Modeling and Schedulability Analysis of Hard Real-Time Distributed Systems based on Ada Components [1]

Julio L. Medina, J. Javier Gutiérrez, José M. Drake, Michael González Harbour

Avda. de los Castros s/n, 39005 Santander - Spain
{medinajl, gutierjj, drakej, mgh}@unican.es

Abstract. The paper proposes a methodology for modeling distributed real-time applications written in Ada 95 and its Annexes D and E. The real-time model obtained is analyzable with a set of tools that includes multiprocessor priority assignment and worst-case schedulability analysis for checking hard real-time requirements. This methodology models independently the platform (processors, communication networks, operating systems, or peripheral drivers), the logical components used (processing requirements, shared resources or remote components), and the real-time situations of the application itself (real-time transactions, workload or timing requirements). It automates the modeling of local and remote access to distributed services. The methodology is formulated with UML, and therefore the software logic design as well as its real-time model may be represented inside any UML CASE tool. The real-time model obtained is analyzable with a set of tools that includes multiprocessor priority assignment and worst-case schedulability analysis for checking hard real-time requirements.

1. Introduction

The Real-Time Systems Annex (D) of the Ada 95 standard [1] allows users to develop single-node applications with predictable response times. Furthermore, there are a few implementations of the Distributed Systems Annex (E), that support partitioning and allocation of Ada applications on distributed systems [2]. One of them is GLADE, which was initially developed by Pautet and Tardieu [3] and is currently included in the GNAT project, developed by Ada Core Technologies (ACT) [5]. GLADE is the first industrial-strength implementation of the distributed Ada 95 programming model, allowing parts of a single program to run concurrently on different machines and to communicate with each other. Moreover, the work in [3] proposes GLADE as a framework for developing object-oriented real-time distributed systems. Annexes D and E are mutually independent and, consequently, the distributed real-time systems environment is not directly supported in the Ada standard [4]. Our research group has been working on the integration of real-time and distribution in Ada 95. We have proposed a prioritization scheme for remote

[1] This work has been funded by the Comisión Interministerial de Ciencia y Tecnología of the Spanish Government under grants TIC99-1043-C03-03 and 1FD 1997-1799 (TAP)

procedure calls in distributed Ada real-time systems [5], and in [6] we focused on defining real-time capabilities for the Ada 95 Distributed Systems Annex in order to allow the development of this kind of applications in a simpler and potentially more efficient way than with other standards like real-time CORBA.

From another point of view, Ada has been conceived as an object-oriented language to facilitate the design of reusable modules in order to build programs that use previously developed components. To design component-based Ada real-time applications it is necessary to have strategies for modeling the real-time behavior of these components, and also tools for analyzing the schedulability of the entire application, to find out whether its timing requirements will be met or not. With the possibility of distribution in real-time applications development, we also need to address issues like the modeling of the communications, or the assignment of priorities to the tasks in the processors and to the messages in the communication networks. On these premises we can think about identifying and modeling basic Ada components that can be useful in the development of real-time distributed programs and for which the schedulability analysis tools can be applied.

This paper proposes a methodology for modeling and performing schedulability analysis of real-time distributed applications, built with basic Ada logical components. The methodology has been designed to facilitate the development of systems written in Ada 95 and using Annexes D and E. The main aspects of the methodology are the following.

- It is based on independently modeling: the platform (i.e., processors, communication networks, operating systems, peripheral drivers), the logical components used (i.e., processing requirements, shared resources, other components), and the real-time situations of the application itself (event sequences, real-time transactions, workload, timing requirements) that provide the elements for the analysis.
- It models the real-time behavior (timing, concurrency, synchronization, etc.) of the Ada logical entities in such a way that there is a complete parallelism between the structure of the code written and the real-time model obtained.
- It allows extracting a model of each high-level logical component, via the instantiation of a generic parameterized real-time model. When all the components are combined together and the associated generic parameters are defined, an analyzable model of the overall system is obtained.
- It automates the modeling of local or remote access to distributed services. If a procedure of a remote call interface is invoked from a component assigned to the same processor node the procedure is modeled as executed by the calling thread; but if the same procedure is invoked from a component assigned to a remote node, the corresponding communication model (with marshalling, transmission, dispatching, and unmarshalling of messages) is automatically included into the real-time situation model that is being analyzed.
- The modeling components as well as the software artifacts of the application are represented and described with UML.

The paper will be organized as follows. Section 2 presents the conceptual environment in which real-time analysis and modeling are considered. In Section 3, we describe the basic structure of the UML real-time view in which our models are hosted, the main abstractions, and the model of the Component class, which is the

element with the highest modeling power. Section 4 discusses and justifies the feasibility of the approach used for mapping the Ada structures into analyzable real-time models. Section 5 presents an example of the modeling and real-time analysis of a simple application. Finally, Section 6 gives our conclusions.

2 Real-Time Analysis Process

The real-time models are based on concepts and components defined in the Modeling and Analysis Suite for Real-Time Applications (MAST). This suite is still under development at the University of Cantabria [8][9] and its main goal is to provide an open-source set of tools that enable real-time systems designers to perform schedulability analysis for checking hard timing requirements, optimal priority assignment, slack calculations, etc.

