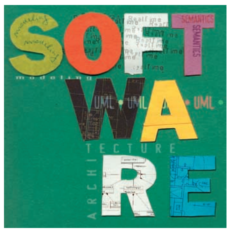
COVER FEATURE

Using UML-Based Rate Monotonic Analysis to Predict Schedulability



The OMG's recent adoption of the Unified Modeling Language profile for schedulability, performance, and timeliness has increased interest in using object-oriented technology to model and implement real-time systems. The authors present guidelines for applying rate monotonic analysis to ensure predictability in these systems.

Hossein Saiedian
The University of Kansas

Srikrishnan Raguraman
Kansas State University

Timeliness is essential in real-time systems, in which a late response is sometimes worse than no response at all because the violation of a single deadline could lead to loss of life or property. However, as Bran Selic pointed out in 1999,[1] timeliness is only one aspect of the complexity in these systems—their interaction with the physical world presents a "fundamental complexity that cannot be eliminated."

To create a predictable system—one in which the timing behavior always falls within an acceptable range—designers must know the period, deadline, and worst-case execution time of each task. System analysts use an appropriate scheduling algorithm to ensure the predictablilty of such a system.

The Object Management Group's adoption of the UML profile for schedulability, performance, and timeliness (www.omg.org/technology/documents/formal/schedulability.htm) has increased interest in using UML and object-oriented technology to model and implement real-time systems. Rate monotonic analysis (RMA) is an extensively researched and successfully implemented technique that can be used in conjunction with the UML profile to analyze schedulability in these systems.

PREDICTING SCHEDULABILITY

Research has shown that RMA solves many of the issues in real-time systems, including predictable real-time industrial computing systems.[2]

The major factors that affect real-time system schedulability include

- the event occurrence pattern—whether the event is periodic or aperiodic;
- each event's deadline—whether the deadline is less than or greater than its period; and
- task interaction—whether tasks need to synchronize with each other or not.

Scheduling algorithms assign priorities to tasks triggered by the arrival of specific events, as the "Priorities in Scheduling Tasks" sidebar describes. These algorithms fall under two main categories. *Static priority* algorithms such as RMA and deadline monotonic analysis assign a fixed priority to each task (and the same priority to every execution of that task). *Dynamic priority* algorithms such as earliest-deadline-first and value-added scheduling assign priorities based on the importance of each scheduling job, which is continuously reexamined in the dynamic context of the scheduler.

The original RMA algorithm, developed in the 1970s by Chung L. Liu and James W. Layland, simply assigned priorities to tasks in the decreasing order of their monotonic rate. The task with the highest monotonic rate was assigned the highest priority, and the priorities for the remaining tasks were assigned in decreasing order.

However, three assumptions in this algorithm made using it in real-time systems impractical. The algorithm assumed that all tasks were independent of one another (noninteracting), that all events were periodic with their deadlines at the end of their period, and that all tasks were perfectly preemptible. Subsequent research improved the algorithm by removing these assumptions, making it suitable for schedulability analysis.[3]

Periodic tasks without interaction

The schedulability of periodic tasks that do not interact with each other depends on their computation time, or *processor utilization*.

Deadline less than or equal to its period. A set of n independent, periodic tasks t_1, t_2, ... t_n, whose deadlines are equal to their period, are schedulable if and only if their combined processor utilization is less than $n(2^{1/n} - 1)$. That is,

$$C_1/T_1 + C_2/T_2 + \ldots + C_n/T_n \leq U_n = n(2^{1/n} - 1),$$

where C_i is the ith task's computation time, T_i is the ith task's period, and U_n is the utilization bound for n tasks. If $n = 2$, then the utilization bound is $2(2^{1/2} - 1)$, or 0.828. For n tasks, the utilization bound degenerates to 0.69, or 69 percent. Consequently, any periodic task set is schedulable by RMA if the tasks' combined utilization is less than 69 percent.

Deadline greater than its period. A basic problem in many real-time systems is allowing a periodically initiated response to complete after the next initiation. To analyze schedulability for this situation, it is necessary to first order the tasks by the increasing monotonic order of their deadlines. It is then possible to compute the worst-case response time for each event by applying a completion time test. If every worst-case response time is less than its deadline, the tasks are schedulable.

Periodic tasks with interaction

In most real-time systems, the tasks are not independent—they synchronize with one another in a mutually exclusive manner to share a resource. For example, consider the following scenario described by Mark H. Klein, John P. Lehoczky, and Ragunathan Rajkumar.[2]

Suppose periodic tasks t_1, t_2, ..., t_n are arranged in descending order of priority, and t_1 and t_n share a common resource guarded by a semaphore S. Task t_n begins execution and enters a critical section using the resource. Task t_1 is initiated next, preempts t_n, and begins execution. During its execution, t_1 attempts to use the shared resource and is blocked on S. Task t_n continues execution, but before it completes its critical section it is preempted by the arrival of one of the tasks, t_2, t_3, ..., $t_n - 1$. Because none of them uses S, any of these tasks can execute to completion before t_n. This creates *priority inversion*, in which a lower-priority task blocks t_1 for a potentially unbounded period of time.

