Proceedings of the 17th World Congress
The International Federation of Automatic Control
Seoul, Korea, July 6-11, 2008

ARTIST: A Distributed Remote Control Lab

Michele Basso [*] **Marco Romagnoli** [*] **Giacomo Innocenti** [*]

[*] *Dipartimento di Sistemi e Informatica, Università di Firenze*
via S. Marta 3, I-50139 Firenze (Italy)
basso@dsi.unifi.it

Abstract: The aim of the paper is to present ARTIST, the remote control laboratory at University of Florence. Students can remotely interact with a number of physical processes exploiting predefined controllers or uploading their own. The main features of the lab are: a modular approach for the realization of different control tasks for the same process, which can be executed on different lab PCs and communicate over a network; a web-service application which exploits a database to facilitate the registration of users, processes and experiments of the laboratory. ARTIST website can be found at http://artist.dsi.unifi.it.

1. INTRODUCTION

In recent years, the fast development of web-based technologies has led to the new concept of remote or virtual labs, i.e. laboratory resources that can be accessed at any time from any place through an internet connection, providing the user of an effective way to perform experiments remotely, while observing the results on his/her computer, locally. Compared to standard lab facilities, a remote lab provides many new capabilities, although the implementation can be rather challenging as well as the lab administration.

In control education, some of the available remote laboratories use Java programming language to perform simulation or to interact with real plants, focusing mainly on the remote laboratory management [Röhrig and Jochheim, 1999, Hua and Ganz, 2003, Rasche et al., 2004] or providing a framework to deal with multiple accesses and to assure consistence. Other interesting examples of this approach can be found in [Casini et al., 2004, Schmid, 1998, Henry, 2007, Hahn and Spong, 2000]. Most of these labs make use of simple interfaces that can be run in a web browser allowing the user to choose the experiment to run, define the reference signal and — in real time during the experiment — modify some controller parameters, observe the output signal in a window and download/archive data for post-processing and analysis.

In this paper we present ARTIST (A Real Time Interactive Simulink-based Telelab), the remote control laboratory at Dipartimento di Sistemi e Informatica of the University of Florence. Its core architecture is mainly based on open source software such as RTAI Linux and faces requirements arising from two distinct point of views: students and lab administrators. The underline framework is based on the RTAI-XML project [Basso et al., 2005], a server component which provides a bridge between a true real time operating system, required to run the most demanding control experiments, and the world of web-services, allowing an easier interaction with a web-based remote control laboratory.

The paper is organized as follows. In Section 2 we introduce ARTIST. In Section 3 we illustrate the main features of the lab from both students and administrators point of views. A detailed description of the underline architecture will be given in Section 4. Finally, in Section 5 some concluding remarks will be given.

2. ARTIST OVERVIEW

In the last decade, the increase in power and speed of personal computers has made it convenient to directly implement (even complex) control algorithms on a standard PC hardware, equipped with standard DAQ boards and a real time multitasking operating system (RTOS), thus eliminating the need of purchasing dedicated hardware. On the other side it is also crucial to provide an easy and intuitive way for programming complex tasks on a modern RTOS. It is important to bring together design of control algorithms and source code generation, exploiting well-known tools for automatic software code generation, such as, for instance, Real Time Workshop (RTW) [Bucher and Balemi, 2006, Yao et al., 2000], fully integrated with the Matlab/Simulink environment [Dixon et al., 2001, The MathWorks, 2007], or Scilab/Scicos and its related automatic code generator Codegen (see [INRIA, 2007]), the latter being available as free software. The remote control lab ARTIST makes use of the above CACSD tools providing several features that are not commonly available in other remote labs. More specifically, students are able to design their control algorithms as Simulink (or Scicos) schemes, upload them on the remote lab through a web interface and run in a distributed real-time network of lab PCs. Moreover, all the files related to any specific experiment for a process remain available for each student for subsequent use.

