Proceedings of the
44th IEEE Conference on Decision and Control, and
the European Control Conference 2005
Seville, Spain, December 12-15, 2005

Adding interactivity to existing Simulink models using Easy Java Simulations

Sebastián Dormido, Francisco Esquembre, Gonzalo Farias, José Sánchez

Abstract— **This paper describes how to use Easy Java Simulations (Ejs for short), a software tool designed to create interactive simulations in Java, to turn existing models created using Simulink into really dynamic, interactive virtual labs. The user basically needs to connect variables in the Ejs model to variables in the Simulink model using an easy-to-use, very natural interface. Ejs takes then care of all the technical tasks required to keep a perfect synchronization between both tools. The main advantage is that existing Simulink models can benefit of the extended graphic capabilities of Ejs for a more realistic visualization and for enhanced interactive possibilities.**

I. Introduction

INTERACTIVITY is a crucial aspect when designing virtual laboratories that are to be used for pedagogical purposes in the field of control engineering. It is interactivity that allows the control engineering student to explore different configurations, values for the parameters, and the qualitative and quantitative response of the system, when trying to acquire not only the theory of the discipline, but also its strategic and intuitive aspects [1],[2].

But true interactivity goes beyond the mere capability of introducing the value of some parameters, running the algorithms, and analyzing and comparing the response of the system in static plots, repeating the process iteratively until a satisfactory configuration is found. True interactivity should allow the user to simultaneously visualize the evolution of the different aspects of the system, and their response in real-time to any change in the parameters that govern the system 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 [3].

Simulink™ is a modeling tool, based in Matlab™ [4] [5], that can be used to create dynamic system models in a graphical way and that has become the de-facto standard software for instructional purposes in control engineering. However, models created with Simulink suffer from a certain lack of interactivity in the sense we just described. A typical instructor would face serious difficulties if (s)he had to develop, using only Simulink, virtual labs with the graphical and interactive capabilities required for what we have termed 'true interactivity'.

This is where Easy Java Simulations comes in handy. Ejs is an open-source (and therefore completely free) software tool designed to create simulations in Java with high-level graphical capabilities and with an increased degree of interactivity, true interactivity. The tool provides its own mechanism for describing models of scientific and control engineering phenomena, and, as such, can be used to create virtual laboratories on its own.

However, this stand-alone use of Ejs has already been described elsewhere [6], and several examples of its use for the creation of virtual labs in the field of control engineering can be found in the literature (see, for instance, [7], [8], and [9]). This paper describes how Ejs can be used to provide existing Simulink models with the level of interactivity required for effective control engineering education. The combination of these two tools brings together the best of both worlds. Authors can use Simulink to develop the model of a control engineering system, and then move to Ejs to create the graphical user interface which provides the necessary visualization and user interaction for this same model.

It is of course possible to build user interfaces with certain interactive capabilities using the facilities included in Simulink only. However, this doesn't provide so varied and rich a set of primitives as Ejs offers. Additionally, the time and effort required to produce an interactive user interface are much higher using Simulink, and the task requires an advanced knowledge of the environment. Ejs offers the author more flexibility, power, and easiness of use when building true interactive user interfaces.

Since nothing illustrates better how to work with software tools than a good example, we will use the classical example of a simple pendulum to see how we can turn this existing Simulink model into a real interactive, dynamic simulation by using Easy Java Simulations. Although we don't assume the reader familiar with Ejs, we will not describe all its features, but concentrate only in what is needed for the goals of this paper. Readers interested in Ejs itself can download the software and a detailed manual from [10].

The paper is organized as follows. In Section II we briefly review the basics of Easy Java Simulations. Section III shows how Ejs and Simulink can cooperate in order to

Manuscript submitted February 15, 2005. This work was supported in part by the Spanish Ministry of Research under Grant DPI2004-01804.

F. Esquembre (corresponding author, fem@um.es) works at the University of Murcia, Spain. S. Dormido, G. Farias and J. Sánchez work at the Spanish National University for Distance Education, in Madrid.

0-7803-9568-9/05/$20.00 ©2005 IEEE

produce interactive virtual labs. A step by step systematic procedure to create and connect variables between Ejs and Simulink is presented in Section IV. Section V shows the mechanism used to control the execution of the model running in Simulink from Ejs. Section VI and VII are devoted to explain how to build a view and run the simulation. Finally, Section VIII gives some concluding remarks and considerations about further work.

II. How Easy Java Simulations works

Ejs structures a simulation in two main parts, the model and the view. (To these two main parts, Ejs adds a first introductory part which we won't describe here.)

