Proceedings
of the 4th International Modelica Conference,
Hamburg, March 7-8, 2005,
Gerhard Schmitz (editor)

G. Ferretti, M. Gritti, G. Magnani, G. Rizzi, P. Rocco
Politecnico di Milano, Italy

Real-Time Simulation of Modelica Models under Linux / RTAI

pp. 359-365

Paper presented at the 4th International Modelica Conference, March 7-8, 2005,
Hamburg University of Technology, Hamburg-Harburg, Germany,
organized by The Modelica Association and the Department of Thermodynamics, Hamburg University of Technology

All papers of this conference can be downloaded from
http://www.Modelica.org/events/Conference2005/

Program Committee

- Prof. Gerhard Schmitz, Hamburg University of Technology, Germany (Program chair).
- Prof. Bernhard Bachmann, University of Applied Sciences Bielefeld, Germany.
- Dr. Francesco Casella, Politecnico di Milano, Italy.
- Dr. Hilding Elmqvist, Dynasim AB, Sweden.
- Prof. Peter Fritzson, University of Linkping, Sweden
- Prof. Martin Otter, DLR, Germany
- Dr. Michael Tiller, Ford Motor Company, USA
- Dr. Hubertus Tummescheit, Scynamics HB, Sweden

Local Organization: Gerhard Schmitz, Katrin Prölß, Wilson Casas, Henning Knigge, Jens Vasel, Stefan Wischhusen, TuTech Innovation GmbH

Real-Time Simulation of Modelica Models under Linux / RTAI

Gianni Ferretti Marco Gritti Gianantonio Magnani Gianpaolo Rizzi Paolo Rocco
Politecnico di Milano, Dipartimento di Elettronica e Informazione
Via Ponzio 34/5, 20133 Milano

Abstract

The paper presents a concept and its implementation software modules to obtain ready to run real-time simulation code directly from Modelica models. Basically, a Modelica special model building block has been developed supporting the definition of the real-time input /output variables, their communication with external tasks or systems (e.g. a real hardware and software controller), and the scheduling of the periodic execution of the simulation task. The special module links to the real-time operating system (Linux with extension RTAI) through a special purpose C library. The real-time simulation of a 7-DOF space robotic arm is presented as a test case.

1 Introduction

Real-time simulation systems are mainly used for testing and check out of control electronics and other components of complex systems ("hardware-in-the-loop" simulation), like power plants, aircraft, vehicles, as well as for training of plant operators, aircraft pilots, and astronauts.
In real-time simulators model inputs must be acquired from external devices each sample time and model equations must be solved within fixed time intervals, so that a selected subset of computed variables can be output the next sampling time. To implement real-time communication with external world and to schedule model execution exactly each sampling time, the simulation software relies on system primitives whose calls are added to the model solution code. Usually, an effort is also necessary to simplify model equations most demanding from the point of view of computational burden.
Commercial tools exist that allow to adapt off-line models to real-time simulations on dedicated hardware. A typical situation consists in porting Simulink models to dedicated hardware using the Matlab Real Time Workshop [2] or the dSPACE TargetLink [6]. Simulink and also Dymola [1] models can be interfaced with dSPACE hardware to allow hardware in the loop simulations. Tools [6] exist also to assist the production of special Simulink models whose simulation can be run on multi-processor hardware.
On the other side, research efforts are spent to port the simulators obtained with open-source modeling tools like MBDyn [3] on real-time [5] possibly distributed platforms, like RTnet [4]. And also, efforts are spent to generate parallel code from Modelica models [10][11], to be eventually executed on supercomputer platforms [12].
This paper deals with the problem of obtaining ready to run real-time simulation code directly from Modelica models, so that already available models can be executed in real-time, and all the powerful Modelica libraries and the features and tools of a Modelica editor / compiler, such as Dymola, can be exploited for the development of new models.
The way this goal has been achieved is illustrated in this paper. In Section 2 the simulation platform requirements and main features are discussed. In Section 3 the proposed real time extension to Modelica is described. In Section 4 a test case of the real-time software modules is presented. A detailed open loop model of the Spider arm, a seven degrees of freedom Italian space manipulator, is exploited. The real-time simulated robot arm can also be controlled through a real-time software controller running on a second workstation. A short description of this software controller is given in Section 5.

