Modelling real-time applications based on resource reservations

Laura Barros, César Cuevas, Patricia López Martínez, José María Drake, and Michael González Harbour
Grupo de Computadores y Tiempo Real
Universidad de Cantabria
39005, Santander, Spain
{barrosl, cuevasce, lopezpa, drakej, mgh}@unican.es

Abstract—**The new MAST 2 specification for modelling and analysis of real-time systems, introduces two new classes named *VirtualSchedulableResource* and *VirtualCommunication Channel*. They are used for modelling the schedulable entities in real-time applications that are designed and executed relying on a resource reservation paradigm. These modelling elements bring together into one single object the two different views that are used to describe a virtual resource through its life cycle. In the initial phase of the application design, a virtual view of the modelling element is used. It describes the fraction of the processor or network bandwidth needed to satisfy the timing requirements of the application. The second view is used afterwards, during the application deployment phase. It describes the real scheduling parameters with which the processor or network scheduler must execute the threads or communication channels that implement the virtual resources in the physical platform. The negotiation process between the application and the resource reservation middleware, which is carried out to make the designed application compatible with the current workload of the platform, can be seen as the pursuit of a configuration that makes both views compatible. This paper describes these new modelling elements and some scenarios where they are used.**

Keywords: Real-time; Resource Reservation; Modelling; Virtual Resources

I. INTRODUCTION

Resource Reservations is a paradigm that is widely applied to the design and execution of hard real-time applications. Its basic principle [1][2][3] in processing resources consists in executing each application thread in a server, which has an assigned fraction of the processor capacity. If during the execution, the thread activity tries to exceed its assigned capacity, the server suspends it temporarily to avoid it. Similarly, for communication resources each message stream is assigned to a communication server that represents a fraction of the communication network bandwidth. The use of this paradigm provides three main advantages:

- *System robustness*: If a system is designed to be schedulable according to its specification, and one or more of its applications exceed their design specifications by requiring more processing capacity than expected, these applications would not meet their timing requirements but the rest of the system would not be affected.
- *Design simplicity*: Using the resource reservation paradigm, the design of the scheduling of an application is accomplished in two stages. First, the application is scheduled independently, based on a virtual execution platform. Afterwards, the virtual platform implementation is negotiated on the physical execution platform. Among other advantages, this allows deploying an application with hard real-time requirements on an open platform whose workload is unknown by the application designer.
- *Reusability of real-time legacy modules*: A legacy subsystem or a reusable software component that implements real-time services can include as metadata the virtual execution platform that describes the resources required to satisfy its timing requirements.

MAST [4] is an open source set of tools that enables modelling and performing timing analysis of real-time applications. MAST can be used to design real-time applications, representing the real-time behavior and requirements together with the design information, and allowing an automatic schedulability analysis. MAST is basically constituted by a modelling methodology, close to that proposed by the OMG's MARTE profile [5][6], and a suite of tools including worst-case response time schedulability analysis, calculation of blocking times, sensitivity analysis through the calculation of slack times, and optimized priority assignment techniques. Version 2 of MAST [7] has been formulated recently and it extends the modelling methodology to advanced real-time paradigms, like resource-reservation and partition-based scheduling. The MAST 2 tools are still under development but, being both MAST 1 and MAST 2 versions based on formal UML metamodels, it is possible to perform simple transformations using MDA strategies and take advantage of the tools available in the current MAST suite for developing systems based on these new paradigms.

The aim of this paper is to describe how MAST 2 can be used to cover the different phases followed in the development and execution of applications based on resource reservations. Section 2 describes the MAST extension that deals with this paradigm, whereas Section 3 describes how the extension is applied to develop real-time applications. The information provided by the different modelling elements is described through a simple example in Section 4. Finally, Section 5 summarizes some current work lines and conclusions.

This work has been funded by the EU under contracts FP7/NoE/214373 (ArtistDesign); and by the Spanish Ministry of Science and Technology under grant TIN2008-06766-C03-03 (RT-MODEL).

