Multitasking Real-Time Control Systems in Easy Java Simulations

Farias, Gonzalo; Cervin, Anton; Årzén, Karl-Erik; Dormido, Sebastián; Esquembre, Francisco

2008

Multitasking Real-Time Control Systems in Easy Java Simulations

G. Farias*, A. Cervin, K. Årzén**,**
S. Dormido*, F. Esquembre***

**Depatamento de Informática y Automática, UNED, Madrid, Spain*
(e-mail: gfarias@bec.uned.es, sdormido@dia.uned.es),
*** Department of Automatic Control, Lund University, Sweden*
(e-mail:{anton,karlerik}@control.lth.se),
**** Departamento de Matemáticas, Universidad de Murcia, Murcia, Spain*
(e-mail: fem@um.es)

Abstract: The paper presents the development of interactive real-time control labs using Easy Java Simulations (Ejs). Ejs is a free software tool that allows rapid creation of interactive simulations in Java. A new TrueTime-based kernel has been designed in Ejs in order to create multitasking real-time system simulations as well as soft real-time applications. The main features of these new capabilities are presented.

1. INTRODUCTION

Control education has to take advantage of the possibilities that the new information technologies provide. One of these new features is the *interactivity* which is a crucial aspect when designing virtual labs for pedagogical purposes. The interactivity allows the user to simultaneously visualize the evolution of the system, and its response on-the-fly to any change introduced by the user. This immediate observation of the gradient of change of the system as response to user interaction is what really helps the student get useful practical insight into control system fundamentals (Heck 1999, Dormido 2004). Hence, interactive virtual labs have a great application potential in control engineering (Sánchez et al. 2005, Dormido et al. 2003, 2005).

An important subtopic of control engineering is embedded real-time control, which deals with feedback control under limited implementation resources (CPU time, communication bandwidth, energy, and memory). In this area, some new tools have recently been developed to create simulations from a research point of view. One of them is the freeware tool TrueTime (Andersson et al. 2005, Ohlin et al. 2007), which is very appropriate for the study of embedded real-time control systems but lacks the interactive features required to create easy-to-use simulations for educational purposes. On the other hand, the freeware tool Easy Java Simulations (Ejs) facilitates the creation of interactive simulations (Esquembre 2004) but is not able to handle multitasking real-time control systems.

A first approach was to combine TrueTime and Ejs in order provide to users the possibility to create multitasking real-time labs (Farias et al. 2007). However in that case a user needs to have a Matlab license in its computer. In this paper we describe a new approach which is a first attempt to expand the functionality of Ejs to handle multitasking real-time control systems. To simplify the implementation, we have decided to adapt the basic real-time task model from TrueTime. The functionality is so far quite limited, only the basic primitives to describe real-time kernels and tasks have been implemented. Advanced TrueTime simulation features like wired and wireless communication networks are not yet supported in Ejs.

The rest of this paper is organized as follows. In Section 2, the tools Ejs and TrueTime are introduced. Section 3 describes the implementation and use of the real-time kernel simulation features in Ejs. Section 4 presents the soft real-time kernel, while Section 5 presents the conclusions.

2. BACKGROUND

2.1 Real-Time Control Systems

A real-time system is a computer program which has to satisfy some time-constraints. The systems are called hard or soft real-time systems if the time-requirements are or not critical to a suitable performance.

Real-time control systems, which execute control algorithms, are traditionally designed jointly by two different types of engineers. The control engineer develops a model of the plant to be controlled, designs a control law and tests it in simulation. The real-time systems engineer is given a control algorithm to implement, and configures the real-time system by assigning priorities, deadlines, etc.

A new interdisciplinary approach is currently emerging where control and real-time issues are discussed at each design levels (Cervin et al. 2003). The development of algorithms for co-design of control and real-time systems requires new tools. One of these new tools is TrueTime, which will be the base for the simulation model of real-time control system implemented in Ejs.