2 The simulation platform

2.1 Requirements

The real-time dynamic simulation software should:

- satisfy the constraints of periodic real-time execution;
- be able to interface itself with external processes, possibly with hardware;

- be easily derived from models developed for the off-line simulation.

2.2 Real-time execution platform

The purpose of obtaining a real-time application imposes the choice of an operating system capable of supporting the execution of real-time processes.
The Linux operating system extended with the Real Time Application Interface (RTAI) [7] has been chosen. This operating system supports the execution of real-time processes, it is open-source, and it is widely used among the scientific community and in the European research centers.

2.3 Interoperability issues

Many custom libraries are available for Linux / RTAI. For the purposes of this project, the COMEDI (COntrol and MEasurement Device Interface) [8] library, developed by the open-source community, is particularly interesting. By means of a set of standard interfaces, COMEDI allows to manage the communication with hardware boards, and so provides a valid support for the data exchange on hardware channels. Actually, a device driver equipped with the COMEDI interface (see Section 5.2) has been exploited. This driver allows the access to an Ethernet board masking it as an acquisition board for analog and digital signals.

2.4 Real-time model generation

The dynamic models for the off-line simulation were developed in Dymola [1], a Modelica editor and compiler. In order to make as easy as possible the production of real-time models, it has been investigated the possibility of deriving the real-time models directly from the off-line models developed in Dymola. After having analyzed the features of Dymola, it has been found possible to use it also for the development and the compilation of real-time models. So, it is possible to both reuse the off line models by rapidly adapting them to the real-time simulation, and to build up some new model from scratch, with the advantage to use all the model libraries and the graphical instruments of such application.

2.5 Numerical integration issues

The features and the performances of the real-time numerical solvers available in Dymola have been analyzed in order to determine the most appropriate algorithm for the case study. It has been found that in order to obtain the maximum processing speed and to avoid the risk of a non convergent solution it is advisable to select the Inline Integration method applied to the Implicit Euler algorithm.

3 Real-time Modelica extension

3.1 Real-time components

A Modelica component has been developed that allows to transform any Modelica model into a model suitable for real-time simulation. This Modelica component, is a Modelica `block` that is called *ModRTAI*, and it can be used in the graphical user interface of Dymola as a normal building block of simulation models. ModRTAI uses a library of functions, called *LibRTAI*, which has been also developed within this project. LibRTAI has been entirely developed in the C language and contains the functions which allow the simulation code to access the RTAI and COMEDI libraries.
More precisely, by adding to the model development environment both a ModRTAI component and the LibRTAI library it is possible to perform the activities mentioned in the project requirements:

- Periodic execution at precise clock ticks of the simulation task, through access to the RTAI application programming interface;
- Management of the communication with the external world, through access to the interface of a COMEDI driver.

3.2 Porting a model to real-time

By means of the two modules which have been developed it is possible to obtain a Modelica model suitable for real-time simulation under Linux / RTAI. Given an off-line Modelica model, the ModRTAI block allows to select the signals which are input to the model (to be acquired from the external world), and the signals which are the output of the model (to be sent to the external world). The ModRTAI block also allows to set a parameter that states the frequency of the periodic task running the simulation. This parameter indicates how much time is left to the numerical solver to evaluate the transient related to one sampling interval. Only if such parameter is equal to the sampling interval, the simulation time should match the physical time elapsed since the instant in which the simulation itself was started; otherwise the simulation time develops slower than the physical time of a factor equal to the ratio between the

task scheduling period and the sampling step of the transient.
The Modelica model has to be compiled with Dymola under Linux, having selected the Inline Integration option and the Implicit Euler as integration algorithm. The functions library LibRTAI should be installed on the workstation onto which the model is compiled. The simulator which is generated in such way is called Dymosim, as a normal simulation binary file obtained by means of compiling with Dymola a Modelica model. The only difference is that the Dymosim binary obtained in such way is runnable in soft[1] real-time under Linux / RTAI. Figure 1 resumes the steps for porting a Dymola model into real-time.