Figure 1 shows a diagram of the toolset and the associated information. At present, MAST handles single-processor, multiprocessor, and distributed systems based on different fixed-priority scheduling strategies, including preemptive and non-preemptive scheduling, interrupt service routines, sporadic server scheduling, and periodic polling servers. The following tools are already available now (✓) or are under development (-):

✓ Holistic and Offset-based analysis
- Multiple event analysis
✓ Monoprocessor priority assignment
✓ Linear HOPA (Holistic) and Linear simulated annealing priority assignment
- Monoprocessor and Distributed simulation

The main goal of the methodology is to simplify the use of well-known schedulability analysis techniques during the object-oriented development of real-time systems with Ada. In this paper we describe only the main characteristics of the modeling technique; we do not deal with the analysis techniques themselves, the least pessimistic of them can be found in [10].

The proposed methodology extends the standard UML description of a system with a real-time model that defines an additional view containing:

- the computational capacity of the hardware and software resources that constitute the platform,
- the processing requirements and synchronization artifacts that are relevant for evaluating the timing behavior of the execution of the logical operations,
- and the real-time situations to be evaluated, which include the workload and the timing requirements to be met.

In the schedulability analysis process, the UML model is compiled to produce a new description based on MAST components. This description includes the information of the timing behavior and of all the interactions among the different components. It also includes the description of the implicit elements introduced by the semantics of the Ada language components that influence the timing behavior of the system. All these elements are specified by means of UML stereotypes. The generated MAST description is the common base on which the real-time analysis toolset may be

applied. Finally, the analysis results may be returned into the UML real-time view as a report for the designer. This process is illustrated in Figure 1.

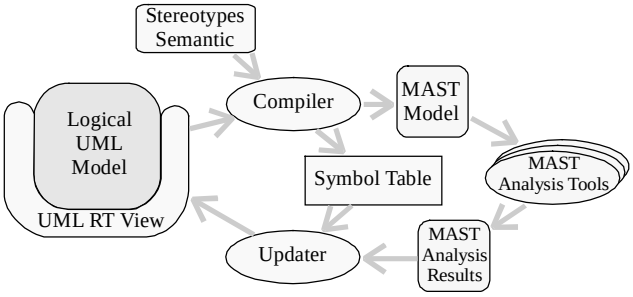

Fig. 1. Components of the real-time analysis process

3 Real-Time Model: UML RT View

The real time model is composed of three complementary sections:

The platform model: it models the hardware and software resources that constitute the platform in which the application is executed. It models the processors (i.e. the processing capacity, the scheduler, the system timers), the communication networks (i.e., the transmission capacity, the transmission mode, the message scheduler, the overheads of the drivers) and the platform configuration (i.e., the connections between processors using the different communication networks). Figure 2 shows the basic components of the platform model.

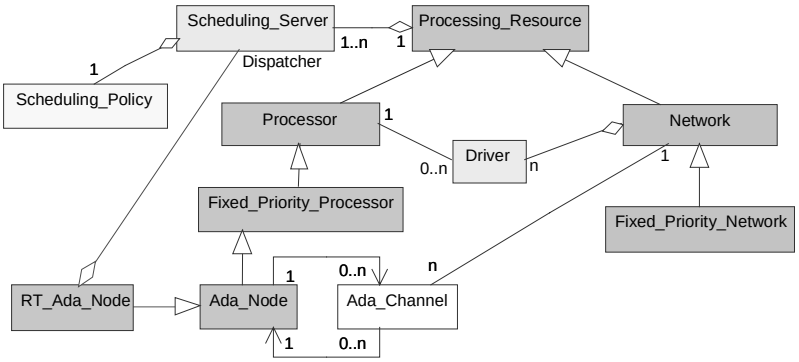

Fig. 2. Basic components of the platform model

The Processing_Resource is the root class of the platform model. It models the processing capacity of every hardware or software target that executes part or all of the modeled system activities. Each Processing_Resource has one or more Scheduling_Servers, in which the execution of the assigned activities is scheduled in

accordance with a specified scheduling policy. At the highest level, the Processing_Resources are specialized as Processors, which have the capacity to execute the application's code, and Networks, which transfer messages among processors. At the lower levels, there are more specialized resources defined, which have specific attributes that quantify the processing capacity or the transmission characteristics.

The logical components model: it describes the real time behavior of the logical Ada components that are used to build the application. A software component is any logical entity that is defined as a design or distribution module in an application. A component may model packages (with libraries or tagged type descriptions), tasks, the main procedure of the application, etc.

The model of a software component describes:

- The amount of processing capacity that is required for the execution of every operation in its interface.
- The identifiers of all other components from which it requires services.
- The tasks or threads that it creates for the execution of its concurrent activities.
- The synchronization mechanisms that its operations require.
- The explicit variations that its code introduces for the value of its scheduling parameters.
- The inner states that are relevant for describing the real-time behavior of its operations.

