Proceedings
of the 3rd International Modelica Conference,
Linköping, November 3-4, 2003,
Peter Fritzson (editor)

Gianni Ferretti, Marco Gritti, Gianantonio Magnani, Paolo Rocco,
Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy:

A Remote User Interface to Modelica Robot Models

pp. 231-240

A Remote User Interface to Modelica Robot Models

G. Ferretti, M. Gritti, G. Magnani, and P. Rocco
Politecnico di Milano - Dipartimento di Elettronica e Informazione
Via Ponzio 34/5, 20133 Milano (Italy)
phone: +39 02 2399 3673, fax: +39 02 2399 3412
e-mail: {ferretti, gritti, magnani, rocco}@elet.polimi.it

Abstract

A remote interface for simulating robots via the Internet is presented. This environment is dedicated to simulations of robotic arms whose models are written in some descriptive language like Modelica. The actual application runs on robot models compiled by Dymola. This work has been developed within a project whose purpose is the Modelica/Dymola implementation of models of space robots and servomechanisms, and their dissemination to partner firms and to the scientific community. The remote simulation environment has been tested with a model of the Spider robotic arm.

1 Introduction

The purpose of the SIMECS[1] (Integrated MEChatronic Simulation for Space systems) project is to build up a library of models of mechatronic components used in space systems. Such components are building blocks for virtual prototyping robots and systems which will be exploited in future space missions.

Typical simulation environments like Dymola [1] or Simulink [2] embed powerful modeling toolboxes for building models of any kind of dynamic system. This gives the user the maximum flexibility and applicability. But once a model has already been built and tested by an expert user of such tools, only a user friendly simulation software is needed, while modeling functionalities become superfluous. That could be happening during the fine tuning of the model parameters, or while inspecting the behavior of the system for programming the sequence of working actions.

Moreover, there are scenarios in which the modeling expert and the ultimate simulation user not only are distinct persons, but also work in different places for distinct organizations that cooperate on the same project. This happens in SIMECS. Within this project, extremely complex models of robotic components were developed as academic research topic; then component models were assembled into models of robot[2] prototypes used as case studies. Working robot models will be finally put at disposal of partner firms which have to build the various parts of one real robot.

The presented scenario is not isolated, because many little establishments exist that do not have the expertise for developing complex models using complex modeling environments, and cannot afford to invest in it, but could improve the quality of their production by means of studying on pre-packed models with *easy-to-use* software tools.

The approach of running simulations remotely, instead of deploying the compiled models on site, is supported by the assumption that who builds the models should also be capable of dimensioning them to the computational power available for simulation. On the contrary, deploying models to the unknown computers of the end users would entail the risk that their computational power could be inadequate for running the simulations in a reasonable time, which would mean to make the deployed models practically unusable.

2 Application Requirements

The SIMECS-RI (SIMECS-Remote Interface) is a visual simulation environment dedicated to simulations of executions of commands assigned to robotic arms which operate in space. Requirements of such an application have been traced from different perspectives.

[1]The project website is online at `http://www.elet.polimi.it/res/simecs/`. The latest version of the client of the application which is described in this paper can also be downloaded there.

[2]As it will be said in section 5, the robot taken into account is the Spider arm, for which different control system solutions were tested by means of a number of simulations.

2.1 Functional Requirements

The application should be flexible enough to deal with any kind of manipulator. This means it should be able to present information about any kind of mechanical chain and motion control system, and it should be also able to initialize and run simulations of robots which accept commands at different abstraction levels.
Information to be presented about the robot model can be divided into three categories:

- Robot model documentation;
- Model parameters and main variables;
- Virtual model of the robot.

The model documentation could only be read and no change to the model should be allowed. This means, for example, that neither links could be attached to or removed from the mechanical chain, nor dynamic or algebraic blocks could be attached to or removed from the control system. The simulation environment should instead allow the user to change the values of the model parameters, such as all the gains in the motion control system, or the masses and lengths of the physical links. The application should possibly remember parameter value changes after the simulation, for further reutilization. Also the main variables of the model should be accessible, and the user should be allowed to plot variables, for transient analysis. Possibly, variable plots should be visible during the simulation also and updated while the simulation is in progress. Virtual prototype animation should be allowed after the simulation, and possibly also while simulation is in progress.
Depending on the model inputs, robot commands that one could assign for simulation are:

$$\text{Direct input} \begin{cases} \text{Open loop: current amplitudes,} \\ \text{Closed loop: canonical set points;} \end{cases}$$

$$\text{Movement} \begin{cases} \text{Joint space,} \\ \text{Cartesian space;} \end{cases}$$

$$\text{High level} \begin{cases} \text{Action,} \\ \text{Task,} \\ \text{Mission.} \end{cases}$$

For each of these categories, a user friendly way for selecting the actual command should be provided.
Finally, the application should also feature an easy way for choosing the initial robot configuration.