II. Modelling Elements For The Resource Reservation Paradigm

The temporal behavior model of a system is conceived in MAST (and in MARTE) as the superposition of two models: the reactive model and the resources model, shown in Figure 1. The reactive model describes, by means of *EndToEndFlow* elements (as e2efA and e2efB), three aspects: the set of steps that, ordered by the control flow, conform the responses to events executed in the application, the generation patterns of the *WorkloadEvents* (coming from the environment or from the clock) which trigger these responses (such as triggerA and triggerB), and the *TimingRequirements* that must be satisfied by the execution of the responses (as tr1 and tr2). The resources model describes the usage of processing or communication resources and of mutual exclusion resources, by the steps belonging to the different *EndToEndFlows*. In the case of passive resources (as a Mutex), for the description of the resource it is enough to specify the synchronization protocol (priority ceiling, priority inheritance, stack resource). However, in the active resources (such as processors and communication networks), the model specifies its *Scheduler* together with its policy. In addition, it is necessary to specify the set of *SchedulableResources*, which are schedulable entities in a processing resource, each of them characterized by its scheduling parameters.

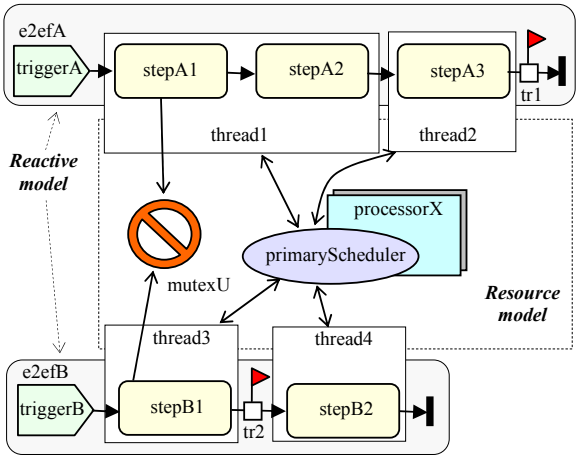

Figure 1: Root elements of the MAST models

The MAST 2 model extension that supports the resource reservation paradigm concerns only the *SchedulableResource* model elements. The new specialized elements decouple the reactive model from the resources model. This enables the design and analysis of the scheduling of an application using a virtual platform based upon resource reservation contracts, and without knowing the resources of the physical execution platform (Figure 2). Bearing in mind that in the development process of an application based on resource reservations the same instance of a *SchedulableResource* is going to be transformed from its virtual view to the real view, the virtual schedulable resource has been defined as inherited from the real schedulable resource, thus maintaining both views. The tool that processes the instance according to the stage of the design process, chooses the appropriate view.

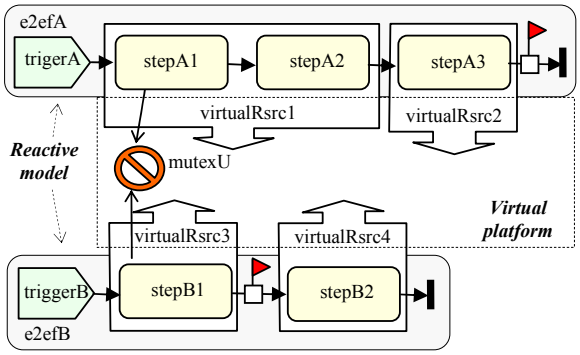

Figure 2: Application model in the virtual resource reservation view

The modelling elements defined in MAST 2 for representing virtual schedulable resources are *VirtualSchedulableResource* and *VirtualCommunicationChannel*. They extend the *Thread* and *Communication Channel* classes respectively, by adding a reference to the *VirtualResourceParams* or *VirtualComm ChannelParams* element that holds the information of the resource reservation contracts. Figure 3 shows the relations among the different modelling elements for the case of processing capacity reservation. For simplicity, the corresponding network elements are not shown.