2.2 Easy Java Simulations (Ejs)

Ejs is a free software tool for rapid creation of simulations in Java with advanced graphical capabilities and a high degree of interactivity. Ejs is different from most other authoring tools in that it has not been designed to make life easier for professional programmers. Rather, it has been conceived for engineering students and teachers; that is, for people who are more interested in the content of the simulation and much less in the technical aspects needed to build the simulation.

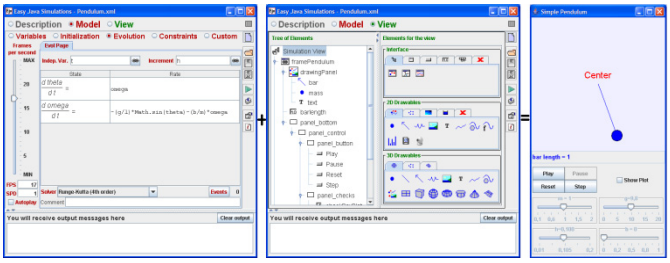

Fig. 1. An Ejs simulation consists of a model and a view.

Ejs structures a simulation in two main parts, the *model* and the *view* (see Fig. 1). The model can be described by means of Java code, ordinary differential equations, or by dynamical models in external applications (such as Matlab/Simulink). The view provides the visualization of the simulated system and the user interface elements required for interaction with the user. The view elements can be chosen from a set of predefined components to build a tree-like structure. Both model and view are easily interconnected so that any change in the model state is automatically reflected by the view, and vice versa, in order to provide a dynamic and interactive visualization of the system.

2.3 TrueTime

TrueTime is a Matlab/Simulink based simulator for networked and embedded control systems that has been developed at Lund University since 1999. The simulator software consists of a Simulink block library and a collection of MEX files. The TrueTime Kernel block simulates a computer node, containing a real-time kernel that executes user-defined tasks and interrupt handlers. The TrueTime Network blocks are capable of simulating a variety of wired and wireless networks.

To create a simulation using TrueTime, the plant dynamics are first modeled using ordinary Simulink blocks. Then, kernel and network blocks, that represent the computer implementation of the controller, are added to the model. For each kernel block, a set of Matlab functions have to be written: one function to initialize the kernel (and possible network interfaces), and one Matlab function for each task and interrupt handler in the real-time system. To model the execution time of a task or interrupt handler, a special code function format is used. A code function is divided into *code segments* according to Fig. 2. The execution of user code is done nonpreemptively in the beginning of each segment, and the code function returns the simulated execution time of the segment. Inside the code functions, the user can access the kernel block inputs and outputs using special kernel primitives *(e.g. ttAnalogIn and ttAnalogOut).*

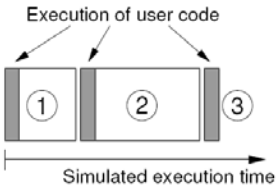

Fig. 2. TrueTime code model. The execution of user code is modeled by a sequence of segments that are executed in sequence by the kernel.

3. SIMULATION OF REAL-TIME KERNEL IN EJS

3.1 Overview

Inspired by the TrueTime Kernel block, a facility to simulate code execution and scheduling was implemented in Ejs. Currently, the Ejs Kernel supports periodic tasks, monitors (mutexes), and various scheduling policies such as earliest-deadline first (EDF) and rate-monotonic (RM) scheduling. Similar to TrueTime, the code of a task is divided into segments as shown in Fig. 2.

Internally, the kernel maintains a number of data structures that represents the tasks, the monitors, the scheduler, etc. Each time the kernel executes, it schedules its next invocation based on the time of the next expected event. An event can be for instance that a task has completed execution of a code segment, or that a task that has been sleeping should wake up.

The Ejs bisection solver will then schedule the event, executing the relevant task code at the proper point in time, see Fig. 3. Similar to TrueTime, the plant dynamics is modeled using the regular ODE functionality.