The solution to this problem is to use *priority inheritance*, in which a lower-priority task inherits the blocked tasks' maximum priority for the duration of the blocking period. The Priority Ceiling Protocol is an implementation of priority inheritance with additional measures to prevent deadlocks and reduce priority inversion to at most one critical section of lower-priority tasks.

To analyze schedulability for such a system using RMA, it is first necessary to determine

- the longest *blocking delay* for each of the higher-priority tasks, and
- an *allocation policy* for the resource—FIFO, highest lockers, priority inheritance, distrib-

Priorities in Scheduling Tasks

To understand how priorities determine schedulability, consider a real-time system with two tasks: τ_1 with period $T_1 = 50$ ms, and τ_2 with period $T_2 = 100$ ms; the tasks' deadlines are at the end of their periods. Task τ_1 has a computation time C_1 equal to 25 ms, and τ_2 has a computation time C_2 equal to 40 ms. Assuming that a single processor will execute both tasks, τ_1's utilization (C_1/T_1) is 50 percent and τ_2's utilization (C_2/T_2) is 40 percent. Because their combined utilization is less than 100 percent, the two tasks are schedulable.

As Figure A shows, the tasks' schedulability depends on their priority. Assigning τ_1 a lower priority would result in τ_1 missing a deadline, but assigning τ_1 a higher priority would result in both tasks executing without missing a deadline. Researchers have shown that reducing a task's period or computation time does not necessarily guarantee schedulability.

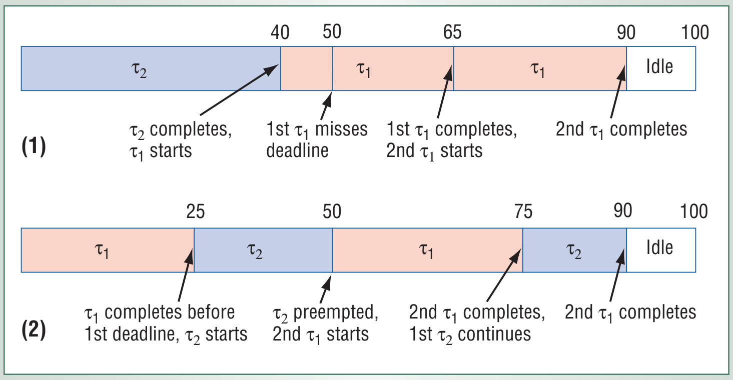

40
50
65
90
100
τ2
τ1
τ1
Idle
(1)
τ2 completes, τ1 starts
1st τ1 misses deadline
1st τ1 completes, 2nd τ1 starts
2nd τ1 completes
25
50
75
90
100
τ1
τ2
τ1
τ2
Idle
(2)
τ1 completes before 1st deadline, τ2 starts
τ2 preempted, 2nd τ1 starts
2nd τ1 completes, 1st τ2 continues
2nd τ1 completes

Figure A. Schedulability of two tasks, with (1) priority (τ_1) < priority (τ_2) and (2) priority (τ_1) > priority (τ_2). Periods: T_1 = 50 ms, T_2 = 100 ms; execution time: C_1 = 25 ms, C_2 = 40 ms.

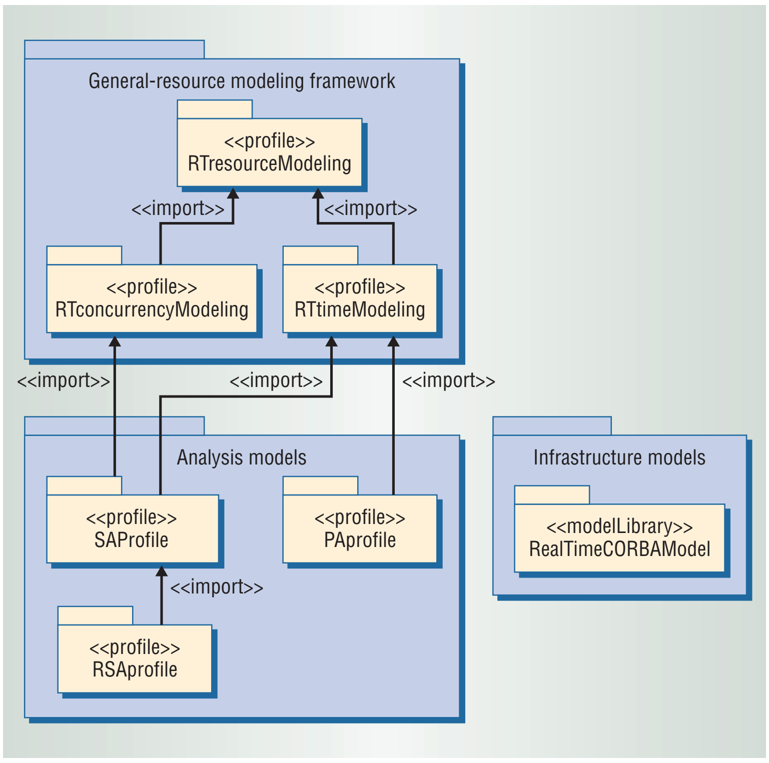


Figure 1. UML profile. The general-resource modeling framework acts as a base for other specialized subprofiles.

uted priority ceiling, and so on; priority inheritance solves all known problems with task synchronization.[2]

The next step is to calculate the worst-case response time for each task by including the blocking delay in the completion time test. If every worst-case response time is less than the deadline, the tasks are schedulable.