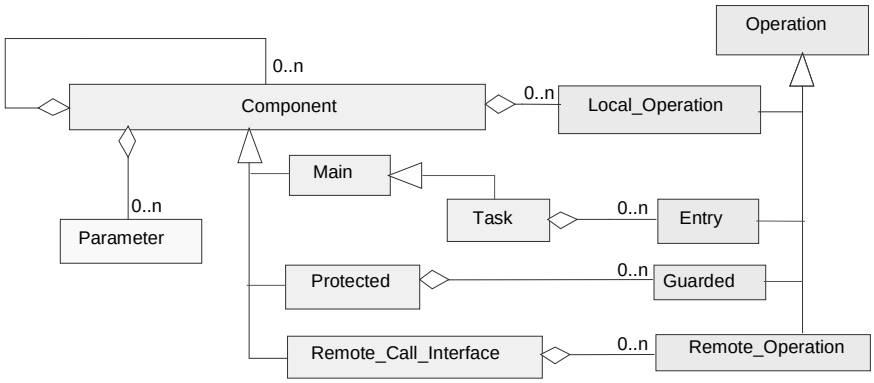

Fig. 3. The software component classes

The Component class (Figure 3) is the root class of the logical model hierarchy, It supplies the timing behavior models of each operation defined in its interface. If the operation is simple, its model is a set of attributes which describe the amount of processing that its execution requires. If the operation is composite, its model describes the sequence of operations that are executed within it. A component is also a container for other components aggregated into it, which are declared as attributes with the <<obj>> stereotype. The aggregated components are instantiated whenever the container component is instantiated. A component may also have parameters, which represent unassigned entities (external used components, operations, external events, timing requirements, etc.) that are used as part of the component's description. Parameters are declared as attributes with the stereotype <<ref>>. When a component

is instantiated, a suitable concrete value must be assigned to the instance's parameters. The components and parameters declared as attributes in a component are visible and may be used in the interface, inside the operations' descriptions and inside its aggregated components.

The Component class is specialized in accordance with the kinds of operations that may be declared in its interface or with the semantics of its operations or parameters. The specialized class Main models the main procedure of the application or of any partition in case of a distributed system. Every Main instance has an implicit scheduling server that models its main thread. The Task class is a specialized kind of Main class that models an Ada task; it also has an implicit aggregated thread (inherited from Main). A Task class may offer one or more Entry type operations in its interface in order to synchronize the calling thread with the internal task thread. The Protected class models an Ada protected object. All of the operations of its interface (Local_Operations or Guarded operations) will be executed in mutual exclusion. For the execution of Guarded operations (i.e., the entries of an Ada protected object) it may be necessary for other concurrent threads to reach a specific state.

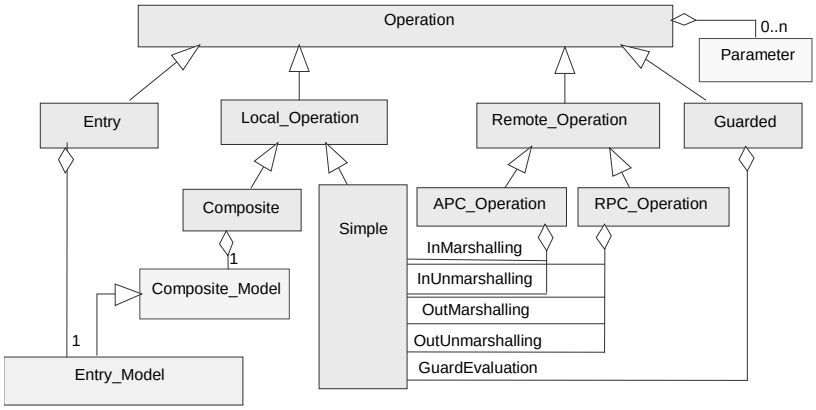

Fig. 4. Main operation classes for modeling the component interface

Every Guarded operation has an aggregated Simple Operation that models the evaluation of the guard. Finally, the Remote_Call_Interface class models a component whose operations may be invoked either from a local thread or from a thread in a remote partition assigned to another processor. Every Remote_Operation has a number of attributes that characterize the transmission parameters, and has several Simple operations aggregated, which model the marshalling and unmarshalling of the arguments of the operations. All of this information is used in the model only when the procedure is invoked remotely. The Operation class and its specialized classes model the timing behavior and the synchronization mechanisms of procedures and functions declared in a logical component's interface. Figure 4 shows the different kinds of specialized operation classes that have been defined.

Real-time situation model: it represents a certain operating mode of the system and models the workload that it has in that mode, which needs to be analyzed by the schedulability analysis tools. The analysis of a real-time system consists of the

identification of all the real-time situations in which the system can operate. The analysis of each situation is performed separately.