2.2 Operational Requirements

Basically, an application like SIMECS-RI should allow the user to choose a robot model from a library of available models, to assign model parameters, and arbitrary initial state, to define assign and simulate the execution of a robot command, and finally to analyze the simulation results.
The precise steps of a typical use case are:

1. choice and loading of the robot model;
2. *model inspection and parameters configuration;*
3. choice of the variables which values should be updated during the simulation;
4. *choice of the initial state;*
5. *robot command definition;*
6. simulation parameters setup;
7. *robot command assignment;*
8. simulation, with online result presentation;
9. loading of the transients of the variables which trend has to be analyzed;
10. variables transient analysis;
11. result saving.

Items that have been emphasized constitute functionalities which should be given special attention during the design phase. These features should be designed in a way that enhances the usability of the robot simulation interface. On the contrary all other functionalities are not affected by the requirement of building an application dedicated to robotic arms.
Documentation could be loaded together with the robot model, and presented simply as a collection of data sheets of the robot mechanical chain and control system. Hyper textual format could be adequate, so the user would navigate inside and outside of robot components and subcomponents, seeing the component connections and relationships, with all model equations explained, and all constant values listed.
Parameters should be well separated from the read-only documentation. They should be presented in a clear way: possibly a list, or maybe a tree in which those belonging to the same component are grouped together under the same branch. Parameters should be also coupled with their descriptions.

The robot virtual prototype should be displayed in a separate window; possibly, the 3D robot model should be surrounded by a model of its real environment. Solid shapes are preferable to wire frames. Changes in parameter values which affect the robot appearance should be reflected immediately in the virtual prototype.

Before performing the simulation, robot initial position should be assigned by choosing the positions of its joints. It could be assumed that the robot is always initially still. This restriction is consistent with the fact that when a command is assigned to a real robot, the robot is always still.

Concerning command definition, if the restriction of dealing only with robotic arms is adopted, robot commands belong to well known categories. So commands and trajectory parameters can be separated from other parameters, and the way a command is issued can be differentiated from a simple change in some input parameter. Robot commands should be edited in a comfortable manner. A visual tool emulating one distinct teach pendant for each level of abstraction is preferable with respect to a textual prompt where each command is issued by entering a line of text.

Finally, robot command assignment could simply coincide with simulation startup.

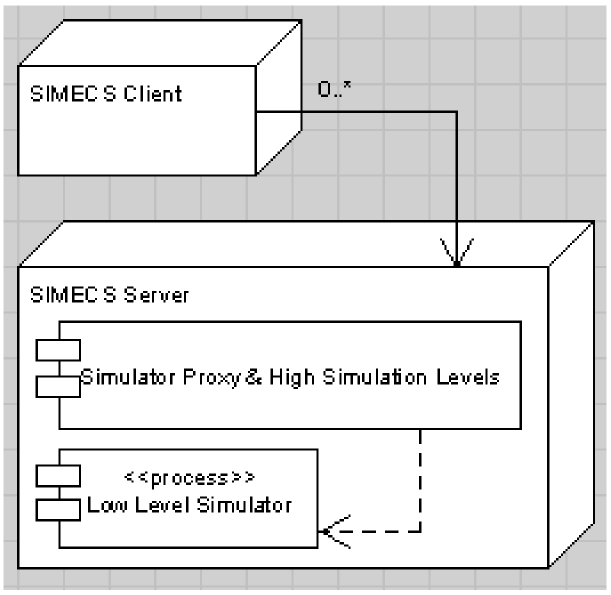


Figure 1: Application overall structure.

2.3 Structural Requirements

The structural requirements of the application are mainly two: first, as said in section 1, simulations have to run on a server computer, while simulation results should be presented on a client computer; second, simulations have to be performed by an application which is external to the SIMECS-RI server.

The choice of exploiting an external software tool for dynamic simulation is motivated by the fact that several tools [1][2][3] already exist which are capable of that, and so it would be anachronistic to start by now the development of a new one. On the other hand, since exploiting an already existing simulation tool means to exchange with it information about robot models and model inputs, most important would be now to compensate the lack of standards in the way robotic systems are defined. This would simplify the interaction with such generic simulation tools.

3 Application Structure

Te SIMECS-RI overall structure is shown in figure 1 through a UML [4][5] deployment diagram. This is a pure client/server strucure. The diagram emphasizes the fact that the application core is by now only a proxy to a distinct process which performs the simulations.

The server can handle multiple connections, which means that many users at a time can perform their own simulation. The maximum number of contemporary users is limited by the computational power of the server, which should be capable of running one simulation per user.

3.1 Server Structure

The structure of the server is shown in figure 2 through an UML component diagram. As it can be seen, a component based architecture has been adopted. The three main components of the application are the Simulation Server, the Low Level Proxy, and the Hot Feedback Manager. The Low Level Simulator Machine, which is the process that actually simulates the robot motion and motion control, does not belong to the application itself. Both the Low Level Simulator Machine and SIMECS-RI have their own configuration files. The last component, i.e. the Action Level Simulator is not currently implemented, and will be useful for future extensions which will be illustrated in section 6.