Aperiodic events

Some critical emergency situations in real-time systems involve aperiodic events, which can be

- *bounded*—have a known interarrival duration;
- *bursty*—have a specific density; or
- *unbounded*—occur in terms of some probability function.

Polling is a well-known and predictable technique for handling such events. However, an aperiodic event that occurs immediately after the polling task checks for events must wait for the entire polling cycle to get serviced.

The need for a periodic polling server and providing an on-demand response to aperiodic events led to the development of *sporadic servers,* which preserve execution capacity equal to the worst-case response time to service an aperiodic event.[3] The arrival of such an event triggers a high-priority interrupt that signals the sporadic server to run. The system schedules the aperiodic event depending upon the priority assigned to it by the sporadic server's designer. Thus, the system can treat aperiodic events as a set of periodic events with their period equal to their minimum interarrival time.

The aperiodic event's execution time should be equal to its worst-case execution time. If there is more than one aperiodic event, as in bursty or unbounded arrival patterns, the interrupt handler queues the events and signals the sporadic server. A limitation of such servers is excessive system overhead due to frequent polling for aperiodic events.

Real-time system predictability depends on both the software's structure and behavior and key attributes of the underlying infrastructure. These describe the quality of service (QoS) that the operating system, network, and other external resources offer. Because designers must specify the software at the highest level and as a structure of interacting dynamic components, object-oriented design is useful for modeling real-time systems.[4]

UML PROFILE

The general-purpose Unified Modeling Language has a broad enough base to capture all the intricacies of real-time systems. The UML profile for schedulability, performance, and time stereotypes core UML elements and adds semantics that supplement, but do not violate, its general semantics.

As Figure 1 shows, the profile is partitioned into hierarchically organized subprofiles dedicated to specific aspects of analysis. At the core of the profile is a general-resource framework that acts as a common base for other specialized subprofiles. This framework consists of three packages:

- *RTresourceModeling* contains general concepts such as action, event, response, scenario, and execution engine. Each concept has a UML-equivalent notation and contains attributes to quantify offered and required QoS.
- *RTtimeModeling* contains a framework for representing time and related mechanisms. For schedulability analysis, the most frequently used elements are clocks and timers to send triggers at a specified interval.
- *RTconcurrencyModeling* contains extensions to incorporate concurrency in actions, responses, and shareable resources.

Within the UML profile, SAProfile describes a specialized framework to analyze real-time situations for schedulability that

- determines if the entire system is schedulable—that is, if it can meet all of the deadlines defined for individual scheduling jobs; and

- helps determine how to improve the system, such as suggesting ways to make an entity schedulable and to maximize system usage.

SAProfile is designed to work with most popular scheduling analysis techniques. It extends existing UML elements with appropriate stereotypes and uses tags to quantify key attributes of real-time concepts. Table 1 lists some frequently used stereotypes and their associated UML elements. Modelers can use SAProfile to perform rate monotonic schedulability analysis on real-time systems.

Table 1. SAProfile stereotypes commonly used to represent a real-time system.

Stereotype	Real-time concept	UML element(s)
<<SASituation>>	Real-time situation	Interaction diagram, object diagram
<<SATrigger>>	Event	Message
<<SAResponse>>	Response	Method
<<SAAction>>	Action	Method
<<SASchedulable>>	Task	Class
<<SAResource>>	Resource	Class
<<SAEngine>>	Processor	Class

USING THE UML PROFILE TO APPLY RMA

The UML profile aids in schedulability analysis using either static or dynamic scheduling techniques. Although using the profile can generate an overwhelming amount of detail, a one-to-one mapping between the real-time terminology and the corresponding UML element (stereotype or tag) helps manage this potential complexity.

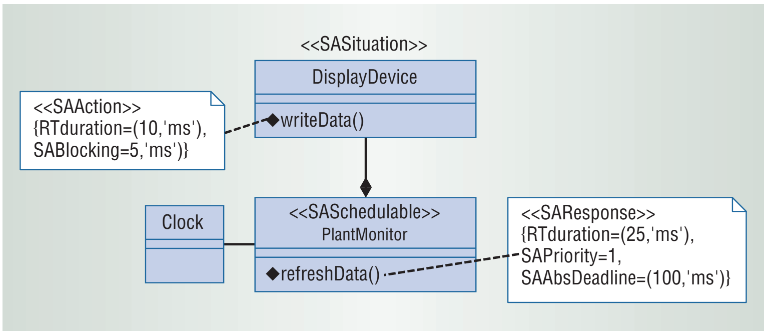
Real-time situations

System analysts generally decompose large real-time systems into smaller units that they analyze independently, and then perform a separate analysis of task interactions. A real-time situation encapsulates timing constraints for part of or, in some cases, an entire system. With SAProfile, analysts can stereotype an object diagram or interaction diagram as <<SASituation>> to use as a context for schedulability analysis.