The model describes the simulated system by means of variables (both state variables and parameters), that completely characterize the system, and of computer algorithms that state how the system evolves in time and how it responds to user interaction. Authors need to declare the variables using a simple table, and write the Java code needed to specify the algorithms. Ejs offers specialized help to solve models based on ordinary differential equations by providing an editor to write these equations and automatically generating the code required using the most popular solvers.

The view provides the visualization of the simulated system, either in a realistic form or using one or several data graphs, and the user interface elements required for user interaction. These view elements can be chosen from a set of predefined, ready-to-use components, to build a tree-like structure in a kind of block construction game for the view. There are elements of several types. Each type specializes in a given visualization or interaction task, but can also be customized using the so-call *properties*, a set of internal values that modify the aspect and behavior of the element on the screen. This way, the job of the author when building the view consists in choosing the right elements from those offered and in customizing them for the interaction desired for the simulation (see Figure 1).

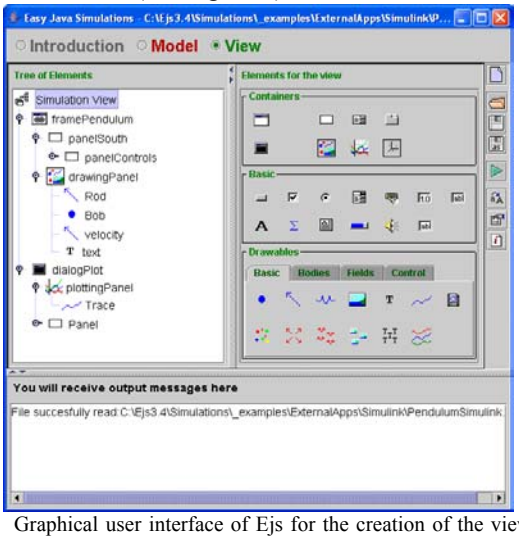


Fig. 1. Graphical user interface of Ejs for the creation of the view of a simulation. The tree at left corresponds to the view of Fig. 3.

Both, model and view need to be interconnected. Any change in the model state must be immediately reflected by the view in order to keep a dynamic, real-time visualization of the system. In turn, any interaction of the user with the view must immediately affect the model so that the intended true interactivity is achieved. This communication is based on *connecting* model variables and view elements properties. This connection is very easily established by typing, in the table of properties of view elements, the names of the model variables we want to connect to the properties.

Once the model and the view have been created and the required connections established, Ejs creates the ready-to-run simulation at a single mouse click, taking care of a good number of technical issues that thus become completely transparent to the author. The result is an independent, high-performance, interactive simulation which can be run either as a stand-alone Java program, or be embedded as an applet in an HTML page.

III. Ejs and Simulink

Although authors can decide to use Ejs' prescribed mechanism to describe the model of the simulation, they can also use Simulink for it. This allows experienced Simulink users to benefit from their know-how to quickly develop interactive simulations, as well as to reuse legacy code and/or existing models.

The procedure is quite simple. It basically consists in connecting Ejs' variables to variables (input, output variables and parameters) of the blocks in the Simulink model. Ejs provides a simple, visual mechanism to do this which turns out to be rather natural, and which requires no modification on the original Simulink model.[1]

We will illustrate this procedure by means of an example. Consider the Simulink model shown in Figure 2, which corresponds to the model of a simple pendulum with friction.

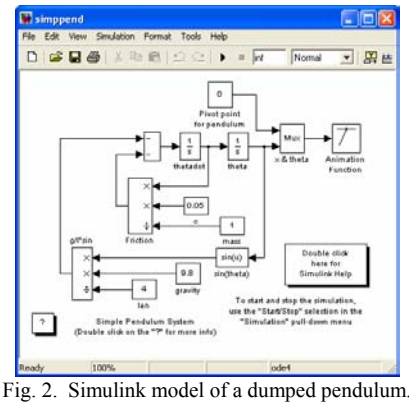


Fig. 2. Simulink model of a dumped pendulum.

We have chosen this example (although not from the field of control engineering) for its simplicity and general applicability. The example derives from one of Simulink's

[1] This feature, new in release 3.4 of Ejs improves a more basic feature of previous releases of Ejs which did require changing the original model.

standard demos of Matlab 6.0 (but it will work well with later versions), which we have modified slightly so that the user can customize the different parameters of the simulation. As the picture shows, the model is a complete working simulation which can be run to display a pendulum of mass 1 and length 4, which oscillates with a friction coefficient of 0.05. The model also includes (what we consider) a rather basic visualization of the oscillating pendulum, by means of an animation block.

