

Java-based remote experimentation for control algorithms prototyping

Yves Piguet¹ and Denis Gillet

Institut d'automatique, Ecole Polytechnique Fédérale de Lausanne
1015 Lausanne, Switzerland

Fax: +41 21 693 25 74, tel: +41 21 693 38 34, e-mail: *forename.surname@epfl.ch*

Abstract

Remote experimentation is a new concept which enables remote users to manipulate a controlled system from a distant location. Major applications are distance learning, scientific benchmarks conducted on common setups, and tele-maintenance. In automatic control, remote experimentation requires flexible mechanisms to implement particular control algorithms and to provide comprehensive information. These functionalities are mandatory in order to control the dynamic behavior of the remote system and to supervise the ongoing operations. An approach based on a real-time interpreter running on the computer which controls the physical system and a client/server architecture written in Java is proposed. The client-side software, which enables the user to exchange information, is integrated within World-Wide Web pages and downloaded transparently from the server-side computer which runs the controller.

1 Introduction

Distance learning has attracted the interest of many universities around the world [7]. Amongst its advantages are: a broader choice of studies even at places where the number of students does not justify the physical presence of qualified teachers; no or fewer constraints on the time table when the courses are attended; and better sharing of expensive laboratory equipment. For standard courses as well as for distance learning modules, an effective pedagogical approach relies on three phases: first presenting the theoretical background with the help of examples, second testing the knowledge with exercises, and third consolidating it with laboratory experiments. This last phase is probably the most difficult to undertake remotely due to the complexity of the practical operations involved and the human interactions requested by students. However, this challenge has been achieved, thanks to the approach proposed.

In automatic control with digital implementation, fortunately, the operations that students have to carry out on the experiment could be limited to the implementation of a control algorithm, to the modification of its parameters, and to the online retrieval of data. In theory, these operations seem to present no insuperable difficulties. However, in practice, care is required about how the different implementation stages are handled to guarantee safety and efficiency of the control solution.

¹Corresponding author. This work has been supported by the Swiss National Foundation for Scientific Research.

At the Institut d'automatique of the Swiss Federal Institute of Technology in Lausanne, most of the experimental setups are controlled by Power Macintosh computers [2]. A real-time kernel with modules written in C controls the experiments via AD/DA boards. The user interface is implemented in LabVIEW (National Instruments). This framework has been recently extended for usage across a network [3]. Problems due to the network, with respect to the limited bandwidth and the variable delay it introduces, are described in [9]. Controllers have a fixed structure, and only their parameters can be changed. This paper deals with the case where the complete algorithm is specified by the remote user. The additional flexibility results in new problems. Reference [1] describes a solution similar to the approach presented in this paper, where the student can control an experiment over the Web and upload control programs written in C. However, for security reasons, access is limited to registered students. Here, this problem is overcome by the use of an interpreter which provides a controlled environment where the remote user does not have access to the file system and other sensitive resources. WinCon, a commercial solution based on the Realtime Workshop of MATLAB, compiles the code of Simulink diagrams and sends it over the network to another computer which controls the experiment [8]; however, limited to Windows 95 and not integrated with the Web, it is not suited for a large distribution of the client software.

The remainder of the paper is structured as follows. In Section 2, the remote experimentation problem is introduced. The requirements for an efficient environment are given in Section 3. The software developed to support local experimentation for prototyping control algorithms is presented in Section 4, and the issues specific to remote operations are detailed in Section 5. Concluding remarks and extensions which are being currently investigated are given in Section 6.