Events. The stereotype <<SATrigger>> represents characteristics of incoming events, known as triggers in UML. Any message can be represented as an event. The stereotype's most important tag, SAOccurrence, defines event arrival patterns, which can be periodic, bounded, bursty, irregular, or unbounded.

Responses and actions. Every event is associated with a response, which is generally implemented as a method of a class and stereotyped as <<SA-Response>>. A response is a sequence of actions, and it defines the root of the action sequence. Each action is stereotyped as <<SAAction>>. Because a response is essentially an action, it inherits all the attributes of an action.

The SAPriority and SAAbsDeadline tags represent the priority and absolute deadline, respectively, of a response. The tag RTduration indicates the response's average computation time, which should be equal to the sum of the durations of each constituent action. The SAWorstcase tag represents the response's worst-case duration, which can be computed during schedulability analysis.



<<SASituation>>
DisplayDevice
<<SAAction>> {RTduration=(10,'ms'), SABlocking=5,'ms')}
writeData()
Clock
<<SASchedulable>>
PlantMonitor
refreshData()
<<SAResponse>> {RTduration=(25,'ms'), SAPriority=1, SAAbsDeadline=(100,'ms')}

Figure 2. Simple real-time situation including a task, a response, and an action.

A response can include actions that might interact with other responses by sharing resources in a mutually exclusive manner. This inevitably results in lower-priority tasks blocking higher-priority tasks, which severely impacts schedulability. The SABlocking tag quantifies the maximum wait time for a task to acquire resources. The SAUtilization tag represents a percentage of the response's utilization of resources.

Tasks. A task is a combination of an event and its associated response. It is generally represented as a class or as an object and can be stereotyped as <<SASchedulable>>. Figure 2 illustrates a simple real-time situation including a task, a response, and an action. The task and the execution engine have a dependency relationship that is stereotyped as <<GRMDeploys>>.

Resources. The UML profile defines a schedulable resource as "an active, protected resource that executes an action and may be shared by other concurrent actions." A resource generally is represented as an object or a class and can be stereotyped as <<SAResource>>. The tag SAAccessControl indicates the policy for controlling concurrent attempts to acquire the resource; its value is an enumeration of FIFO, PriorityInheritance, Highest-Lockers, NoPreemption, and DistributedPriority-Ceiling. A schedulable resource and the execution engine have a dependency relationship that is stereotyped as <<SAOwns>>.

Execution engine QoS. The processor that executes schedulable tasks plays a crucial role in schedula-

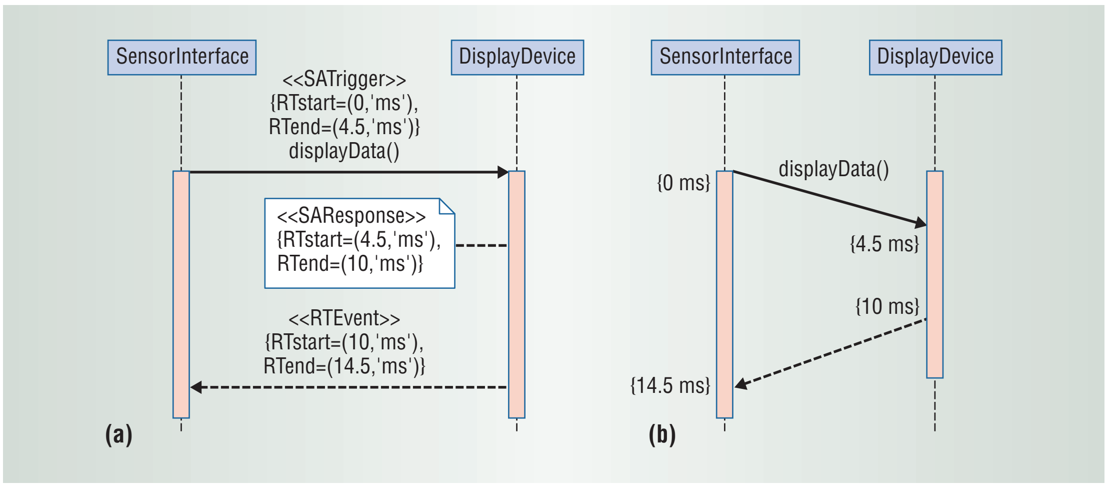


Figure 3. Two equivalent sequence diagrams showing different notions of time. (a) Timing details via stereotypes (as part of profiles), further enhanced by means of a UML "constraint" box. (b) Minimal timing detail, with a focus on active lifelines.