It is important to notice that the inheritance relationship implies that a *VirtualSchedulableResource* is a *Thread*, so the tools that process it as a real schedulable resource will find within it the reference *SchedulingParameters* that characterizes it as a *Thread*. Likewise, the optional nature (0..1 multiplicity) of the *Scheduler* and *SchedulingParameters* references, makes it possible to create a *VirtualSchedulableResource* instance without any reference to the physical execution platform.

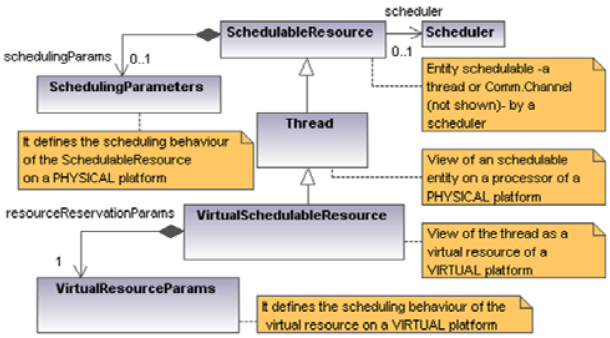

Figure 3: New classes in the resource reservation MAST model

The concrete classes that extend *ResourceReservationParams* define the behavior of the *VirtualSchedulableResource* (or of the *VirtualCommunication Channel*). This root class has been defined as abstract to include future types of virtual resources. Figure 4 shows the virtual resources currently defined, which follow the periodic replenishment strategy [11][12] for fixed priority scheduling: the periodic server [8], the deferrable server [9] and the sporadic server [10] models defined in the bibliography.

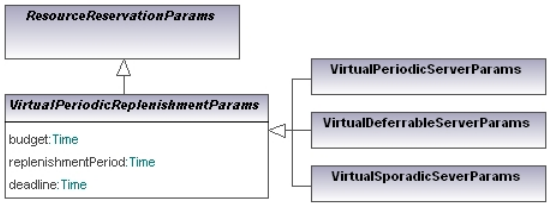

Figure 4: Resource reservation parameters classes in MAST 2

The attributes defining any type derived from *VirtualPeriodicReplenishmentParams* are the following:

- *Budget: Time* => Minimum execution capacity per server period.
- *Deadline: Time* => The server guarantees that a piece of work of size less than or equal to the budget and requested for a server with full capacity will be completed before the server's deadline.
- *Period: Time* => The period of the replenishment mechanism. The virtual resource will guarantee that every period, the part of the application running on it will get, if requested at the start of the period, at least the specified budget on the processing resource on which the associated schedulable resource is running.

The *VirtualCommChannelParams* class is similar, except for the *Budget* attribute, which is defined as the number of bits of transmission capacity.

The different server models vary in the granularity of the capacity replenishment and in how the platform responds when the application time is beyond the budget capacity. As an example, Figure 5 shows the worst-case response time of an activity of duration t_a in a virtual schedulable resource with *DeferrableServerParams* (Budget t_B, Deadline t_D and Period t_P). For this kind of virtual schedulable resource it is possible to evaluate the maximum response time t_x of an activity with duration t_a in the following manner:

$$If\ t_b \geq t_a \Rightarrow t_x < t_D - (t_B - t_a)$$

$$If\ t_b < t_a \Rightarrow \begin{cases} N_B = \left\lceil \frac{(t_a - t_b)}{t_B} \right\rceil \\ t_x < N_B t_R + t_D + t_a - t_b - N_B t_B \end{cases}$$