Notice that, if we want to modify any of its variables or parameters, we will need to stop the model, edit it in a manual form, and re-run it again, which is not a very interactive procedure. Instead, we want to provide to this model a user interface like the one shown in Figure 3. This interface is fully interactive, and responds to user interaction in real-time. In particular, the pendulum can be set to new initial conditions of angle and angular velocity by just dragging the bob or the velocity vector in the view to a new position.

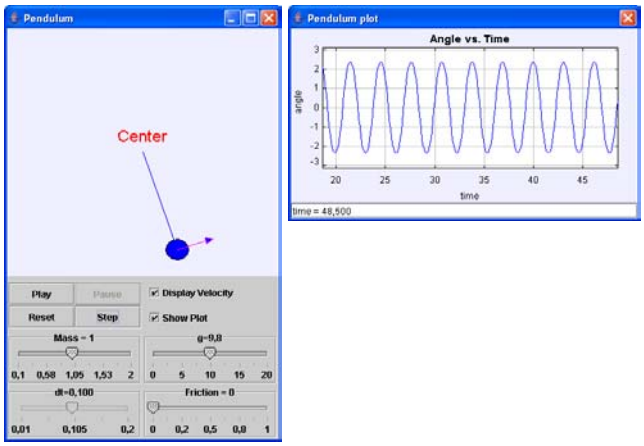


Fig. 3. The intended view for the simulation of the pendulum.

IV. Creating and Connecting Variables

The first step we need to complete in Ejs is to provide a description of the system in terms of variables. For this, we create in Ejs the table of variables of Figure 4.

Variables | Initialization | Evolution | Constraints | Custom

Basic Variables

External File _examples/ExternalApps/Simulink/simppend.mdl

Name	Value	Type	Dimension	Connected to
time	0.0	double		
dt	0.1	double		
mass	1.0	double		
length	1.0	double		
gravity	9.8	double		
friction	0.0	double		
angle	Math.PI/6.0	double		
velocity	0.0	double		

Fig. 4. Table of variables in Ejs that describe the motion of the pendulum.

This table has been created in a special page that includes a text field devoted to select what is called an *external file*, and which we use to select the Simulink model of the pendulum, *simppend.mdl*, located in the subdirectory *_examples/ExternalApps/Simulink* of Ejs' standard distribution. Selecting this file automatically instructs Ejs to begin working with Simulink.

Notice that the table of variables includes a column labeled *Connected to*, which appears now empty. The whole procedure consists in filling the cells of this column with the variables of the Simulink model which must be synchronized with the corresponding Ejs variable in the table. To help doing this, we can bring-in the so-called *connection dialog*, for each variable in turn, that will help us select variables from the Simulink model.

The connection dialog displays initially, as shown in Figure 5, the global variables and parameters of the model. If we want to select the variables of any of the blocks in the model, we can either select the block in the combo box included in the dialog or, more naturally, display the Simulink model itself (clicking on the check box at the upper-left corner of the dialog) and directly click on the desired block. The connection dialog will then display the variables and parameters of the block selected.

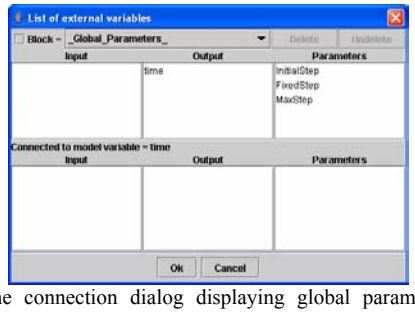


Fig. 5. The connection dialog displaying global parameters of the Simulink model.

The connection dialog displays three white areas in its upper half part. These areas list separately *input* variables, *output* variables, and dialog *parameters* of the selected block. To select one of the variables we only need to double click on it and it will be added to the corresponding area in the lower half-part of the dialog. Unselecting is carried out in a similar way.