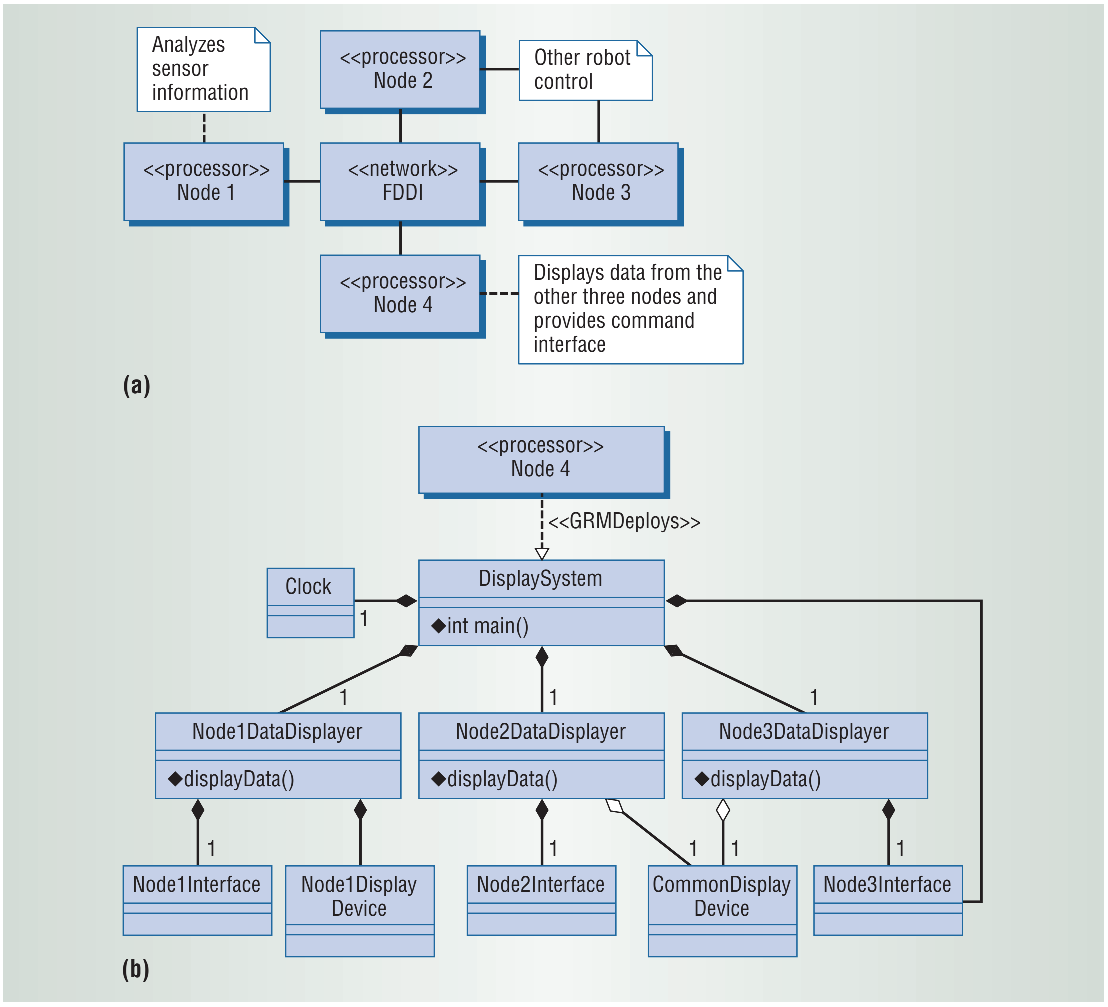


Figure 4. Real-time robotics application. (a) Deployment view shows the fiber-distributed data interface (FDDI) network and four processing nodes. (b) Node 4 display processor specification.

bility analysis. Key QoS attributes of the execution engine include context-switch time, scheduling policy, processor rate, and priority range. Most real-time situations have only one execution engine that is represented as a class or node and stereotyped as <<SAEngine>>.

Representing time in diagrams

When analyzing real-time situations, it is often important to clearly identify when critical events occur. For hard deadlines, an *interaction diagram* can model a trigger's start and end times and its response. The UML profile extends the sequence diagram to include "timing marks" to better visualize an event's start and end times. A *collaboration diagram* helps to visualize situations that involve a set of triggers and responses. These diagrams also can visualize concurrent behavior.