2 Remote experimentation for algorithm prototyping

This section provides an overview of the remote experimentation environment discussed in this paper. The experiment is located in a laboratory, while the user conducts the experimentation from a remote location; typically the computer room of a university, a distant laboratory, or home. The control algorithm runs on the local computer which drives the experiment, so that the feedback loop (if there is one) does not suffer from the delay that the network would introduce. This computer also acts as a server to broadcast data and cope with user requests. The supervision of the operation is conducted remotely by means of a client software which commu-

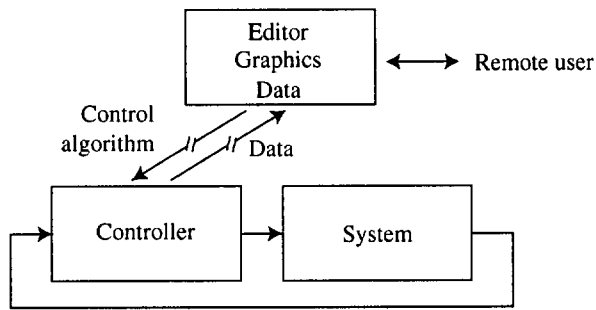


Figure 1: Remote experimentation using uploaded control algorithms.

nicates through the network (Internet or a local area network) (see Fig. 1).

3 Requirements of remote experimentation

In order to complement or to replace effectively local experiments, where the user can interact directly with the system and see it running, remote experimentation requires several additional features. Otherwise, there is a large risk of having an impractical setup, either because of too restricting limitations or an overwhelming complexity. Below is a list of what is important to achieve a successful remote experimentation setup. This list is designed to be as exhaustive as possible to give a complete picture of where remote experimentation for algorithm prototyping is heading.

- *Ease of use:* In remote experimentation for distance learning, not only the physical experiments are distant, but also the support staff. Difficulties in making the system react to user requests are much more discouraging when no support is immediately available. An effective collaborative environment, while important, cannot replace a good user interface.
- *Flexible algorithm formulation:* Even if a large set of controllers can be reduced to a fixed structure (e.g. polynomials or state-space representation for linear time-invariant controllers), it is important to have the freedom that only a programming language can provide. This enables non-linear controllers, any kind of excitation and identification, and any kind of integration algorithm if the controller is formulated in continuous time. One of the purposes of remote experimentation is to enable several research team access to a single experiment; flexibility is also important to cater to different approaches and different formulations.
- *Online supervision of the experiment:* Once an experiment has been started, the user must be able to supervise what happens and to check that everything runs as expected. In the case of long experiments, being able to detect quickly whether something goes wrong is mandatory to avoid wasting time producing worthless results or even to prevent damage to the experiment. Despite the fact that some information can only be acquired by a person sitting in front of the experiment (such as odors of heated components and vibrations), real-time data, audio and video feedback should be broadcasted in order to provide to a remote user the feel-

ing of being present at the experiment location (this is why remote experimentation is often referred as telepresence).

- *Downloading experimental data:* In addition to the online visualization of remote data, it is often useful to have the possibility to download them for batch processing or for playing back the experiment off-line. This also enables better generation of graphical representations using other software.
- *Server security:* The security concerns valid for any kind of server also apply to a server controlling an experiment. These include preventing the remote user from disturbing the good functioning of the server, from getting access to private resources (information, processing time, storage, or local network access), or from masquerading as someone else. In addition, the process should be protected against any actions that can either damage or destroy it, by a malicious user or by accident. This latter point may be more difficult to enforce, because too strict limitations could prevent the user from bringing the system to its limits, which is often required when validating a new control solution.
- *Cross-platform client-side software:* Different computer architectures and operating systems (OS) coexist amongst the intended user community, and nothing indicates that this situation will change in the foreseeable future. Especially for students, it is not possible to impose a reference computing platform. Thus the client software should be available from those most widely used (currently Windows, Mac OS, and Linux).
- *Easy installation of the client-side software:* The intended users are often not computer experts, and the person responsible for the whole setup is by definition not physically present for troubleshooting. Therefore, the installation of the client software should be as simple as possible, and should not assume the existence of third-party software which is difficult to install. The use of software which can be freely distributed is also a huge advantage for educational applications.
- *Easy access to a library of algorithms:* The best way to assess the capabilities of a system is to see what other people have done. For remote prototyping, a single set of default parameters is not sufficient. The more flexible the environment is, the more important it is to provide several examples. In addition, for research purposes, access to what other people have done facilitates comparisons and collaboration.
- *Flexibility to improve the whole environment:* Especially when the environment is still under development, the communication protocol shared by the server-side and client-side software may evolve. This requires either compatibility amongst different versions or a mechanism to upgrade the client software transparently.
- *Client security:* The client might not trust the source from which he obtains the client-side software. An electronic signature system can certify that the software is written by someone trusted, or the client can run in a controlled environment where it has no access to the resource and private data of the client computer.

