Int. J. Engng Ed. Vol. 19, No. 3, pp. 445–454, 2003
Printed in Great Britain.

0949-149X/91 $3.00+0.00

A XML-based Framework for the Development of Web-based Laboratories Focused on Control Systems Education*

R. PASTOR, J. SÁNCHEZ, and S. DORMIDO
Dpto. de Informática y Automática, UNED, Avda. Senda del Rey no. 9, 28040 Madrid, Spain.
E-mail: rpastor@dia.uned.es

In this paper, a novel framework, termed RELATED, is described that allows Web-based labs focused on the teaching/learning of control systems to be published. With this approach, XML files are used to connect the laboratory elements (plant, simulations, controllers, experiments), conduct the remote system access, and define different experiments. Teachers can thus create and publish new Internet-based control labs using legacy code and control elements regardless of their locations on the Internet. A Java-enabled browser is the only tool needed by students to operate the remote experiments.

INTRODUCTION

AT THE MOMENT, practical education in control topics faces several problems that can be reduced to two: no room for didactical setups and lack of financial resources. To solve these deficiencies, there are many works focused on the development of virtual (simulation-based) and remote (didactical setups) laboratories conducted across the Internet [1–9].

However, all these remote and virtual laboratories are isolated efforts by different university groups. The idea of using different educational centers' software and hardware is not contemplated at all, and the work already carried out by others is not fully exploited. That is to say, to date a methodology or a standard for the construction of networks of virtual/remote laboratories based on previous developments do not exist.

For creating these networks of virtual/remote labs, new tools and languages are needed for defining and integrating the elements that constitute a control lab (plants, controllers, user interfaces, experiments, access permissions, etc). Once the components are declared by means of these new definition languages, the tools carry out the integration of a new virtual/remote laboratory irrespective of the component locations and release it in the WWW. These components (plants, control code, models, etc.) can belong to other previous developments, but the graphical user experimentation interface will hide these details. In this way, the distributed or local architecture of the telelab is transparent for users.

The ultimate objective of this line of work is the creation of an RCLML DTD (Remote Control Laboratory Markup Language). The benefits of this approximation are manifold. First, the developers (teachers) can express the system's attributes, the system's control commands and responses, the data streams, the command and inquiry formats, the communication mechanisms, and the documentation in an extremely flexible yet highly structured XML format. Code generation and (where real-time constraints permit) run-time interpretation enable teachers to make frequent changes to RCLML text files, which are automatically reflected in the application code, and thus in the telelab architecture. At the same time, this DTD enables the XML parser to validate the XML input files, thereby guaranteeing that the telelaboratory description (i.e., elements and experiments) are complete and in line with the constraints imposed by the teachers. The RCLML DTD can be extended to cover new control elements or accommodate changing constraints of existing components, often without making changes to previously written laboratory descriptions.

Similar developments applied to other fields are the VIML [11] specification for describing the elements of a network of virtual instruments, and the IML [12] and NASA AIML [12] specifications.

This paper focuses on part of this DTD. First, details of the environment characteristics and system architecture are given. Next, a full example of remote on-line control of a small-scale plant is described. Finally, the educational outcome and conclusions are presented.

MAIN ATTRIBUTES OF RELATED

RELATED (**RE**mote **La**boratory Ex**t**end**ed**) is an XML-based framework for the development of

* Accepted 18 February 2003.

new paradigms on Internet-based laboratories. RELATED has been defined for the teleoperation of academic control systems as simulation problems or real-time plants (didactical setups). The main idea is to define an abstract entity called *RLAB system* by an XML DTD (the ongoing RCLML DTD) and publish it in an *RLAB Control Web Server* so that students can have remote access to it. Therefore, it will be possible to work with any RLAB system plugged into the web server and share different resources such as plants, simulation engines, or simply, control code. The main RELATED attributes are:

1. Internet access to different RLAB systems services by an RLAB control web server.
2. Teleoperation for any RLAB system defined in an RLAB control web server. Digital certificates and signatures, which are generated for each RLAB system user, provide security and integrity.
3. System specifications via XML. Different XML files allow teachers to define different types of RLAB systems. An RLAB system could be made up of local elements or use components located in other university departments.
4. Platform independency and portability. The RELATED framework is fully developed in Java language. The RLAB system could run on any Java platform such as Wintel boxes, Unix/Linux systems, or Java Cards. Thus our software architecture combines the platform-independent processing capabilities of Java with XML power.
5. Local code reusability. Any control code developed for the local operation of an RLAB system can adapt for implementing new RLAB systems.
6. Control code repository. The code of other RLAB systems can be used to build other remote RLAB systems.
7. Open architecture. Changes are easily made in the system. For example, the addition of new system features just consists of 'adding new XML lines to the definition files'.

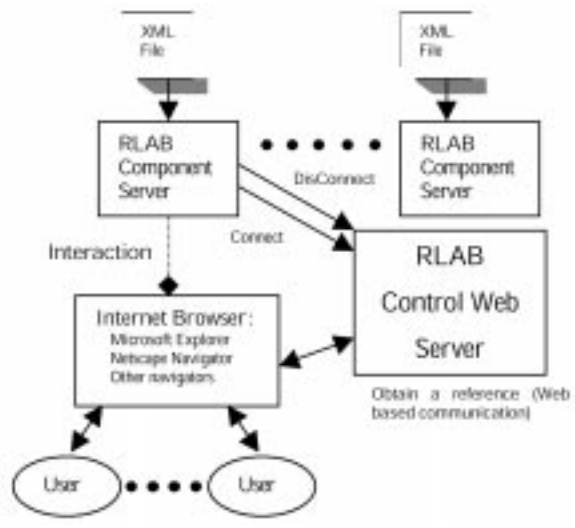

Fig. 1. Logical outline of the RELATED system architecture.

SYSTEM ARCHITECTURE

RELATED system architecture is shown in Fig. 1. There are several RLAB systems (formally RLAB component servers) connected to the RLAB control web server. First, it lets users connect to a particular RLAB system provided that students have enough permission. This main web server can also point to the RLAB system web pages of other associated HTTP servers, i.e., it works like a publication system. In this sense, self-explanatory documents about local system behavior can publish, for example, hands-on assignments or information pages. Hence, users browse the RLAB control web server pages in order to find the RLAB system where the experiments are conducted.

XML-BASED DEFINITIONS

The E**x**tensible **M**arkup **L**anguage (XML) is the universal format for structured documents and data on the Web [10], and it has become an industrial standard for different software environments. XML is used for describing the structure and elements of an RLAB system, so every RLAB system is defined with one XML file, termed the definition file. According to the RCLML DTD specification, a system is composed by different elements: a plant, several controllers, some experiments and different GUI. So a system is described in this file by means of a set of XML tags that represent these elements or system objects. At present, these objects are modules, variables, parameters, views, references, and experiments.

Any XML definition file has two sections:

- *Static behavior section*. It includes modules, references, and views. Variables and parameters are dynamic elements since they are changed by the users. Yet as both objects belong to system and modules, they are included in this file section.
- *Dynamic behavior section*. Only experiments are defined here.

As previously mentioned, an RLAB system is defined by a set of objects (they will be explained below): variables, parameters, modules, experiments, views, and references. Thus an RLAB system can be considered as a black box that can be manipulated via its variables as happens in block-oriented applications (e.g., Simulink or LabView). The ⟨system⟩ XML tag defines a system and only one tag is allowed on each definition file.

A **module** is an entity with variables and

parameters. This entity allows the code to be run inside a local thread while a user is conducting a remote experiment. At present, RELATED supports two types of implementation:

- Native code developed for the local platform where the RLAB system is defined. It is very useful to run the real-time control code or any kind of code developed for control purposes. The programming interface is the Sun's JNI specification (*Java Native Interface*). So, the binary compatibility of native method libraries across all Java virtual machine implementations on a given platform is attained.
- Java code organized as a set of classes included in a Jar file. This Jar file can be located in a URL. This means that any user can develop his/her own control code and use it to build the RLAB system.