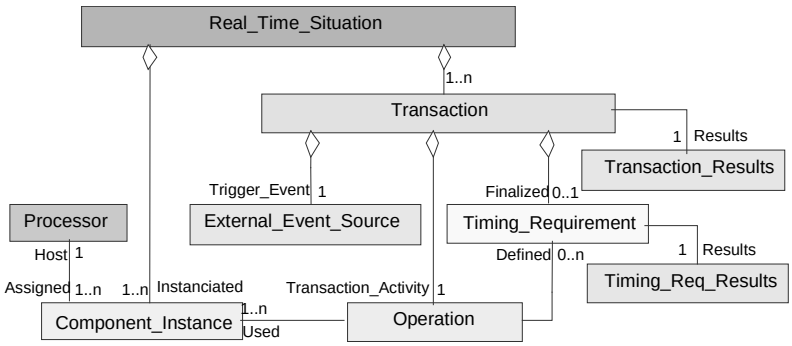

Fig. 5. Main classes of a real-time situation

Figure 5 shows the basic modeling components that constitute a real-time situation model. It is described by means of the declaration of the system configuration in that operating mode and of all the transactions that are expected to occur in it. The system configuration is described through the declaration of all the instances of the software components that participate in any of the RT_Situation activities and the declaration of its deployment on the platform.

The workload of the system in a real-time situation is modeled as a set of transactions. Each transaction describes the non-iterative sequence of activities that are triggered by the events generated by a specific external event source. Besides, these events serve as a reference for specifying the timing requirements of the RT_Situation. The nature of the stream of events is specified by an instance of any of the classes derived from the External_Event_Source abstract class. This class has been specialized into single, periodic, and aperiodic event sources. The aperiodic class specializes again into sporadic, unbounded, and bursty. The activity that is launched by the Trigger_Event is specified by the Transaction_Activity. It references an operation that is declared in the interface of an object instantiated from that activity. In each transaction we also describe the set of timing requirements that must be met. The one named Finalized is shown in the transaction declaration and stands for the final state that is reached when all the transaction activities are completed. The others are specified as arguments in the invocation of the Transaction_Activity operation; they are assigned to the operation parameters and in this way they are attached to the designated states, which were considered relevant from the timing point of view. Each timing requirement is modeled by an object instantiated from any of the classes derived from the Timing_Requirement abstract class. These classes are specialized according to the event that is taken as the reference for the calculus. They are called Global if they refer to the Trigger_Event of the transaction, and Local if they refer to the beginning of the last activity before the state to which the timing requirement is attached. They are also specialized according to other criteria like whether they represent a hard or soft deadline, whether they limit the amount of output jitter, etc.

Any object of a class derived from the classes RT_Situation, Transaction, Timing_Requirement or Scheduling_Server has an object linked, which is instantiated from the corresponding specialization of the Result class, which in turn holds the set of variables into which the results of the schedulability analysis tools can be stored.

4 Real-Time Model of the Basic Ada Structures

Even though the modeling and schedulability analysis methodology presented is language independent and is useful for modeling a wide range of real-time applications, the semantics of the high-level modeling components defined and the syntax and naming conventions proposed are particularly suitable and certainly adapted to represent systems conceived and coded in Ada.

The RT-model has the structure of the Ada application: The MAST Component instances model the real-time behavior of packages and tagged types, which are the basic structural elements of an Ada architecture:

- Each Component object describes the real-time model of all the procedures and functions included in a package or Ada class.
- Each Component object declares all other inner Component objects (package, protected object, task, etc.) that are relevant to model its real-time behavior. It also preserves in the model declarations the same visibility and scope of the original Ada structures.

A Component object only models the code that is included in the logical structure that it describes. It does not include the models of other packages or components on which it is dependent.

The RT Model includes the concurrency introduced by Ada tasks: The <<Task>> components model the Ada tasks. Each task component instance has an aggregated Scheduling_Server, which is associated with the processor where the component instance is allocated. Synchronization between tasks is only allowed inside the operations stereotyped as <<Entry>>. The model implicitly handles the overhead due to the context switching between tasks.

The RT model includes the contention in the access to protected objects: A protected MAST component models the real-time behavior of an Ada protected object. It implicitly models the mutual exclusion in the execution of the operations declared in its interface, the evaluation of the guarding conditions of its entries, the priority changes implied by the execution of its operations under the priority ceiling locking policy, and also the possible delay while waiting for the guard to become true. Even though the methodology that we propose is not able to model all the possible synchronization schemes that can be coded using protected entries with guarding conditions in Ada, it does allow to describe the usual synchronization patterns that are used in real-time applications. Therefore, protected object-based synchronization mechanisms like handling of hardware interrupts, periodic and asynchronous task activation, waiting for multiple events, or message queues, can be modeled in an accurate and quantitative way.

The RT model includes the real-time communication between Ada distributed partitions: The model supports in an implicit and automated way the local and

remote access to the APC (Asynchronous Procedure Call) and RPC (Remote Procedure Call) procedures of a Remote Call Interface (RCI), as described in Annex E of the Ada standard. The declaration of an RCI includes the necessary information for the marshalling of messages, their transmission through the network, their management by the local and remote dispatchers and the unmarshalling of messages to be able to be modeled and included automatically by the tools.