Successful setups may still omit one or several of the above requirements. For instance, the remote control of an inverted pendulum or an electrical drive, currently used at the Institut d'automatique, lacks some flexibility, but it focuses more on ease of use and quality of feedback.

4 Real-time interpreter

At the Institut d'automatique, fulfilling many of the points outlined in Section 3 has been achieved in two phases. During the first, the part of the problem not related to the use of the network has been addressed. It was found quickly that the flexibility of the approach could be used to solve many requirements for remote experimentation. This section describes the software used by students during their laboratory experiments to apply quickly to real systems algorithms which they develop and simulate with MATLAB and Simulink [4]. The second phase, which extends the software environment to remote operation, is detailed in Section 5.

4.1 Reasons for new software

MATLAB and Simulink are widely used in the engineering world in automatic control, and also at the Institut d'automatique for research and education. However, they are not well suited for real-time experimentation because they are closely integrated with the OS of the computer which they run on and cannot be called from an interrupt routine. The solution used here is to rewrite the algorithms in C, where most calls to the OS (such as memory allocation, file access, and the graphical user interface) can be avoided easily. Unfortunately, experience shows that the learning curve is steep and prevents the students from writing new real-time routines during their laboratory projects. Preexisting controllers (such as proportional-integral-derivative (PID) controllers, Smith predictors, and adaptive pole-placement-based controllers) have been used for many years; implemented in C and interfaced with LabVIEW (a software package from National Instruments for graphical interfaces and control of external devices with its own graphical language), they are invaluable tools for learning the basic concepts of automatic control. Nevertheless, they do not replace the actual coding of an algorithm. Thus an interpreter which implements a large subset of the MATLAB syntax has been written with the primary purpose to be run in real-time. Using the same syntax as MATLAB permits to write, debug, and simulate the program in the MATLAB/Simulink environment, and does not require learning of a new language.

A separate program, called RT, has been developed for the Macintosh. It permits the real-time execution of MATLAB-like scripts and the communication with input/output (I/O) boards. Support for real-time operations is provided by the Extended Time Manager, the part of the Mac OS which handles periodic execution and timing. Sampling times in the order of 1 ms can be achieved easily, which is fast enough for many laboratory experiments. The Extended Time Manager has existed since 1991 with no significant usage changes, and it should still be present in the next major Mac OS update in 1999. Relying on the services of the OS has the important advantage of reducing the changes when a new I/O board is used. Concerning the I/O, a simple plug-in module scheme has been designed. Each board requires a plug-in, written as a shared library (dynamic link library or DLL in the jargon of other OSs), which implements four functions for initialization, reading, writing, and termination. Modules for GW Instruments MacADIOS and National Instruments PCI 1200 boards have been written.

4.2 Algorithm formulation

In Simulink, the systems to be simulated are built from simpler elements such as adders, transfer functions, and elementary nonlinear functions. More complex blocks can be created by enclosing simple blocks in subsystems, or by writing a function in the MAT-

```
t = rt'time           % get current time in sec.
rt'wait(delay)       % wait based on the prev.call
y = rt'read(chan)    % read the value of sensors
rt'write(chan,u)     % set the value of actuators
```

Listing 1: Functions for real-time and input/output.

LAB language. Such functions, called *s-functions*, implement a general state-space model. Their arguments and calling sequence are imposed by Simulink, so that the number of states, initial state vector, new state or state derivative, and output can be obtained during the simulation. The s-function syntax is a natural choice for RT. In this way, provided a model of the system to be controlled is available, the whole controlled system can be easily simulated before the controller is run by RT with the real system. Currently the s-functions supported by RT are restricted to discrete-time.

