A tool for simulation and fast prototyping of embedded control systems

Luigi Palopoli Giuseppe Lipari
Luca Abeni Marco Di Natale
Paolo Ancilotti
ReTiS Lab, Scuola Superiore S. Anna
Via Carducci 40 - I-56127 Pisa, Italy
{palopoli,lipari,luca,marco,paolo}@sssup.it

Fabio Conticelli
ABB Corporate Research, Italy
fabio.conticelli@it.abb.com

ABSTRACT
This paper presents a set of C++ libraries, called RTSIM, aimed at realizing a joint simulation of a continuous plant and of a real-time embedded controller. The libraries permit a separate specification of the functional behaviour of the controller and of the software platform to be used for its deployment. In particular, it is possible to provide an accurate modeling of the concurrent architecture of the control tasks and of the run-time support offered by the operating system for the real-time scheduling of the shared resources (CPU, memory buffers, network links). In this way, its is possible to compare different scheduling solutions by evaluating their simulated performance in the domain of the control application. Moreover, the tool can be utilized to the tune up such design parameters as the activation frequencies of the tasks. The application of the tool is shown on a meaningful robotic case-study.

1. INTRODUCTION
Many computer based embedded control applications can intuitively and efficiently be implemented as a set of interacting parallel *tasks* operated by a multitask real-time operating system.

The performance of the controller is affected in many (sometimes subtle) ways by design choices related to the behaviour of control tasks with respect to time.

In fact, the execution times of the tasks, along with the physical limitations of sensors and actuators, impose upper boundaries on the maximum achievable sampling rates (for the sensors). Moreover, the scheduling of multiple control tasks sharing limited computational resources determines stochastic delays in their response times. Crucial design issues for controlling these factors are the selection of the appropriate task execution rates and the choice of (possibly) the best real-time scheduling policy for the shared resources (CPU, memory, communication links).

Classical approaches to control design often disregard these problems assuming a highly deterministic behaviour for the controller: the time interval between the acquisition of the sensor data and the release of the controller output is negligible or fixed. This choice is motivated by the difficulties encountered in defining an analytical model for stochastic response time variations deriving from data-dependencies and scheduling jitter. For the same reason, real-time computing theory has endeavoured to ensure time determinism on the implementation side. However, the notion of determinism used by real-time theory is the respect of deadlines which is generally difficult to relate to the performance of the control system.

As a consequence, an architectural design strictly complying with the deadline constraints can result into a poorly performing system. When these problems occur, the typical approach is to scale up the power of the hardware platform whereas a better tuned software design could often accomplish the same goals with low cost solutions. For instance in [19] we showed that, if the tasks execution times have a wide variance, the price of occasional overload situations can be outweighed by the "average" improvement of the controller behaviour achieved by using high sampling rates. This technique can pay off up to 50% performance improvement for certain classes of applications (e.g. robotic applications using multimedia sensor data), especially if soft real-time scheduling algorithms are used to control the system's timing behaviour in overload conditions. In the absence of well-settled analytical approaches, an integrated design of the control law and of the software architecture has to strongly rely on the use of simulation techniques.

However, the best known CAD tools are either focused on simulating the plant and the functional architecture of a controller or on performing simulations on the discrete events evolution of the computer system. The present work proposes a collection of C++ libraries, called RTSIM, aimed at bridging this gap. By using an intuitive object oriented framework, a separate specification can be provided for the functional model of the controller and for the software architecture. In this way, different architectural solutions can be tested and evaluated using performance indices belonging to the domain of the control application. The best known solutions for real-time CPU scheduling (either on single or on multiprocessor boards) and for bounded delay sharing of resources are available as predefined classes. Moreover, a well defined programming framework guides the user in developing his own scheduling solutions. The functional specification of the system is provided interconnecting a set of reusable components following a data-flow approach. The ensemble of functional and architectural model is directly implementable on a target real-time operating system (without the need of additional run-time support, such as virtual machines). Consequently the simulated performance turns out to be very close to the performance of the real system. For what concerns the simulation of the plant, RTSIM exploits the functionality of a powerful mathematical library, called OCTAVE [9], embodying the *state of the art* solutions for the integration of differential equations. The tool suite is completed by a set of classes defining stochastic parameters (*job*s durations, packet lengths etc.) with assigned probability distributions. Moreover, it is possible to record the events on a trace file and to collect statistics on the performance indices of interest over multiple runs.

The paper is organized as follows. In Section 2 the base modeling concepts are introduced. In Section 3 a brief description of the different components of the tool and of their interaction for performing the simulation is provided. In Section 4 the experiences made by applying the tool on a realistic robotic case study are reported. In Section 1.1 the related work is presented. In Section 5 we give our conclusions and present the future work.

1.1 State of the art

The best known tool suite for simulating real-time systems is MATLAB. Regarding the simulation of embedded control systems, MATLAB fits very well for modeling and simulating the continuous part of the plant and the functional structure of the controller. Moreover, it permits to automatically generate a prototype for a real-time controller (by the **Real-Time Workshop** tool). However, it is not easy to model generic Hardware/Software entities and there is little or no direct support for modeling and simulation of real-time scheduling algorithms. In [10], the authors propose a MATLAB toolbox to simulate a real-time scheduler in a SIMULINK block. This allows, to a given degree, the simulation of timing properties and the assessment of the performance of real-time controllers against changes in the timing attributes of tasks.