5 An Example: Teleoperated Machine Tool

The Teleoperated Machine Tool (TMT) example shows how to model and analyze a real-time distributed system with the presented methodology. The system platform is composed of two processors interconnected through a CAN bus. The first processor is a teleoperation station (Station); it hosts a GUI application, with which the operator manages the tool jobs. It also has a hardware "Emergency-Stop-Button". The second processor (Controller) is an embedded processor that implements the controller of the machine tool servos and of the associated instrumentation.

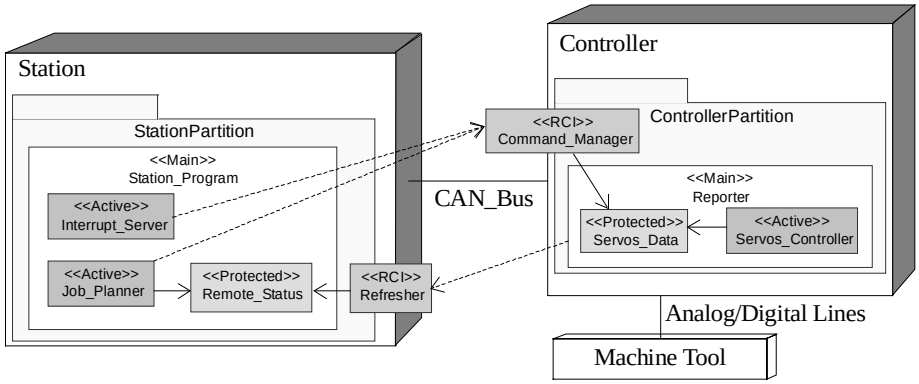

Fig. 6. TMT deployment diagram

5.1 Software Description: Logical Design

The software is organized in two Ada partitions, one for each processor. Each partition has its own Main program and offers a Remote Call Interface to the other. Each processor has a suitable implementation of the System.RPC package with guaranteed real-time features. The main components and the relations among them are shown in Figure 6.

The Controller partition's main program (The_Reporter:Reporter) acquires the status of the machine tool, with a period of 100 ms, and then notifies about it. Its active object, Servos_Controller has a periodic task that is triggered by a periodic timer every 5 ms. The RCI Command_Manager offers two procedures, one for processing the remote commands coming from the station and the other one to handle

the sporadic "Halt" command raised by the emergency stop button. All the Controller components communicate among them through the protected object Servos_Data.

The Station partition has a typical GUI application plus two active and one protected objects. Job_Planner is a periodic task that monitors and manages the jobs executing in the machine tool. The Interrupt_Server task handles the hardware interrupt generated by the emergency stop button. The RCI Refresher exports a procedure that updates the remote status data sent by the controller task. Remote_Status is a protected object that provides the embodied data to the active tasks in a safe way.

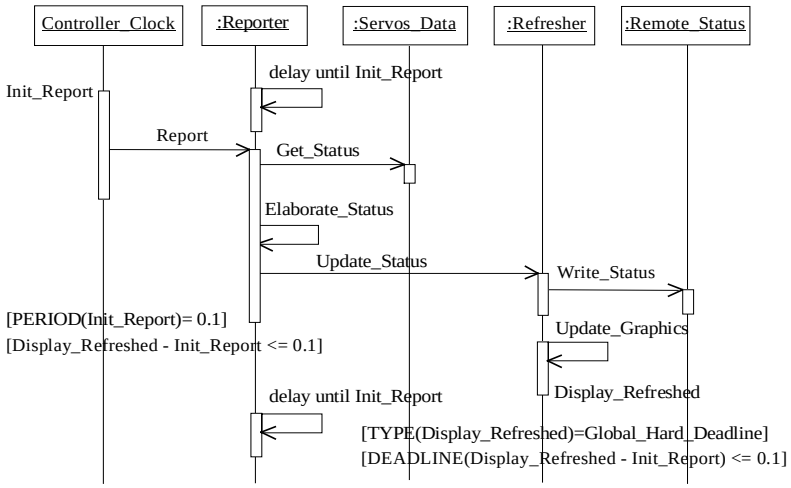

Fig. 7. Sequence diagram of the Report_Process transaction

In this example we model and analyze the normal operating state of the system, which is named as the Teleoperated_Control real-time situation. It contains four transactions with hard real-time requirements. They are all triggered by external or timed events:

- The Control_Servos_Process transaction executes the Control_Servos procedure with a period and a deadline of 5 ms.
- The Report_Process transaction transfers the hardware status data, in order to refresh the display and remote status data, with a period and deadline of 100 ms.
- The Drive_Job_Process transaction is activated each second, and its purpose is to analyze the job status and send the proper commands to the machine tool.
- The Do_Halt_Process transaction is activated by the emergency stop button. It is sporadic and has a minimum interarrival time of 5 s, and a deadline of 5 ms.

Figure 7 shows the functional description of the Report_Process transaction, using a sequence diagram.

5.2 UML Real-Time View of the TMT

The three sections of the UML real-time view will be described next.

5.2.1 TMT Platform Model

It describes the processing capacity of the three processing resources of the system: the Station and the Controller processors, and the CAN Bus network.