However, s-functions, though powerful, are complicated and non-intuitive to write for beginners, because they are called at different times with requests for different kinds of information. In addition, s-functions are cumbersome for applications where the same computation is not carried out at each sampling time. For instance, an experiment might begin with a phase of open-loop identification, where the system input is a pseudo-random binary signal and the output is collected for later processing; a phase of identification, where the measurements are batch-processed to give the parameters of the system; a phase of controller synthesis; and a phase of closed-loop experimentation, for checking the performances of the controller. The content of the state of the whole experiment would change for each phase; putting it in the single state vector of the s-function would become over complicated. A modular approach is impractical because it is difficult to split the phases into separate functions.

For cases where a Simulink simulation is not required and a faster and simpler development is desirable, RT also accepts sequential programs where the state vector is replaced with named variables. Four functions have been added to the interpreter for real-time operations and input-output with the system (see Listing 1). Compare the code of a proportional-integral (PI) controller implemented as an s-function (Listing 2) and as a sequential function (Listing 3).

In MATLAB, the result of an expression is displayed when the statement where it appears has no final semicolon. This makes debugging easier; by removing a few semicolons, both the progress of the execution and where the results do not correspond to what they should can be seen. The real-time interpreter also implements this feature. Real-time output is stored in an intermediate buffer, and the user-interface part of RT periodically dumps it to an output window.

4.3 Integration with other software

RT is a stand-alone application and has the user interface necessary to load functions, display textual output, and choose which I/O module to use. It can also be controlled from MATLAB or Simulink with AppleEvents, the inter-application communication system of the Macintosh. Commands were written to permit installation of s-functions, start and stopping of the real-time operation, changing parameters, and retrieval of the experimental data. A set of Simulink blocks enables the real-time display of measurements in scopes.

```

function [sys,x0] = contrPI(t,x,u,flag)

kp = 1;           % controller parameters
Ti = 10;
r = 5+2*sign(sin(0.5*t)); % reference signal
Ts = 0.1;        % sampling period

switch flag      % flag tells what to do
  case 0        % structure
    sys = [0,1,1,1,0,1];
    x0 = 0;    % initial state
  case 2        % next state
    sys = x + r - u; % new state
  case 3        % control signal
    sys = kp * (r - u + Ts * x / Ti);
  case 4        % next sampling time
    sys = t + Ts;
  otherwise    % ignore other requests
    sys = [];
end

```

Listing 2: Discrete-time PI controller implemented as an s-function.

5 Remote experimentation

The real-time interpreter described in the previous section can be easily extended for remote experimentation, because it is already based on a client-server architecture. The parts missing are what is specific to the network and to the clients: the communication protocol, the management of multiple potential users, security, and ease of installation.

5.1 Client-server architecture

In order to implement the client and the communication part of the server, the Java system [10] was chosen. Java (at least in its current implementation under non-real-time OSs) is not well suited for real-time operation, because it is often still slower than pure C, it uses a garbage collector which slows down the execution at random time, and it cannot be interfaced with the real-time architecture of the underlying OS. In addition for the project described in this paper, the interpreter already exists in C and a port to Java would be difficult and time-consuming to establish. For these reasons, the real-time system and the interpreter are written as a shared library loaded by the Java application which implements communication with the client. The client itself is a Java applet, i.e. a program which runs inside a Web page (see Fig. 2).