The Simulation Server waits for requests incoming from the client, and parses and executes them once they arrive. To serve the requests, the Simulation Server relies on the interfaces that the Low Level Proxy exports. The client of the Simulation Server

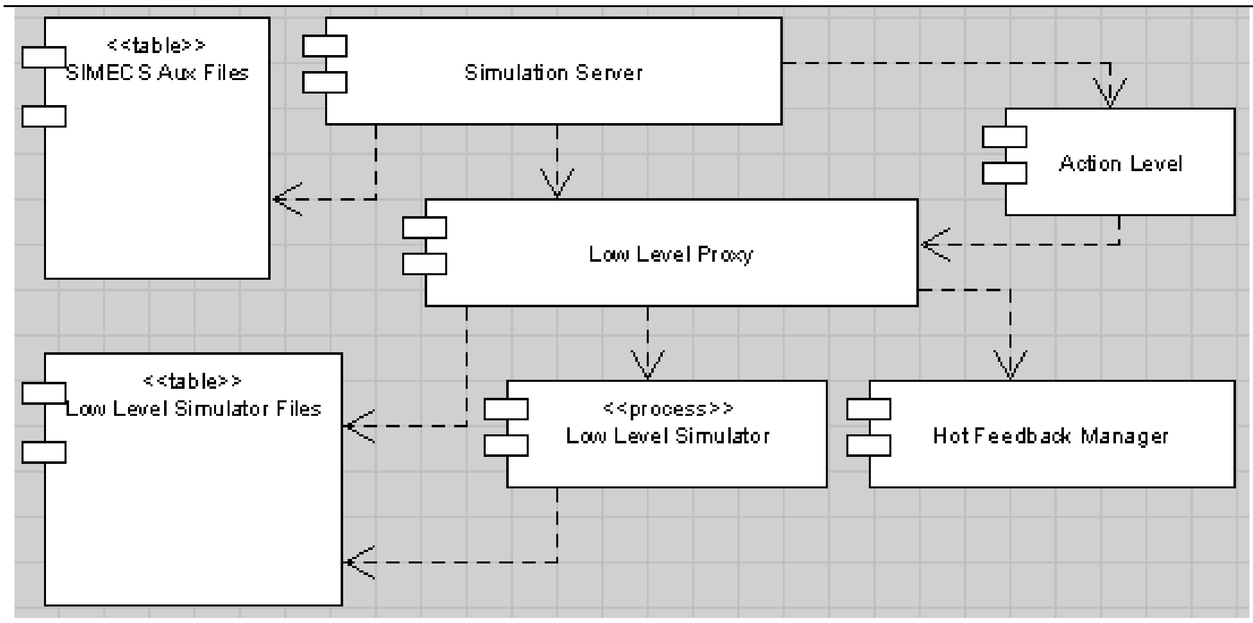


Figure 2: Server structure.

is a component of the SIMECS-RI client. These two physical components together constitute a conceptual *Communication* component.

This design has been adopted because it is general, and does not obstruct any kind of possible future development. A client/server mechanism has been adopted in spite of a remote procedure call mechanism for decoupling the two main parts of the application. Indeed, a communication layer based on an application-level protocol allows the client and the server to be implemented by means of different technologies. This also allows any one to implement a client for the SIMECS-RI, provided that he respects the communication protocol. On the other hand, abstracting a conceptual communication component allows at any step of the development process to change the way communication is performed, without affecting any other part of the application. This simplifies the way both the communication protocol and the communication mechanism can be changed, if this is considered useful.

The Low Level Proxy exports some interfaces used for gathering the robot model and the simulation results from the files of the external simulator, and some other interfaces used for controlling the external simulator itself. Building a component dedicated to the translation of simulator independent commands into simulator dependant ones has been useful during the design phase because it allowed to abstract from their implementation in the exploited simulator. A separate Low Level Proxy component is useful for the maintenance of the application also, since it allows to more easily change the external simulator by means of changing the proxy, and letting the rest of the application unchanged.

Finally, the Hot Feedback Manager watches for the results (i.e. the transient of the variables) incoming from the simulator while a simulation is in progress, and outputs these results to the client, so they can be immediately presented to the user. Results are gathered by the Low Level Proxy from the simulator through some mechanism of interprocess communication, and fired as events to the Hot Feedback Manager. Event paradigm has been adopted since result arriving is unpredictable. The Hot Feedback Manager can reduce the sampling rate of the results that are forwarded. Not exceeding a prefixed forward rate prevents the communication channel from saturations. If a user wants to analyze the entire transient of a variable he can in any case reload it when the simulation is terminated.

3.2 Client Structure

The structure of the client is shown in figure 4 through a UML component diagram. The two main components of the client are the User Interface and the Simulation Client. The event paradigm has been adopted for exchanging information between them. This means that as soon as the User Interface receives a command from the user as a signal coming from the computer hardware, whenever this command implies a request to the SIMECS server, the User Interface itself propagates the signal as an event to the Simulation Client.