In recent years many interesting tools have been proposed for the analysis and simulation of complex systems with timing constraints. TimeWiz, from Timesys corp., is a commercial toolset aimed at the analysis and simulation of timing constraints with respect to schedulability. The toolset is being integrated with UML design frameworks which allows the description of complex systems in a fairly general way, but it does not allow (to our knowledge) to interactively evaluate the performance of control systems against changes in the task architecture and/or in the scheduling policies.

Many tools have been proposed for the formal verification of real-time systems modeled as "time automata". The best known proposals are KRONOS [7] and UPPALL [4]. An important evolution of the concept of timed automata are hybrid automata [11, 8]. However, the use formal techniques must face the explosion in the number of possible states to be analyzed when the complexity of the system and of the timing constraints grows.

The GIOTTO programming abstraction [30] has been devised to develop hybrid control applications consisting of periodic tasks. The language is primarily aimed at the design and prototyping of time-predictable control systems by using the conventional paradigm of separating the functional from the timing behavior (hard schedulability requirements). Time predictability (schedulability) is obtained by means of restricting the design to a time-triggered architecture (and scheduling) [14]).

The influence of the computing and network delays on the controller performance has been treated in several papers in the control literature [2, 24, 23, 17, 18].

An integrated design of real-time control systems encompassing performance and schedulability concerns was first proposed in [21]. In this work an optimization procedure for the activation frequencies of control threads is proposed; the goal is maximizing the controller performance under schedulability constraints. The paper is inspired to the evaluation approach for embedded controllers suggested in [25]. Other noteworthy results on this problem are presented in [13]; the authors first map the classical control design parameters onto the end-to-end requirements of the controller and then apply the method of period calibration to derive the execution parameters of each thread so that the end-to-end requirements are respected.

2. MODELING PRIMITIVES OF RTSIM

A real-time controller can be designed and analyzed considering the system's *functional behaviour* expressed in terms of the processed data and of the computations performed upon them, and the *HW/SW architecture* for the implementation platform.

Obviously, a given functional behaviour can be implemented by a set of possible architectures. For this reason, it is useful to define two distinct models of the system corresponding to the functional and to the architectural viewpoints. In fact, the separation of functional and architectural design permits to easily test different implementations for the same functional specification in order to identify the solution which best fits the performance/cost tradeoffs.

In this section, the most important modeling primitives of RTSIM for defining both the functional and the architectural specification are described.

Modeling the functional behaviour. For the RTSIM tool, the separation between the functional and architectural specification is aided by the use of a well structured data-flow approach for providing the functional model of the system. Data-flow models are growing more and more popular in the design of real-time software [28] and they are used in a variety of tools including Simulink, Ptolemy [15], Giotto [30]. The functional abstractions of RTSIM are essentially of two types: computing units and storage units.

Computing units are used to perform the computation while the storage units are used to exchange data between different computing units or between the controller and the external environment. A computing unit is endowed with a set of input ports and of output ports which must be connected to storage units. Each computing unit can respond to three different external commands. The first command, called **read** is used to acquire external data from the storage units connected with its input ports. The second one, called **execute**, computes an output value, while the third one, called **write**, is used to write the output into the storage units connected with the output ports. Notice that no particular model is required to specify the **execute** method. Thus, a computing unit can be a finite state machine, a digital filter, a proportional integral derivative (PID) controller, or whatever is needed in the controller's structure.

The storage units are of three types: input buffers, memory buffers or output buffers. Input buffers serve as an interface between the environment and the controller. From the point of view of the environment they can be thought of as sensors performing a measure on a quantity continuously varying in time. From the controller's side, an input port models an I/O card whose content changes when a sampling command is received. We remark that input buffers can only be connected to the input ports of a computing unit. Output buffers can be used to model actuators and can only be connected to the output ports of a computing unit. They model digital to analog converter i.e. when a computing unit writes a new data into an output buffer, the value is retained up to the next writing. Memory buffers can be accessed either for reading or for writing and realize the communications among different computing units. This is done by simply connecting a memory buffer to an output port of the computing unit acting as "writer" and to an input port of the computing unit acting as "reader". It is important to observe that the semantics of the storage units is very simple; they are essentially a way for establishing a communication between different computing units and between a computing unit and the plant. No particular assumption is made on such issues as the hardware implementation, the resource scheduling and so forth.

Modeling the architecture of the system.

The definition of a target architecture for a controller entails a set of decisions.

The application is first decomposed into a set of concurrent processes or *tasks*. A task is an infinite stream of instances (jobs) which can be activated either by a periodic timer or by external events. A job corresponds to the execution of a sequence of instructions realizing a functional behaviour. In our model, the **read**, **write** and **execute** commands of a computing unit can be associated to the instructions of a task. Thus, one or more computing units can be mapped onto a task. Then, tasks are assigned to the physical resources (nodes) of the system. On each node runs a real-time kernel, with a scheduling policy and a synchronization protocol. Finally, nodes communicate through some kind of network (Ethernet, CAN bus, etc.).