In Fig.1 a simplified schematic of the architecture is reported. The physical processes interact with a real-time distributed domain (RTDOMAIN), where a number of control tasks (TARGETs) can be executed on different machines communicating over a LAN. Every control task running in the domain can be separated in software

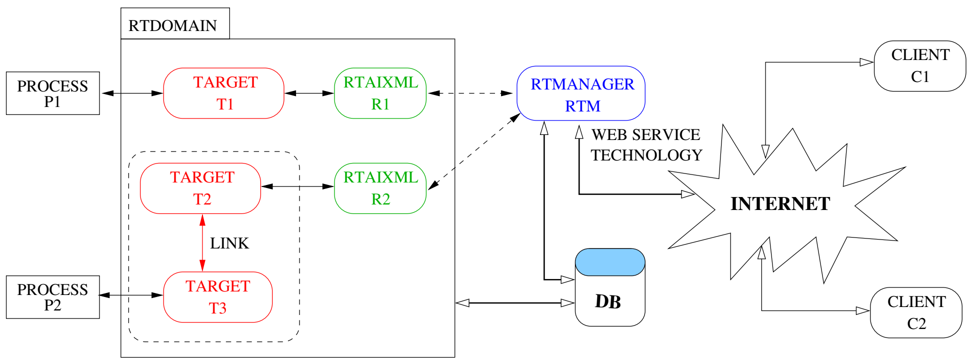


Fig. 1. Architecture remote control laboratory scheme (functional diagram).

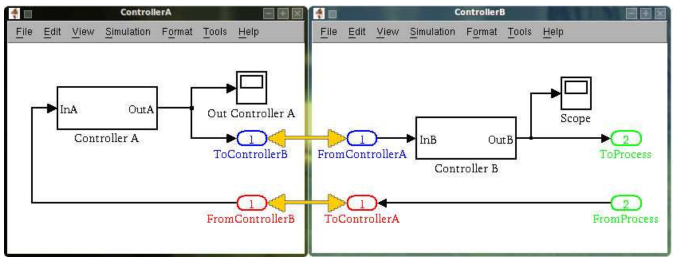


Fig. 2. Controller composed by two modules and its links.

modules. Every module is automatically generated by CACSD tools available from the RTAILab project [Bucher and Balemi, 2006], that allows to create real time C code from a block diagram model. Clients can manage an experiment (i.e. a process controlled by its target) from outside the real-time domain through jRTAILab[Basso et al., 2005] – a generic multi platform Java application which is embedded in a web page and communicates with the domain via a web service provided by the RTAI-XML server component.

The laboratory framework supports the development of real time distributed control systems where different tasks of the same controller can be executed on different machine connected among a real time network. Extending the module concept (see Section 4.3), it is possible to create real time distributed tasks. A module is a part of the controller task that runs on a single machine. Designing a controller through modules it is possible to deploy every module on different machines exploiting every peculiarity of the available hardware. For example, if a powerful hardware is available in the laboratory but it is not equipped with any acquisition board, one can divide a controller in a processing module to be executed on such a machine and an acquisition module deployed on another machine. Is is also possible to divide a controller in modules by sampling time, grouping low sampling time tasks to be executed on "slow" machines and fast sampling time modules deployed on more powerful hardware. Each module is connected to the other components through links (see Section 3.3) to exchange data and influence the execution of the other modules blocking them until some conditions are fulfilled. This approach gives the opportunity to create a module set to be employed in many controller tasks as shown in Section 4.3.