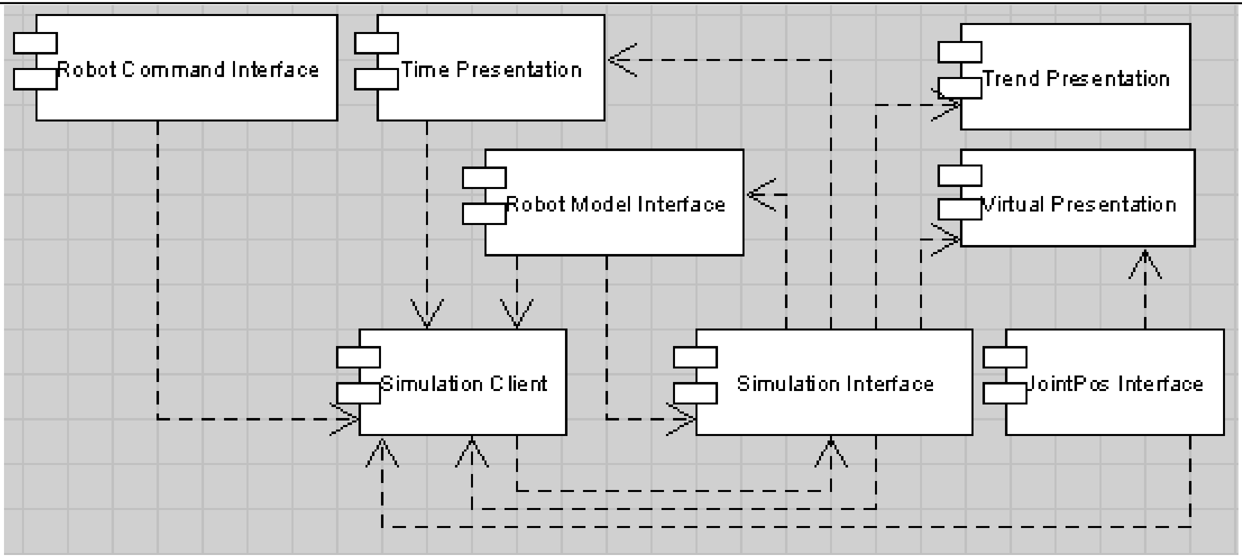


Figure 3: User interface structure.

Also when the Simulation Client reads results coming from the server, it fires events for propagating data to the User Interface.

The Simulation Client component is the counterpart of the Simulation Server component of the SIMECS-RI server. Its purposes are to translate the requests of the user into strings of the communication protocol and to dispatch the results coming from the SIMECS-RI server.

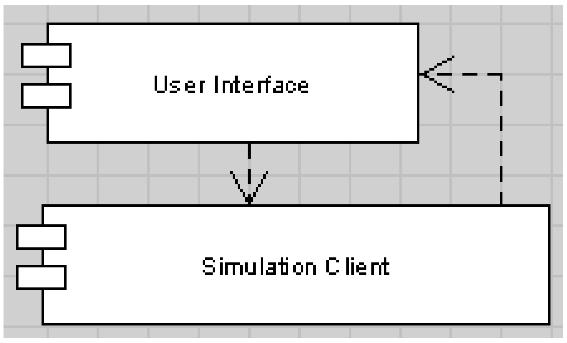


Figure 4: Client structure.

The User Interface is a super-component. Figure 3 shows the User Interface decomposed into its sub-components. For assuring the maximum decoupling, their interconnections also are asynchronous. The main sub-component is the Simulation Interface; other sub-components are the Joint Position Interface, the Robot Command Interface, the Robot Model Interface, the Time Presentation, the Trend Presentation, and the Virtual Presentation.

The Simulation Interface is responsible of instantiating all other visual sub-components and of assigning them a place on screen. Visual components are dedicated to a specific type of interaction with the user. The Joint Position Interface allows the user to drive the joints of the robot into a desired position. The Robot Command Interface allows the user to build and assign a path in the Cartesian or joint space. The Robot Model Interface allows the user to inspect the robot parameters and variables, to change the parameters values, and to select the variables which transient he wants to see. The Time Presentation indicates the progress of the simulation, with respect to its total time; it also allows the user to pause or abort the simulation. The Trend Presentation displays the transient of a variable, through an interactive plot[3]. Finally, the Virtual Presentation displays the robot virtual prototype within its workspace.

4 Application Technology

The SIMECS-RI has been implemented in Java [6]. Java is a pure object-oriented programming language, and exploiting of interfaces to the classes it allows the realization of well decoupled software components. Moreover, Java is a general purpose language, and it is provided with libraries that can be exploited for building applications which span over many different programming fields.

A language which is pure object-oriented makes it harder to obtain inconsistencies between the UML

[3]User can see more than one plot at a time, by creating multiple instances of the Trend Presentation component.

structural design and its implementation. This improves the software quality, and makes it easier to maintain and evolve the application. Improvements are important also in the design in-the-large, were Java interfaces have been exploited to export the methods of the various components.

JavaBeans technology has been adopted for the implementation of the SIMECS-RI client. Beans are the Java proposal for component-oriented software architectures. Beans interact exchanging events, and are well suited for light-weight visual components[4] like those of the SIMECS-RI client.

