority=2, SArelease=2ms, SAworstcase=4, SAusedresource=(1,(R2,3),(R1,1),0)}
<<SAction>> {SApriority=3, SArelease=0ms, SAworstcase=4, SAusedresource=(1,(R1,3),(R2,1),0)}

Figure 4: Timing diagram showing deadlock avoidance using Table 1

priority scheduling. A model (using UML 2.0 Sequence diagram and UML 2.0 Timing diagram) has been developed to represent deadlock occurrence as a drawback of Priority Inheritance Protocol. Further, Priority Ceiling Protocol is used in another improved model (using UML 2.0 Sequence diagram and UML 2.0 Timing diagram) to overcome this difficulty.

As the UML profile for Schedulability, Performance, and Time is clearly biased towards fixed priority scheduling (such as Rate Monotonic), we like to extend the specification to comprise dynamic scheduling (such as Earliest Deadline First). In future we plan to develop a model of dynamic priority scheduling (such as Earliest Deadline First) for the prevention of deadlock in RTS.

References

[1] Bertolino, A., Angelis, G. D., Bartolini, C., and Lipari, G. A uml profile and a methodology for real-time systems design. In *Proceedings of the 32nd EUROMICRO Conference on Software Engineering and Advanced Applications, IEEE Computer Society Washington, DC, USA*, pages 108–117, 2006.

[2] Bichler, L., Radermacher, A., and Smith, A. S. J. Evaluating uml extensions for modeling real-time systems. In *Proceedings of 7th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS 2002)*, pages 271–8, Los Alamitos, CA, USA, 2002. IEEE Computer Society.

[3] C.L., L. and J.W., L. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, Jan 1973.

[4] Dijkstra, E. W. Cooperating sequential processes. In Genuys, F., editor, *Programming Languages*, pages 43–112. Academic Press, 1968.

[5] Douglass, B. Designing real-time systems with uml, embedded systems programming, 1998.

[6] Jigorea, R., Manolache, S., Eles, P., and Peng, Z. Modelling of real-time embedded systems in an object-oriented design environment with uml. In *2nd ARTES Graduate Student Conference*, Chalmers University of Technology, Goteborg, Sweden, Mar 2000.

[7] Klein, M., Ralya, T., Pollak, B., Obenza, R., and Harbour, M. G. *A Practitioners Hand book for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems*. Kluwer Academic Publishers, Boston , MA, 1993.

[8] Kuster, J. M. and Stroop, J. Consistent design of embedded real-time systems with uml- rt. In *4th IEEE International Symposium on Object Oriented Real-Time Distributing Computing ISORC 2001*, pages 31–40, Los Alamitos, CA, USA, 2001. IEEE Computer Society.

[9] Lampson, B. and Redell, D. D. Experience with processes and monitors in mesa. In *Proceedings of the 7th ACM Symposium on Operating Systems Principles*, pages 43–44, 1979.

[10] Lampson, B. and Redell, D. D. Experience with processes and monitors in mesa. *Communications of the ACM*, 23(2), Feb 1980.

[11] Lavazza, L., Quaroni, G., and Venturelli, M. Combining uml and formal notations for modelling real-time system. *ACM Software Engineering Notes*, 26(5):196–206, Sep 2001.

[12] Lehoczky, J. and L.Sha. Performance of real time bus scheduling algorithms. *ACM performance evaluation review Special issue*, 4(1), May 1986.

[13] Lehoczky, J., L.Sha, and Strosnider, J. Enhancive aperiodic responsiveness in a hard real time environment. In *Proceedings of IEEE Real Time System Symposium*, 1987.

[14] Leung, J. and Merrill, M. A note on preemptive scheduling of periodic real time tasks. *information processing letters*, 11(3), Nov 1980.

[15] Mok, A. *Fundamental Design Problems of Distributed Systems for The Hard-Real-Time Environment*. PhD thesis, Laboratory for Computer Science, Massachusetts Institute of Technology, 1985. Available as Technical Report No. MIT/LCS/TR-297.

[16] Object Management Group. *OMG. OMG Unified Modeling Language Specification*, omg document- formal/03-03-01 edition, March 2003.

[17] Object Management Group, Massachusetts, USA. *UML Superstructure Specification - version 2.0*, 2004.

[18] Object Management Group. *OMG. UML 2.0 Superstructure Specification*, omg document - formal/05-07-04 edition, July 2005.

[19] Object Management Group. *OMG. UML Profile for Modeling and Analysis of Real-Time and Embedded systems*, omg document - realtime/05-02-06 edition, Feb 2005.

[20] Object Management Group. *OMG. UML Profile for Schedulability, Performance and Time Specification*, omg document -formal /05-01-02 edition, Jan 2005.

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

[22] Selic, B. and Rumbaugh, J. *Using UML for Modeling Complex Real-Time Systems*. Objec-Time Limited, 340 March Rd., Kamata, Astario, Canada, 1998.

[23] Sha, L., Rajkumar, R., and Lehoczky, J. P. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Transactions on Computers*, 39(9):1175–1185, 1990.

[24] Valiente, M. C., Genova, G., and Carretero, J. Uml 2.0 notation for modeling real time task scheduling. *Journal of Object Technology*, 5(4), May-June 2006.