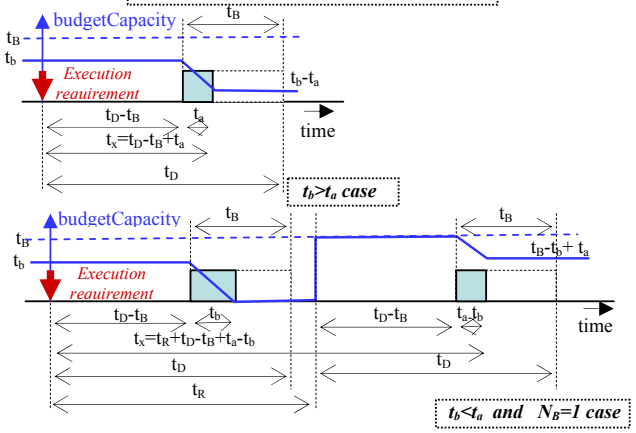

Figure 5: Worst case response time for the deferrable server

III. Supporting Real-Time Application Development

Figure 6 shows the four phases followed in the development of a real-time application based on the resource reservation paradigm: (1) The application is designed relying on a virtual platform. (2) The application is analysed to verify if it satisfies its timing requirements. (3) The instantiation of the virtual platform is negotiated with the execution platform. (4) The application is executed. The information and the models used in the process are also shown in the figure.

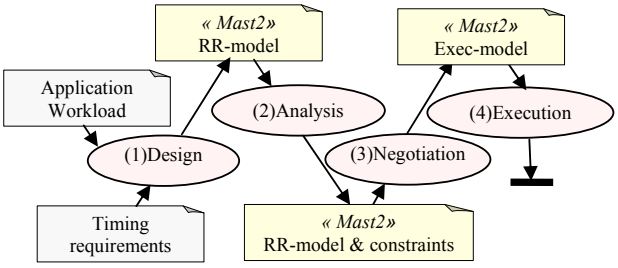

Figure 6: Phases in the design and execution of a real-time application

A simple example, called *ServoControl*, is used in this paper to illustrate the design process of a real-time system based on the resource reservation paradigm. It implements the controller of a servo engine, which is executed with a period of 10ms and a deadline of 5ms on a distributed platform composed of two processors, *Central* and *Remote*, interconnected by a CAN bus with a throughput of 1Mb/s. Figure 7 shows the reactive model of the application. It comprises only one *EndToEndFlow*, which involves seven *Step* elements, representing the activities executed in both processors and the transmission of messages through the bus.

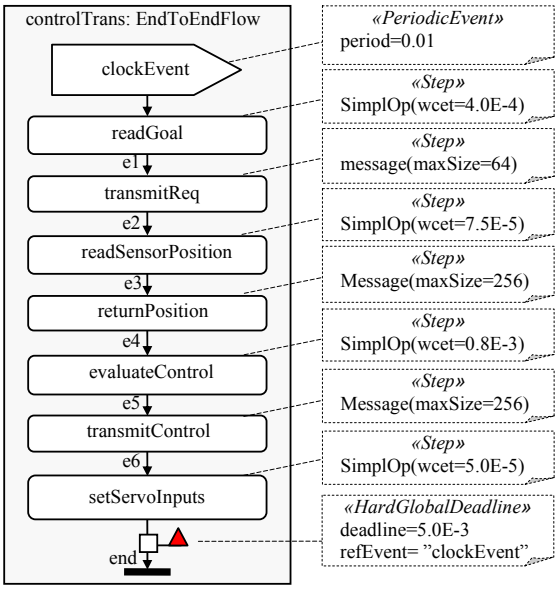

Figure 7: Reactive model of the ServoControl example

A. RT-Application specification and design.

The real-time application design consists in defining a virtual execution platform (i.e. a set of virtual resources), assigning to each step the virtual resource where it is executed, and according to this mapping, assigning values to the attributes of the virtual resources of the virtual platform. It is important to remark that this process can be done without knowing the physical platform where the application will be really executed.

Different design strategies may be applied to define the virtual platform. The simplest one consists in assigning an independent virtual resource per transaction and per processing node that take part in the application. In this case the replenishment period of each virtual resource is made equal to the minimum interarrival time of the transaction trigger, and the budget must be equal to the sum of the *wcet*s of the *Step*s executed in the same virtual resource. In this strategy the deadline attributes are kept indeterminate until the negotiation phase. The virtual platform that results from this design criterion for the *ServoControl* example is shown in Table I. The RR-Model (see Figure 6) that describes the application design is formulated as a MAST 2 model, which includes the reactive model (the "*controlTrans*" *EndToEndFlow* shown in Figure 7), and the virtual resources of Table I.

