An XML Modular Approach in the Building of Remote Labs By Students: A Way to Improve Learning

http://dx.doi.org/10.3991/ijoe.v9iS5.2662

R. Pastor-Vargas, D. Sánchez, S. Ros, R. Hernández, A. Caminero, L. Tobarra,
A. Robles-Gomez, M. Castro, E. San-Cristóbal, G. Diaz, M. Tawfik
Spanish University for Distance Education, Madrid, Spain

Abstract—**Practical knowledge is increasingly getting more attention in the higher education, especially in the field of engineering education. Engineering is a discipline which has, by its own definition, a large amount of practical contents. Allowing students to interact with real equipment can give to them a qualitative knowledge that cannot be obtained in other way. In this situation, remote laboratories are especially useful. This paper presents a way of reusing remote labs code by its XML representation. The idea is that students learn not only the basic subjects of the course, even more, that students do not only manipulate lab equipment. The idea is to give the students a practical way to check the potential of work organization and to give a taste of the powerfulness of code reusing, that is, in the end, a way to improve work efficiency.**

Index Terms—**Reusing code, distributed systems, remote laboratories, engineering education.**

I. INTRODUCTION

In the past days, engineering students were used to manipulate complex mathematic expressions and, in most cases, graduates began his working life without a practical view. But today, increasingly, enterprises demand more specialized students and, what is more important, enterprises demand graduates capable of starting to work as quickly as possible. To improve the learning process in this way, it is important to give to the students a taste of real situations. It's important for the students to develop an intuitive knowledge of the theoretical concepts they are studying. This can be done via laboratory practices.

Laboratory practices are very useful, but in some scenarios like distance education or with a large number of students, another concept is needed. The solution in these scenarios is the use of remote laboratories. There are tons of works about remote laboratories [1-4] but in most cases, they lack a structured development system or the possibility of code reusing [5]. This last feature is very important, especially in case of allowing students to build their own virtual or remote laboratories. It is very easy and simple to provide laboratory modules to students when a researcher/lecturer uses a modular approach to the developing. These modules can be combined with student's modules (like a puzzle) to build new laboratories (virtual o remote). In this case, this is achieved by RELATED.

RELATED is a framework that provides a structured methodology to develop virtual and remote laboratories [6]. Moreover, RELATED has a modular architecture and the laboratories produced using this framework have modular architectures too. This modular architecture is the key piece to allow code reusing. In fact, every laboratory module and even every laboratory developed using RELATED could be expanded, reused or combined with other modules.

It is important to promote motivation between students [7] because motivated students perform better in general competencies, in specific competencies and also acquires better knowledge. Involving students in the development of a remote lab promotes motivation [8] so a development of a remote and virtual laboratory and its application in a university course is presented.

This course is called "Real time systems" and is a part of a degree in Industrial Engineering. The main learning goal is to understand the basics of Real Time Operating Systems (RTOS) and programming techniques, regarding with issues like concurrency, real time clocks and tasks planning. Also, as Industrial Engineers, they need to learn how to represent industrial models used in control processes. So, they have to program a real time process representing a PID (Proportional Integral Derivative) module and also learn how to use it in a distributed way. To achieve this, students will work in the developing of a real-time control system for a DC motor using RELATED. This is done easily thanks to the RELATED modular approach.

II. RELATED MODULAR STRUCTURE

RELATED is a framework focused in the development of remote/virtual laboratories based in the concept of component and using the MVC (Model-View-Controller) paradigm. A laboratory can be defined as a set of components which are part of the laboratory. The components can be categorized as model components (M in MVC paradigm) or view components (V in MVC paradigm). The model components are called "modules" and they will be executed by the RLAB server controller (C in MVC paradigm). The view components will be executed by the RLAB client controller in the user/student computer. Both of them, RLAB server/client controllers, are Java based applications so they can be executed on several operating systems with Java support.

In order to select which components will be used in a practical experience, a new component called "experiment" is defined. The experiment declares which components will be part of the practical experience. As the definition of an experiment is declarative, a XML based language will be used. This language is called LEDML (Experimentation Description Markup Language) and, basically, declares all components of the laboratory (including model, view components and experiments). In Fig. 1, it is shown the relationship between components and laboratories.