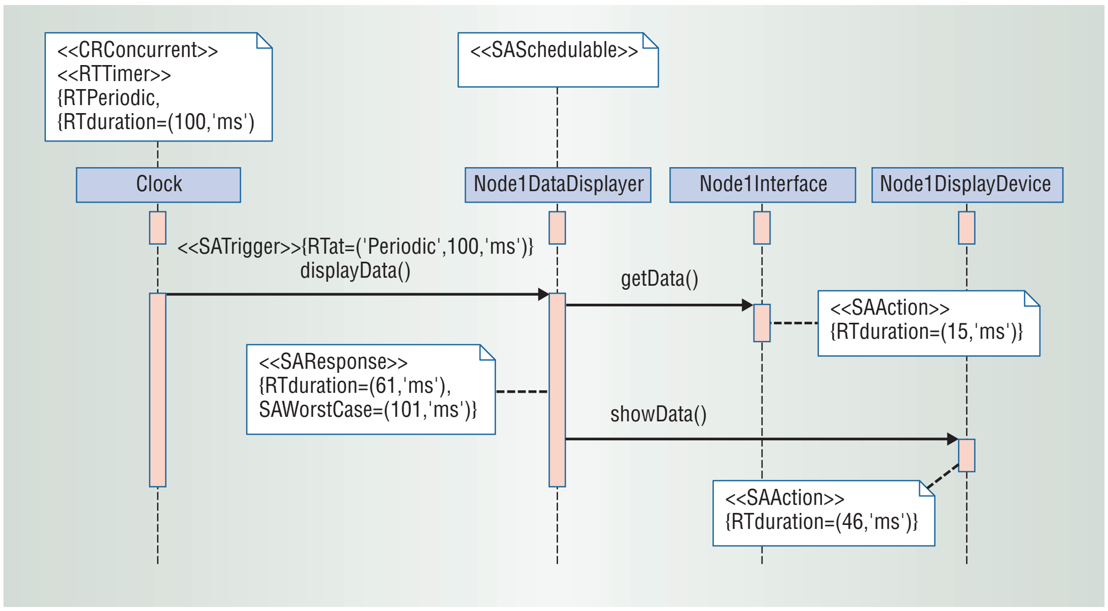


Figure 5. Sequence diagram for displaying node 1 data. <<RTClock>> sends a trigger to a Node1Displayer object at constant 100-ms intervals. Node1Displayer responds by displaying data from node 1, with a worst-case response time of 101 ms.

Table 2. Timing requirements for node 4 tasks.

Task	Time period (T_i)	Execution time (C_i)	Deadline (D_i)	Priority (P_i)	Blocking delay (ms)	Resource sharing
Display node 2 (τ_1)	80	20	80	High	5	Yes (with τ_3)
Display node 1 (τ_2)	100	61	200	Medium	5	No
Display node 3 (τ_3)	300	30	300	Low	None	Yes (with τ_1)

Figure 3 shows two equivalent sequence diagrams.

CASE STUDY

A case study in which a collaboration diagram represents a real-time situation demonstrates the use of UML real-time profile elements for schedulability analysis. The case study—originally described by Klein, Lehoczky, and Rajkumar[2]—involves an application in which a robot system uses a distance sensor that measures the shape of pipes by moving around. As the deployment view of the system in Figure 4a shows, the application has a fiber-distributed data interface network with four processing nodes: Nodes 1, 2, and 3 are dedicated to robotics processing, while node 4 has an operator's console for sending commands to the other three nodes and displaying their system data measurements.

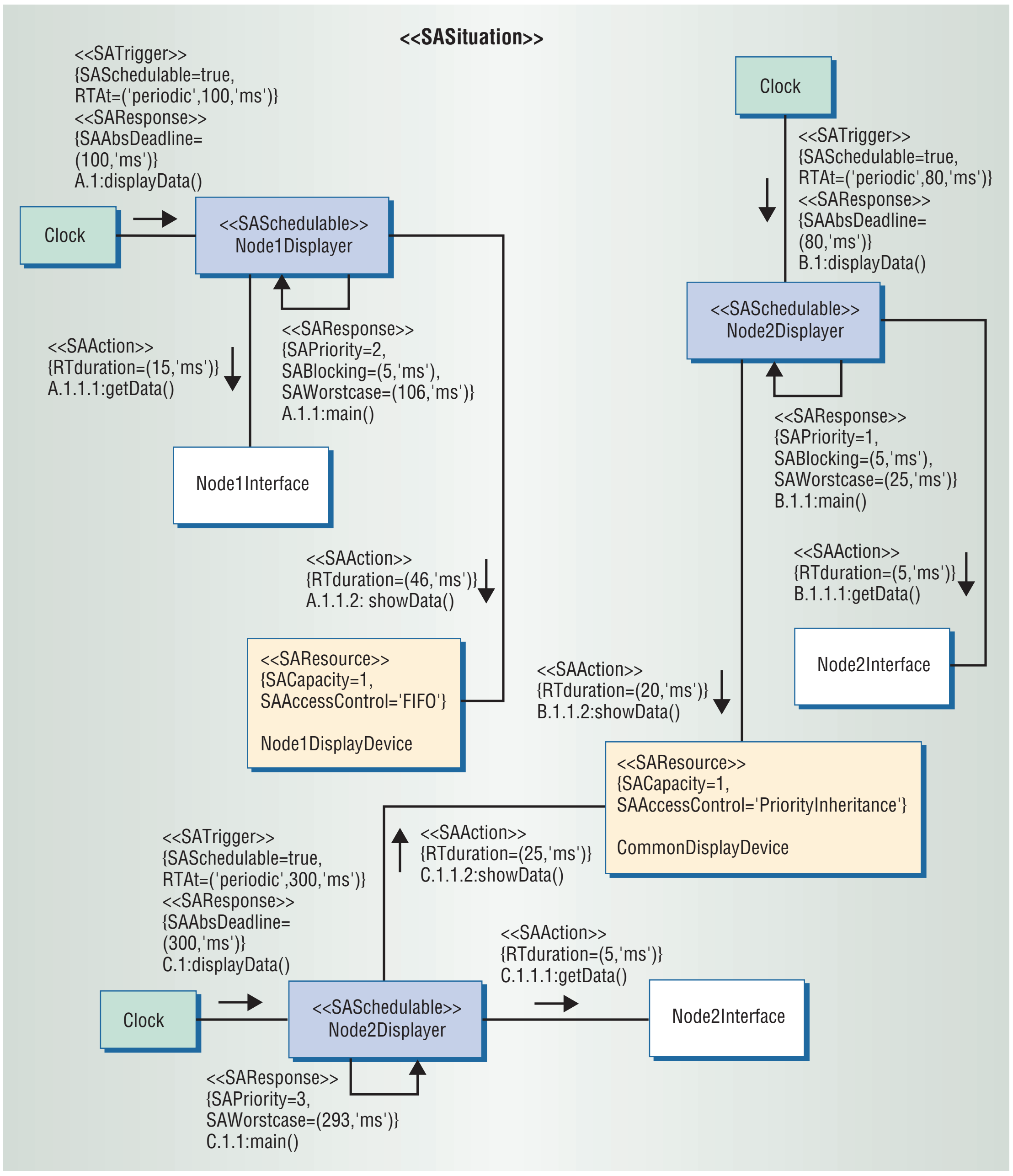
The system is required to meet each task's deadline on each node. For illustration purposes, this example considers schedulability only for node 4, which performs a data display function. The display system in node 4 contains three tasks—τ_1, τ_2, and τ_3—to display data sent from the other three nodes. Tasks τ_1 and τ_3 share a common display device, while τ_2 has a dedicated display device. Table 2 lists the timing requirements for the node 4 tasks. A scheduler assigns each job to its execution engine according to a scheduling policy.

