



Interactive Real-Time Control Labs with TrueTime and Easy Java Simulations

Gonzalo Farias¹, Karl-Erik Årzén², and Anton Cervin²

¹ Departamento de Informática y Automática, UNED, Spain,
gfarias@bec.uned.es

² Department of Automatic Control, Lund Institute of Technology, Sweden,
{karlerik, anton}@control.lth.se

Abstract. This paper presents the development of interactive real-time control labs using TrueTime and Easy Java Simulations. TrueTime is a freeware Matlab/Simulink based tool to simulate real-time control systems, and Easy Java Simulations allows rapid creation of interactive simulations in Java. Authors can use TrueTime to develop the simulation of a real-time control system, and then move to Easy Java Simulations to link the system and create the graphical user interface which provides the visualization and user interaction. The combination of these tools brings together the best of them.

1 Introduction

Control education has to adapt to the new scenario that the information technologies provide. In this context interactive virtual labs take advantage of these new possibilities and improve the understanding of the system behavior [1, 2]. The interactivity should allow 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. TrueTime is a MATLAB/Simulink based tool [3–5], which facilitates co-simulation of controller task execution in real-time kernels, network transmissions, and continuous plant dynamics. The tasks are processes that control continuous-time plants modeled as ordinary Simulink blocks. However, models created with Simulink suffer from a certain lack of interactivity in the sense described. A typical instructor would face difficulties if (s)he had to develop, using only Simulink, virtual labs with graphical and interactive capabilities. This is where Easy Java Simulations (Ejs) comes in handy. Ejs is a software tool designed to create simulations in Java with high-level graphical capabilities and with an increased degree of interactivity [6]. The paper is organized as follows. In Section 2 both tools TrueTime and Ejs are introduced. In Section 3 main aspects of the integration between TrueTime and Ejs are commented. Section 4 shows three examples of this approach. Finally, Section 5 presents the main conclusions and further work.

2 TrueTime And Easy Java Simulations

In this Section both TrueTime and Easy java Simulation (Ejs) are described briefly. TrueTime is a Matlab/Simulink based tool to simulate real-time systems, and Ejs is an authoring tool to create simulations in Java. Since Ejs has a link to reuse Simulink models, authors can create interactive real-time control labs combining both tools.

2.1 A Brief Overview Of 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 (see Fig. 1) and a collection of MEX files. The kernel block simulates a real-time kernel executing user-defined tasks and interrupt handlers. The various network blocks allow nodes (kernel blocks) to communicate over simulated wired or wireless networks. TrueTime can be downloaded from <http://www.control.lth.se/truetime/>.

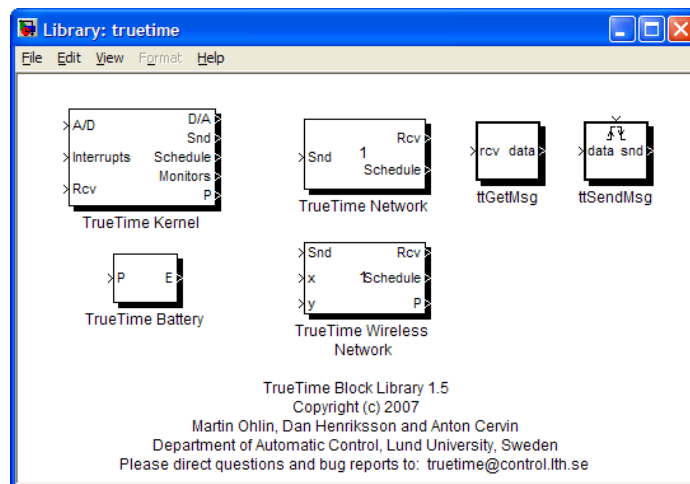


Fig. 1. The TrueTime 1.5 block library

To create real-time simulations TrueTime provides mainly two kinds of blocks, TrueTime Kernel and TrueTime Network. The TrueTime Kernel block simulates a computer node with a generic real-time kernel, A/D and D/A converters, and network interfaces. The block is configured via an initialization script. The script may be parameterized and the programmer may create objects such as tasks, timers, interrupt handlers, etc., representing the software executing in the computer node. The TrueTime Kernel block supports various pre-emptive scheduling