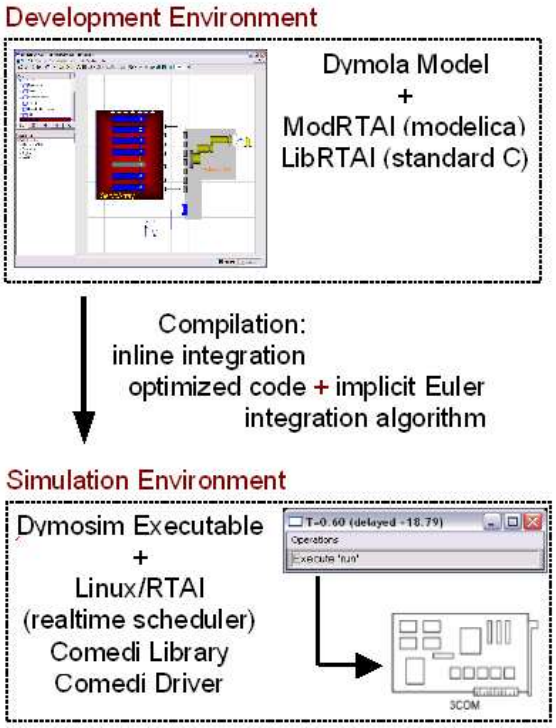

Figure 1: Development of a real-time Modelica model using Dymola

In order to allow the data exchange with external processes, the Dymosim simulator should be run on a workstation on which the COMEDI library is installed, and the drivers of the peripheral board onto which should flow the data signals in input and output to the simulator are installed too. By now, the simulator has been tested using an Ethernet communication board, which directs the signals to another workstation. The Ethernet board, equipped with drivers with a COMEDI interface, emulates a data acquisition board, as described in Section 5.2.
The entire procedure has been tested using a model of the Spider robotic arm with detailed descriptions of motors and transmissions (see Section 4), which was developed in a former research [14][13]. In this model the outputs are the seven motor positions, while in input the model receives the seven motor current set-points, and a digital signal controlling the brakes that in the home position block the motors axes. Experimental results about the performance of the Dymosim real-time simulator obtained with this model will be illustrated in Section 4.

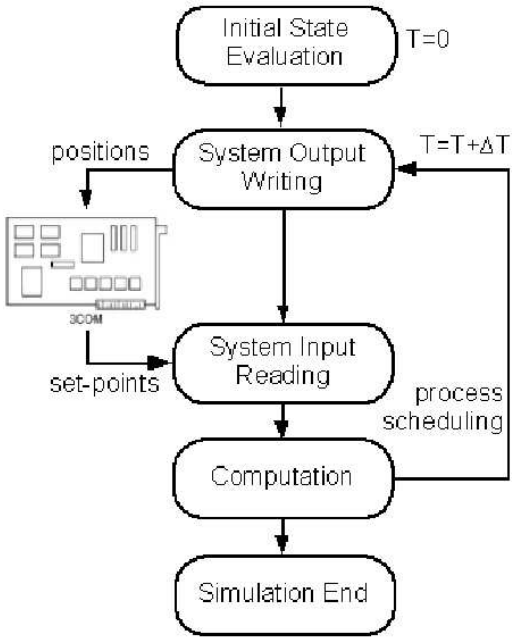

Figure 2: The simulation process execution cycle.

3.3 The simulator process

The real-time execution of the simulation code has the purpose of computing, at the chosen frequency, the solution that represents the evolution of the model state. The output of the model should be passed to an external control unit, which could be another workstation running the control procedure, or a dedicated hardware controller. The control system, on the basis of the received data, computes a control action that is sent to the workstation on which the real-time simulator is running. The simulator evaluates the new model state on the basis of the received input. This elementary cycle, illustrated in Figure 2, is executed till the end of the simulation.

[1] In Linux / RTAI a *hard* real-time process runs in the Linux kernel space and has more strict timing constraints, while a *soft* real-time process runs in the Linux user space and has more loose timing constraints. In Section 4 it is explained why the soft real-time solution has been preferred.