Figure 8 shows the platform model and a close up on the Controller partition and the CAN_Bus network. The processing capacity of the processor and all the attributes of its platform are modeled by the RT_Ada_Node MAST_Controller component. It is an embedded processor that executes the controller partition on top of a minimal real-time kernel. Controller has a periodic ticker timer, which is used for timing the periodic tasks allocated on it. Only the communication and system scheduling servers are explicitly declared. The priority and the scheduling policy assigned to the RCI partition dispatcher are specified by means of attribute The_policy (with an Interrupt_FP_Policy value assigned) and its argument The_priority (30).

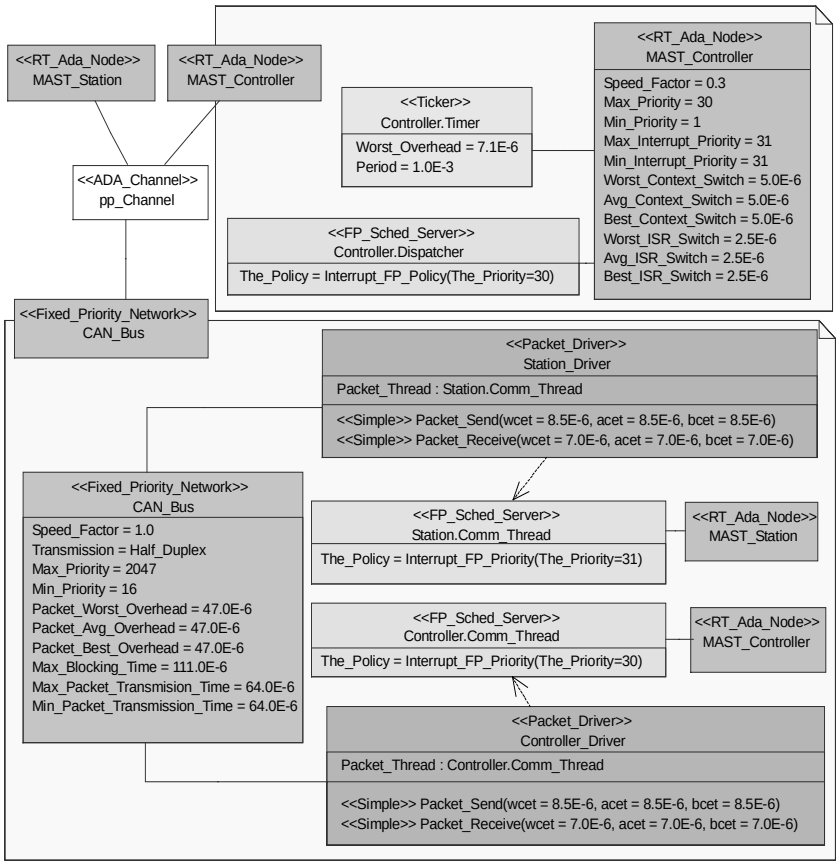

Fig. 8. Partial description of the TMT platform model

The specific values of the Controller processor parameters are:

Max_Priority= 30 Min_Priority= 1	Range of priorities allowed by the Ada compiler and MaRTE OS.
Max_Interrupt_Priority= 31 Min_Interrupt_Priority=31	Range of priorities allowed for hardware-interrupt handlers.

Worst_Context_Switch=5.0E-6 Avg_Context_Switch=5.0E-6 Best_Context_Switch=5.0E-6	Estimated context switch times between the threads of the application.
Worst_ISR_Switch= 2.5E-6 Avg_ISR_Switch= 2.5E-6 Best_ISR_Switch= 2.5 E-6	Estimated context switch times between a thread and an interrupt service routine and viceversa.
Speed_Factor= 0.3	The Controller processor has 30% of the processing capacity of the reference processor.

The CAN_Bus network is a packet-oriented half-duplex communication channel. Each packet has a frame header with 47 bits and a data field with 1 to 8 bytes. The bus transfer rate is lower than or equal to 1 Mbit/s. The packet transfer is prioritized, so a packet is never transferred if its priority is lower than the priority of any other packet that is waiting for being transferred. The real-time model of the CAN bus is described by means of the Fixed_Priority_Network component and the Packet_Drivers that run on the processors where the partitions that access the network are deployed.

The MAST CAN_Bus component has a Fixed_Priority_Network stereotype and it describes the transfer capacity of the channel. The attributes of this object that characterize its real-time behavior are:

Max_Priority= 2047 Min_Priority= 16	Priority range allowed for the messages on the CAN bus.
Packet_Worst_Overhead=47.0E-6 Packet_Avg_Overhead=47.0E-6 Packet_Best_Overhead=47.0E-6	This is the overhead due to the non-data bits of the standard CAN Bus Message Frame format. Non-data bits represent 47 bits per Data Frame (i.e., per packet).
Transmission= Half_Duplex	The transmission mode of the CAN bus is half-duplex.
Max_Blocking= 111.0E-6	The longest network's blocking time due to a packet transfer of maximum length.
Max_Packet_Transmission_Time= 6.4E-5 Min_Packet_Transmission_Time= 6.4E-5	Maximum and minimum times required for the transmission of the data bit field of a single packet frame (8 bytes/packet).
Speed_Factor= 1.0	The transmission time values refer to an implementation of the ISO 11898 standard for CAN operating at 1 Mbit/s.