3. HOW DOES IT WORK?

Before going into the details of ARTIST lab it is necessary to define:

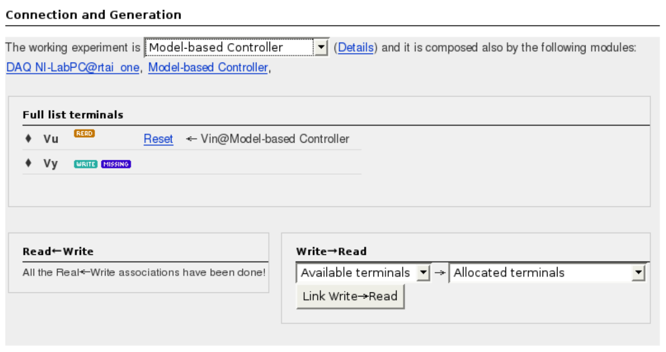
- A process is the physical plant to be controlled.
- A module is a single controller task and all the related information including the real time executable.
- An experiment is a set of modules related to the same process.
- A link is a connection between two modules (see Section 3.3 for details).

3.1 Students

In order to perform their experiments students must carry out the following steps:

(1) connect to http://artist.dsi.unifi.it and access the lab with the provided username and password;
(2) choose a lab process and one of to two classes of experiments:
    - start an experiment using a predefined controller;
    - edit an experiment by designing a new controller.

    By choosing the second option the next points will show how to create the controller and test it within the remote lab.
(3) the selected process is presented together with the necessary documents and information (pdf and/or web links) to obtain a mathematical model of the plant. Students must create the controller based on the information provided. At the present time, the remote lab accepts Simulink schemes, whereas Scicos models will be shortly available.
(4) within the remote lab, each module can be run on a different machine. Students create their own modules inside an experiment exploiting the web tools provided by the lab. Each module is composed by
    - a Simulink model, representing a controller task;
    - an optional data file, that contains the necessary variables to be loaded before the model is compiled;
    - links to other modules in the same experiment for inter-task communication.
(5) once uploaded, the controller scheme is automatically analyzed in order to search for inport/outport Simulink blocks that represent the links to the other modules.



Connection and Generation

The working experiment is Model-based Controller (Details) and it is composed also by the following modules: DAQ NI-LabPC@rtai_one, Model-based Controller,

Full list terminals

- Vu READ Reset ← Vin@Model-based Controller
- Vy WRITE MISSING

Read←Write

All the Real←Write associations have been done!

Write→Read

Available terminals → Allocated terminals

Link Write→Read

Fig. 3. Web tool to create links between modules.

(6) student need to define through a specific interface how the modules are linked together before starting the compilation of the executables.

(7) when all the executables involved in the experiment are available it is possible to:
   (a) click on the “connect” button: the lab checks the status of the experiment;
   (b) click on “setup experiment”: the lab starts each real time executable on the selected machine and setup the rtaixml server to accept external connections (see Section 4 for more details);
   (c) start jRtaiLab to interact with the experiment viewing signals and changing controller parameters on-line.

These steps represent the standard way to perform an experiment from scratch. The web interface has been created to be as simple as possible and let students perform their experiment on real time machine without the difficulties that normally arise when dealing with real time systems.

3.2 Administrator

The main aim of the laboratory administrator is to manage and maintain the remote laboratory. Main tasks are:

- setup the experiments.
- check the safety and the correct use of the remote lab.

As far as the former task is concerned the remote lab provides a set of tools to publish experiments on the web. Two categories are available:

(1) “View category”, this category create a full public experiment. Students do not need to create any controller, but can start the experiment exploiting the default controller and change on-line the parameters of the controller. This category is useful for students at the beginning of a course in control systems (i.e., controllers can be as simple as a PID).

(2) “Control category”, this experiment category provides the opportunity to test any kind of controller (student specific) on the physical processes. The lab administrator creates (via an administrator web interface) a stack of modules based on a level priority. Each element of the stack is a module which can be *public* or *private*. In the first case students have access to the model and data files of the module, otherwise the module is invisible and only its links toward the student controller are available.

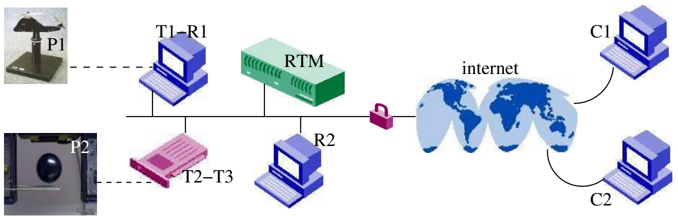


Fig. 4. real time domain: Hardware diagram.