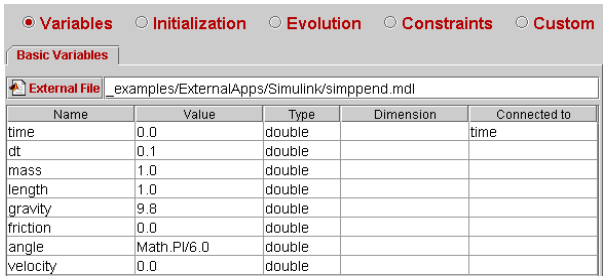
The category from which we select our variable has an effect on what we can do with it:

- Input variables can be freely changed from Ejs. That is, we can use any of Ejs' mechanisms to change them, at any time, either in the model or in the view. Any change we do to them will be immediately reflected in the model.
- Output variables can, on the contrary, only be read by Ejs. Hence, any change we do to their associated variables in Ejs won't affect their value in the Simulink model. This has an important exception, though, which consists in output variables of integrator blocks, as discussed below.
- Finally, parameters can also be changed from Ejs, but Simulink only guaranties using the new value if the model has not started running.

Although this may seem like a complicated distinction, it is actually not such, since each category is typically used

exactly as we would expect it to be. For instance, it is very unlikely that we want to change the time directly in our model. Instead, we expect Simulink to advance the time by iteratively running the model.

Once we have selected the variable we want, clicking the *Ok* button establishes the connection. See Figure 6.


Variables Initialization Evolution Constraints Custom

Basic Variables

External File _examples/ExternalApps/Simulink/simppend.mdl

Name	Value	Type	Dimension	Connected to
time	0.0	double		time
dt	0.1	double		
mass	1.0	double		
length	1.0	double		
gravity	9.8	double		
friction	0.0	double		
angle	Math.PI/6.0	double		
velocity	0.0	double		

Fig. 6. The table of variables reflects now that Ejs variable *time* is connected to the time in the Simulink model.

As a second example of connection of variables, we will connect the mass of the pendulum in Ejs' table of variables to the input variable of the Simulink model where the mass is used to compute the frictional force. This is the third input of the block called *Friction*, which appears selected in Figure 7.

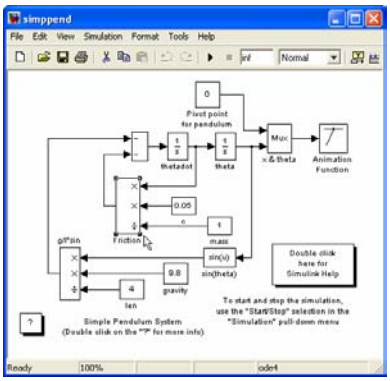
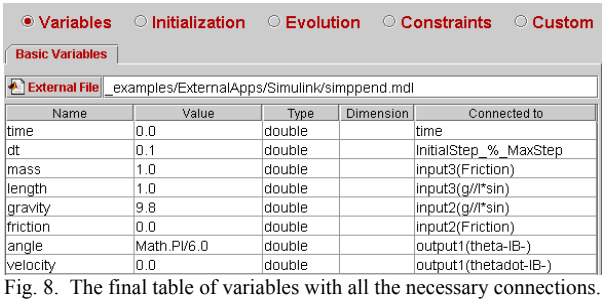


Fig. 7. The mouse points to the input of the block where the mass of the pendulum is used to compute the frictional force.

Selecting the third input of this block would result in the table displaying the connection of the mass with the *input3* variable of the block called *Friction*. Figure 8 shows the final state of the table of variables once we have established all the necessary connections. Notice that there is a variable, *dt*, connected to two parameters of the model (which is perfectly valid), and that the angle and the (angular) velocity are connected to variables of blocks which have a name with the suffix *–IB–*.

This suffix expresses a peculiarity of the variables. These variables are actually output variables of a special type of Simulink block called *Integrator Block* (because it integrates a differential equation). Even when they are output variables and, in principle, they should not be changed by Ejs, most users would want to do this, because this implies changing the initial condition of the system. For example, in our model, this means setting new values to either the angular position or the angular velocity of the pendulum.

Changing the initial conditions of a differential equation


Variables Initialization Evolution Constraints Custom

Basic Variables

External File _examples/ExternalApps/Simulink/simppend.mdl

Name	Value	Type	Dimension	Connected to
time	0.0	double		time
dt	0.1	double		InitialStep_%_MaxStep
mass	1.0	double		input3(Friction)
length	1.0	double		input3(g//l*sin)
gravity	9.8	double		input2(g//l*sin)
friction	0.0	double		input2(Friction)
angle	Math.PI/6.0	double		output1(theta-IB-)
velocity	0.0	double		output1(thetadot-IB-)

Fig. 8. The final table of variables with all the necessary connections.

modeled with Simulink is not impossible, but is a bit traumatic, in the sense that Simulink needs to reset the integrator blocks and restart them. For this reason, Ejs provides the special Java method *_external.resetIC()*, that does this task and that we will need to invoke whenever we want Simulink to accept any change we do in Ejs to any of the variables *angle* or *velocity*.