In real-time systems, tasks have time constraints, often expressed as deadlines: for example, a typical time constraint for a periodic tasks is that each job must finish before the next activation. In order to ensure the respect of such constraints, it is necessary to know in advance the requirements of each task on the computation resources (e.g. the processor time needed for executing a job, the maximum time a task locks a communication resource, the maximum length of the packets it delivers on a network link and so forth). Furthermore, time-predictable scheduling algorithms have to be used for the scheduling of the shared resources.

RTSIM allows to model and simulate all these aspects. In particular, the tool was originally designed for experimenting different real-time scheduling solutions. The resource requirements of the tasks can be specified either by deterministic or stochastic parameters. Moreover, the best known state of the art algorithms for scheduling CPU such as Cyclic executive, Fixed Priority,(RM) [16], Earliest Deadline First (EDF) [16], Proportional share [29] are provided as predefined objects. The performance of the schedulers can be enhanced by using aperiodic servers (Polling server [12], Sporadic Server [27], Constant Bandwidth Server [1], etc). It is also possible to choose the most appropriate protocol to handle shared resources as Priority Inheritance/Priority Ceiling [22], Stack Resource Policy [3]. Finally, a clear programming framework allows for an easy development and experimentation of novel solutions.

With the help of RTSIM, it is possible to easily vary the hardware/software architecture, assessing the performance of an application both in the system domain (response time of each single task) and in the application domain (performance index of the controller). The distribution of tasks to nodes, the network, the scheduling algorithm, the synchronization mechanism, the task periods: all these parameters can greatly influence the system behaviour. RTSIM allows to evaluate different solution in order to minimize the cost of the solution and maximize the application performance.

3. DESCRIPTION OF THE TOOL

RTSIM is a collection of C++ libraries which can be used inside user defined programs. The modeling technique described in the previous section suggested to decompose the software into three interacting packages, as shown in Fig. 1.

The package denoted as "Numerical Package" is used to simulate the plant. The second package, called "CTRLIB" is used to describe the functional model of the system. The "RTLIB" package is used to map the functional description onto a deployment architecture.

The way the three packages are used and interact one another will be described in detail in the next section on a practical example.

The Numerical Package. The main purpose of the numerical package is to provide programming models for continuous time plants. A plant is described by means of its state variables, differential equations and so on. The lan-

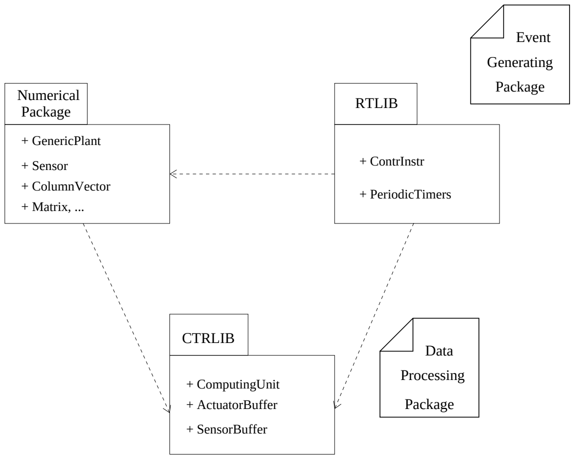

Figure 1: Main components involved in a RTSIM based simulation of a real-time controller

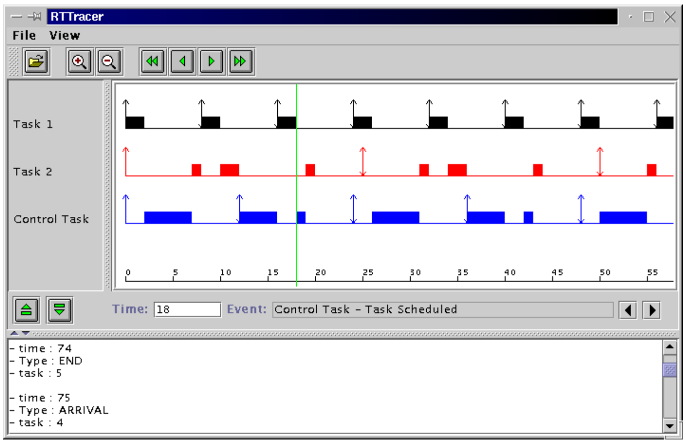

Figure 2: Graphical representation of the output of a RTSIM simulation.

guage mechanism used to build a plant model is inheritance from an abstract class defined in the library.

From a structural viewpoint, the numerical package is a software layer built on the top of a library which provides such services as differential equation integration and linear algebra. The current implementation is based on the OCTAVE library, which is a freely available tool embodying the best known algorithms for numerical computation. The presence of a software abstraction layer allows to replace OCTAVE with any other similar solution without affecting the structure of the simulator. An object implementing a plant is connected to a set of other objects implementing the sensors (input buffers) and the actuators (output buffers). The integration is simply performed by issuing a call on a method exported by the plant, for every interval separating two consecutive events.

CTRLIB. The functional model of the system is expressed using the classes of the CTRLIB package.

According to the adopted data-flow model, CTRLIB offers two types of components: computing units and storage units. Both of these components are framed within a hierarchy of classes. In order to specify a computing unit, the programmer has to derive it from an abstract class and has to provide an implementation for three pure virtual methods: **read**, **execute** and **write**. Once the class is defined, the programmer can instantiate objects from it to be used in different contexts. For example, a class implementing a PID controller is likely to be a reusable component.