In order to provide a standardized interface, the module implementation must offer a predetermined set of methods. This aspect is very significant since it enables the modules to be reused to elaborate other RLAB systems (local or remote). The most important methods of the module interface are:

- *Init( )*. It returns a Boolean value according to module initialization.
- *Start( )*. It executes the thread that runs the local code implementation every sampling time.
- *Pause( )*. Used to pause modules that work with simulations.
- *Resume( )*. Used to resume modules that run simulations.
- *Stop( )*. It stops the running thread of the module.
- *Exit( )*. Used to free resources in the local code implementations.
- *GetVarValue(var)*. Used to retrieve the value of a variable in the local implementation. This method allows the signal values to be obtained in real time.
- *SetVarValue(var, value)*. Used to change the value of a variable in the local implementation.

The ⟨module⟩ XML tag defines a module and the ⟨implementation⟩ tag indicates which code will be run. So, a dynamic library (DLL for Windows, SO for UNIX) is needed in a native implementation, and the Jar file URL and the main class name are provided in a Java implementation.

A **variable** is an object that stores a value (double, float, integer, long, boolean, or string) and that can be changed while the system is running an experiment. A variable is always associated with a module, since the module implements the *get* and *set* functions to access these variables. A variable is defined by the ⟨var⟩ XML tag.

A **parameter** is like a variable but it cannot be changed while the system is running an experiment. Parameters are useful for initializing the internal variables of a module implementation. There are default parameters for modules and views (e.g., the ExecutionTime parameter defines the maximum time estimation in running local code). A parameter is defined by the ⟨param⟩ XML tag.

A **view** defines a GUI form composed of graphical components and multimedia capabilities (video, animation, sound). These components allow users to view and manipulate the remote data defined in an RLAB system. A view is implemented like a module, i.e., with a standardized interface, but different methods are needed. These are:

- *Show( )*. It presents the GUI form to the user.
- *Hide( )*. Used to hide the GUI form and run the finalization tasks.
- *Update( )*. Used to update the graphical components of the GUI form.

The ⟨view⟩ XML tag just defines a view and the syntax looks like a module definition.

A **reference** is a URL that allows information to be published using HTML pages. Any information about local RLAB systems can be added by references. The ⟨reference⟩ XML tag defines this object.

An **experiment** describes the dynamic behavior of an RLAB system, for example, a controlled system, the manual operation, etc. An experiment is defined using the static definitions of modules and variables in the following way:

- An experiment needs to run one or more modules (one is mandatory) that must be running while the experiment is going on, i.e., there will be several threads running concurrently. The system must provide synchronization among these threads for its correct operation regardless of the local or remote nature of these modules (the local or remote execution of modules is possible by means of a ⟨run⟩ tag).
- If a module needs data from another module (remote or local) then input ports must be defined using several ⟨in⟩ XML tags. These input ports make possible to set data on the module from different sources.
- If a module sends data to another module (remote or local) then outport ports must be defined using some ⟨out⟩ XML tags. With these output ports a module get data from different sources (local or remote data).
- If disturbances are introduced in a variable or it is necessary to program timing changes on several variables then the ⟨set⟩ tag is used.
- Finally, experiment duration is set by the ⟨duration⟩ XML tag.

An RLAB system could include as many experiments as the system administrator adds to the definition file and users could also add their own experiments to the definition file.