V. Controlling the Execution of the Model

Once we have connected variables in Ejs and variables in Simulink, Ejs can take control of executing the Simulink model in a synchronized way to the execution of its own main loop, using the utility method:

_external.step (int times);

A call to this method has the following effects:

1) It first pushes the values of any Ejs variable which is connected to a Simulink input variable. Variables connected to Simulink parameters are also pushed, but they only have a real effect in the model if the simulation has not started running, that is, only when the time is still equal to 0. Ejs variables connected to Simulink output variables are not pushed, except those belonging to an integrator block.
2) It runs the Simulink model as many steps as the parameter in parentheses states.
3) It finally retrieves the value of all Simulink output variables which are connected to Ejs variables.

A final change that we may want to do to the original Simulink model is to remove any visualization included in it. In our case, this amounts to suppressing the *Animation Function* block. However, one of our requests was not to modify the existing model (after all, we may want to run it independently). For this reason, the connection dialog provides a way to instruct Ejs to remove it from the model, but *only when it runs together with Ejs*.

This is done by selecting the block we want to suppress and clicking on the *Delete* button on the upper row of the connection dialog. Blocks that have been suppressed can be recognized because they appear dimmed in the list of blocks of the combo box of the connection dialog. See Figure 9.

VI. Building a View for the Simulation

The Ejs model for the simulation is now ready. (Notice that we didn't need to program any algorithm, since the task of solving the model is entirely done by Simulink.) It is now

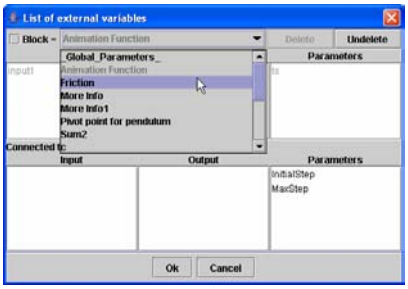


Fig. 9. List of blocks of the model. Blocks deleted appear dimmed.

time to prepare the view for it. The tree of elements for the view desired is shown in Figure 1 and the view itself in displayed in Figure 3. This task is accomplished in exactly the same way as for any other simulation created with Ejs, and we won't describe it here in detail. There are, however, a couple of minor peculiarities.

The first one derives from the fact that the Ejs variable *dt* is connected to the parameters *InitialStep* and *MaxStep*, to help us control from the user interface the integration step of the differential equation solver of the Simulink model. Because these are parameters, Simulink will only accept changes to them before actually running the simulation, i.e. when the simulation time is zero. For this reason, we decide to let the user change *dt* only when *time==0*. This is what we need to type in the *Enabled* property field of the slider view element for this variable (see Figure 10). Hence, when the time starts running, the slider will be disabled and the user won't be able to modify a variable that, on the other hand, will have no real effect in the execution of the simulation.

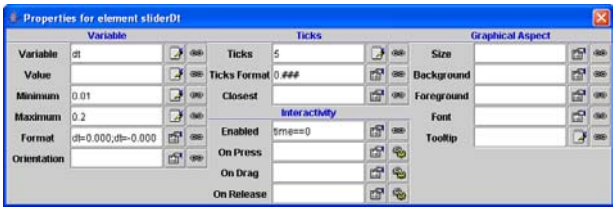


Fig. 10. Table of properties for the slider that modifies *dt*.

The second peculiarity has to do with the interaction of the user with the simulation intended to modify the initial conditions of the system, that is, Ejs' variables *angle* and *velocity*. This can be done by dragging either the bob of the pendulum or the (Cartesian) velocity vector. For this reason, the table of properties of these two view elements need to include an action that will invoke the *_external.resetIC()* method so that Simulink will accept the changes to these variables. See Figure 11.

This is worth repeating: if we forget to include these calls to *_external.resetIC()*, interacting with either the bob or the velocity vector of the pendulum might temporarily change the angle and angular velocity of the pendulum, *but the corresponding initial conditions of the Simulink model won't be affected*.