In order to understand the basic idea behind the proposed “Control category”, let us consider a common scenario for a generic remote lab where it is not considered as a best practice to give direct access to the lab hardware (usually for safety reasons). Therefore, a possible solution is to provide for each process: *(i)* a low level software component that masks the hardware layer; *(ii)* a supervisor that monitors the experiments and prevents unsafe conditions. In ARTIST it is quite simple to realize this control architecture. The lab administrator needs to create two private modules, one for the low level hardware layer (DAQ module) and another for the supervisor (Supervisor module). Then, the two modules are linked and grouped in an experiment of the *Control category*, with the required signals available through inports/outports toward the student controller. At this time a new experiment is available to the students, they can interact and control the underlying physical process as if they were directly connected to the process, but actually they are sending the control data and reading the measured sensors from the Supervisor module that masks the underlying structure preventing damages to the process.

3.3 Modules links

Module links represent the way modules can communicate among each other. In a Simulink model, a link is represented by an inport block (down-link) or an outport block (up-link) whereas inside the real time domain these are translated into RTAI mailboxes (read or write). If a student requires to split his/her controller into two modules he/she needs to create two Simulink files as shown in Fig.2. In this example the idea is to execute Controller A on PC A and Controller B on PC B, Controller A needs to send data to Controller B (blue inport and outport blocks) while it receives data from Controller B through red inport and outports blocks. Within the lab engine, to correctly create the real-time executables it is necessary to preliminarly “link” such blocks through the tool provided by the web interface as shown in Fig.3. The green input and output blocks in ControllerB are used to send and receive data from the process and they have to be linked with blocks provided by the experiment.

4. AN INTEGRATED AND DISTRIBUTED REAL TIME FRAMEWORK

In this section we will go deeper into the details of the hardware and software architecture of the remote lab. ARTIST is based on an extended version of RTAI-XML, an integrated framework whose conceptual and functional diagram is shown in Fig.1, whereas Fig.4 maps its basic actors on hardware. Each physical process $P\{n\}$ is interfaced with a Hard Real Time (HRT) control task $T\{n\}$

(the target), situated inside the *Real Time Domain*. In general, a target can be generated using Rapid Control Prototyping (RCP) based on RTAI-Lab code generator, or implemented directly using the Application Programming Interfaces (API) [DIAPM, 2007] of the underlying real time operating system (Section 4.1). These components represent the core of the control system and they are autonomous during the execution phase, although they require additional software for remote monitoring and managing purposes. This job is accomplished by soft real time actors $R\{n\}$ (rtaixml servers), automatically inserted in the domain by the framework, collecting information from the targets and dispatching such information over the network through a web services approach. This solution completely divides data processing from information representation.

The rtaixml server manages single user↔target interaction providing an interface to connect/disconnect the target from the user and to change, if required, the target parameters. Using XML to communicate over HTTP, one external client $C\{n\}$ can operate on the target using a web services approach, obtaining information about the selected target and changing its status using remote procedure calls.

The whole real time domain is managed by a soft real time process RTM (RT Manager) that receives the connection requests from the external client $C\{n\}$ and provides the right information to interact with the target through the rtaixml server $R\{n\}$. RTM provides functions useful to manage and check the RT domain.

The following subsections will show some technical details of the proposed architecture.

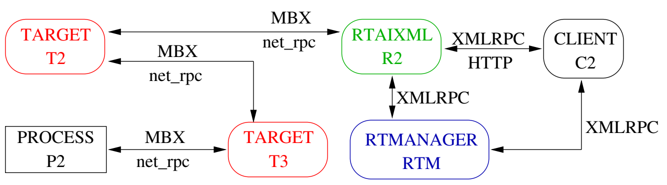


Fig. 5. Advanced client-server architecture with protocol specification.