PUBLISHING AN RLAB SYSTEM

After describing the objects defining an RLAB system, we now list the steps that a teacher has to take to publish an RLAB system in an RLAB control web server. They are as follows:

1. Establish the static behavior of the system in terms of variables and parameters. These variables must be classified into groups, since a system is composed of many kinds of variables and parameters.
2. Define a module implementation for each group of variables and parameters. The teacher chooses a native or pure Java implementation for each module, and then the implementation must be programmed.
3. Develop views. Several GUI forms can visually help to describe and understand the dynamic behavior of the system.
4. Begin the documentation task. The results are HTML pages that are included in the system as references.
5. Define the dynamic behavior. First, the teacher decides what features in the system are public for users. Second, he/she has to write the experiments using the modules defined in the previous static behavior section.

Steps 1–5 produce the XML definition file. This file is loaded into the RLAB Publish Application (see Fig. 2a) for exporting the new system to RELATED users. The definition file is parsed selecting the *Load XML-File* menu item, and if no errors are found, the RLAB system is published in the main web server. From now on, the new RLAB system can be managed across the Internet.

To manage the services available in a published RLAB system, an applet has been developed (see Fig. 2b). This applet allows users to start and stop experiments, show and hide views, or open the references in new windows. This applet also has graphic capabilities in order to show different scopes for the variables defined in the modules.

(a)

(b)

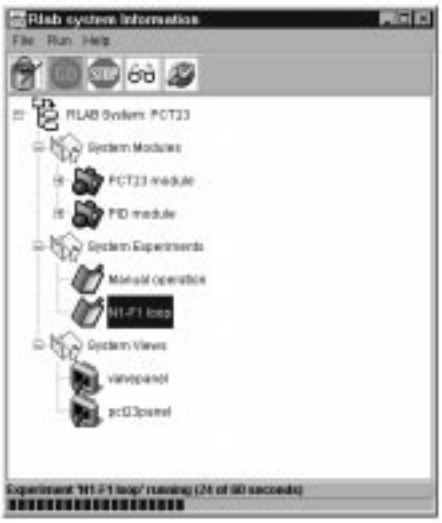

Fig. 2. Interfaces: (a) RLAB Publish Application GUI; (b) main applet window.

BUILDING AN EXPERIMENT

The example presented here uses as its system a small-scale pasteurization plant, the PCT-23 (see Fig. 3). In this plant, the typical working procedure is energy exchange between two liquid flows with different temperatures: cool and warm. A Windows application called CILab (Control Interactive Laboratory) was developed for the local operation of this didactical setup.

As mentioned before, CILab is a standalone application. So, initially, the PCT-23 teleoperation is not possible. However, thanks to the features of the RELATED framework, the CILab code can be reused and the experimentation with the plant across the network can become a reality. In this example, a single PID loop is considered and liquid flow is controlled with a pump.

The procedure for writing the XML definition file of an RLAB system based on this plant is described below. Afterwards, an experiment with both local and remote implementation of a PID controller is carried out. The steps defining the XML file are:

Step 1. In this example, just three PCT-23 variables are considered: F1 (liquid flow), N1 (liquid pump),

Fig. 3. PCT-23 plant.

```
<module name='PCT23 module'>Manual operation module
<param name='ExecutionTime' type='long' value='100'>Thread time</param>
<var name='F1' type='double' initial='0' max='1500' min='0' units='ml/s'>Liquid Flow</var>
<var name='SP_F1' type='double' initial='0' max='250' min='0' units='ml/s'>Setpoint for F1</var>
<var name='N1' type='double' initial='0' max='100' min='0' units='%'>Liquid pump</var>
<implementation type='JNI' libname='pct23.dll'>Native code</implementation>
</module>
```

Fig. 4. XML definition for the 'PCT23 module'.

and SP_F1 (set point for F1). A PID implementation with seven variables and one parameter is also needed:

- 'y'. It represents the plant output (in this case F1).
- 'u'. It is the control action generated for the PID (N1).
- 'ref' represents the set point for the 'y' variable (SP_F1).
- 'Kp'. PID proportional gain.
- 'Ti'. PID integral time in seconds.
- 'Td'. PID derivative time in seconds.
- 'Automatic' variable (boolean value).
- 'Inverse' parameter (boolean value).