The SIMECS-RI graphical user interface has been implemented using the *Swing* library for windowing, the *Java2D* library for graph plots, and the *Java3D* [7] library for the robot virtual presentation. Thanks to Java portability, the SIMECS-RI client can be run on any platform[5] for which such libraries exist.

The SIMECS-RI server, on the other hand does not need an interface to the user, but one for interacting with the simulator process, and it is constrained to run on the same platform for which the simulator has been compiled. In the actual version of the SIMECS-RI server, Dymosim has been exploited as external simulator. Dymosim is a Windows executable file automatically generated by Dymola [1], by means of translating the Modelica code into C code, and then compiling the C code. Since Dymola compiles a new Dymosim executable for every Modelica model, a library of Dymosim executables is stored on the server, each of which corresponding to a single compiled model.

The Dymosim executable reads the simulation settings and the actual values of the parameters of its own model from an input file, and stores into an output file the variables transient evaluated during the simulation. So, the Proxy Simulator can actually exchange information with the simulator by writing its input file, and by reading its output file.

Dymosim allows another way also for exchanging data: the DDE (Dynamic Data Exchange) [8] interprocess communication technology. DDE is Windows native, and is based on shared memory areas. Communication through DDE has been made possible in Java by exploiting the JNI (Java to Native Interface) API. It is so by means of DDE that the Proxy Simulator gathers from Dymosim the data that are forwarded to the SIMECS-RI client while simulation is in progress.

[4]For example, all the *javax.swing* library of windowing components is implemented through JavaBeans.

[5]SIMECS-RI has been tested successfully, by now, on Linux and Windows.

5 A Case Study: The Spider Arm

The SIMECS-RI has been tested on compiled Modelica models of the Spider robotic arm, which is shown in figure 5. By now, three different types of motion control systems have been modeled and applied to the same blocks of the Spider mechanical chain and actuators array [10]. These are: joint independent control, operational space motion control, and operational space hybrid control. At current stage of development, SIMECS-RI is able to work with the first of them.

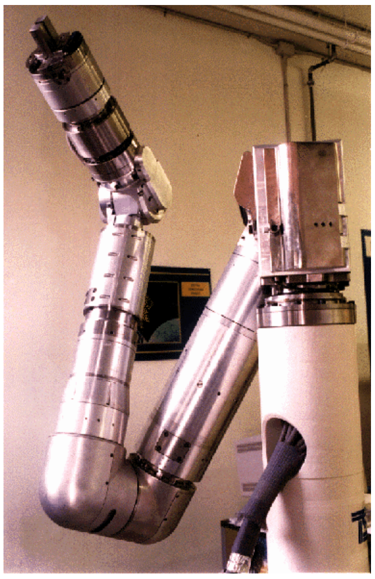


Figure 5: The Spider arm.

5.1 The Robot Model

The Spider model with joint independent control system has been provided with a joint space trapezoid speed trajectory generator which accepts as inputs the initial and final positions of each joint, and the percentages of the maximum values of joint speed and acceleration. The model [10] features *P/PI* cascade controllers with motor and joint position sensors, dynamics of brushless two-phase motors, current controllers, elastic transmissions with backlash and friction, and a seven degrees of freedom multi-body chain with optional payload. A total of more than 12,000 equations are listed at compile time.

The Modelica model of the motor (see[6] figure 6) describes the electrical dynamics of the two phases, the electro-mechanical conversion (block `EMF_2`), the

[6]All figures referred within this section are Dymola [1] schemes.