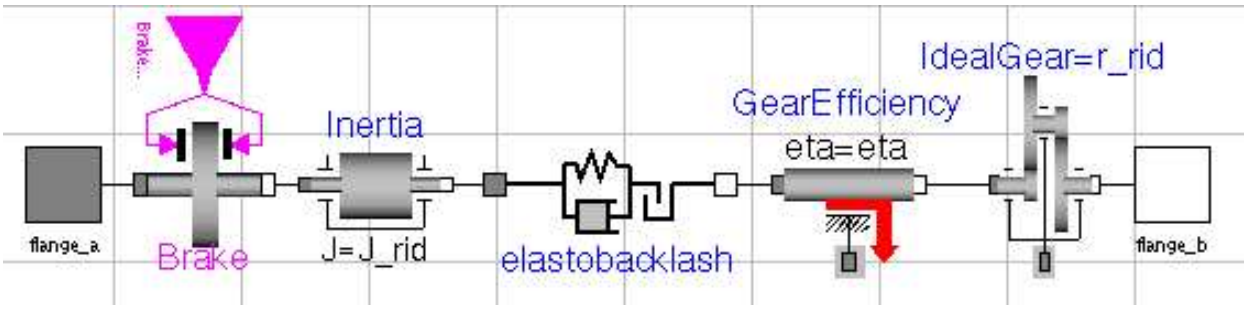

Figure 3: Modelica scheme of the transmission of the real-time Spider robot model.

At every wakeup of the periodic process, the simulator has to evaluate within a maximum time span[2] the next model status, and the corresponding outputs. This time span is equal to the scheduling period of the simulator process. Since the control loop should be closed on a digital (either hardware or software) regulator, it is mandatory that at any wakeup the simulator process is fed with inputs sampled at a constant frequency, and in the same way it yields as output some signal sampled at a constant frequency. This leads to the necessity to use a fixed step algorithm for the numerical integration, just as the Implicit Euler algorithm is.

As it has been said in Section 3, in order to synchronize with real-time a standard Dymosim process, and in order to exchange the model inputs and outputs with the external world, some calls to the external C functions implemented in the LibRTAI have been added in the ModRTAI block. These functions are:

- `RTAIGetInputSample(...)`
- `RTAIGetData(...)`
- `RTAIPutOutputSample(...)`

At every step, when the numerical solver tries to solve all the equations of the model, these three C functions are called in this order. The first and the third ones assign the external input variables to the vector of model inputs, and the vector of model outputs to the external output variables. By means of these two functions, the data are only put in some internal buffer of the LibRTAI module. Instead, the *real* data exchange with the external process / hardware is done by `RTAIGetData(...)`.

[2]There is no way to set a constraint on a Dymola numerical integrator forcing it to yield a result at every step within a physical maximum time. The missed deadline is checked by comparing the theoretical time in which the result should have been yield with the time in which the Dymosim process has actually released the control.

If `initial()` or `terminal()` are true in the Modelica code, `RTAIGetData(...)` respectively performs the initialization or the finalization of the the real-time process associated with Dymosim; otherwise such function performs the real data exchange and then it suspends the process.

During the initialization, the RTAI `rt_task_init(...)` function is called to initialize a new RTAI task, and the RTAI `rt_task_make_periodic(...)` function is called to make this task a periodic one. Other RTAI APIs are called to set the real-time scheduler, and to initialize and start the RTAI timer at the chosen frequency. During the finalization, the `rt_task_delete(...)` function is called, and the RTAI timer is stop. During a normal periodic call, the RTAI `rt_task_wait_period()` is called at the end of `RTAIGetData(...)` in order to suspend the periodic task.

4 Experimental results

As it has been said in Section 3.2, the proposed system has been tested on a detailed model of the Spider robotic arm [13][14] consisting of a total of more than 12,000 equations listed at compile time. The robot model includes a seven degrees of freedom mechanical chain, built with the old Modelica MultiBody library, and an array of seven servomechanisms, each of which featuring the dynamics of a brushless two-phase motor, an analog current controller, an elastic transmission with backlash and a brake on the motor axis. A detailed scheme of the elastic transmission used in the servomechanisms is shown in Figure 3.