Step 2. There are two groups of variables (plant and PID modules) that are implemented separately in modules using the two different types of implementation: native code for the PCT-23 variables, and Java code for the PID variables. These two modules are named 'PCT23 module' and 'PID module'. Both definitions are shown in Figs 4 and 5.

Step 3. Two views (see Fig. 6) were developed for direct user interaction:

- Valves panel, useful for opening and closing PCT23 valves.
- Global diagram, which allows pump values to be changed and other variables in the PCT23 plant to be seen.

Step 4. Currently no references are defined.

At this point, the static behavior section of the XML definition file is already fulfilled and the dynamic behavior must be written, that is to say, the experiments.

Step 5. Two experiments are considered in this contribution:

- *An experience of the PCT23 manual control*: No connections are needed and the experimentation time is set to infinite.

```
<module name='PID module'>PID execution module
<param name='ExecutionTime' type='long' value='50'>Thread time</param>
<param name='INVERSE' type='boolean' value='false'> Inverse action</param>
<var name='Y' type='double' initial='0' max='1000' min='0' units='seconds'>Output variable</var>
<var name='U' type='double' initial='0' max='100' min='0' units='seconds'>Control variable</var>
<var name='REF' type='double' initial='0' max='1000' min='0' units='seconds'>Reference variable</var>
<var name='Kp' type='double' initial='0.1' max='100' min='0' units='N/A'>Proportional Gain</var>
<var name='Ti' type='double' initial='1' max='1000' min='0' units='seconds'>Integral Time</var>
<var name='Td' type='double' initial='0' max='1000' min='0' units='seconds'>Derivative Time</var>
<var name='Automatic' type='boolean' initial='true' max='1' min='0' units='none'>Var for automatic/manual switching</var>
<implementation type='JAVA' jarfile=file://c:/rlab_xml/pid.jar classname='PIDModule'>PID algorithm</implementation>
</module>
```

Fig. 5. XML definition for the 'PID module'.

(a)

(b)

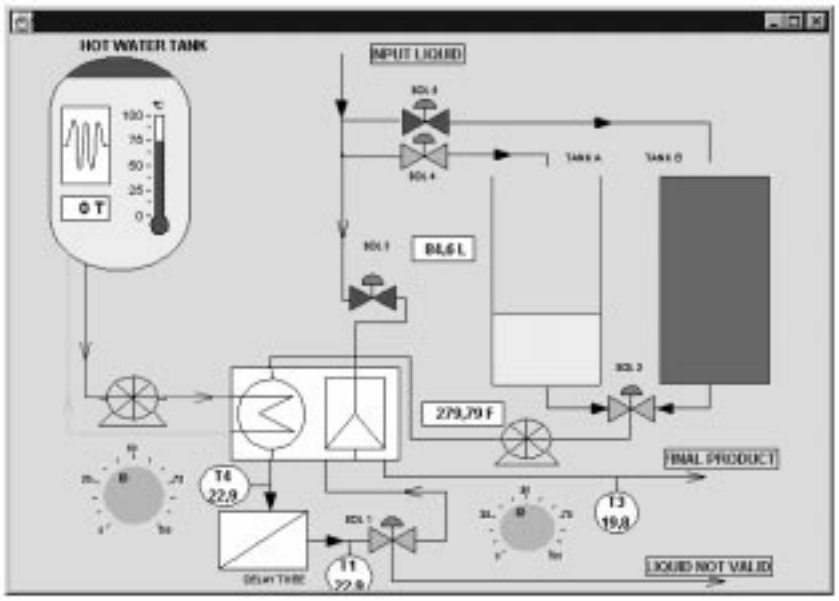

Fig. 6. Views: (a) valves panel (b) PCT23 global diagram.