So, an RLAB (Remote LABoratory) system it defined using a formal specification based on LEDML. As this specification is written in XML, all components have associated a XML fragment defining its properties. Using this approach, it is possible to reuse "XML fragments" to include the components in other laboratories in an easy way.

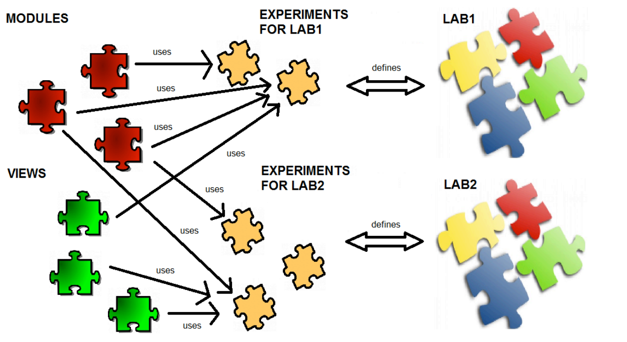


Figure 1. RELATED Modular Structure based on components.

A. *Modules*

As it was mentioned before, the model components are called "modules". These modules, which are run-able entities (Java coded), are running by the RELATED server controller facilities in order to get/set data from/to the equipment, and this data will be sent over Internet to the RELATED client controller.

In order to provide a standardized interface for controlling module execution, the module implementation must offer a predetermined set of methods (implementing the IRLABModule java interface). 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:

1. *Init()*: It returns a Boolean value according to module initialization. It can be used to get information from module or initialize some module behavior.
2. *Start()*: It marks the starting of module execution. Useful to start module operations (like starting communication threads with real equipment)
3. *Run()*: It will be called in every experiment sample. It can be used to update module information.
4. *Pause()*: Used to pause modules that work with simulations.
5. *Resume()*: Used to resume modules that run simulations.
6. *Stop()*: It marks the stopping of module execution.
7. *Exit()*: Used to free resources in the local code implementations.
8. *GetVarValue(var)*: Used to retrieve the value of a variable in the module implementation. This method allows the signal values to be obtained in real time. It is used internally by RLAB controller to update module information.
9. *SetVarValue(var, value)*: Used to change the value of a variable in the module local implementation. Also, it is used internally by RLAB controller to update module information.

Once the module has been implemented (following the before guidelines), it is mandatory to declare its XML fragment. This XML fragment will be part of the laboratory specification, as it will be shown in the next sections.

B. *Views*

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 (IRLABView java interface), but different methods are needed. These are:

1. *Show()*: It presents the GUI form to the user.
2. *Hide()*: Used to hide the GUI form and run the finalization tasks.
3. *ReceiveData(vars)*: Used to update the graphical components of the GUI form.
4. *SendData(vars)*: Used to send data changes in variables associated to the view.

As before, in order to include as a component in the laboratory, it is mandatory to write the corresponding XML definition. One important aspect in the specification of views is the feature to associate modules variables to views. To do that, it can be used the <use> tag in the view specification. For example, to use a variable called "myVar" from a module specification (called "myModule") simply it has to be declared as a <use> tag, indicating the internal name used in the view (property as of the <use> tag):

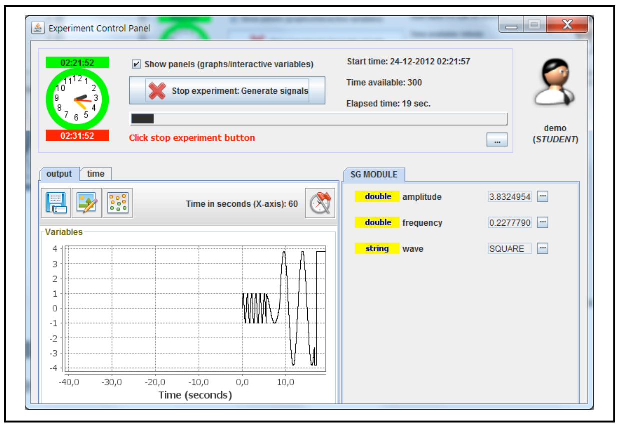
```
<view name="myviewnam">
      …
    <use name="myVar"
         module="myModule"
         as="myVarNameInView"/>
              …
</view>
```

The RLAB controller will do the corresponding assignment in the ReceiveData() method, setting the "myVarNameInView" variable of view with the data from module.