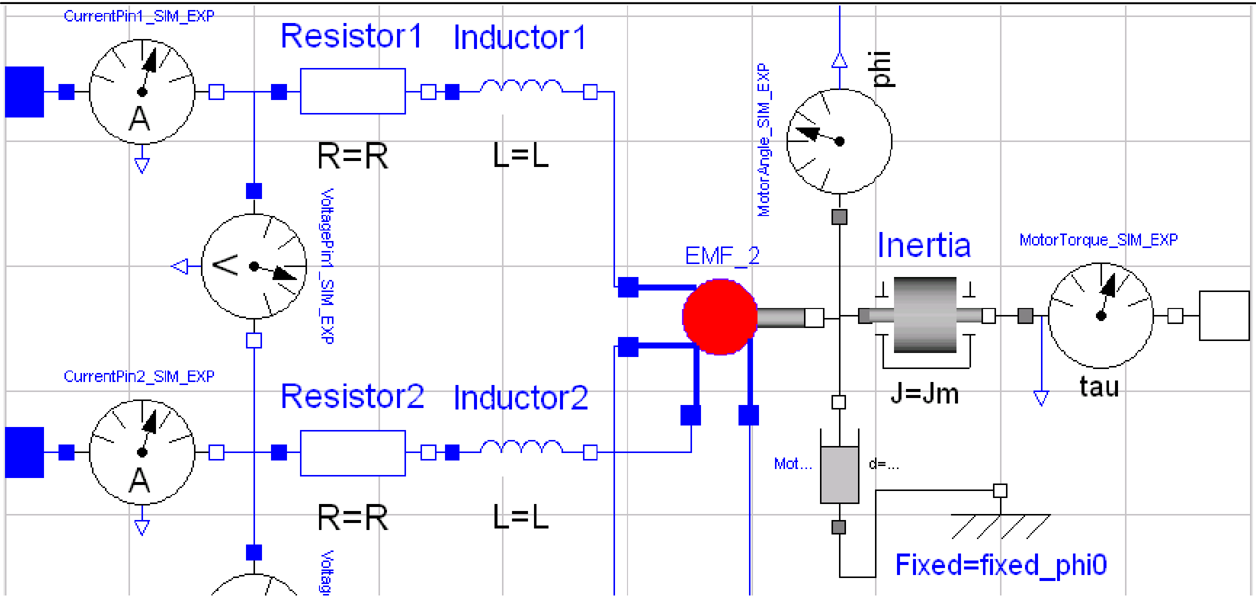


Figure 6: Scheme of the Modelica brushless two-phase motor model.

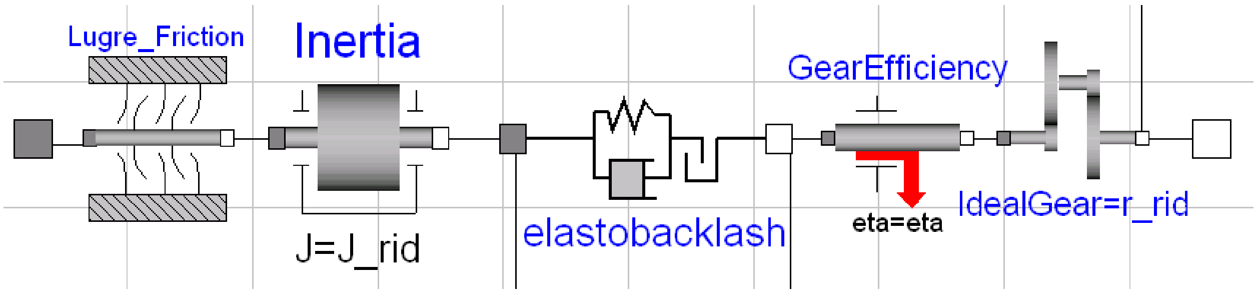


Figure 7: Scheme of the Modelica elastic gear model.

equivalent rotor inertia, the viscous friction of the motor, and it also includes the encoder. The electro-mechanical conversion block offers the interesting possibility of simulating the effect of the most important torque disturbances due to the motor dynamics, like the ripple caused by torque phase unbalanced, the ripple due to shape functions imperfections, and the detent torque, which is present also when current is null.
The Modelica model of the analog current controller includes two analog *PI* regulators with anti-windup compensation. This component allows to set the value of the current offset, which is useful to simulate sensor polarization. This way, this component can reproduce a torque disturbance on the motor.
A realistic transmission model has been built using the Rotational objects taken from Modelica Standard Library. The `Gear_Box` Modelica component (see figure 7) includes:

- a continuous-nonlinear friction model (LuGre);
- a mechanical efficiency model;
- an equivalent gear train inertia model;
- torsion flexibility, damping and backlash models;
- an ideal reducer model.

The analog[7] joint control system is equipped with two resolvers for each joint (at motor and load sides). The control scheme is constituted by an inner loop (*PI* part), for motor speed control, and by an outer loop (*P* part), for joint position control. Anti-windup compensation mechanism and velocity feed-forward are also present in the inner loop.
The Spider mechanical chain model has been implemented exploiting the Modelica Multi-body library. In order to make the SIMECS-RI application able

[7] A digital control system has been implemented and tested as well, but in order to maximize the speed of the remote simulations, its equivalent analog version has been finally preferred.

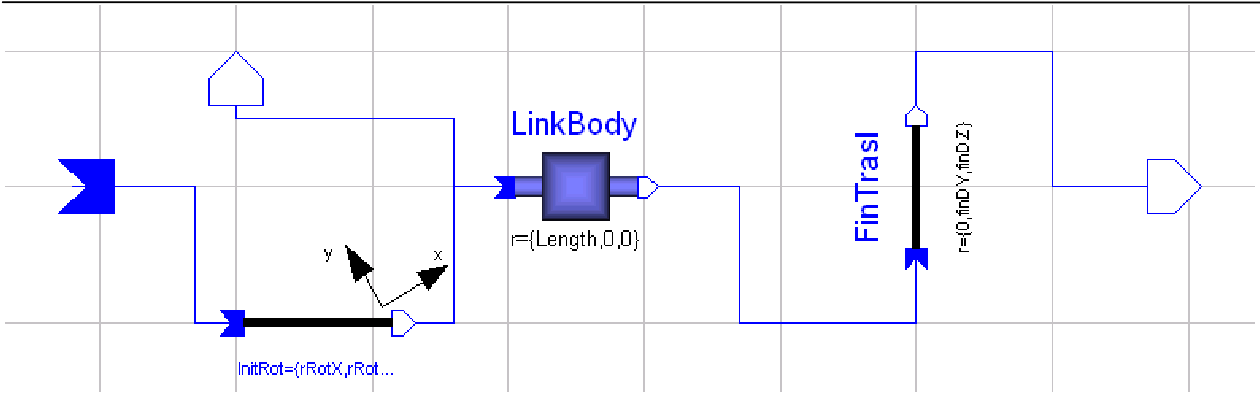


Figure 8: Scheme of the Modelica link model.

to graphically represent and animate robots, a general way of specifying robot kinematical chains with graphical appearance has been conceived. This allows to change link lengths at simulation time without loosing the link appearances.