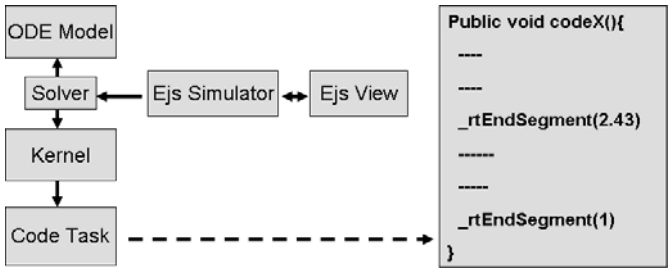

Fig. 3. Diagram of a real-time system simulation in Ejs.

3.2 Kernel Primitives

There are three main kinds of kernel primitives in Ejs. The first type is related the initialization of the kernel, the second one deals with tasks, and in the last group there are special functions to synchronize tasks and to get scheduling data.

Kernel primitives: The method *_rtInitKernel* is used to initialize the kernel. It is possible to declare the priority function desired (predifined or custom) and whether the scheduler should produce output data. It is also possible to

choose simulation mode or soft real-time application mode (see Section 4).

Task primitives: This is the main group of primitives; here there are methods to create periodic tasks (*_rtCreatePeriodicTask*), methods to set parameters of the tasks (like *_rtTaskSetPeriod* to set the period), and also methods to get parameters of the tasks (like *_rtTaskGetDeadline* to get the relative deadline). Finally, there is a special function to specify the end of a segment in the task code (*_rtEndSegment*, see below). The task code is entered as a method in the *Custom* section in Ejs.

Special primitives: In this group there are primitives to creating and using monitors for task synchronization (*_rtCreateMonitor*, *_rtEnterMonitor*, *_rtExitMonitor*). There are also methods to get schedule output data for a task (*_rtTaskGetSchedule*).

The functions implemented so far allow the creation of simple real-time control systems in an easy way. However, they do not yet support the simulation of for instance multiple kernels that communicate over a network.

There is a slight difference between the TrueTime and Ejs models to describe the task code. In TrueTime, a *switch-case* structure is used to determine the segment to execute. In the Ejs kernel, the code is divided into segments by the primitive *_rtEndSegment*(*value*). This function indicates to Ejs that the end of a segment has been reached, and the *value* argument is used to return the execution time of the code segment. Another difference is that, in Ejs, the variables declared in the *Variables* subsection are global, hence the task code can directly use the variables of the ODE model. This eliminates the need of special kernel primitives for reading inputs and writing outputs.

3.3 Example 1: Simple PID Control of a DC Servo

As a first example, we show how a simple real-time control simulation can be created in Ejs. The simulation itself shows how the execution time of the controller induces a delay in the feedback loop that might deteriorate the performance. In Fig. 4, the code to define the kernel and the task is shown. This code is entered in the *Initialization* section in Ejs.

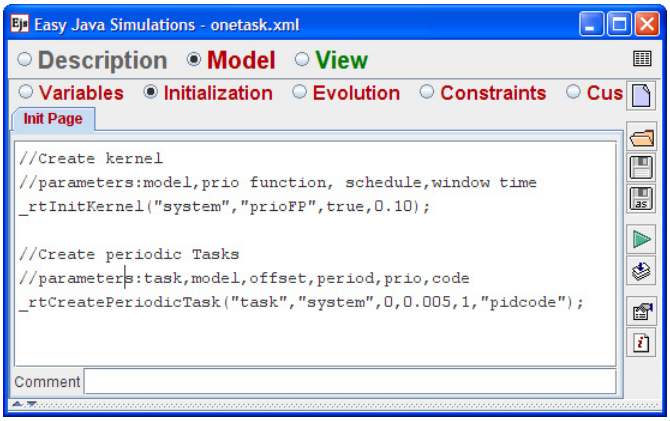

Fig. 4. Creation of the kernel and task for the first example.