4.1 The real time guarantee

A major requirement in implementing digital control systems is assuring determinism in the execution time. This aspect, often neglected in the design step, needs to focus also on timing constraints by choosing a suitable hardware/software platform. Since our main goal is to provide a reliable remote laboratory based on a real time framework, we adopted a solution based on a standard PC running a fully featured operating system with real time capabilities. A Real Time Operating System (RTOS) provides programming interfaces allowing to meet process deadlines generally (Soft Real Time – SRT) or deterministically (HRT).

In particular, our lab platform is based on the open source operating system Linux running a kernel patched using RTAI extension. RTAI has a UniProcessor (UP) specific scheduler and two for MultiProcessors (MP), in which case it is possible to choose between a Symmetric MultiProcessor (SMP) and a MultiUniProcessor (MUP) scheduler.

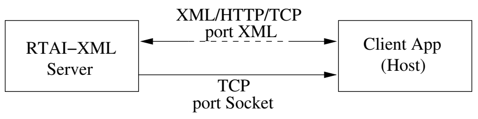


Fig. 6. Details of the client-server interaction.

4.2 The RTAI-XML workflow

In order to understand what happens on the backstage of the remote lab when a student works on it, it is necessary to detail the extended RTAI-XML architecture implemented in the remote laboratory.

The first step toward control system implementation is the simulation of the controlled plant. This requires designing an appropriate Simulink model using blocks from the standard built-in libraries, including the detailed model of the plant. Using this scheme the designer can simulate different control strategies and algorithms, selecting the solution that fits better the required specifications. In order to convert the scheme into a real time target, the automatic C-code generation based on Real-Time Workshop (for Simulink) is exploited. The student must isolate the controller from the model of the plant inserting the inport Simulink blocks instead of the sensor measurements and the outport blocks instead of the controller outputs. When the model is analyzed by the remote lab engine, inport and ouport blocks are searched within the scheme and the necessary module links are created as shown in Fig.3. Once the link operation is terminated and the student asks for the generation of the real time executable, the following actions are performed:

(1) the model and data files are downloaded by RTM from the DataBase (DB) if the DB version is newer than the local one;
(2) the model is checked and adjusted to fit the requirements of the RTAI-Lab code generation project;
(3) some model scheme modifications are also carried out, such as:
    - *inport and outport blocks* are substituted with RTAI mailboxes. In particular the module links provide the necessary information to substitute each inport or outport block with the corresponding RTAI mailbox (read or write MBX). Therefore, exploiting RTAI mailboxes it is possible to transform Simulink models communicating through inport and outport blocks with other real time executables exchanging data through real time mailboxes;
    - *Simulink scopes* are substituted with RTAI scope in order to remotely monitor the scheme signals.
(4) the data file is loaded and the model is compiled by RTW that exploits RTAI-Lab to generate a real time executable for Linux RTAI;
(5) the real time executable is stored in the database ready to be run as soon as the student requests to start the experiment.

The execution phase starts when the student presses the “Setup Experiment” button of the web interface. This phase is operated by RT Manager, which is in charge of running specific target upon client request. Indeed, RTM performs the following actions:

- locates on the DB all the modules composing the requested experiment;
- checks for resources availability (process and related modules);
- starts each target $T\{n\}$ on the right machine as configured in the module properties;
- executes the rtaixml server $R\{n\}$ dedicated to the client-target communication;
- updates the DB information to keep track of the execution.

In the context of a real time distributed framework, the communication among actors (see Fig.5) is one of the crucial features which deserves special attentions. Since the introduction of web services there have been conflicting positions concerning advantages and disadvantages of applying such a technology to real time applications [Strothman, 2002]. The main advantage can be summarized in the increase of interoperability among different platforms, removing the restrictions due to the hardware limits. The disadvantage is generally concerned with the overhead present in data transmission due to XML and HTTP which are sent over the network, increasing the required bandwidth without increasing the delivered information. For this reason, web services are often not suited to communicate the amount of data an application might require in a real time domain.