In figure 8 the Modelica scheme of the `Link` component in the Spider mechanical chain is shown. As it can be seen, a central translation with associated body is preceded by an initial rotation and is followed by a final translation. The initial rotation has a null component around the axis of the joint preceding the link, and the final translation has a null component along the axis of the link. The central translation can have a non-zero component only along the x axis, and its value equals the length of the link.

Shapes that can be attached to the robot links are currently modeled in separate Modelica components. Many elementary solid shapes can be attached to the same robot link (i.e. mass plus roto-translation), but only one shape object can change its dimension along one of its axes according to the link length. The resizable shape can be virtually of any type, but it is more appropriate if it has a constant orthogonal section with respect to its resizable axis. This in order to avoid misshaping the robot link appearance in the 3D model, when the length is changed. Modelica shapes that are appropriate in this sense are `Cylinder`, `Pipe`, `Box`, and `Beam`.

5.2 The Application

The main window of SIMECS-RI is shown in figure 9, with the joint controlled Spider arm model loaded. As it can be seen, the window is divided in three main sections: at the left side there is the parameters tree, in the middle is the robot virtual presentation, and at the bottom is the joint command panel. Additionally, a simple toolbar is placed at the top of the window. The toolbar is used to access the simulation parameters setup window, the model documentation window, and the models list window. The initialization panel is also accessible by means of this toolbar.

The initialization panel (which is not shown in figure 9) is a popup menu, and is identical to the one that is shown in figure, which is used to assign destinations in a joint space path. The initialization panel is constituted by an array of sliders, one per joint. When the user changes the robot joints positions, the 3D robot model is immediately updated.

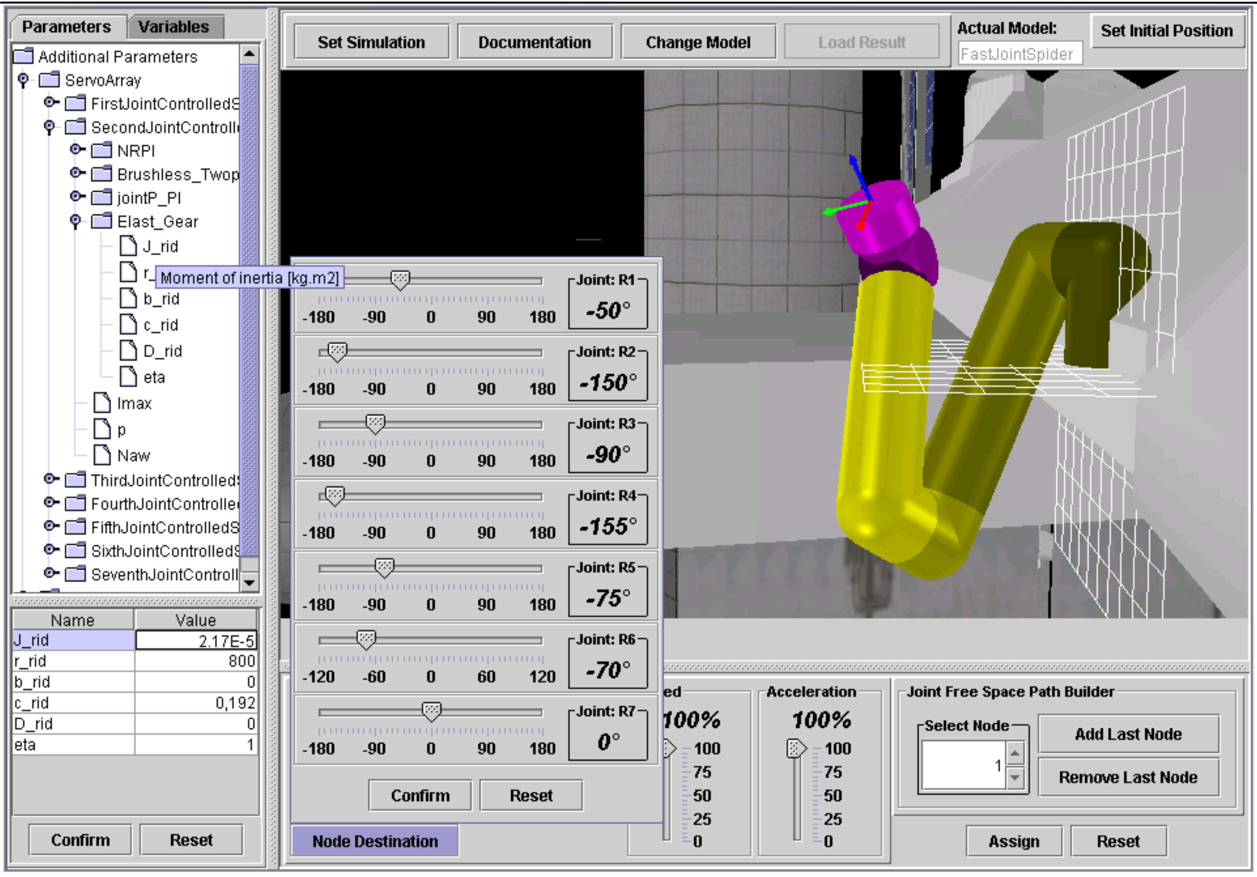
In practice, the user can move the robot by acting directly on its joints. This is also known as kinematical simulation. By means of this feature, the user can immediately perceive the position of the robot moving within its environment, and so he/she can easily place the robot in the desired initial position.