- *A PID control for F1 and N1*: In this case, connections between the 'PCT23 module' and the 'PID module' have to be established in order to change the N1 value ('u' variable in the 'PID module') and reach the set point. Thus three connections are defined using the ⟨in⟩ and ⟨out⟩ tags. Additionally, some changes in the SP_F1 are programmed in order to verify the correct operation of the PID algorithm.

Both experiment definitions are shown in Figs 7 and 8.

After the previous steps have been completed, one XML definition file is obtained and released to users by means of the RLAB Publish application. Figure 2b shows graphically the two parts of the PCT32 system defined with the previous XML file: the system modules (static section) and the system experiments (dynamic section).

Now, starting and stopping experiments in the plant can be done. Once the PID Control experiment is started, it is possible to select and view real-time data and change 'interactive' variables using the RLAB system information applet (Fig. 9).

Running a remote pid module

The above example shows how to build an experiment using local modules. In this new example, the PID module with a different XML definition file has been moved to an RLAB system located in another machine (see Fig. 10). A new experiment is now defined using the remote PID module and the PCT23 module. The previous PCT23 System experiment is modified for using this remote PID module (see Fig. 11).

EDUCATIONAL OUTCOME

If the existence of remote laboratories resolves many of the problems raised, the creation of consortia or networks of telelabs considerably increases the benefits. Benefits which, on the one hand, affect the two parties involved: the students will have a complete set of activities that do not depend on the available university equipment, whereas the teachers will have different platforms to support their teaching based on the master lesson, thereby minimizing the purchase and

```
<experiment name='Manual operation' sampleTime='1000'>
<duration type='User'>Infinite time</duration>
<run module='PCT23 module'>
<interactive names='SP_F1, N1'/>
</run>
</experiment>
```

Fig. 7. Manual operation experiment.

```
<experiment name='N1-F1 loop' sampleTime='100'>PID Control for F1
<duration type='Time' time='60'/>
<run module='PCT23 module'>
<in name='N1' source='local' module='PID module' var='U'/>
<out name='F1' source='local' module='PID module' var='Y'/>
<out name='SP_F1' source='local' module='PID module' var='REF'/>
<set name='SP_F1' time='0' value='0.0'/>
<set name='SP_F1' time='2' value='200.0'/>
<set name='SP_F1' time='15' value='350.0'/>
<set name='SP_F1' time='30' value='550.0'/>
<set name='SP_F1' time='40' value='415.0'/>
<set name='SP_F1' time='55' value='0.0'/>
<interactive names='N1, SP_F1'/>
</run>
<run module='PID module'>
<set name='Kp' time='0' value='0.1'/>
<set name='Ti' time='0' value='1'/>
<interactive names= 'U, Y, REF'/>
</run>
</experiment>
```

Fig. 8. PID control experiment.

maintenance costs. Additionally, the access to different experimental resources thanks to the net enables the laboratory to be more integrated in the educational curriculum both horizontally and vertically. Horizontally, because students can access laboratories of different subjects (control, robotics, vision, electronics, artificial intelligence, hydraulics, etc.) irrespective of the existence or not of a laboratory at their teaching center. Vertically, because the student can do practical activities of increasing difficulty within the same subject or range of subjects. Consequently, well or badly equipped laboratories do not exist but rather just one environment in which to work, to the advantage of the whole university community. This community can thus offer more integral, complete, varied and state-of-the-art education.

CONCLUSIONS

Although the paradigm of the virtual and remote laboratory accessible via the Internet is presently at a relatively mature stage of development, there are still no standards for defining experiments and connecting teleoperated environment elements. Most researchers carry out similar developments and paradoxically make the mistake that they are precisely trying to solve with the creation of experimentation tele-environments: minimize costs and maximize uses. It is therefore obvious that global initiatives are necessary in RELATED to group, or at least share, all these efforts for a common purpose: the training of pupils.