The first instruction in the code, *_rtInitKernel*, is used to indicate that a simulated kernel will be used. The kernel is linked to the model called "system", which is shown in Fig. 5. The other parameters of the *_rtInitKernel* method indicate the selected priority function (fixed-priority scheduling), whether the schedule output data is needed, and if it is, how much schedule data time is required.

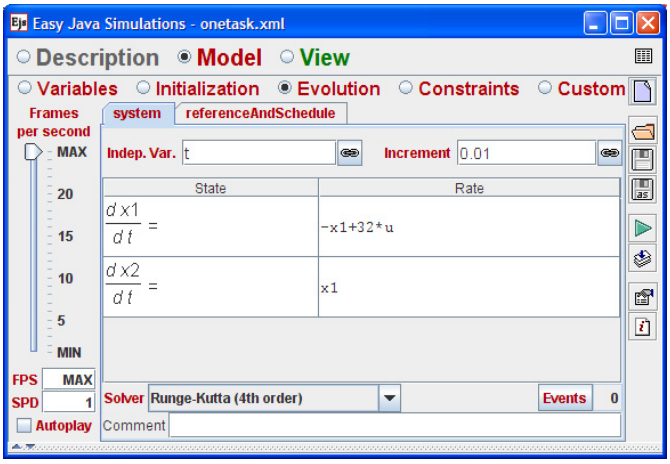

Fig. 5. Ordinary differential equations of the model system using the editor of Ejs. The models are defined in the *Evolution* section in Ejs.

The function *_rtCreatePeriodicTask* creates a period task. The first parameter gives the name of the task ("task"), while the second parameter indicates the model used ("system"). The next parameters are the task offset (0), period (0.005), and priority (1), respectively. The last parameter specifies that the method "pidcode" is the code function for the task. This method is shown in Fig. 6.

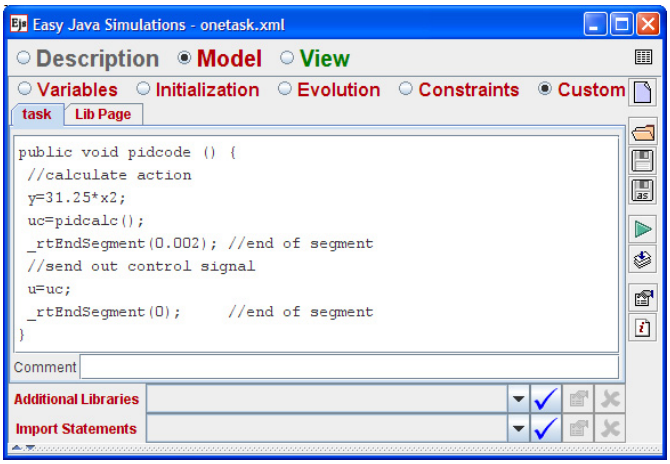

Fig. 6. The method "pidcode" used by the task created in the first example. The code functions are defined in the *Custom* section in Ejs.

In the first segment of the code function, the control action is calculated using the custom function "pidcalc". The segment is ended by *_rtEndSegment(0.002)*, specifying the execution time of this segment. In the second code segment, the control action variable *u* is delivered to the plant. In this case the

execution time for the segment is 0. After the second segment, the task should sleep until the next period.

Typical results of this example are shown in Fig. 7, where the reference, control and output signals are presented for two cases. In the first case (*a*) the *Execution Time* is 2[ms] and in the second case (*b*) the *Execution Time* is 9[ms]. In both cases the *Period* is 12[ms]. Note that the performance of the first case is better than the second one.

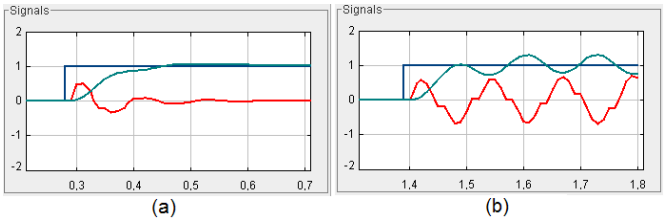

Fig. 7. Simulation of the first example: (*a*) The signals when *Execution Time* is 2[ms]. (*b*) The Signals when *Execution Time* is 9[ms]. In both cases the *Period* is 12[ms] and the control parameters are the same.