The tests have been done running the simulation on a workstation, and a control action playback on another workstation. More precisely, the control action was not computed during the simulation, but it was a record of the control action of a simulated control system dur-

ing an off-line simulation of the same command given to the robot. The real-time model has been tested on a 3GHz Pentium IV workstation with 512Kb of 2^{nd} level cache. This processor ensures enough computational power to simulate the model with an integration step of $1ms$[3] and a wakeup period of $1.5ms$ for the corresponding scheduled process.
It has been experienced an average of 1% of faults, i.e. periods in which the simulator has not been able to evaluate the corresponding model transient. The graph in Figure 4 shows the CPU time not used by the simulator at each period, during the first 2 seconds of a simulation. A `move` command was simulated, and the command execution was started at $1.1s$, which explains why at this instant the free CPU time dramatically decreases.

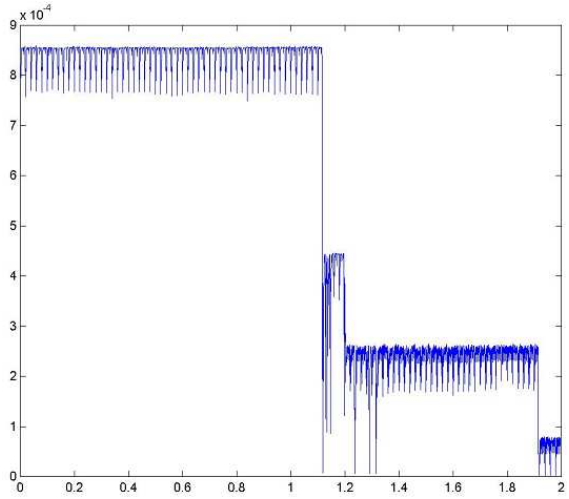

Figure 4: CPU idle time during simulation. Simulation time (in seconds) on x-axis; CPU idle time (in seconds) on y-axis.

This result proves that the model is still not ready to be simulated in a time equal to the physical time on the testing workstation, since the transient computation occupies half of the CPU time if the robot is still, but practically all the CPU time if the robot is moving. In order to reach the purpose to simulate with a scheduling period of $1ms$, it is mandatory to simplify the model, or to use a more powerful workstation.
The graph in Figure 5 shows the physical time span between two subsequent process wakeups in the same simulation as before. Figure 6 shows a detail of the graph in Figure 5. The maximum variation of the wakeup period is of 40%, while the average variation is of 1.34%. The picks of variation in the scheduling period are a consequence of the beginning of large transients, due to the motion start, but the continuous period variation during the movement and also before the beginning of the movement, are due to the soft real-time nature of the chosen scheduler.

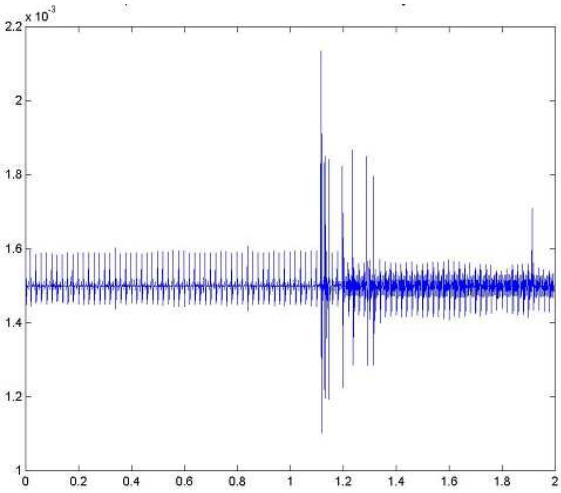

Figure 5: Dymosim task scheduling interval. Simulation time (in seconds) on x-axis; scheduling interval (in seconds) on y-axis.