RELATED is the first release of a new framework developed for connecting control systems

(a)

(b)

(c)

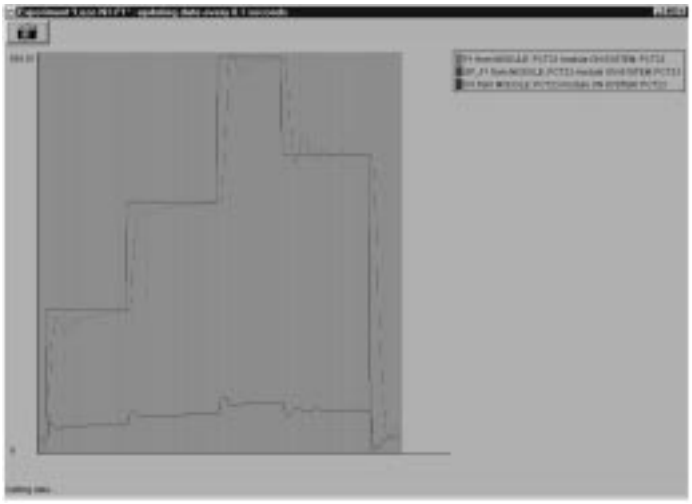

Fig. 9. (a) Selecting data. (b) Changing values. (c) Plotting real-time data from PCT-23.

```
<?xml version='1.0' encoding='utf-8' ?>
<system name='RLAB PID' type='0'>Export a PID System
<module name='PID module'>PID execution module
<param name='ExecutionTime' type='long' value='100'>Thread time</param>
<param name='INVERSE' type='boolean' value='false' >Inverse action</param>
<var name='Y' type='double' initial='0' max='1000' min='0' units='seconds'>Output variable</var>
<var name='U' type='double' initial='0' max='100' min='0' units='seconds'>Control variable</var>
<var name='REF' type='double' initial='0' max='1000' min='0' units='seconds'>Reference variable</var>
<var name='Kp' type='double' initial='0.1' max='100' min='0' units='N/A'>Proportional gain</var>
<var name='Ti' type='double' initial='1' max='1000' min='0' units='seconds'>Integral time</var>
<var name='Td' type='double' initial='0' max='1000' min='0' units='seconds'>Derivative time</var>
<var name='Automatic' type='boolean' initial='true' max='-1' min='-1' units='none'>Var for automatic/manual switching</var>
<implementation type='JAVA' jarfile='file://d:/rlab/PID_JAVA/pid.jar' classname='PIDModule'>PID algorithm</implementation>
</module>
</system>
```

Fig 10. New definition file only with the PID module.

```
<?xml version='1.0' encoding='utf-8' ?>
<system name='FULL REMOTE EXPERIMENT' type='0'>Using two remote modules
<experiment name='REMOTE N1-F1 loop' sampleTime='100'>Remote PID Control for F1
<duration type='Time' time='180'/>
<run module='PCT23 module' type='external' source='//62.204.197.96/PCT23' ExecutionTime='100'>
<in name='N1' source='//62.204.197.98/RLAB PID' module='PID module' var='U'/>
<out name='F1' source='//62.204.197.98/RLAB PID' module='PID module' var='Y'/>
<out name='SP_F1' source='//62.204.197.98/RLAB PID' module='PID module' var='REF'/>
<set name='SP_F1' time='0' value='0.0'/>
<set name='SP_F1' time='10' value='300.0'/>
<set name='SP_F1' time='70' value='200.0'/>
<set name='SP_F1' time='140' value='100.0'/>
<interactive names='SP_F1, N1'/>
</run>
<run module='PID module' type='external' source='//62.204.197.98/RLAB PID' ExecutionTime='100'>
<interactive names='Y, U, REF' />
</run>

</experiment>
</system>
```

Fig. 11. New experiment using the remote PID module.

(real plants, simulations, control algorithms) across the Internet using XML language. This approach lets teachers create new Internet-based control labs using legacy code and control elements regardless of location.