C. *Experiments*

Experiments define the subset of components used in the practical experience, so it does not have implementation issues. Only it is a declarative component. The key idea is declaring modules and views used in the experiment, using the <run> and <open> tags in its declaration.

RELATED provides some useful services like painting variables in trend graphics or the capability to provide internal editors for some type of variables. These capabilities are declared in the experiment definition using the <paint> tag (trend graphics) and <interactives> tag (edition). For every <paint> tag defined in an experiment, one tab is built automatically on the left/bottom part of the experiment panel. It is possible to group some modules variables in one tab, assigning a list of variables names in the “names” attribute of the <paint> tag. In the same way, a tab is created on the right/bottom part of the experimentation panel for the <interactives> tag (one by module). In Fig. 2 is shown the experimentation panel running an experiment. This experiment is defined as part of a virtual laboratory which specifies a signal generator module and the experiment.


Experiment Control Panel
02:21:52
02:31:52
Show panels (graphs/interactive variables)
Stop experiment: Generate signals
Click stop experiment button
demo
(STUDENT)
output
time
Time in seconds (X-axis): 60
Variables
Time (seconds)
SG MODULE

Start time: 24-12-2012 02:21:57
Time available: 300
Elapsed time: 19 sec.

double	amplitude	3.8324954
double	frequency	0.2277790
string	wave	SQUARE

Figure 2. Experimentation panel running an experiment.

TABLE I.
REFERENCE VALUES TO VALIDATE STUDENT’S RTOS INSTALLATION

Command	Max (µs)	Min (µs)	Avg (µs)
cyclictest -t1 -p 80 -n -i 10000 -l 1000 -q	55	26	36
cyclictest -t1 -p 80 -i 10000 -l 1000 -q	120	38	72
cyclictest -t1 -p 80 -i 500 -n -l 1000 -q	63	33	18
cyclictest -t1 -p 80 -i 500 -l 1000 -q	102	30	70

Once the fundamentals of RELATED are detailed, in the next section it will be shown how the students build their own laboratories.

III. PRACTICAL PROCEDURE OF BUILDING MODULAR LABS BY STUDENT

Students do the practical work in several stages, guided by a student’s manual. In the case of the “Real Time Systems” course, these are stages which students must follow:

1. Learn how RELATED works (facilities and general services)
2. Prepare a computer with a RTOS (Real Time Operating System).
3. Develop a PID controller in Real Time Java in a modular way. This PID controller will run as a high priority process in the RTOS prepared in point 2.
4. Develop an RLAB module, in order to communicate with the real time process representing the PID, using a network protocol based on RMI (Remote Method Invocation, [9], [10]). Also the module configuration XML fragment is got.
5. Test the behavior of the controller with a virtual laboratory of a tank, combining the PID module developed with a tank virtual lab (also, module based).
6. Test the behavior of the controller with a remote laboratory comprised of a DC motor, in the same way that the virtual tank laboratory developed in the last step.

Stages 1, 4 and 5/6 are general steps in the building process of virtual/remote labs with RELATED. Stages 2 and 3 are more specific for the learning goals of this course.

Students are provided with perfectly working laboratories, this is, the virtual laboratory of the tank, and the remote module of the DC motor, which are developed previously by the lecturer/professor. Students must reuse them (using only XML tags) to expand its capabilities with the PID controller.

In next sections, details about the different stages will be shown.

IV. REAL TIME PID DEVELOPMENT

There are several ways to develop a real-time system, but in this case student must focus in developing real time software using a prepared computer with real-time capabilities. To achieve this, the first requirement is a computer and a RTOS (Real Time Operating System). A RTOS based on RT Linux [11] will be installed and set-up. The first step is the installation, configuration and checking of the RTOS capabilities.