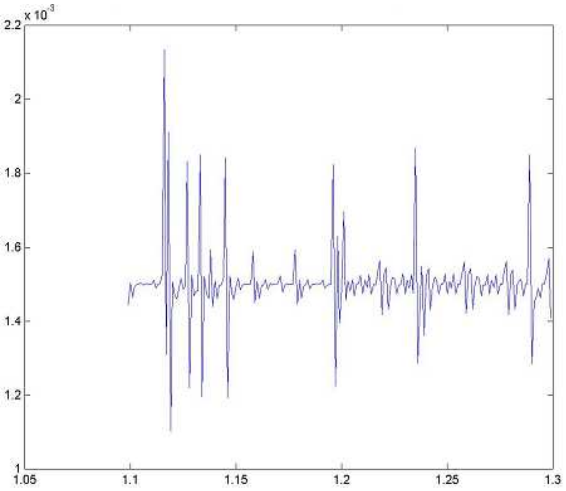

Figure 6: Dymosim task scheduling interval: detail. Simulation time (in seconds) on x-axis; scheduling interval (in seconds) on y-axis.

The choice of a soft real-time scheduler has been imposed by the nature of the executable binary code generated by Dymola. In fact, the Dymosim native code executes some operating system calls that do not exploit the RTAI API[4]. The OS calls of Dymosim impose a continual switch from a real-time context to a non-

[3] This is equal to the sampling frequency of a typical axis control cycle of an industrial robot controller.

[4] For example, all mathematical functions in the numerical integrator do not call the RTAI API.

realtime context. If the simulator is declared hard real-time at process startup, the continual context switch is from hard real-time to non-realtime, while if the simulator is declared soft real-time at process startup, the continual context switch is from soft real-time to non-realtime. The first kind of switch is much more time consuming than the second kind, and leads to worse performances.

5 Real-time control

5.1 The software control application

Within the same project framework, a real-time software control system [16] has been developed which emulates several functionalities of robot controllers. This control application can be easily adapted to interact with simulations of robotic arms driven at joint level. Thanks to the adoption of COMEDI drivers (which have standard interfaces, as stated in section Section 5.2), the control system can control without distinction a physical system, or a model based simulation of the system itself, provided that the two systems have the same number of input and output channels, disposed in the same order.

The software control system can be coupled with the real-time simulator, and each one of these two applications can be used as test bench when adding new features to the other one. So, the software control system can be used to test new and more refined robot models, and to analyze their behavior, if compared to the behavior of the corresponding real robot, while the real-time simulator can be used to test some innovating control solutions, without taking the risk of damaging the robot hardware.

The Linux / RTAI operating system has been chosen for the control application too, for the same reasons explained in Section 2.2. To support the design of the control application, the OROCOS (Open RObot COntrol Software) [9] framework has been chosen. The control application is by now capable of executing the position control in joint space for a six[5] degrees of freedom robot. The OROCOS control application can execute a standard control cycle, with signals exchanged in Real-Time with the controlled system; moreover, it can publish the variables internal to the controller and the signals received from the controlled process. Internal variables are published to non-realtime applications external to the controller, for reporting purposes. Also, the OROCOS control application can accept the robot motion commands from a program script, or from some external non-realtime application.

[5]Due to this limitation, the cross tests with the Spider model have been done blocking the seventh joint of the robot.

5.2 The closed-loop data acquisition

Both the control and the simulation applications should be able to transmit and to acquire signals on a hardware communication channel. In order to make any application unaware of the presence of hardware or software on the other side of the control loop it has been decided to implement COMEDI drivers for the communication boards. The COMEDI package has been chosen because it is an open-source product widely used in the field of automation. Indeed COMEDI provides a standard for drivers of DAQ (Digital AcQuisition boards) under Linux.

A COMEDI driver for 3COM 3C90x(B) [15] Ethernet boards and a COMEDI driver for COMAU BIT3 AT CARD [15] boards of the COMAU C3G-9000 controller have been developed. Both boards are accessible from real-time processes: the first one is used by now for the data exchange between the OROCOS control system and the Dymosim simulated process, while the second one (whose driver is still in a test phase) will be used for the data exchange with all robots supported by the COMAU C3G-9000 controller.

6 Conclusions and future work

A design concept and the related implementation software for obtaining real time simulation code from standard Modelica / Dymola models and related software has been presented. They permit to develop ready to run real time simulation code by fully exploiting the powerful libraries and tools that Modelica / Dymola make available for the model development phase. The real time simulation of a detailed model of a 7-DOF space arm has been afforded as a test bench for the software, and has proved its versatility and correctness.