The usage of storage units is slightly different according to whether they are input buffers, output buffers or memory buffers. Input buffers are abstract classes which play the role of sensors (from the viewpoint of the plant). Thus, if $\mathbf{x}$ is the state vector of the plant, an input buffer produces a vector output according to a function $\mathbf{h}(\mathbf{x}, t)$. The programmer is required to implement such a function by writing a virtual method, called **updateSensor**, which can read the plant state via a method call. A predefined method, called **sample**, can be used to sample the value of the sensor upon the occurrence of certain events (for example a timer expiration). The sampled value can be read by a computing unit calling the **getValue** method. Output and memory buffers can be used with no other efforts than defining the width of the data vector when an object is instantiated. Output buffers export the **putValue** method to the computing units and the **getCommand** method to the plant. Memory buffers export to the computing units both the **getValue** and the **putValue** methods.

The RTLIB Package.

RTLIB is a library designed to simulate the timing behaviour of a real-time software system. In this context, because of space constraints, we just report some of its basic features.

RTLIB is based on a discrete event model. A system, in RTLIB, is an interconnection of *entities*, implemented as objects, which receive and generate *events*. When an entity receives an *event*, an event handler method is invoked which updates the entity internal status, determines method calls on other entities and the generation of other events.

A real-time task is an entity characterized by four events: *activation*, *schedule*, *suspension*, and *end of the job*. By setting the task parameters appropriately, it is possible to simulate a time–driven (typically periodic) or event–driven execution model. A task is constructed as an aggregation of other entities called *pseudo–instructions*: a pseudo–instruction has a duration (which can be fixed or randomly chosen with an arbitrary probability distribution), and three events, schedule, suspension and termination.

Tasks are handled by kernels which can handle single or multiprocessors nodes. A kernel implements the scheduling policy, the communication and the synchronization protocols among tasks. Schedulers, resource managers, aperiodic servers are implemented as classes which can be plugged into a kernel or replaced by other components through the use of well defined programming interfaces.

Finally, kernels communicate through networks: a network

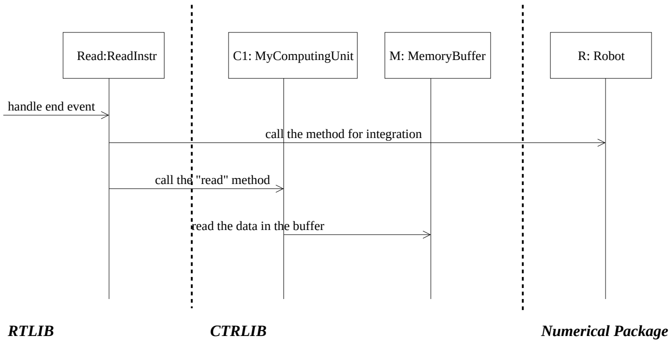

Figure 3: Sequence diagram showing the interactions which take place when an end event for a pseudoinstruction is handled.

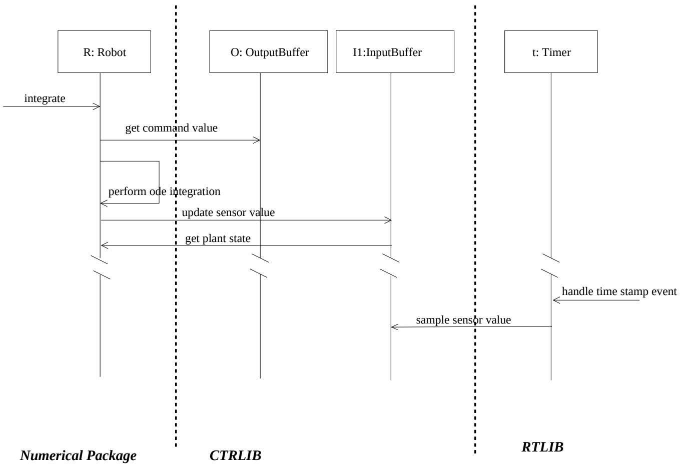

Figure 4: Sequence diagram showing: 1) how the integration is performed, 2) how the sensor values are sampled.

consists of a network link and an interface for each kernel it is connected to. Tasks on different nodes communicate by means of messages, that are transferred on the network link. Each message can have a deadline, depending on the communication protocol. RTLIB implements a generic dedicated link (which models point to point connections), the Ethernet and the CAN bus.

Summarizing, once a structure of tasks has been specified along with the underlying hardware/software architecture, RTLIB permits to construct a timed sequence of events associated with the jobs activation and termination instants, detailing, if necessary, with the beginning and the termination of pseudo-instructions. The occurrence of an event can be recorded into a simulation trace or accounted for to collect statistics. In this way it is possible to gather a great deal of information on the system timing behaviour, including the maximum response time of a task, the number of preemptions it undergoes, the end-to-end delay of a pipeline of tasks and so on. Figure 2 shows an execution trace reconstructed using the RTLIB simulator. The trace visualization tool is part of RTLIB suite and is entirely written in Java to enhance portability.