The UML diagrams illustrate the suitability of the profile for performing rate monotonic schedulability analysis. Tasks τ_1, τ_2, and τ_3 are schedulable jobs that will be executed in node 4. A scheduler will assign each schedulable job to its execution engine depending on a scheduling policy. Figure 4b shows the UML structural representation for the display processor system in node 4.

DisplaySystem, the aggregate class that represents the system, runs on a single processor in node 4. It contains various classes that aid in displaying system data from each of the three nodes. Note that this class diagram is a descriptor diagram, not an instance-based diagram. In the class diagram, a realization relationship (stereotyped as <<GRMDeploys>>) exists between the node and the DisplaySystem, meaning that the DisplaySystem is deployed in the node 4 execution engine. There are many more such realization relationships at the instance level.

Each of the three classes—Node1Displayer, Node2Displayer, and Node3Displayer—is an active, protected resource that executes an action to display data. These classes correspond to tasks τ_1, τ_2, and τ_3 in Table 2.

Two real-time situations, each stereotyped as <<SASituation>>, provide a context for schedulability analysis: displaying data without task interaction and displaying data with task synchronization.



<<SASituation>>
<<SATrigger>> {SASchedulable=true, RTAt=('periodic',100,'ms')} <<SAResponse>> {SAAbsDeadline=(100,'ms')} A.1:displayData()
Clock
<<SASchedulable>> Node1Displayer
<<SAAction>> {RTduration=(15,'ms')} A.1.1.1:getData()
<<SAResponse>> {SAPriority=2, SABlocking=(5,'ms'), SAWorstcase=(106,'ms')} A.1.1:main()
Node1Interface
<<SAAction>> {RTduration=(46,'ms')} A.1.1.2: showData()
<<SAResource>> {SACapacity=1, SAAccessControl='FIFO'} Node1DisplayDevice
Clock
<<SATrigger>> {SASchedulable=true, RTAt=('periodic',80,'ms')} <<SAResponse>> {SAAbsDeadline=(80,'ms')} B.1:displayData()
<<SASchedulable>> Node2Displayer
<<SAResponse>> {SAPriority=1, SABlocking=(5,'ms'), SAWorstcase=(25,'ms')} B.1.1:main()
<<SAAction>> {RTduration=(5,'ms')} B.1.1.1:getData()
Node2Interface
<<SAAction>> {RTduration=(20,'ms')} B.1.1.2:showData()
<<SAResource>> {SACapacity=1, SAAccessControl='PriorityInheritance'} CommonDisplayDevice
<<SATrigger>> {SASchedulable=true, RTAt=('periodic',300,'ms')} <<SAResponse>> {SAAbsDeadline=(300,'ms')} C.1:displayData()
<<SAAction>> {RTduration=(25,'ms')} C.1.1.2:showData()
<<SAAction>> {RTduration=(5,'ms')} C.1.1.1:getData()
Clock
<<SASchedulable>> Node2Displayer
Node2Interface
<<SAResponse>> {SAPriority=3, SAWorstcase=(293,'ms')} C.1.1:main()

Figure 6. Collaboration diagram for task synchronization scenario, with blocking delays. Triggers from a concurrently executing clock periodically execute three schedulable resources. Two of the tasks also share the resource CommonDisplay-Device.

Displaying data without task interaction

Sequence diagrams are useful for providing a detailed description of triggers and responses. Figure 5 is a sequence diagram for displaying node 1 data, which corresponds to task τ_2 in Table 2. A clock, stereotyped as <<RTClock>>, sends a trigger to a Node1Displayer object at constant 100-ms intervals. Node1Displayer responds to the trigger by displaying data from node 1—<<SAResponse>> – Node1Displayer::displayData()—which has a worst-case response time of 101 ms. Similar sequence diagrams can be drawn for tasks τ_1 and τ_3.