3.4 Example 2: Control of Three DC Servos

We now extend the previous example to the case of three PID-tasks running concurrently on the same CPU, controlling three different servo systems. The effect of the scheduling policy on the global control performance is demonstrated.

The graphical user interface for the simulation is shown in Fig. 8. The user can modify different parameters like: reference type, control settings, jitter and the execution time of the controller. Also the user may choose between two scheduling policies, rate monotonic (RM) and earliest deadline first (EDF). In the RM case, a task with shorter period has higher priority. In the EDF case, the priority of each task is dynamic and depends on the minimum absolute deadline. If RM is selected, then the task with the longest period is strongly affected by the other tasks, whereas if EDF is selected, then all the tasks will be affected by each other.

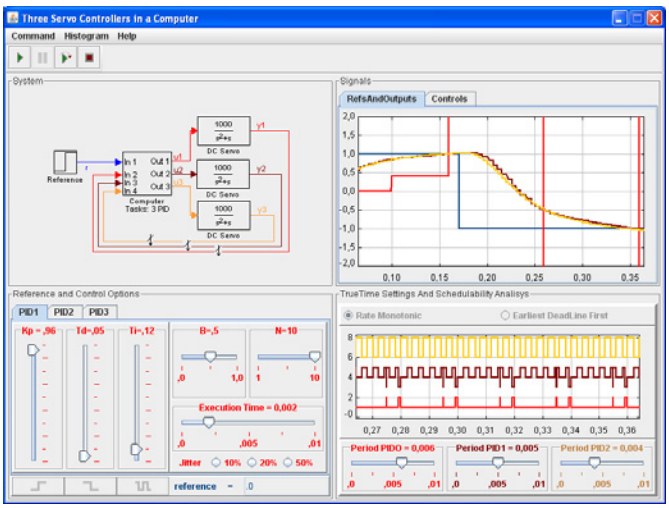

Fig. 8. Graphical user interface for the second example.

In the time plots of Fig. 8, it is seen that RM scheduling causes the control loop with the longest period (and hence lowest priority) to go unstable.

4. SOFT REAL-TIME CONTROL USING EJS

4.1 Overview

In some cases, users may want to execute their real-time models as real applications, and in real time (possibly even interfaced to real plants). In principle this is not possible, since Ejs generates ordinary Java programs without any timing guarantees. However, given a fast enough computer, a Java application may be *time-stable* over a long execution. Hence, as long as the user does not care about hard time constraints and just wants to execute an application (for instance to control a laboratory process), s(he) can use a special Ejs kernel to do it in an easy and fast way.

The special kernel allows the execution of tasks as Java Threads, but using the scheduling algorithm defined by the user. Of course, the time accuracy of the final application depends on the Java Virtual Machine, the operating system, the hardware, etc.

Fig. 9 shows how a soft real-time application can be implemented in Ejs. The kernel, tasks, and even the Ejs View are now Java threads. The Ejs View can be considered as a soft aperiodic task, since it will be executed only if the kernel is idle. The threads (the kernel and the tasks) are executed when they have a *token* (implemented as a Java monitor). The *sleep* and *wait* methods of Java threads and *System.nanotime* method are used to control the time constraints.

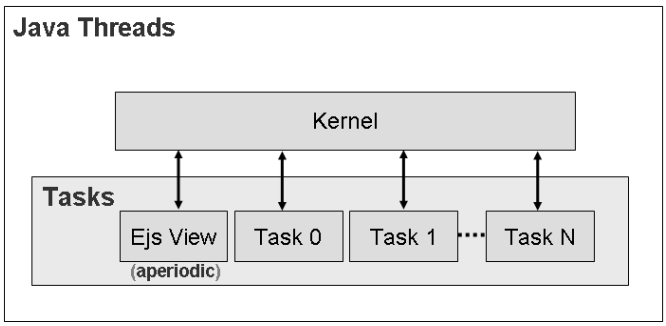

Fig. 9. Diagram of a soft real-time application described in Ejs.