5.2.2 Real-Time Model of the Logical Components

The RT Logical Model describes the timing behavior of all the logical modules that affect the real-time response of the system. As an example, Figure 9 shows the modeling elements that describe the temporal behavior of the MAST_Reporter class, which is the main program of the Controller partition. MAST_Reporter is an example of a <<Main>> component used both as a container and also as a periodic task. Its inner objects, The_Data and The_Controller, are declared as attributes of their corresponding types. Other attributes are the scheduling policy and the priority. For example the description of the composite operation Report is formulated in the aggregated activity diagram and follows the syntax proposed in [9]. The arguments of its simple operation Elaborate_Report, give value to the worst, average, and best case execution times expected for the operation. Arguments for the operations in the declaration of protected object The_Data (MAST_Servos_Data) and task The_Controller (MAST_Servos_Controller) are omitted for the sake of clarity. Notice that The_rc argument of Report is used in the invocation of the APC procedure Update_Status of the Refresher RCI, which in turn is defined inside MAST_Reporter as the Remote_Refresher referencing attribute.

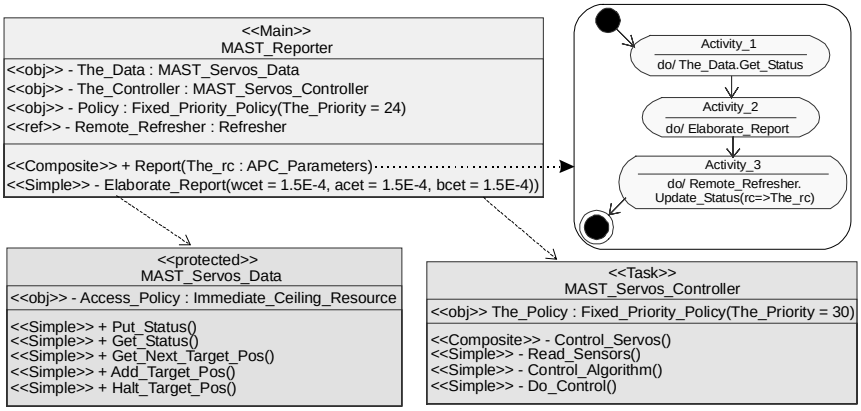

Fig. 9. TMT logical components model: MAST_Reporter description

5.2.3 Model of the Real-Time Situations

The real-time situation to analyze is formulated by means of the four mentioned transactions (Control_Servos_Process, Report_Process, Drive_Job_Process, and Do_Halt_Process). Figure 10 shows the model of the Report_Process transaction, whose functional behavior has been described by the sequence diagram in Figure 7. This real-time situation requires the instantiation of four components: The_Reporter, The_Command_Manager, The_Refresher, and The_Station_Program. In Figure 17 The_Reporter is an instance of the MAST_Reporter component and is hosted in the MAST_Controller processor.

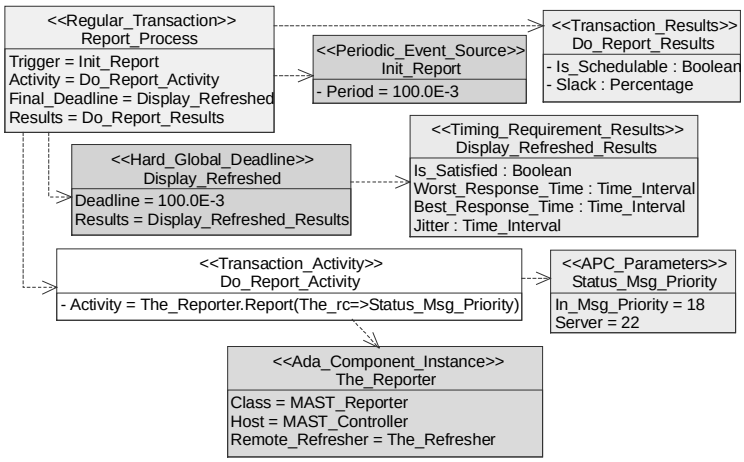

Fig. 10. Transaction Report_Process

The transaction is declared by means of a class diagram that is an instance of Regular_Transaction. Its Trigger attribute is set to the external event Init_Report, and the Final_Deadline attribute is set to the Display_Refreshed timing requirement. The

external event source Init_Report is an instance of the Periodic_Event_Source class and its attribute value indicates that it has a period of 100 ms. The timing requirement Display_Refreshed is an instance of the concrete class Hard_Global_Deadline and its deadline attribute is also 100 ms. This means that the state Display_Refreshed must be reached before 100 ms after the Trigger event (Init_Report). The transaction activity is the Report operation of the component object The_Reporter. The detailed sequence of activities that constitute the transaction is obtained by recursively invoking all the operation models declared in the logical model, with the concrete values of their parameters assigned. The actions in the activities invoke the operations and the swim lanes represent the scheduling servers (i.e., the threads) in which they execute.