As Table 2 indicates,

- Node2Display (τ_1) and Node3Display (τ_3) have their deadlines at the end of their period,
- Node1Display (τ_2) has its deadline after the end of its period, and
- the event arrival pattern is periodic (<<RTClock>>).

The first step in determining system schedulability is to calculate the utilization bound of tasks τ_1 and τ_3. According to RMA, the Node2Display and Node3Display tasks are schedulable if $\Sigma\ C_i/T_i \leq U^*$, where $1 \leq i \leq n$ and $U^* = n(2^{1/n} - 1)$. Calculating the summation, (20/80) + (30/300) = 0.35, which is $\leq U^* = 0.828$ ($n = 2$). Therefore, tasks τ_1 and τ_3 are schedulable.

Because the deadline for Node1Display task (τ_2) is greater than its period, determining whether it is schedulable requires applying a completion time test to calculate the worst-case response time.

Performing this calculation involves a four-step process. Letting $i = 2$,

Step 1: Compute the first approximation $S_0 = B_i + \Sigma C_j$, where $1 \leq j \leq 2$ and B_i is the blocking delay of the ith job $S_0 = 0 + 20 + 61 = 81$ ms.
Step 2: Set a counter k to 1
Step 3: Calculate the next approximation until $S_{n+1} = S_n$
$S_{n+1} = B_i + kC_i + \Sigma \text{ ceiling } (S_n/T_j)$, where $1 \leq j \leq i - 1$
$S_1 = 0 + 61 + 2(20) = 101$ ms
$S_2 = 0 + 61 + 2(20) = 101$ ms
Step 4: Determine the response time, $E_{i,k}$. If the response time is greater than the period, then increment k and go to step 3.
$E_{i,k} = S_n - T_i\,(k - 1)$
$E_{2,1} = 101 - 61(0) = 101$ ms

Because the worst-case response time for Node1-Displayer is less than the absolute deadline of 200 ms, task τ_2 is also schedulable.

Displaying data with task synchronization

If two or more tasks must be synchronized to share a single resource, Klein and colleagues[3] suggest determining a blocking delay for each task and applying a Priority Inheritance Protocol because of the potential for an unbounded priority inversion. As the blocking delay column in Table 2 indicates, the maximum blocking delay for tasks τ_1 and τ_2 is 5 ms. Although task τ_2 does not share any resource, it still incurs a 5-ms blocking delay due to potential priority inversion.

Collaboration diagrams help to visualize a set of triggers and responses scheduled on a single processor. In the diagram in Figure 6, triggers from a concurrently executing clock periodically execute three schedulable resources. Two of the tasks also share the CommonDisplayDevice resource. The diagram includes annotations for the blocking delays for the responses to the Node1Displayer and Node2Displayer tasks.

The analyst, optionally with the aid of a tool, can perform the completion time test again to determine whether the three tasks are schedulable in this situation. Blocking delays must be considered while performing the test. The completion time test reveals that the worst-case response time is 25 ms for task τ_1, and 106 ms for task τ_2. The worst-case response time for task τ_3, which does not incur blocking, is 293 ms. The worst-case time for each task is less than its deadline, thus all three tasks are schedulable.

Because real-time system designs share many commonalities with the object-oriented design paradigm, modelers have experienced a relatively smooth transition in the adoption of UML-based designs for real-time systems. Although no currently available commercial tools strictly comply with the UML profile, new applications such as Tri-Pacific Software's RapidRMA (www.tripac.com/html/prod-toc.html) aim to fill that gap. Computer-aided software engineering tools are invaluable for analyzing a real-time situation for its schedulability and suggesting improvements in maintaining a balanced system. However, even with CASE tools, real-time designers who use the UML profile for RMA must be prepared to contend with a great deal of detail. ■

References

1. B. Selic, "Turning Clockwise: Using UML in the Real-Time Domain," *Comm. ACM*, vol. 42, no. 10, 1999, pp. 46-53.
2. M.H. Klein, J.P. Lehoczky, and R. Rajkumar, "Rate-Monotonic Analysis for Real-Time Industrial Computing," *Computer*, Jan. 1994, pp. 24-33.
3. M. Klein et al., *A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems*, Kluwer Academic, 1993.
4. B.P. Douglass, *Real-Time UML: Developing Efficient Objects for Embedded Systems*, 2nd ed., Addison-Wesley, 1999.

Hossein Saiedian *is a professor and associate chair in the Department of Electrical Engineering and Computer Science, and a member of the Information and Telecommunication Technology Center at the University of Kansas. His research interests include software process improvement, object technology, and software architecture. Saiedian received a PhD in computer science from Kansas State University. He is a senior member of the IEEE, and a member of the IEEE Computer Society and the ACM. Contact him at saiedian@eecs.ku.edu.*

Srikrishnan Raguraman *is a software engineering consultant specializing in object-oriented design. He received a BE in computer science from Madras University, India, and is currently completing an MSE at Kansas State University. Raguraman is a member of the IEEE Computer Society. Contact him at sri@ksu.edu.*