One limitation of our implementation is due to the fact that it is not possible to stop a thread from another thread. (This option was deprecated from Java authors because it is inherently unsafe.) This makes it impossible to correctly preempt a task that is currently executing a code segment. Hence two options were introduced: the concurrent and non-concurrent approach. In the concurrent case, a preempting thread is released and is allowed to execute in "parallel" until the preempted thread has finished. In the non-concurrent case, preemption will not occur until the code segment is finished.

Fig. 10 illustrates the two cases. There are two tasks, *T0* with three segments and *T1* with four segments. Task *T1* has higher priority than task *T0*. The behavior depends on the case selected:

- Non-concurrent case: Task *T1* has to wait for the end of the current segment of task *T0*. When the segment of task *T0* is finished the first segment of task *T1* is executed. The next segment of task *T0* will be executed when the schedule policy indicates so.

- Concurrent case: Task *T1* starts execution at the release time, even though a segment of task *T0* is still executing. The concurrence situation will finish when the task *T0* segment finishes its code (i.e. when an *_rtEndSegment* is executed). The next segment of task *T0* will execute when the schedule policy decides so.

Both cases have advantages and disadvantages. The concurrent case executes the task at the release time, but the concurrence introduces an unpredictable jitter in the execution of both tasks. Moreover, preempting a segment could lead to unwanted side-effects. The non-concurrent case introduces release jitter in the execution of the tasks.

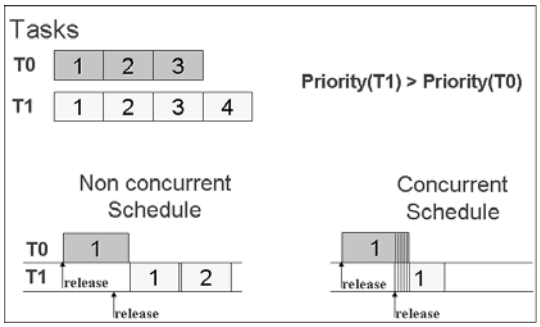

Fig. 10. Example of concurrent and non concurrent cases. There are two tasks with three and four segments respectively. Task *T1* has a higher priority than task *T0*. In concurrent case task *T1* is executed even if the segment of *T0* has not finished yet, and non concurrent case task *T1* has to wait for the end of the segment of task *T0*.

4.2 Example: A Soft Real-Time Application

We now show an example of a real-time application. This example is very simple and has no real pedagogical value. However, it shows the approach that can help create, in an easy way, interactive (and even remote) real control laboratories. Remote laboratories have an scheme (Fig. 11) where the tasks require different priorities in order to provide a good performance for end users (Dormido 2007).

In our example, we can suppose that we just need two tasks (communication and control tasks) to implement the server side for a remote lab. The creation of the kernel and two tasks, *task0* and *task1*, is shown in Fig. 12. Since in this case any ODE model can be used (actually a real plant would be used) the first parameter of *_rtInitKernel* is null. Note that in this case the execution will use the concurrent approach.

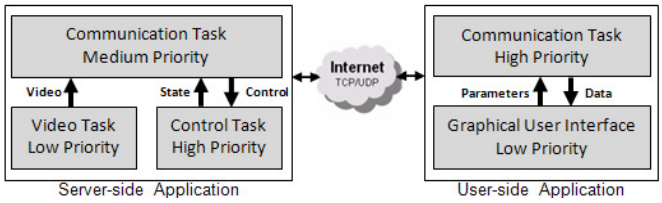

Fig. 11. Application and tasks diagram of an interactive remote lab.