RTAI-XML tackles the latter problem exploiting two separate communication levels as depicted in Fig.6. At the first level, when the client ($C\{n\}$) request involves general information about the target ($T\{n\}$) state as, for instance, retrieving the signal and/or parameter structure, the server $R\{n\}$ adopts a web services approach, communicating via XML/HTTP/TCP. Conversely, if a high data rate is involved, rtaixml server switches to the next communication level, sending raw data directly over TCP, such that the overhead is reduced and the transmission is optimized for the available bandwidth.

The solution adopted for communication between client $C\{n\}$ and rtaixml $R\{n\}$ is inherently soft real time, both at the network (i.e. TCP/IP and ethernet protocols) and the software (data buffering and asynchronous calls) levels. Therefore, a completely different communication technology is used to exchange real time data among targets. RTAI is a standard POSIX operating system and offers the needed communication tools like mailboxes, FIFOs and semaphores among processes running on the same machine and extends such tools in a remote way with the net_rpc layer, built on top of RTnet hard real time network driver [RTN, 2007].

On the external side, the basic features of a web service approach are provided. Such an architecture is completely independent from the hardware outside the real time domain. At the same time it allows to communicate and interact between generic external applications and the real time domain.

4.3 Modular and distributed architecture

In a multi-tasking real time environment the operating system provides standard Inter-Process Communication (IPC) tools between tasks. Our framework exploits communication channels offered by the standard POSIX in terms of mailboxes where signals can be directly accessed for read/write operations. Data can therefore be exchanged allowing the integration of real time tasks generated by Matlab/Simulink with tasks generated by Scilab/Scicos. Such an approach allows to easily divide problems into distinct targets grouped into RT packages, implementing complex algorithms more efficiently. Fig.7 shows different scenarios and provides useful hints on how to model a controller structure within ARTIST:

- *single target*: the standard approach in a simple control system is just to have one target interacting both with the process and with the client through rtaixml server (see Fig.7a);
- *hardware–software*: often it is natural to separate hardware-related tasks (data acquisition and normalization) from the control logic (see Fig.7b). This approach allows both for hardware independence (i.e. boards change after failure) and software abstraction (mainly in order to test different algorithms without the need to modify the underlying structure).
- *reader–writer*: working with a networked and distributed framework, it could be interesting to have one client enabled to completely manage the process (with full permissions to modify system parameters and target status), but maintaining the possibility for lighter clients to only monitor specific real time signals. This approach can be exploited by the reader–writer package, as shown in Fig.7c.
- *supervisor–control*: this structure is mandatory for achieving supervisory control, that is the controller action is filtered by the supervisor module which monitors the process state and switches the controller output on/off accordingly (see Fig.7d).

The above examples stress the attention on the need for a correct target (module) dependency management within a package (experiment). This feature is implemented in ARTIST via suitable module links (see Section 3.3) and exploiting a module hierarchy: each module can be allocated to a specific level which allows the definition of an execution order among modules.

It is important to remark that an experiment can be executed on a cluster of real time machines by means of modules separation and communication links. This feature allows to exploit the available hardware taking into account hardware and/or architectural constraints. For example, if a powerful PC is available in the domain but it is not equipped with any acquisition board, one can divide a controller in a processing module to be executed on such a machine and an acquisition module deployed on another machine.