TABLE I. VIRTUAL PLATFORM AFTER DESIGN

VirtualResource	Assigned steps	R. Period	Budget	Deadline
VR_Central (vr1)	readGoal evaluateControl	10 ms	1.2 ms	?
VR_Bus (vr2)	transmitReq returnPosition transmitControl	10 ms	576 bits	?
VR_Remote (vr3)	readSensor setServos	10 ms	0.125 ms	?

When other design paradigms are used, the criteria to define the number of virtual resources and to assign steps to them may be different. Hence, if the *wcet* of the different steps executed in a processor has a large variability, or if some of its steps require access to a mutex with a high utilization, it can be advisable to use several virtual resources to schedule the steps of this processor. Likewise, if the application design is based on a component-based paradigm, the deployment of the components in the platform is unknown, so it is convenient to assign the virtual resources to each individual component in order to facilitate their later deployment. In this case, a virtual resource can be assigned for each invocation of the component services that is made in the different transactions. Hence, a strict ownership relation is kept between the component and its virtual resources.

B. RT-Application analysis

The analysis phase of a real-time application based on a resource reservation paradigm deals with establishing the relations among the virtual platform attributes that make the application satisfy its timing requirements. These requirements are formulated in the RR-model as *Timing_Requirement* modelling elements.

The design criterion described in the previous section guarantees that, when the execution of a step is required, there is enough budget to execute it without waiting for the next replenishment. Besides, its response time (t_x) can be delimited by the expression $t_x<t_D-(t_B\text{-wcet})$. In the case that the *EndToEndFlow* transaction is linear, for each timing requirement imposed on it, a restriction among the virtual resources attributes can be obtained. For example, in the *ServoControl* application, the restriction introduced by the timing requirement is the following:

$$vr1.t_D-(vr1.t_B\text{-readGoal.wcet})+ vr1.t_D-(vr1.t_B\text{-evaluateControl.wcet})+$$
$$+vr2.t_D-(vr2.t_B\text{-transmitReq.wcet})+ vr2.t_D-(vr2.t_B\text{-returnPosition.wcet})+$$
$$+vr2.t_D-(vr2.t_B\text{-transmitControl.wcet})+ vr3.t_D-(vr3.t_B\text{-readSensor.wcet})+$$
$$+vr3.t_D-(vr3.t_B\text{-setServos.wcet}) < t_{GD}$$

This expression leads to the following numerical solution:

$$2\ vr1.t_{D+}\ 3\ vr2.t_{D+}\ 2\ vr3.t_D \le 9.182\ ms$$

When the reactive model is more complex (e.g., when the control flow among the steps relies on fork, join, branch, or merge relations), the previous restrictions described as inequalities cannot be deduced directly, so the result of the analysis would be a set of concrete n-tuples of deadline attributes that make the application schedulable. These values can be obtained using the MAST 2 model and the MAST analysis tools.

The result of the analysis process of the real-time system is the RR-model&constraints model, which it is the result of adding to the previous RR-model the set of restrictions between the attributes of the virtual resources obtained from the analysis process. Again, it is important to notice that the analysis process is made without knowing neither the physical platform in which the application will be executed nor the other applications that will share the same physical platform.

C. RT-Application negotiation

The negotiation process is executed as a step previous to the deployment and execution of the application on the physical execution platform. It is an online process performed by the resource reservation service that must be provided by the execution platform. This negotiation requires knowing the RR-model&constraints of the application, the deployment plan, and the current workload of the physical platform. This process results in the assignment of values to the scheduling parameters of the threads and mutexes of the application that is being negotiated, and the reassessment of the threads and mutexes of the workload that was already executing, in order to adapt it to the new situation.