With reference to the class of mechatronic systems models, additional work has to be spent to speed up model execution by refining or simplifying models of those phenomena that most affect the computational burden, like, for instance, non linear friction at low speed. Indeed, while the model exploited for the actual tests can be simulated in a time which is of the same order of magnitude of physical time on a high-end mono-processor system, models including friction

equations are much slower and cannot be proposed for the purposes of real-time simulations. An alternative approach would be to move on a multi-processor platform, provided that a Modelica compiler tool for parallel code generation is adopted.

Acknowledgments

The authors would like to thank the Italian Space Agency and the Ministry of Instruction, University, and Research for the support to this research, in relation to the I/R/217/02 ASI contract and the OASYS project.

References

[1] Dymola Multi-Engineering Modeling and Simulation [Online]. Available: http://www.dynasim.se/

[2] Real-Time Workshop: Generate optimized, portable, and customizable code from Simulink models [Online]. Available: http://www.mathworks.com/products/rtw/

[3] MBDyn - MultiBody Dynamics Software [Online]. Available: http://www.aero.polimi.it/~mbdyn/

[4] RTnet - Hard Real-Time Networking for Linux / RTAI [Online]. Available: http://www.rts.uni-hannover.de/rtnet/

[5] Attolico M, Masarati P. A Multibody User-Space Hard Real-Time Environment for the Simulation of Space Robots. In Proceedings of the 5^{th} Real-Time Linux Workshop 2003, Valencia, Spain, Real-Time Linux Foundation, November 9-11, 2003.

[6] dSPACE - Solutions for Control [Online]. Available: http://www.dspaceinc.com/

[7] Cloutier P, Mantegazza P, Papacharalambous S, Soanes I, Hughes S, Yaghmour K: DIAPM-RTAI Position Paper. In Proceedings of the 2^{nd} Real-Time Linux Workshop 2000, Orlando, Florida, USA, Real-Time Linux Foundation, 27-30 November, 2000.

[8] COMEDI - The Linux Control and Measurment Device Interface [Online]. Available: http://www.comedi.org/

[9] OROCOS - Open Robot Control Software [Online]. Available: http://www.orocos.org/

[10] Aronsson P, Fritzson P. Parallel Code Generation in MathModelica / An Object Oriented Component Based Simulation Environment (Conference paper). In Proceedings of the Parallel / High Performance Object-Oriented Scientific Computing Workshop, POOSC01 at OOPSLA01, Tampa Bay, Florida, USA, 14-18 October, 2001.

[11] Aronsson P, Fritzson P. Multiprocessor Scheduling of Simulation Code from Modelica Models (Conference paper) in Proceedings of the 2^{nd} International Modelica Conference, DLR, Oberpfaffenhofen, Germany, Modelica Association, 18-19 March 2002.

[12] Nyström K, Aronsson P, Fritzson P. GridModelica - A Modeling and Simulation Framework for the Grid (Conference paper). In Proceedings of the 45^{th} Conference on Simulation and Modelling, Copenhagen, Danemark, Scandinavian Simulation Society, 23-24 September 2004.

[13] Ferretti G, Gritti M, Magnani G, Rocco P, Viganò L. Object-Oriented Modeling of a Space Robotic Manipulator. In Proceedings of the 8^{th} ESA Workshop on Advanced Space Technologies for Robotics and Automation 2004, Noordwijk, The Netherlands, ESTEC, 2-4 November 2004.

[14] Viganò L. Modellistica del Braccio Robotico Europa con Analisi del Controllo nello Spazio Operativo [in Italian]. Milano, Italy: Master's Thesis, Politecnico di Milano, 2003.

[15] Minazzi P. Sviluppo di Driver COMEDI in Ambiente Linux / RTAI [in Italian]. Milano, Italy: Master's Thesis, Politecnico di Milano, 2004.

[16] Cappellini S, Consonni A. Progetto e Realizzazione di un'Applicazione Real-Time a Componenti per il Controllo di Robot Manipolatori [in Italian]. Milano, Italy: Master's Thesis, Politecnico di Milano, 2004.