5. CONCLUSIONS

Laboratory experiences in control systems education are an important issue in students formation. ARTIST, the remote lab at the University of Florence offers the possibility

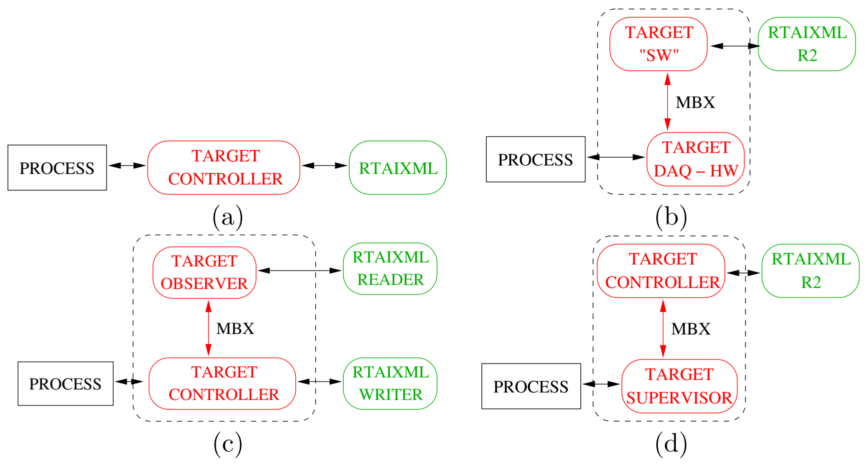


Fig. 7. Scenarios: (a) standard approach, (b) hardware–software, (c) supervisor–control, (d) reader–writer.

to operate with real plants via an internet connection. Students can design their control algorithms within a CACSD software environment and connect to the real processes in an easy way. The use of Simulink, in both standalone simulations and real experiments, permits students to speed up their learning curve. Laboratory architecture assures real time control performance and distributed location of software/hardware.

REFERENCES

Rtnet, hard real-time networking for linux/rtai. [Online] http://www.rts.uni-hannover.de/rtnet, 2007.

M. Basso, R. Bucher, M. Romagnoli, and M. Vassalli. Real-time control with linux: A web services approach. In *IEEE Conference on Decision and Control and European Control Conference*, pages 2733–2738, 2005.

R. Bucher and S. Balemi. Rapid controller prototyping with matlab/simulink and linux. *Control Engineering Practice*, 14(2):185–192, 2006.

M. Casini, D. Pratichizzo, and A. Vicino. The automatic control telelab: A web-based technology for distance learning. *IEEE Control Systems Magazine*, 24(3):36–44, 2004.

DIAPM. *RTAI, Real Time Application Interface.* 2007.

W. E. Dixon, D. M. Dawson, B. T. Costic, and M. S. de Queiroz. Towards the standardization of a matlab-based control systems laboratory experience for undergraduate students. In *IEEE American Control Conference*, pages 1162–1166, 2001.

H. H. Hahn and M. W. Spong. Remote laboratories for control education. In *IEEE Conference on Decision and Control*, pages 895–900, 2000.

J. Henry. Engineering lab online. University of Tennessee at Chattanooga – [Online] http://chem.engr.utc.edu/, 2007.

J. Hua and A. Ganz. Web enabled remote laboratory (r-lab) framework. In *33rd ASEE/IEEE Frontiers in Education Conference*, pages T2C–8–T2C–13, 2003.

INRIA. Scilab and scicos. [Online] http://www.scilab.org, 2007.

A. Rasche, B. Rabe, P. Tröger, and A. Polze. Distributed control lab. In *1st International Workshop on E-learning and Virtual and Remote Laboratories (VIRTUAL-LAB'2004)*, pages 150–160, 2004.

C. Röhrig and A. Jochheim. The virtual lab for controlling real experiments via internet. In *IEEE International Symposium on Computer Aided Control System Design*, pages 279–284, 1999.

C. Schmid. The virtual lab vclab for education on the web. In *IEEE American Control Conference*, pages 1314–1318, 1998.

J. Strothman. Will web services replace hmi? *Intech - Research triangle park NC*, 49(9):34–37, 2002.

The MathWorks Inc. Matlab and simulink. [Online] http://www.mathworks.com/, 2007.

Z. Yao, N. P. Costescu, S. P. Nagarkatti, and D. M. Dawson. Real-time linux target: A matlab-based graphical control environment. In *IEEE International Symposium on Computer-Aided Control System Design*, pages 173–178, 2000.