algorithms such as fixed-priority scheduling and earliest-deadline-first scheduling. It is also possible to specify a custom scheduling policy. The TrueTime Network block and the TrueTimeWireless Network block simulate the physical layer and the medium-access layer of various local-area networks. The types of networks supported are CSMA/CD (Ethernet), CSMA/AMP (CAN), Round Robin (Token Bus), FDMA, TDMA (TTP), Switched Ethernet, WLAN (802.11b), and ZigBee (802.15.4). The blocks only simulate the medium access (the scheduling), possible collisions or interference, and the point-to-point/ broadcast transmissions.

2.2 A Brief Overview of Easy Java Simulations

Ejs is an authoring tool designed for rapid creation of interactive simulations in Java. Ejs is different from most other authoring tools in that Ejs is not designed to make life easier for professional programmers, but has been conceived by science teachers, for science teachers and students. 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. Ejs can be downloaded from <http://www.um.es/fem/Ejs/>.

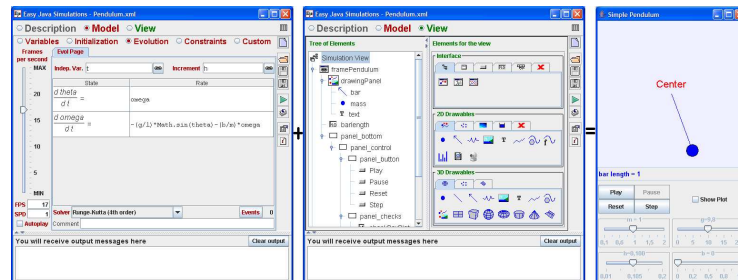


Fig. 2. Ejs uses the model and the view to create the simulations

Ejs structures a simulation in two main parts, the model and the view (see Fig. 2). The model can be described by means of Ordinary Differential Equations, Java code or external applications. 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 components to build a tree-like structure. 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 and on-the-fly visualization of the system.

3 Integration TrueTime and Easy Java Simulations

Ejs has an special link with Matlab/Simulink. In this case Ejs is used to create the graphical user interface and the TrueTime model to provide the system behavior.

3.1 Improving the Link

The connection with Simulink models allows users develop in an easy and fast way interactive simulations. The procedure, described in [7, 8], is quite simple. It basically consists in connecting Ejs' variables to signals (input, output or parameters) of the blocks in the Simulink model. Ejs also provides a set of predefined methods that allow the users read and write variables in the Matlab workspace, for instance to read the variable *myMatlabVar* from Matlab workspace the next instruction could be used: `myEjsVar=_external.getDouble("myMatlabVar");`

These methods are similar (same signature) to functions defined in the Matlab Engine Library. In this way the integration between TrueTime and Ejs is very direct, but to improve the performance of the initial results it was needed to consider two features of TrueTime simulations, i.e., zero crossings evaluations and scheduler data.

The first aspect is taken into account because the simulation with TrueTime involves a lot of zero-crossing functions, which produces slow, though good enough, simulations using the TrueTime Simulink blocks. However the simulations in the Ejs-TrueTime combination are even much slower. To solve this obstacle a new way to link Ejs and Simulink was created. Until now the updating of the Ejs' variables was done in every step of the Simulink solver, but with the new link the variables are updating at fixed time intervals, which involves a faster and "smooth" simulation. In the Fig. 3 more details are presented. To indicate this link the user just needs to specify the time interval in the External File option in Ejs (see Fig. 4).

The second feature considered was the schedule output signals generated by TrueTime. These signals are rather important to understand the performance of the real-time system, so it is necessary to catch all samples from schedule data. However, Ejs has a mechanism to avoid waiting a long time for a Matlab variable, so sometimes a few samples from schedulers were lost and the plot of these signals was not good. Hence the new method `_external.setWaitForEver(true)` was added in Ejs to wait until the sample is caught.