5.3 Real-Time Analysis and Scheduling Design of the System

The TMT example has been analyzed using the MAST set of tools [8], which let us calculate the blocking times, ceilings for PCP_Resources, optimum priority assignment, transaction slack, and system slack. We have chosen the Offset-Based Analysis method, which is the least pessimistic[10]. Table 1 shows the most relevant results obtained from the schedulability analysis tools. In this table, we have compared the worst-case response times of each of the four triggering events of the RT situation with their associated deadlines. The priorities assigned to the tasks were calculated with the HOPA algorithm integrated in MAST.

Table 1. Analysis results for the four transactions in the real-time situation

Transaction/Event	Slack	Worst response	Deadline
Control_Servos_Process	19.53%		
End_Control_Servos		3.833ms	5 ms
Report_Process	254.69%		
Display_Refreshed		34.156ms	100 ms
Drive_Job_Process	28.13%		
Command_Programmed		177.528ms	1000 ms
Do_Halt_Process	25.00%		
Halted		4.553ms	5 ms

Although the overall schedulability is interesting information, it does not tell the designer whether the system is barely schedulable, or it has margin enough for changes. In order to get a better estimation of how close the system is from being schedulable (or not schedulable), the MAST toolset is capable of providing the transaction and system slacks. These are the percentages by which the execution times of the operations in a transaction can be increased yet keeping the system schedulable or decreased if necessary. Table 1 also shows the transaction slacks obtained for the example. We can see that the execution time of every activity of the Report_Process transaction can be increased by 254.69% and the system will still be schedulable. From a global point of view, the execution time of all the operations of the system could be increased by 2.34%, since that is the system slack calculated by MAST.

6 Conclusion

In this work we have presented a methodology for modeling the real-time behavior of Ada applications, with the appropriate level of detail to guarantee that the schedulability analysis, the optimum priority assignment, and the slack calculations using the generated model, can be applied.

Its main feature is that it allows modeling complex Ada components (packages, tagged types, protected objects, tasks, etc.), independently of the application in which they are used, which in turn may serve as the basis for the support of a design methodology for real-time systems based on Ada reusable components.

The modeling power of the proposed methodology is capable of covering most of the software patterns and programming building blocks that are widely used in the development of the majority of analyzable Ada real-time applications. Of course non real-time applications may use complex synchronization structures that cannot be modeled with the proposed approach, since they don't have a predictable timing behavior. But these structures are not commonly used in the real-time application environment.

The methodology presented in this paper is currently being implemented in the UML-MAST toolset. The description and implementation of this toolset can be found at: `http://mast.unican.es`

References

[1] S. Tucker Taft, and R.A. Duff (Eds.) "Ada 95 Reference Manual. Language and Standard Libraries". International Standard ISO/IEC 8652:1995(E), in LNCS 1246, Springer, 1997.

[2]L. Pautet and S. Tardieu, "Inside the Distributed Systems Annex", Intl. Conf. on Reliable Software Technologies, Ada-Europe'98, Uppsala, Sweden, in LNCS 1411, Springer, pp. 65-77, June 1998.

[3] L. Pautet and S. Tardieu: "GLADE: a Framework for Building Large Object-Oriented Real-Time Distributed Systems. Proc. of the 3rd IEEE". Intl. Symposium on Object-Oriented Real-Time Distributed Computing, (ISORC'00) Newport Beach, USA, March 2000.

[4]Luís Miguel Pinho, "Distributed and Real-Time: Session summary", 10th.Intl. Real-Time Ada Workshop, IRTAW'01, Ávila,Spain, in Ada Letters, Vol. XXI, Number 1, March 2001.

[5] Ada-Core Technologies, Ada 95 GNAT Pro, http:// www.gnat.com/

[6] J.J. Gutiérrez García, and M. González Harbour: "Prioritizing Remote Procedure Calls in Ada Distributed Systems", ACM Ada Letters, XIX, 2, pp. 67-72, June 1999.

[7] J.J. Gutiérrez García, and M. González Harbour: "Towards a Real-Time Distributed Systems Annex in Ada", ACM Ada Letters, XXI, 1, pp. 62-66, March 2001.

[8] M. González Harbour, J.J. Gutiérrez, J.C. Palencia and J.M. Drake: "MAST: Modeling and Analysis Suite for Real-Time Applications" Proceedings of the Euromicro Conference on Real-Time Systems, Delft, The Netherlands, June 2001.

[9] J.L. Medina, M. González Harbour, and J.M. Drake: "MAST Real-Time View: A Graphic UML Tool for Modeling Object-Oriented Real-Time Systems" RTSS'01, London, December, 2001

[10] J.C. Palencia, and M. González Harbour, "Exploiting Precedence Relations in the Schedulability Analysis of Distributed Real-Time Systems". Proceedings of the 20th IEEE Real-Time Systems Symposium, 1999.