This approach fulfills many of the requirements outlined in Section 3. Indeed, the server is protected against poorly designed or malicious real-time programs because the interpreter prevents access to the functions of the OS. Since Java is available on many OSs, client software is cross-platform, and could not be easier to install since it is downloaded and run automatically when the user selects the correct link in the Web browser. The code is typically much smaller than for a complete application; the current implementation is less than 16 kbytes. A library of different algorithms can be made available with a set of pages, each of them containing the client applet with a preloaded program. Each time a page is displayed, the browser checks whether a version of the applet more recent than the content of its cache exists on the server; thus the

```

function contrPI'seq

kp = 1;           % controller parameters
Ti = 10;
length = 10;     % experimentation length
Ts = 0.1;        % sampling period
sum = 0;         % sum of past errors

while rt'time < length % main loop
  y = rt'read(1);    % read syst.output ADC#1
  r = 5+2*sign(sin(0.5*rt'time)); % ref. signal
  e = y - r;        % error
  sum = sum + e;    % sum of past errors
  u = kp * (e + Ts * sum / Ti); % control signal
  rt'write(1,u);    % write syst.input DAC#1
  rt'wait(Ts);     % wait for next sample
  % 1 liter. is exactly Ts
end

```

Listing 3: Discrete-time PI controller implemented as a sequential function.

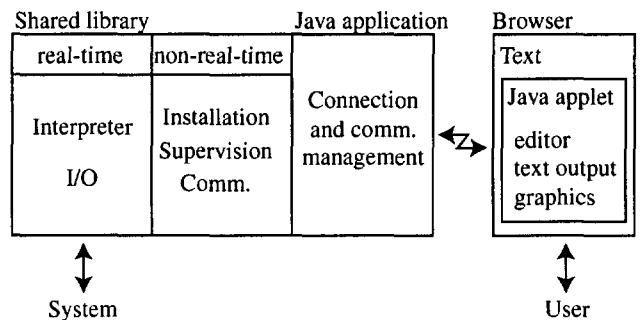


Figure 2: Architecture for remote real-time experimentation.

communication protocol between the server and the client can be improved or even replaced without any problems. Finally, the applet itself runs in a controlled environment (the Java system) where it cannot do any damage to the client computer.

Writing the server application in Java has the additional benefit of simplifying the networking code. Contrary to Unix, Mac OS only supports non-preemptive multitasking, and asynchronous I/O functions must be used to avoid blocking the whole system. Java follows the Unix model where blocking functions (such as those which listen for a new connection) make the central processing unit (CPU) available for other tasks. Using the Macintosh, this means that most of the dirty code is hidden in the Java implementation. Note that Java is still an immature technology which is evolving quickly, and its stability is suboptimal. Improvements can be expected over the next few years.

5.2 Communication protocol

The remaining requirements for a successful remote experimentation setup depend on the way the server and client are designed. Currently the client establishes a TCP (Transmission Control Protocol) connection (reliable and based on a bidirectional stream) with the server, which is not necessarily located on the same computer as the server used to deliver Web documents. Both the client and the server send packets to each other for various requests. A watchdog

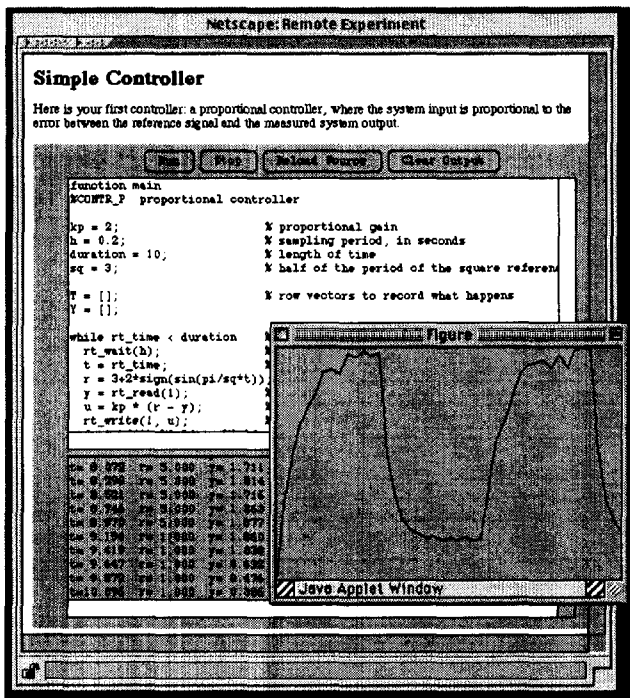


Figure 3: Client running as a Java applet in a Web browser. Measured data are displayed as a graphic in a separate window.

runs on the server, so that if the client crashes, a timeout stops the experiment and closes the connection permitting another client to connect.