Some insight into the simulation. In order to perform the simulation, an important property of the systems under analysis can be exploited. This property consists in the fact that the interactions between the plant and the controller take place only in correspondence with specific events. On the one hand, in the time interval separating two writing on the output buffer, the differential equations of the plant can be integrated assuming constant values in the actuators. On the other hand the plant state can be observed through the objects simulating the sensors only when an event associated to the sampling is generated. Therefore, the simulation of the plant and of the controller can be carried out almost independently.

In order to highlight the interactions between different components of RTSIM which take place upon the occurrence of some meaningful events, consider the diagram in Figure 3. The boxes represent RTSIM objects involved in a simulation. The diagram is partitioned according to the three different packages objects belong to. The horizontal arrows denote interactions between the objects realized via method call. The diagram shows a sequence of method calls which follow up the termination event of the **Read** pseudoinstruction. According to the programming framework of RTLIB (see [6]), the event has a target entity, in this case the **Read** object, which is responsible of handling it. The first action performed by **Read** is calling a method on the **Plant** object. This method call determines the integration of the differential equation up to the instant related to the event. After the plant state has been updated, the two simulations regarding the plant and the controller have been synchronized. The second action is a call on the “read” method of the computing unit associated with the pseudoinstruction, which, in its turn, reads the data from the buffer.

It is also very interesting to observe how the integration is performed (diagram in Fig. 4). At the beginning of the integration the value of the command variables, contained in the output buffer, is acquired. Then the integration can be performed (by calling the appropriate functions of the OCTAVE library) assuming constant values for the input throughout the integration interval. At the end of the integration, the values contained in the input buffers, which model the sensors, are updated. We observe that keeping the sensor value up to date while the integration is performed would be a useless effort for, as stated above, the interactions between the plant and the controller may take place only at the end of each integration interval (corresponding to the RTLIB generated events). In the same figure, after a while, a clock stamp event is triggered for the timer object. The event handling method issues a call to a method of input buffer $\mathbf{I}_1$ determining the sampled value to be updated.

4. AN APPLICATION EXAMPLE

Problem description. In order to provide a deeper explanation of the concepts shown so far, a simplified description of a meaningful application is presented. The controlled plant is an autonomous mobile robot. The control goal is to track a moving target until an assigned displacement, expressed in a frame attached to the robot, is achieved (see Fig. 5). The control problem is analyzed with full details in [20] and the proposed solution is based on the back-

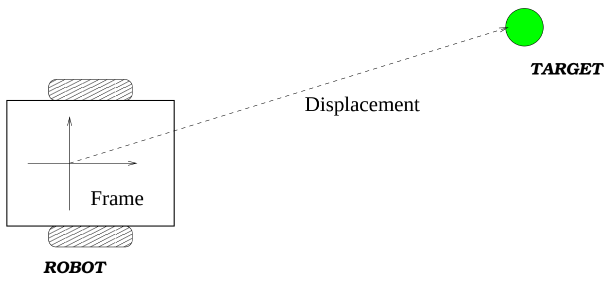

Figure 5: Scheme of the control example

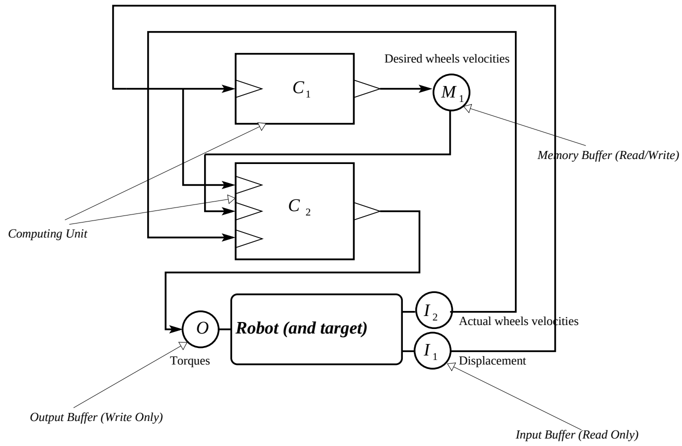

Figure 6: Functional scheme for the controller analyzed in the case study

stepping approach, which leads to hierarchical nested subcontrollers. If the robot wheels are driven by motors which can be commanded through the armature currents, the resulting controller comprises two hierarchical levels. The first level produces a reference value for the angular velocities of the wheels in order to make the actual relative displacement between the robot and the target converge to the desired value. The second level produces the torques to be applied to the actuators to make the actual angular velocity of the wheels converge to the reference value. The input to the first level is the measured displacement, while the second level is fed with the measured displacement, the measured angular velocity of the wheels and the reference value produced by the first level.

Modeling the system. The first step for simulating the presented application in the framework of the RTSIM software tool, is to provide a consistent modeling for the robot and for its controller. The model is constructed as a collection of C++ objects; some of them are defined by simply instantiating RTSIM predefined classes. For other objects it is necessary to write first the implementation of their classes using inheritance from RTSIM abstract classes. In Fig. 8 some of the objects composing the model of the system under analysis are shown. The boxes denote objects, whereas the lines denote the existence of a relation between the connected objects.

As far as the robot model is concerned, the programmer can resort to the classes provided by the numerical package. In particular, a **Robot** class can be derived from an abstract class named **GenericPlant**. The most important programming effort is represented, in this case, by providing the differential equation for the robot dynamics by an implementation of a pure virtual function inherited from **GenericPlant**.