3.2 Creating the Interactive Simulations

After the improvement phase, the linking between TrueTime models and Ejs was direct, in this situation just two main activities were done, selecting the signals and connecting the global variables (see Fig. 4). The procedure to select the signals from TrueTime model is the same as in any other Simulink model, and is simple because the link between Ejs variables and Simulink signals is done by a single mouse click. The procedure to connect global variables and

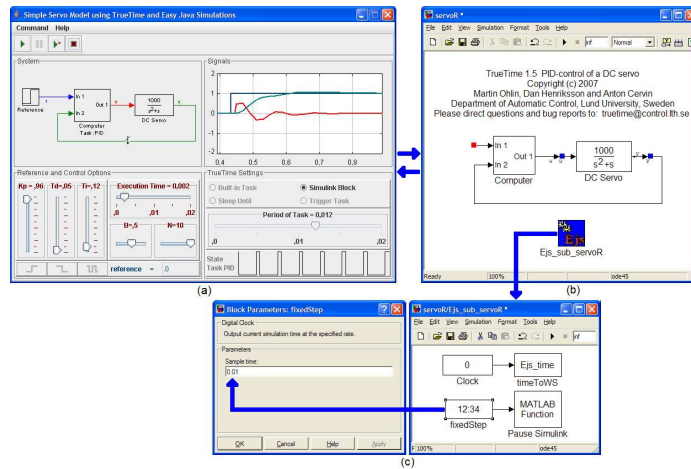


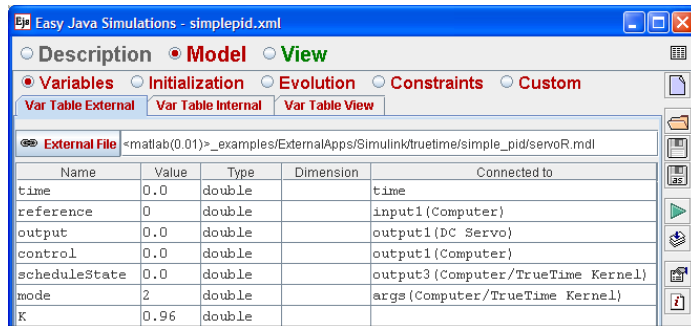
Fig. 3. Ejs simulations using a TrueTime model. The graphical user interface is shown in (a), the Simulink model is presented in (b), in (c) the submodel Ejs_sub_servoR shows the blocks to speed up the simulation. Note that the fixed time interval is equals to 0.01

Ejs variables requires more work, but it is also simple. In order to affect any variable in the TrueTime code functions, it just needed to declare them as global and use the read and write predefined methods in Ejs to set or get the values. An example of a modified code function is presented in Listing 1.1. Finally the method `_external.step()` is used in the Evolution section to simulate the Simulink model.

Listing 1.1. `pidcode1` function modified to set a link between Ejs and TrueTime. Commented lines are original lines of the function

```

function [exectimeAux , data]=pidcode1(seg , data_) % Ejs
% function [exectime , data]=pidcode1(seg , data)
global data exectime; % Ejs
switch seg,
case 1,
    r = ttAnalogIn(data.rChan); % Read reference
    y = ttAnalogIn(data.yChan); % Read process output
    data = pidcalc(data , r , y); % Calculate PID action
    exectimeAux=exectime; % exectime = 0.002*rand;
case 2,
    ttAnalogOut(data.uChan , data.u); % Control Signal
    exectimeAux = -1; % exectime = -1;
end
    
```



Easy Java Simulations - simplepid.xml

Description
 Model
 View

Variables
 Initialization
 Evolution
 Constraints
 Custom

Var Table External
 Var Table Internal
 Var Table View

External File <matlab(0.01)>_examples/ExternalApps/Simulink/truetime/simple_pid/servoR.mdl

Name	Value	Type	Dimension	Connected to
time	0.0	double		time
reference	0	double		input1(Computer)
output	0.0	double		output1(DC Servo)
control	0.0	double		output1(Computer)
scheduleState	0.0	double		output3(Computer/TrueTime Kernel)
mode	2	double		args(Computer/TrueTime Kernel)
K	0.96	double		

Fig. 4. Setting a link between Ejs and a TrueTime model, note the fixed time interval updating is used and the Ejs' variables are connected to inputs or outputs of Simulink's blocks

4 Examples

In this section three examples using the TrueTime-Ejs integration will be shown. The first example is an introduction to the TrueTime simulation environment. The second one is an extension of the first one. Finally, the third example uses TrueTime Network block to show a distributed control application.