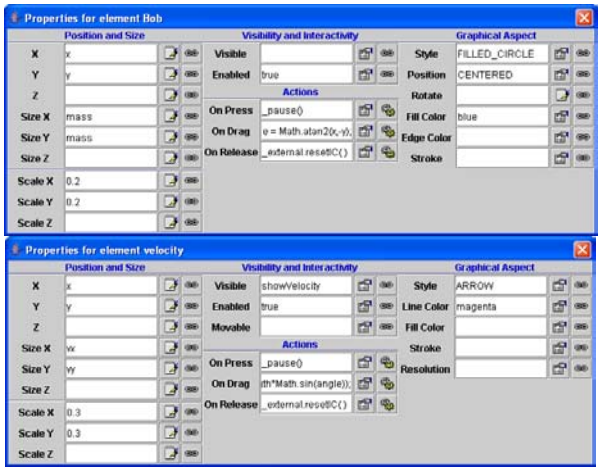


Fig. 11. Tables of properties for the bob and the velocity vector. *x, y, vx* and *vy* are auxiliary variables defined to translate the angular magnitudes to Cartesian coordinates.

VII. Running the Simulation

The simulation is ready to run. Clicking the *Run* icon in Ejs interface makes the miracle. The simulation is generated, compiled, compressed into a JAR file, and run as an independent process.[2] The execution of the simulation automatically launches Matlab/Simulink in the background so that Ejs can ask it to solve the model for it. The visualization and the user interaction appear on the screen and work as designed to provide the interactivity we wanted for our model.

It is out of the scope of this paper to describe the technical procedures that take place behind the scenes. Basically, what Ejs does is to launch Simulink, load the original model, and change it a little bit in order to provide the internal hooks for Ejs to communicate with the model through Matlab's workspace and to tightly control its execution. The communication takes place through a DLL library written in C and some utility Java classes that we created for this task, and is therefore very efficient. All this work is, however, completely transparent to the user.

VIII. Conclusions and Further Work

Adding interactivity to Simulink models can improve the pedagogical use of existing simulations in the field of control engineering education. Easy Java Simulations is a valuable tool not only to create simulations on its own, but also to cooperate with Simulink to bring to life previously existing models. We have tried to describe briefly how this cooperation takes place. With a handful of mouse clicks, authors can quickly connect Ejs and Simulink, combining the best of both worlds.

The mechanism described in this paper opens a whole world of applications in several fields of study. Our work in

[2] Although Ejs generates an applet for the simulation, security features embedded in current Web browsers do not allow simulations that use Matlab/Simulink to be run in applet mode.

the next future includes creating a number of virtual labs for automatic control based in this successful combination. We are also working to adapt our ideas and tools to control remote laboratories [11] [12], as well as to connect Ejs to other interesting tools, such as Sysquake™ [13].

There is still some technical work to be done, too. The tool has still some problems to communicate with models that include integrator blocks of subtypes other than the standard (and most frequently used) one. We are currently working to solve this problem.

Easy Java Simulations, the software tool, a complete English manual for it, and a document describing the connection of Ejs with Matlab and Simulink, can be downloaded for free from Ejs' Web server at http://fem.um.es/Ejs. The example described in this paper can be found in the standard distribution of Ejs version 3.4, in this server, in the subdirectory of the *Simulations* directory called *examples/ExternalApps/Simulink*.

REFERENCES

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

[2] S. Dormido, "Control learning: Present and future" IFAC Annual Control Reviews, vol. 28, pp 115-136, 2004

[3] J. Sánchez, S. Dormido, F. Esquembre. "The learning of control concepts using interactive tools". Computer Applications in Engineering Education (to appear).

[4] www.mathworks.com

[5] The MathWorks, Inc. ''Using Simulink'',Version 3. 1990-1999.

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

[7] S. Dormido, F. Esquembre, "The Quadruple-Tank Process: An Interactive Tool for Control Education", European Control Conference ECC'03, Cambridge (England), September 2003

[8] S. Dormido, C. Martín, R. Pastor, J. Sánchez, F. Esquembre, "Magnetic Levitation System: A Virtual Lab in Easy Java Simulation", American Control Conference ACC'04, Boston (USA), july 2004

[9] J. Sánchez; S. Dormido; R. Pastor; F. Esquembre , "Interactive learning of control concepts using Easy Java Simulations", Plenary Lecture, IFAC Workshop Internet Based Control Education IBCE'04, Grenoble (France), september 2004

[10] Easy Java Simulations' home page at http://fem.um.es/Ejs

[11] R. Pastor, C. Martín, J. Sánchez, S. Dormido. ''A Xml Framework Customization for a Servo Motor Web based Laboratory'', AIChE Annual Meeting, San Francisco (EEUU), November 2003

[12] R. Pastor, J. Sánchez, S. Dormido. ''Web-based virtual lab and remote experimentation using Easy Java Simulations'' (accepted), 16th IFAC World Congress, Praha, July 2005

[13] www.calerga.com