A possible solution for the functional specification of the controller for the example application is based on two computing units: $\mathbf{C}_1$ and $\mathbf{C}_2$. The reference value of the angular velocity is communicated through a memory buffer while two input buffers are used for acquiring the plant data and an output buffer is used for issuing the command variable. The resulting structure is shown in Fig. 6. The diagram in Fig. 8 shows the object implementing $\mathbf{C}_1$ and the two storage units ($\mathbf{M}$ and $\mathbf{I_1}$) it is connected to. The relation between $\mathbf{I}_1$ and the plant is motivated by the necessity for $\mathbf{I}_1$ of reading the state variables of the robot.

For what concerns the architectural model, a simplified solution could be the one shown in Figure 7. The data from the plant are acquired by two sensors. The first sensor is a camera used to determine the relative displacement between the robot and the target; suppose it is sampled with period T_1. The second sensor is a pair of tachimetric encoders used to read the wheels' angular velocity; this sensor is sampled with period T_2. The computations specified within $\mathbf{C}_1$ and $\mathbf{C_2}$ are implemented by two periodic tasks τ_1 and τ_2, modeled using the appropriate RTLIB task class. The object **Tau1**, shown in Fig. 8 is used to model the τ_1. The task is constructed aggregating three pseudoinstructions (to be executed in sequence) labeled in the figure as **Read**, **Exe** and **Wri**. Such pseudoinstructions are related with **C1** so as to trigger the appropriate "functional" behaviour. In the proposed architecture, the tasks share a single CPU; thus, the model has to include a scheduler (shown as an object in the diagram). For this example, accesses to the storage units by the tasks are assumed to be entirely asynchronous (that is multiple tasks can use the shared buffers simultaneously without being blocked). There are many lock-free protocols in the literature which allow for this type of semantics (see [5]). Other types of protocols (for instance mutual exclusion semaphores, priority inheritance and so forth) are available as RTLIB classes and could easily be used. The object diagram shows also a timer object used for triggering the periodic sampling, into the $\mathbf{I}_1$ input buffer, of the relative displacement between the robot and the target.

Some experimental results. In this section, we report some simulation data collected utilizing the tool on the case study. We remark that the modeling of the system has been very accurate both for the plant and for the controller. In particular, referring to the architecture shown in Fig. 7, the tasks' execution times were assumed to be random variables uniformly distributed in the range $[40ms, 100ms]$ for task τ_1, and in the range $[1ms, 2ms]$ for task τ_2. The difference in the two ranges are due to the fact that the computational load of extracting the distance between the robot and the target from the grabbed image has been attributed to task τ_1. Moreover, a periodic task having fixed computation time $C_3 = 4ms$ activated with period $T_3 = 20ms$ was introduced to account for residual processor stealing activities (for ex-