In the creation of the tasks the values for the periods and offsets are given in seconds. The scheduling policy is fixed priority scheduling, and the *task1* has the highest priority. Further, *task0* starts after one second, and one and half seconds later the *task1* will execute the first segment.

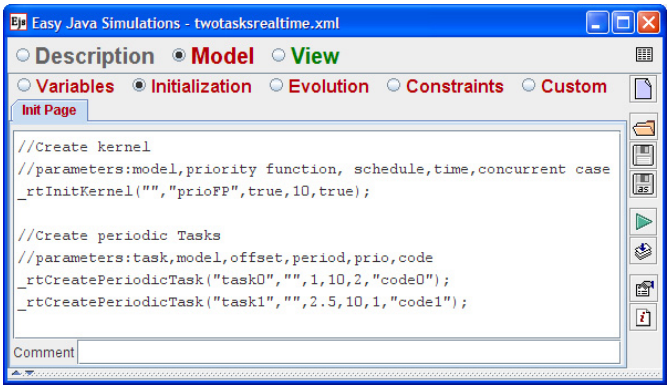

Fig. 12. Creation of the kernel and tasks for the real-time application.

The code for both tasks is presented in Fig. 13. Both code functions call the custom method *sleepAndEndSegment()*, which generates a pause (using the Java sleep method) of the thread for 1000 milliseconds. After the pause, the method *_rtEndSegment()* is executed to indicate the end of the segment to the kernel. Hence, *task0* has three segments of one second, and *task1* has two segments of one second.

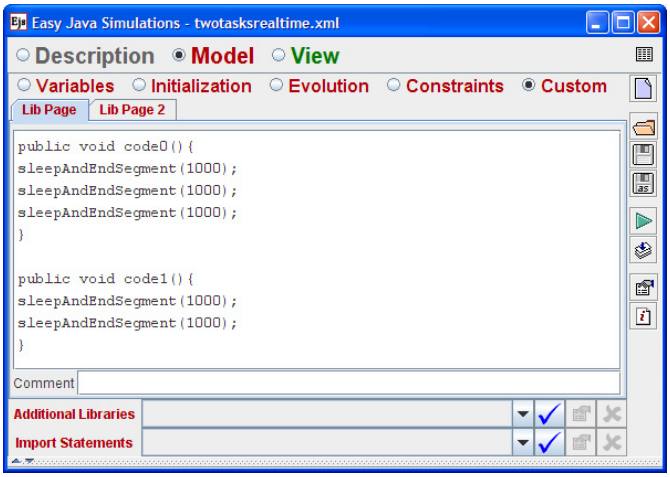

Fig. 13. Code for both tasks used in the real-time application.

When the application is executed, it is possible to view the state of both tasks for the first ten seconds. In the Fig. 14, the schedule signals are presented. In this case we can observe that the state of the *task1* (upper signal) indicates that the first segment is executed at the release time (two and half seconds). Hence, the execution of this segment is concurrent with the execution of the second segment of *task0* for half a second.

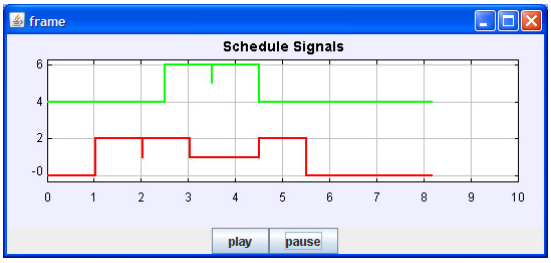

Fig. 14. Schedule data for both tasks in the concurrent case. Note that the tasks are executing concurrently for half a second.

If we select a non-concurrent case for this application, the schedule signals are now different to the previous case. The results are presented in Fig. 15, and it can be noticed that the first segment of *task1* is delayed until the end of the second segment of *task0* before executing.