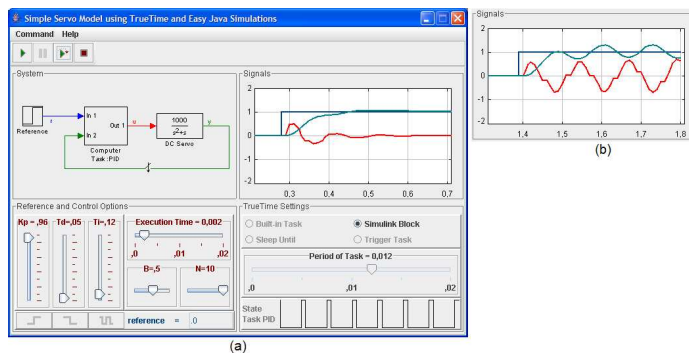


Fig. 5. (a) Graphical User Interface of the first example. Note that it is possible to change the control parameters on-the-fly using the sliders. (b) The Signals when Execution Time is incremented from 2[ms] to 9[ms]

4.1 First Example: The Simple PID Servo Controller

The example considers simple PID control of a DC-servo process. The process is controlled by a controller task implemented in a TrueTime kernel block. Four



Fig. 6. Graphical user interface of the second example. Figures: (a) is the main view, (b) show histograms of input-output latencies or response time for three tasks, (c) is the dialogCode where the user can read and write the code of PID computation. Note the schedule data of the red task and its control performance

different modes of implementation of the controller task are provided: Built-in Task, Simulink Block, Sleep Until and Trigger Task. The graphical user interface of the example is shown in Fig. 5. The user can modify different parameters like: reference type, control settings, execution time of the controller and the mode of the implementation of the tasks. In the Fig. 5b the performance of the controller is shown when the execution time change from 2[ms] to 9[ms].

4.2 Second Example: The Three Servo Controllers

This example extends the simple PID control 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 is shown in Fig. 6. Main view has the same appearance than the previous example. There is also an auxiliary dialog to show histograms of input-output latencies or response time of three tasks, and there is another dialog to enable the user to modify the code for calculate the PID action.

In the main view the user can modify the parameters for three controllers as in the first example, but also can add some jitter to the execution time. Note also the scheduler data for the tasks is presented in this window. The policies are chosen from radio buttons Rate Monotonic and Earliest Deadline First. Note

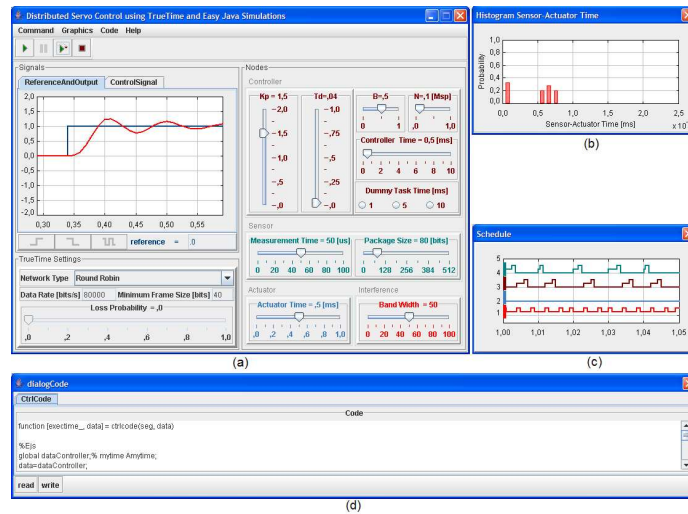


Fig. 7. Graphical user interface of the third example. Figures: (a) is the main view, (b) show histograms of sensor-actuator time, (c) is the scheduler data for the four nodes, (d) is the dialogCode where the user can read and write the code of controller

in Fig. 6b the input-output latencies for three task, since the rate monotonic policy was selected the task 1 (red) has the lowest priority and the performance is not good. The dialogCode can be used for instance to modify the algorithm to compute the PID controller.