Notice that, from the user point of view, initialization consists *only* in choosing the joint positions of an initially motionless robot. All internal states of the model components[8] which have to be updated for maintaining consistency with the initial joint positions are automatically computed by the SIMECS-RI server on the basis of some algebraic expressions stored in an auxiliary file. These expressions state the relations between the joint positions and the unknown quantities when the robot is in an equilibrium state[9], and should be supplied with the robot model.

Robot model parameters and variables trees share the same space in the window, and it is possible to switch

[8]In the actual model these are motor initial positions, and the initial states of the pseudo-derivatives blocks of the control systems.

[9]For example, the algebraic relation between each motor angle and the correspondent joint angle is simply $q_m = nq_l$, where n is the gear ratio. The effect of an elastic transmission is neglected since the robot is supposed to operate in zero-gravity conditions.



Name	Value
J_rid	2.17E-5
r_rid	800
b_rid	0
c_rid	0,192
D_rid	0
eta	1

Figure 9: The SIMECS-RI graphical user interface.

from one to the other by choosing the corresponding tab at the top left corner. Parameter values can be changed by means of the table at the bottom left corner, while parameter descriptions appear as tool tip texts. Variables plots are accessed by double-clicking the variable names in the variables tree.

The 3D graphics model is built by interpreting information stored in the simulator input file. This can be done provided that the the guidelines sketched in section 5.1 are followed while the modeler builds the kinematical chain of the robot and its graphical appearance

The joint command panel is used for building and assigning joint free space paths, which are sequences of path nodes. Every node has a destination (i.e. a *via* point of the overall path), and two parameters which state speed and acceleration reduction to be applied in the corresponding path segment. Paths that can be assigned are a subset of the ones that can be defined by means of the PDL2 [9] `move` instruction. Actually, only the first node of any path is really issued to the simulator. This limitation is applied for compliance with the trajectory generator of the model.

6 Future Work

SIMECS-RI is a complete simulation environment for robotic arms moving in free space. By now, robot commands can be given at joint level. The most natural way of extending such an application is to make it capable of dealing with more complex simulations, always within the field of articulated robotics.

First, it is planned to handle the case of simulations of contact situations between the robot and its surrounding environment: this includes both the case of a manipulator grasping objects and the case of a robot whose end effector slides onto a surface; second it is planned to allow to perform simulations of movements in an environment with obstacles. Extensions to such cases not only imply to design and implement novel command interfaces for robotic arms, but also to design the corresponding command interpreters, and, last but no least, to build models that can handle such new complexities.

It is planned also to modify the SIMECS-RI server in order to make it able to interface itself with hardware-in-the-loop simulators, where only the robot electro-

mechanical parts are simulated, while the motion controller is a real one.
Finally, the application user interface can be extended to allow telemanipulation of a real robot.

Acknowledgments

The authors would like to thank Gianpaolo Cugola for its important contribution to the design of the SIMECS-RI application, Paolo Donadeo for the implementation of the Virtual Presentation component, and Luca Viganò for having provided all Modelica models of the SIMECS library.

References

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

[2] *Simulink: Design and Simulate Continuous and Discrete-Time Systems* [Online]. Available:
`http://www.mathworks.com/products/simulink/`

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

[4] *OMG UML Specification* [Online]. Available:
`http://www.omg.org/uml/`

[5] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modelling Language User Guide*, Addison-Wesley Pub Co; 1st ed. September 1998.

[6] B. Eckel, *Thinking in Java*, Prentice Hall PTR; 3rd ed. December 2002.

[7] D. J Bouvier, *Java3D API Tutorial*, Sun Microsystems Inc. [Online]. Available:
`http://developer.java.sun.com/developer/onlineTraining/java3d/`

[8] C. Petzold, *Programming Windows*, Microsoft Press; 1st ed. 1988.

[9] *Linguaggio di Programmazione PDL2 - Versione 3.0* [in Italian], COMAU S.p.A. Robotics Division; 1992.

[10] L. Viganò, *Modellistica del Braccio Robotico Europa con Analisi del Controllo nello Spazio Operativo* [in Italian], Master's Thesis, Politecnico di Milano; June 2003.