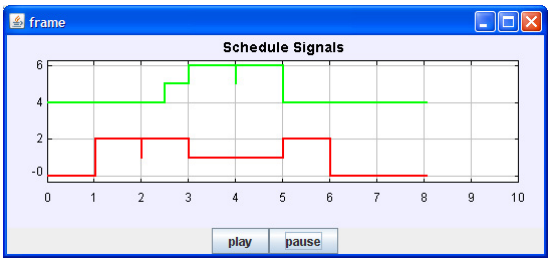

Fig. 15. Schedule data for both tasks in the non-concurrent case. Note that the *task 1* is delayed for half a second until the *task 0* finishes its second segment.

Regarding to the interactive remote lab, probably the performance of the concurrent case is better since at least the control task (the highest priority task here) will start to execute when the period is reached, that is very important in particular, if the user introduces some disturbing in the plant.

5. CONCLUSION

In this paper a new experimental feature in Easy Java Simulations is presented. The feature allows non-programming instructors to create simulations of real-time control systems or even soft real-time applications.

The task model and the structure of the kernel is based on TrueTime, a freeware Matlab/Simulink based tool. The tasks are defined using code functions that are further divided into segments. Users of TrueTime will find that Ejs scheme to describe real-time systems is quite similar to the one TrueTime provides. However, only a basic set of TrueTime functions has been implemented so far.

Using this implementation, users can describe simple real-time control systems in an easy way, with all the interactive features that Ejs provides to create virtual labs for pedagogical purposes.

Future work will mainly consist in adding new functionality based in the TrueTime model. But we are also interested in exploring how much the new possibilities actually allow us to create soft real-time applications in order to control real plants.

ACKNOWLEDGEMENTS

This work has been supported by the by the Comisión Interministerial de Ciencia y Tecnología(CICYT) under Grant DPI2007-61068, and by the IV PRICIT (Plan Regional de Ciencia y Tecnologia de la Comunidad de Madrid), under Grant S-0505/DPI/0391.

REFERENCES

Andersson M., Henriksson D., Cervin A. and Årzén K. (2005) Simulation of wireless networked control systems, *Proceedings of the 44th IEEE Conference on Decision and Control and European Control Conference ECC*, Seville, Spain.

Cervin A., Henriksson D., Lincoln B., Eker J. and Årzén K. (2003) How does control timing affect performance?, *IEEE Control Systems Magazine* **23(3)**, 16–30.

Dormido, S., Esquembre, F. (2003) The Quadruple-Tank Process: An Interactive Tool for Control Education, *Proceedings of the European Control Conference ECC,* Cambridge, England.

Dormido S. (2004) Control learning: Present and future: *IFAC Annual Control Reviews*, **vol. 28**, 115-136.

Dormido, S., Esquembre, F., Farias, G., Sánchez, J. (2005) Adding interactivity to existing Simulink models using Easy Java Simulations, *Proceedings of the Conference Decision and Control-European Control Conference*.

Dormido, S., Vargas, H., Sanchez, J., Duro, N., Dormido, R.; Dormido-Canto, S., Esquembre, F. (2007) Using web-based laboratories for control engineering education, *Proceeding of International Conference on Engineering Education*, Coimbra, Portugal.

Esquembre F. (2004) Easy Java Simulations: A software tool to create scientific simulations in Java: *Comp. Phys. Comm.* **156**, 199-204.

Farias, G., Cervin A. and Årzén K. (2007) Interactive Real-Time Control Labs with TrueTime and Easy Java Simulations, *Proceedings of the International Multiconference on Computer Science and Information Technology*, Wisla, Poland.

Heck B.S. (editor) (1999) Special report: Future directions in control education *IEEE Control Systems Magazine*, **Vol. 19**, No. 5, 35-58.

Ohlin M., Henriksson D., Cervin A. (2007) TrueTime 1.5 Reference Manual: Manual, Department of Automatic Control, Lund University, Sweden.

Sánchez J., Dormido S., Esquembre F. (2005) The learning of control concepts using interactive tools: *Computer Applications in Engineering Education*, **Vol. 13**, No 1, 84-98.