The deployment plan describes the assignment of the virtual resources of the application to the processing resources of the platform. Likewise, it includes the assignment of the kind of scheduling parameters (fixed-priority, EDF deadline, partition timetable, etc.) to the virtual resources, according to the scheduling policy applied to the scheduler of the processing resource assigned to them. The need to access this information reveals the importance of modelling both the virtual resource of the virtual platform and the schedulable resource of the final

resources model with a single element. Table II shows the resources model of the *ServoControl* application.

TABLE II. RESOURCE MODEL AFTER NEGOTIATION

VirtualResource/ Schedulable Rsrc	Assigned steps	R. Period/ priority	Budget/ scheduler	Deadline
VR_Central (vr1) /centrThr	readGoal evalControl	10 ms /14	1.2 ms /centrSch	2.51
VR_Bus (vr2) /commChannel	sendReq rtrnPosition sendControl	10 ms /146	576 bits /busSch	0.83
VR_Remote (vr3) /remiteThr	readSensors setServos	10 ms /22	0.125 ms /rmtSch	0.5

When the virtual platform established in the RR-model has all of its attributes assigned (i.e., in the case of complex reactive models), the negotiation process consists in building a timing behaviour model by gathering the application model and the current workload model, and executing a schedulability analysis. Different sets of values can be used until a schedulable solution is found.

However, when the result of the analysis phase is a set of restrictions among the virtual resources attributes (and not a set of concrete values), there is a higher negotiation capacity. In this case, the global model composed by the current workload of the platform plus the virtual platform model is partially specified, since the budgets and replenishment periods of the virtual resources have their definitive values assigned, but the deadlines do not have concrete values assigned yet. However, these deadlines must satisfy the set of restrictions obtained in the real-time analysis of the application. The negotiation task consists of looking for a set of values for these deadlines that not only satisfies the required restrictions, but also leads to a final schedulable workload including the prior application contracts together with the new one.

The online negotiation process requires a quick analysis of possible priority assignments and of the schedulability of the global model. The design strategy applied leads to global models composed of several, although very simple, elements. A typical model may be composed of hundreds of virtual resources, each of them with a periodic triggering pattern, with a period equal to the replenishment period, executing an activity whose *wcet* is equal to the budget, and with a deadline less than or equal to the period. If the platform relies on fixed-priority schedulers, the priority assignment criterion consists in assigning the maximum priority to the thread that implements the virtual resource with the minimum deadline, and using classical RMA for the shedulability analysis. The model is more complex if the activities of the virtual resources use mutual exclusion resources, although there are other known RMA techniques for these cases. In our case, a simplified version of the MAST analysis tool [4] is used. It makes it possible to analyse the system with hundreds of virtual resources in tenths of a second.

The negotiation process usually requires performing hundreds of schedulability analysis because the deadlines of the virtual platform have to be modified, keeping the imposed restrictions caused by the timing requirements of the application, until finding a schedulable global configuration.

In Table II, the results of the negotiation for a certain prior workload are shown. The values of the deadlines attributes are the output of this negotiation, and they are compatible with the set of restrictions imposed by the timing requirements of the application, and with the workload of the processing resources of the physical platform. Likewise, the scheduling parameters (priorities) of the threads of the physical platform with which the timing resources are implemented are also assigned.

D. Execution of the RT-Application

Once the virtual resources are instantiated in the physical execution platform (if the negotiation process has been successful), the execution of the application is launched using the resource reservation API of the middleware, binding the threads of the application with the existing virtual resources.

IV. CURRENT STATUS AND FUTURE LINES

The MAST 2 metamodel has been recently proposed. Therefore, updating the tools that form the complete modelling and analysis suite will take some time. Taking advantage of the fact that both MAST 2 and MAST 1 are defined as formal metamodels, MDA strategies have been used to develop the design and analysis tools that use the resource reservation paradigm. These tools process complete MAST 2 models, transforming them into other MAST 2 models that exclusively use modelling elements compatible with MAST 1, to whom the currently available MAST tools are applicable. Thus, the application model is unique and supports the whole resource reservation-based development cycle for real-time applications. However, in each stage, temporary models suitable to the current available MAST tools are generated by lightweight model-to-model (M2M) tools implemented with the ATL language.