A full example of real-time control via the Internet has been presented. This example shows that the use of local code to develop a complete Internet control application could be quite simple. Finally, remote module execution in a shared-distributed information system has been demonstrated, and how a full PID implementation algorithm can also be used for educational experiments.

For further information about the software's availability and a complete description of the system's details, readers can e-mail the authors.

REFERENCES

1. J. W. Overstreet and A. Tzes, An Internet-based real-time control engineering laboratory, *IEEE Control Systems Magazine*, **19**(5), 1999, pp. 19–34.
2. C. Schmid, A remote laboratory using virtual reality on the Web. *Simulation*, **73**(1), 1999, pp. 13–21.
3. A. Zolnay, A. Lassó, H. Charaf, and I. Vajk, Configurable remote, platform independent control system, in *Advances in Control Education 2000* (L. Vlacic and M. Brisk, eds), Pergamon, Great Britain (2001) pp. 319–324.

4. D. Gillet, H. A. Latchman, Ch. Salzmann, and O. D. Crisalle, Hands-on laboratory experiments in flexible and distance learning, *Int. J. Eng. Educ.*, (2001), pp. 187–191.
5. C. Salzmann and P. Huguenin, A distributed architecture for teleoperation over the Internet with application to the remote control of an inverted pendulum, *Lecture Notes in Control and Information Sciences 258: Nonlinear Control in the year 2000*, 1, Springer-Verlag, London (2001) pp. 399–407.
6. C. C. Ko, B. M. Chen, J. Chen, Y. Zhuang, and K. C. Tan, Development of a Web-based laboratory for control experiments on a coupled tank apparatus, *IEEE Trans. Education*, **44**(1) 2001, pp. 76–86
7. K. Ling, Y. Lai, and K. Chew, An online Internet laboratory for control experiments, in: *Advances in Control Education 2000* (L. Vlacic and M. Brisk, eds), Pergamon, Great Britain (2001) pp. 173–176.
8. W. Sheng, L. Choo-Min, and L. Khiang-Wee, An integrated Internet based control laboratory, in: *Advances in Control Education 2000* (L. Vlacic and M. Brisk, eds), Pergamon, Great Britain (2001) pp. 49–54.
9. J. Sánchez, F. Morilla, S. Dormido, J. Aranda, and P. Ruipérez, Virtual control lab using Java and Matlab: A qualitative approach, *IEEE Control Systems Magazine*, **2**, (2002).
10. Available: http://www.w3.org/XML/
11. Available: http://www.nacimiento.com/VIML/.
12. Available: http://pioneer.gsfc.nasa.gov/public/iml/.
13. T. Ames, L. Koons, K. Sall, and C. Warsaw, Using XML and Java for astronomical instrumentation control, *SpaceOps 2000*, Toulouse, France (2000).

R. Pastor received his Physics degree in 1994 from Madrid Complutense University and he is currently working on his thesis on virtual and remote laboratories. Since 2000 he has been working at UNED Department of Computer Sciences and Automatic Control as an Assistant Professor. His current research interests are real-time control systems, distributed systems, virtual and remote labs and their applications in higher degree courses.

J. Sánchez received his Computer Sciences degree in 1994 from Madrid Polytechnic University and his Ph.D. in Sciences from UNED in 2001. Since 1993 he has been working at UNED Department of Computer Sciences and Automatic Control as an Assistant Professor. His current research interests are the design of new systems for control education, virtual labs, telepresence, multimedia, and the use of the Internet in education.

S. Dormido received his Physics degree from Madrid Complutense University (1968) and his Ph.D. degree with a thesis on 'Adaptive Sampling' from the University of the Basque Country, Spain, in 1971. In 1981, he was appointed Full Professor of Control Engineering at UNED Faculty of Sciences. Since 1986 he has been Head of UNED Department of Computer Sciences and Automatic Control. His scientific activity includes various topics from the control engineering field: computer control of industrial processes, adaptive systems, model-based predictive control, robust control, and modeling and simulation of continuous processes.