Experimental data can be displayed in two ways. First, textual output is sent to the client and displayed while the real-time program is executing. On the server, data generated in real-time are buffered, fetched by the Java server, and sent in packets to the client. The second way to observe the experimental data is with the help of graphics, built either incrementally when new data are available or in batch with a single plot command (see Fig. 3). Incremental graphics are useful to supervise the functioning of the setup; batch plots are better suited to analyzing final results such as an identified nonparametric model, the Fourier transform of signals, or frequency responses resulting from a phase of identification.

6 Conclusions

In this paper, the requirements of a remote experimentation environment designed to carry out algorithm prototyping have been reviewed. RT, software developed at the Institut d'automatique, has been described. It is based on an interpreter which uses a syntax similar to that of MATLAB and which runs in real-time on the Macintosh. This local software solution has served as a base to implement an environment for remote experimentation. The communication solution is based on a client-server architecture written in Java. The client is an applet running in a Web browser. The user can write code, upload it and make it run by the real-time interpreter, and obtain results as textual data, real-time sliding signals in a scope, or batch graphics.

RT has been used since 1997 by both teachers and students at the Institut d'automatique. Feedback from the students is excellent; it shows the usefulness of a simple environment to quickly implement algorithms and get a better feeling of how to apply theory to real-world systems.

The current implementation of remote experimentation based on the real-time interpreter and Java is still a proof-of-concept which lacks some of the features required for public use; amongst them, access control to prevent unauthorized use and permit potential users to book time in advance; and the video and audio feedback, which could be provided separately but would be better integrated into the Web browser. The approach presented here will complement other software tools which implement fixed-structure controllers for standard algorithms.

Instead of putting the client in a Web browser, it can be included with an interactive computer-aided design software such as SysQuake [5, 6]. This will permit obtaining a model based on remote experimentation, remotely updating the controller, and complementing simulations with experimental data.

References

- [1] A. Bhandari and M. H. Shor. Access to an instructional control laboratory experiment through the World Wide Web. In *Proc. of 1998 American Control Conf.*, Philadelphia, 1998.
- [2] D. Gillet, R. Longchamp, and D. Bonvin. Integrated workbench for laboratory projects in automatic control. In *Int. Conf. on Computer Aided Learning and Instruction in Science and Engineering*, Lausanne, 1991.
- [3] D. Gillet, C. Slatzmann, R. Longchamp, and D. Bonvin. Telepresence: An opportunity to develop practical experimentation in automatic control education. In *Proc. of 1997 European Control Conf.*, Brussels, 1997.
- [4] The MathWorks, Inc., Natick, Mass. *MATLAB, the Language of Technical Computing — Using MATLAB*, 1997.
- [5] Y. Pigué, U. Holmberg, and R. Longchamp. Multi-model weighted pole placement. *European Journal of Control*, 3:216–226, 1997.
- [6] Y. Pigué, U. Holmberg, and R. Longchamp. Instantaneous performance display for graphical control design methods. In *Proc. of IFAC'1999*, Beijing, 1999.
- [7] S. E. Poindexter. Using the web in your courses: The how-to's and the why's. In *Proc. of 1998 American Control Conf.*, Philadelphia, 1998.
- [8] Quanser Consulting. Quanser consulting home page. <http://www.quanser.com/>.
- [9] C. Salzmann, H. A. Latchman, D. Gillet, and O. D. Crisalle. Requirements for real-time experimentation over the internet. In *Proc. of International Conference on Engineering Education 1998*, Rio de Janeiro, 1998.
- [10] Sun Microsystems, Inc. Java technology home page. <http://www.java.sun.com/>.