4.3 Third Example: Distributed Servo Control

This example simulates distributed control of a DC-servo. The example contains four computer nodes, each represented by a TrueTime kernel block. A time-driven sensor node samples the process periodically and sends the samples over the network to the controller node. The control task in this node calculates the control signal and sends the result to the actuator node, where it is subsequently actuated. The simulation also involves an interfering node sending disturbing traffic over the network, and a disturbing high-priority task executing in the controller node. The Graphical User Interface is shown in Fig. 7. The main view allows the user to modify parameters of the network and nodes. There are also three auxiliary dialogs to show an histogram of end-to-end latency, to modify the code for the controller node, and to show the scheduler data for the four tasks.

In the main view the user can modify control parameters and also add a dummy disturbing high-priority task with different execution times. In the same view, but in the section Sensor the user can modify the measurement time and the package size. This last parameter is important to see the effect of the size of

the package in the control performance. The section BandWidth allows the user to increase bandwidth used by the interference node. The network parameters can be modified in the TrueTime Settings section.

5 Conclusions

Interactivity is crucial aspect in virtual labs that are to be used for pedagogical purposes in the field of control engineering.

In the context of simulation of real-time systems, TrueTime is a freeware that provides functionalities that simulates control models under resources constraints. However from a teaching point of view, and since True-Time is a Matlab/Simulink based tool, to create an interactive simulations using Matlab features could demand hard work. That's why Ejs was used.

Ejs is also a freeware tool designed to create quickly interactive simulations in Java. Besides, Ejs has a direct link with Matlab/Simulink models, and is not difficult to get a first version of the interactive lab after a few minutes. However it was necessary to improve the link in order to get more robust and faster simulations. That finally involved new features that made possible the acceleration of the simulation and prevented the loss of scheduler data.

Three examples of TrueTime models were developed. These allow the user to modify parameters like execution time, jitter, and see how the whole system is affected.

The creation of this kind of labs is not difficult, and probably in order to get a final version a couple of days is needed. However most of the time the designer will not worry about Java programming details, but about finding new functionalities to add to the lab.

Further work could involve further development of web-based labs or the creation of a link between Ejs and Scilab version of TrueTime.

References

1. Heck B. S. (editor): *Special report: Future directions in control education: IEEE Control Systems Magazine*, Vol. 19, No. 5, (1999) 35–58.
2. Dormido S.: *Control learning: Present and future: IFAC Annual Control Reviews*, Vol. 28, (2004), 115–136.
3. Ohlin M., Henriksson D., Cervin A.: *TrueTime 1.5 Reference Manual.: Manual*, Department of Automatic Control, Lund University, Sweden, (2007).
4. Cervin A., Henriksson D., Lincoln B., Eker J. and Årzén K.: *How does control timing affect performance?* IEEE Control Systems Magazine 23(3), 16–30,(2003).
5. Andersson M., Henriksson D., Cervin A. and Årzén K.: *Simulation of wireless networked control systems*, In Proceedings of the 44th IEEE Conference on Decision and Control and European Control Conference ECC (2005). Seville, Spain.
6. Esquembre F.: *Easy Java Simulations: A software tool to create scientific simulations in Java*, Comp. Phys. Comm. 156, (2004), 199–204.
7. Sánchez J., Dormido S., Esquembre F.: *The learning of control concepts using interactive tools*, Computer Applications in Engineering Education, Vol. 13, No 1, (2005) 84–98.

8. Dormido S., Esquembre F., Farias G., Sánchez J.: *Adding interactivity to existing Simulink models using Easy Java Simulations*, In Proceedings of the 44th IEEE Conference on Decision and Control and European Control Conference ECC (2005). Seville, Spain.