As an example, we describe below the model transformation that leads to a model that allows the search of a solution by analysing the application using the MAST schedulability analysis tools. This approach is required when the application has a complex control flow and the timing analysis cannot be performed analytically according to the diagrams shown in Figure 5 and the equations derived from them.

The schedulability analysis is accomplished by transforming the MAST 2 model in such a way that each *VirtualSchedulable Resource* or *VirtualCommunication Channel* is replaced by a *Thread* or *CommunicationChannel* respectively, executed by an independent *ProcessingResource* where a given workload is also being executed. This load introduces contention that forces the worst-case response time in the activity scheduled by the virtual resource. Of course, this worst-case response time value must be compatible with the deadline and budget set in the virtual resource. Figure 8 shows the response time of an activity of *wcet* equal to the budget as a function of the period of the load activity. The *wcet* of the load activity is chosen as the maximum value that makes both activities (application and load) schedulable. We need to find a value of the load period that leads to a response time equal to the deadline of the virtual resource when the *wcet* of the activity is t_B. A load period of t_D (see Figure 8) is the only one

also causing a worst-case response time when the activity executed in the virtual resource has a duration $t_a<t_B$. This period is the one used in the model transformation. Figure 9 shows the modelling elements that form the analyzable model (compatible with MAST 1 tools) of the activity scheduled by the virtual schedulable resource *vrx*. The elements *vrxProc* and *vrxSch* are the processor and the scheduler where the *vrx* thread, which executes the activity of the virtual resource in the analysis model, is scheduled. *RxThLoad* is the higher-priority thread that executes the load activity modelled by the *vrxEtEF EndToEndFlow*.

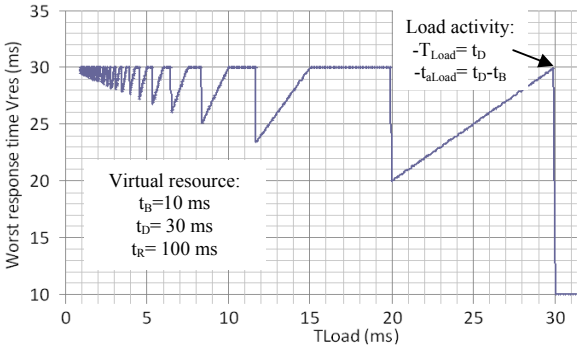

Figure 8: Selecting the load period for the virtual scheduling analysis

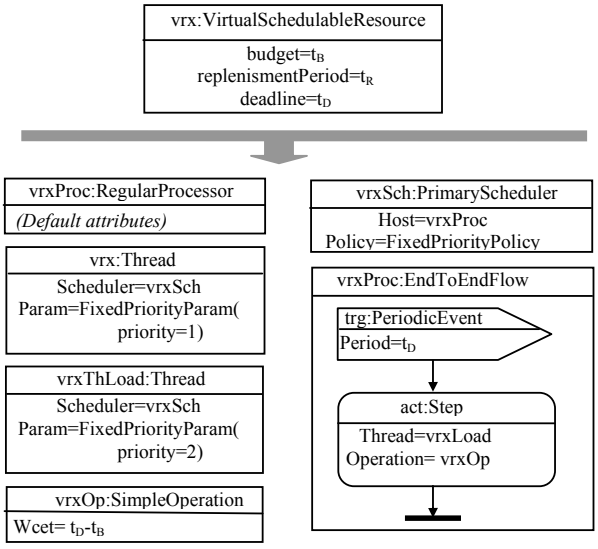

Figure 9: MAST 1 type model for the virtual scheduling analysis

Figure 10 shows the result of the schedulability analysis corresponding to the ServoControl example carried out in this work using the schedulability analysis tools currently available in MAST. Apart from verifying the application schedulability, it also assesses a set of worst-case response times that match those previously obtained from the equations in Section 3.