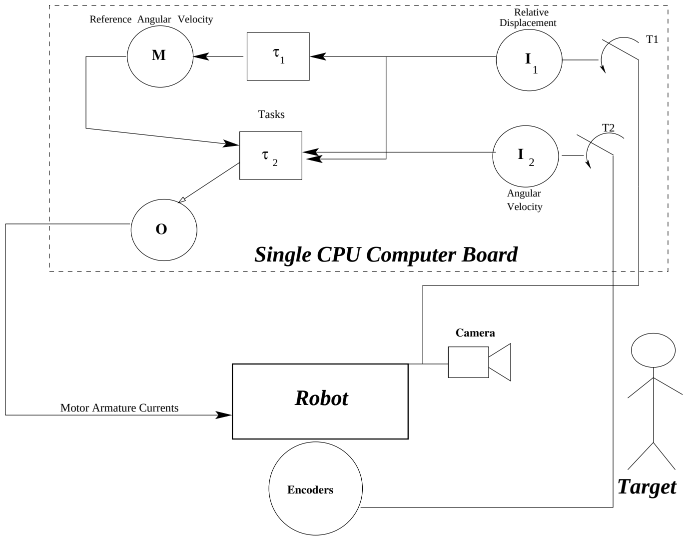

Figure 7: Control implementation scheme applied to the case study

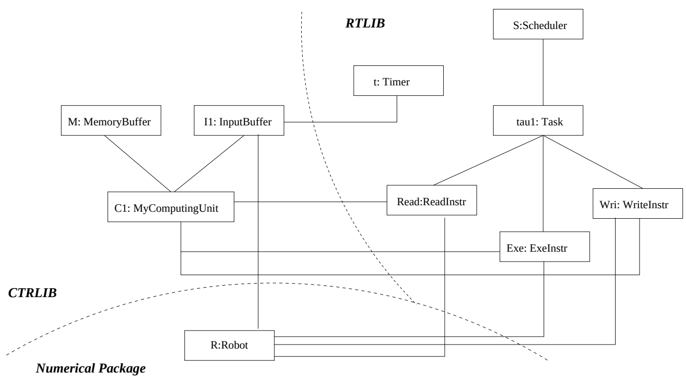

Figure 8: Partial object diagram modeling the system proposed in this example

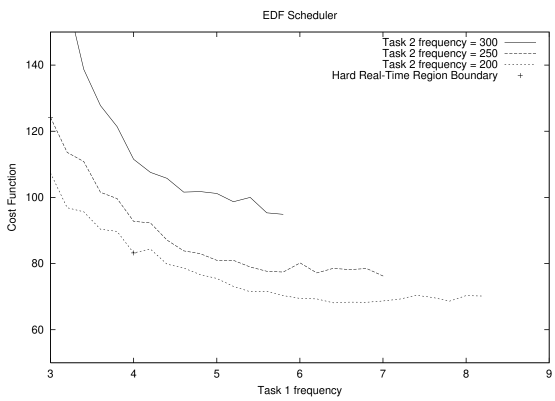

Figure 9: Cost function variations obtained using the EDF algorithm

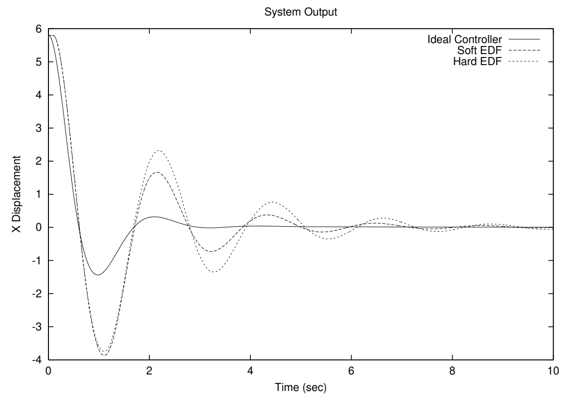

Figure 10: Dynamics of the first output

ample data logging). The robot physical parameters were identified on a mobile robot provided by the IDEA s.r.l. company.

In order to produce a quantitative evaluation of the system's performance [26], the following stochastic cost function, averaged through multiple runs, was considered

$$\mathcal{J} = E\left[\int_0^{+\infty} (\mathbf{y}^T Q \mathbf{y} + \mathbf{u}^T R \mathbf{u}) dt\right] \quad (1)$$

where $E[.]$ denotes the expectation value. For the performance evaluation to be fair, it is useful to consider the difference $\Delta\mathcal{J} = \mathcal{J} - \mathcal{J}_c$, where $\mathcal{J}_c$ is the ideal cost function, obtained considering null computation time and activation periods for the tasks. The study was performed on a set of trajectories, assuming that the robot tracks a still target. For periodic sets of tasks like the one being considered, the real-time scheduling theory [16] guarantees that each job of the tasks finishes within its deadline (i.e. its next periodic activation), if the periods are chosen according to the schedulability condition $\sum \frac{C_i}{T_i} \leq U_m$, where C_i is the worst case execution time of the i-th task, T_i is its activation period and U_m is a constant equal to 1 if the Earliest Deadline First algorithm and to 0.69 if the Rate Monotonic algorithm is used.

To assess the impact of the scheduling parameters on the system performance, expressed through the $\Delta\mathcal{J}$ cost function, different activation frequencies $f_1 = \frac{1}{T_1}$, $f_2 = \frac{1}{T_2}$ were tried. All the experiments were performed by running 30 repetitions of each simulation, and estimating the *mean* cost function (as expressed in Equation 1). For all points the 95% confidence interval resulted to be lower than the 3% of the mean value. The ranges of possible frequencies were chosen taking also into account the the different physical characteristics of the sampled devices. Figure 9 shows cost function $\Delta\mathcal{J}$ obtained on a specific trajectory using the EDF scheduling algorithm, when f_2 is held constant and f_1 is made to vary. An interesting result is that in the considered region the best frequency assignment under EDF resulted in $f_1 = 6.4Hz$ and $f_2 = 200Hz$, which does not respect the schedulability condition ($\sum \frac{C_i}{T_i} = 1.291 > 1$). This result can be visually observed in Figure 10 where the dynamics obtained in a simulation run for a frequency assignment respecting the schedulability condition (denoted as "Hard EDF") and for the best assignment (denoted as "Soft EDF") are shown. The explanation for this behaviour is that the strict respect of the schedulability condition imposes very slow sampling rates. On the contrary, activation rates beyond this limit bring to better performance since the price of occasional deadline misses is outweighed by the reduced sampling delays. However, if the chosen frequencies are too high, the controller behaviour becomes completely unpredictable and the performance degrades sharply. This simple consideration explains a minimum in the curves in Figure 9. A more detailed discussion on this case study can be found in [19], where novel scheduling solutions to improve the system performance are evaluated.

5. CONCLUSION AND FUTURE WORK.

In this paper a tool for the joint simulation of a plant and of a real-time embedded controller is presented. By using hybrid techniques the tool permits to provide a realistic modeling for many implementation related issues which are not usually accounted for during the controller design. The tool consists of a complete set of C++ libraries for modeling, simulating and gathering statistical profiles of performance metrics. As a particular application, the tool has been employed to study the impact of the real-time scheduling choices when a multi-task implementation on a single CPU is chosen for the controller. A planned future work is to use the same modeling framework to study the performance in case of multiprocessor and/or network distributed control applications. Moreover, a simulink-like modeling GUI for constructing the models of the system is under way.

6. REFERENCES

[1] L. Abeni and G. Buttazzo. Integrating multimedia applications in hard real-time systems. In *Proceedings of the IEEE Real Time Systems Symposium*, Madrid, Spain, December 1998.

[2] K. Åstrom and B. Wittenmark. *Computer Controlled Systems, (3rd ed.)*. Prentice Hall, 1997.