This paper demonstrates the capacity of the models conforming to MAST 2 to support in a uniform way the different phases of development of a real-time application based on a resource reservation paradigm. This contribution is a starting point towards updating the MAST tools in order to support the new advanced design paradigms for real-time systems covered by MAST 2.

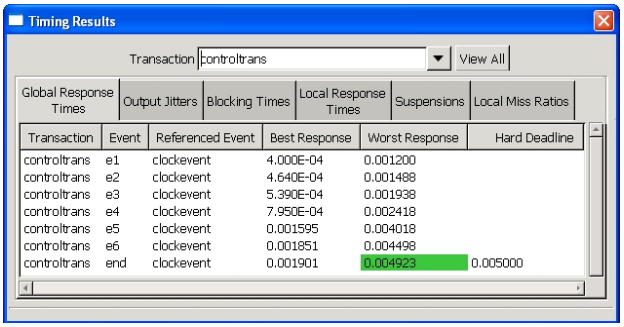

Figure 10: Results of the analysis of ServoControl using MAST 1 tools

The use of a formal UML metamodel to define MAST 2 allows it to cover in a unified manner the models corresponding to different design strategies for real-time systems. Likewise, MAST 2 represents a suitable environment to build a new generation of lightweight tools based on model transformation rules (instead of source code) that easily allow dealing with the new development paradigms for real-time systems recently appeared.

REFERENCES

[1] C. W. Mercer, S. Savage, and H. Tokuda, "Processor capacity reserves: an abstraction of managing processor usage", Proc. 4th Workshop on Workstation Operating Systems (WWOS-IV), 1993.

[2] C. W. Mercer, R. Rajkumar, and J. Zelenka, "Temporal protection in real-time operating systems", Proc. 11th IEEE Workshop on Real-Time Operating Systems and Software, 1994, pp. 79-83.

[3] R. Rajkumar, K.Juvva, A. Molano and S. Oikawa, "Resource kernels: A resource-centric approach to real-time and multimedia systems" Proc. SPIE/ACM Conf. on Multimedia Computing and Networking, 1998.

[4] M. González Harbour, J.J. Gutiérrez, J.C.Palencia and J.M.Drake, "MAST: Modeling and Analysis Suite for Real-Time Applications", Proc. 22nd. Euromicro Conf. Real-Time Systems (ECRTS 2001), 2001. MAST tool: http://mast.unican.es/

[5] Object Management Group. "UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE)" version 1.0, OMG doc. formal/2009-11-02, 2009.

[6] J.Medina and A. García Cuesta, "From composable design models to schedulability analysis with UML and UML profile for MARTE". Proc. of CRTS 2010. 3rd Workshop on pompositional Theory and Technology for Real-time Embedded Systems November 2010.

[7] C. Cuevas, J.M. Drake et all, "MAST 2 Metamodel" http://mast.unican.es/simmast/MAST_2_0_Metamodel.pdf.

[8] J.P.Lehoczky, L.Sha and J.K.Strosnider, "Enhanced aperiodic responsiveness in hard real-time enviroments",Proc. IEEE RTSS 1987.

[9] J. Strosnider, J. Lehoczky and L. Sha, "The Deferrable Server Algorithm for Enhanced Aperiodic Responsiveness in Hard Real-Time Environments", IEEE Transactions on Computers , 44 (1), January 1995.

[10] B. Sprunt, L. Sha, J. Lehoczky, "Aperiodic task scheduling for hard real-time systems", Journal of Real-Time Systems, vol 1, July 1989.

[11] S. Saewong, R. Rajkumary J.P. Lehoczky M.H. Klein:"Analysis of Hierarchical Fixed-Priority Scheduling" Proceedings of the 14 th Euromicro Conference on Real-Time Systems (ECRTS.02).

[12] R.I. Davis and A. Burns: "An Investigation into Server Parameter Selection for Hierarchical Fixed Priority Pre-emptive Systems" 16th International Conference on Real-Time and Network Systems (RTNS 2008