[3] T. P. Baker. A stack-based allocation policy for realtime processes. In *IEEE Real-Time Systems Symposium*, december 1990.

[4] J. Bengtsson, K. Larsen, et al. Uppal: a tool-suite for automatic verification of real-time systems. In Alur, Hentinger, and Sontag, editors, *Hybrid Sytems III*, Lecture notes in computer science. Springer Verlag, 1996.

[5] G. Buttazzo. *"HARD REAL-TIME COMPUTING SYSTEMS: Predictable Scheduling Algorithms and Applications*. Kluwer Academic Publishers, Boston, 1997.

[6] A. Casile, G. Buttazzo, G. Lamastra, and G. Lipari. Simulation and tracing of hybrid task sets on distributed systems. In *Real Time Computing Systems and Applications*, 1998.

[7] C. Daws, A. Olivero, et al. The tool kronos. In Alur, Hentinger, and Sontag, editors, *Hybrid Sytems III*, Lecture notes in computer science. Springer Verlag, 1996.

[8] A. Deshpande, A. Gollu, and L. Semenzato. The shift programming language for dynamic networks of hybrid systems. *IEEE Trans. Automatic Control special issue on Hybrid Systems*, April 1998.

[9] J. W. Eaton. Octave. www.octave.org.

[10] J. Eker and A. Cervin. A matlab toolbox for real-time and control systems co-design. In *Proc. of The Real-Time Computiong Systems and Applications*, Hong Kong, China, December 1999.

[11] T. Henzinger, P.-H. Ho, and H. Wong-Toi. Hytech: A model checker for hybrid systems. *Software Tools for Technology Transfer*, 1997.

[12] L. S. J.P. Lehoczky and J. Strosnide. Enhanced periodic responsiveness in hard real-time environment. In *Proc. of the IEEE Real-Time Systems Symposium*, 1987.

[13] N. Kim, M. Ryu, S. Hong, M. Saksena, C. Choi, and H. Shin. Visual asessment of a real-time system design: a case study on a cnc controller. In *Proceedings of the IEEE Real-time Systems Symposium*, 1996.

[14] H. Kopetz, A. Damm, C. Koza, M. Mulazzani, W. Schwabla, C. Senft, and R. Zainlinger. Distributed fault-tolerant real-time systems: The mars approach. *IEEE Micro*, 9(1), February 1989.

[15] E. A. Lee. Computing for embedded systems. In *IEEE Instrumentation and Measurement Technology Conference*, Budapest, Hungary, May 2001.

[16] C. L. Liu and J. Layland. Scheduling alghorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1), 1973.

[17] J. Nilsson, B. Bernhardsson, and B. Wittenmark. Some topics in real-time control. In *Proc. American control conference*, Philadelphia, June 1998.

[18] J. Nilsson, B. Bernhardsson, and B. Wittenmark. Stochastic analysis and control of real-time systems with random time delays. *Automatica*, 34(1):57–64, 1998.

[19] L. Palopoli, L. Abeni, F. Conticelli, M. D. Natale, and G. Buttazzo. Real-time control system analysis: an integrated approach. In *Proc. of the Real-Time Systems Symposium*, Orlando, Florida, December 2000.

[20] L. Palopoli, F. Conticelli, and B. Allotta. Multi-level stabilizing control of nonholonomic vehicles and its multirate digital implementation. In *Proc. of International Conference on Robotics and Automation*, S. Francisco, USA, 2000.

[21] D. Seto, J. Lehoczky, L. Sha, and K. Shin. On task schedulability in real-time control systems. In *IEEE Real Time System Symposium*, December 1996.

[22] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE transaction on computers*, 39(9), september 1990.

[23] K. Shin and X. Chui. Computing time delay and its effects on real-time control systems. *IEEE Transactions on Control Systems Technology*, 3(2):218–224, June 1995.

[24] K. Shin and H. Kim. Derivation and application of hard deadlines for real-time contol systems. *IEEE Transactions on Systesms, Man and Cybernetics*, 22(6):1403–1412, November/December 1992.

[25] K. Shin, C. Krishna, and Y. Lee. A unified method for evaluationg real-time computer controllers and its application. *IEEE Transactions on Automatic Control*, AC30(4):357–366, April 1985.

[26] K. Shin, C. Krishna, and Y. Lee. A unified method for evaluationg real-time computer controllers and its application. *IEEE Transactions on Automatic Control*, AC30(4):357–366, April 1985.

[27] B. Sprunt, L. Sha, and J. P. Lehoczky. Aperiodic scheduling for hard real-time systems. *The Journal of Real-Time Systems*, 1989.

[28] D. Stewart, R. Volpe, and P. Khosla. Design of dynamically reconfigurable real-time software using port-based objects. *IEEE trans. on Software Engineering*, 23(12), 1997.

[29] I. Stoica, H. Abdel-Wahab, K. Jeffay, S. K. Baruah, J. E. Gehrke, and C. G. Plaxton. A proportional share resource allocation algorithm for real-time, time-shared systems. In *Proceedings of the IEEE Real Time Systems Symposium*, December 1996.

[30] C. M. K. Thomas A. Henzinger, Benjamin Horowitz. Giotto: A time-triggered language for embedded programming. Technical report, University of California at Berkeley., 2000.