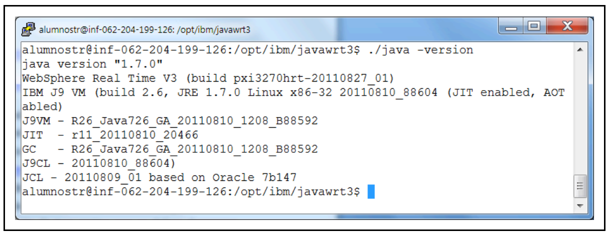
In any case, students are provided with a preinstalled RTOS in a university available computer with the purpose of test and comparing the results or to use it directly, if the student gets stuck with the installation of the RTOS. To access to this RTOS, students must use a standard SSH client (like Putty) and connect to a given IP address with a given login and password.

To perform several tests with the RTOS the cyclictest [12] command is added. This command allows the running of several tests of validation of the RTOS capabilities, in special in the aspect of system latency. For getting more details about the working of this tool, students can run the cyclictest command with the –h option or inspect the [12] reference. In Table 1, the latency values for several cyclictest commands on the preinstalled RTOS (on the university available computer) is given to student, in order to test in their RTOS installation is correct (values must be similar).

When the RTOS is ready, it’s necessary to choose a programming language to implement the real time code. In this case Java with real-time extensions [13] is used (RTJ, Real Time Java). Java was not a very common language in the field of real-time systems in the past but, it is getting more attention, especially since RTJ extensions has changed the working of garbage collector and the use of memory. Also, students have deeper knowledge of Java programming language (in past courses they have programming courses where they learn this programming language).

It is necessary to install a JVM (Java Virtual Machine) with real-time support; in this case IBM Real Time WebSphere will be used [14]. Consequently, the next step is to get the virtual machine working in the RTOS. Student must check that the Java Virtual Machine is working fine; this can be seen in Fig. 3.

Next step is the proper developing of a simple PID, which will be running as a real-time process in the RTOS. This PID must allow communication with external software using a RMI (Remote Method Invocation) interface.



```
alumnostr@inf-062-204-199-126:/opt/ibm/javawrt3$ ./java -version
java version "1.7.0"
WebSphere Real Time V3 (build pxi3270hrt-20110827_01)
IBM J9 VM (build 2.6, JRE 1.7.0 Linux x86-32 20110810_88604 (JIT enabled, AOT
abled)
J9VM - R26_Java726_GA_20110810_1208_B88592
JIT  - r11_20110810_20466
GC   - R26_Java726_GA_20110810_1208_B88592
J9CL - 20110810_88604)
JCL - 20110809_01 based on Oracle 7b147
alumnostr@inf-062-204-199-126:/opt/ibm/javawrt3$
```

Figure 3. Testing the Real Time Java implementation

This external software (the RLAB module) will be located in a different host than the RTOS host. This way, the PID can be used with the tank simulated model and with the DC motor.

To let the PID communicate with RELATED, several steps must be done; first of all, the PID will run as a real-time thread, this is shown in Fig. 4. A real-time thread can be defined in RTJ, extending the RealtimeThread class and configuring the thread parameters (for example, the PriorityParameters object, shown in Fig. 4). Additionally, a run method is needed to implement real time process functionality. In this case, as it is shown in Fig. 5, it can be seen the `run()` method is the way to calculate each control value in a specific time moment. Fig. 6 shows the procedure to start the PID controller, which is in basis to instantiate the thread and calling its start method. Fig. 7 shows the remote interface for RMI communications, which could be used from external RMI client software (using network communication) to get the PID calculated value. There are two important methods defined in the remote interface for controlling the execution of the RealtimeThread in the RTOS host: `startController()` and `stopController()`. These methods will be used in the RELATED module defined in the next section as part of the corresponding code for `init()` and `stop()` methods in a module. This allows controlling the remote execution in the RTOS host of the RealtimeThread model representing the PID.

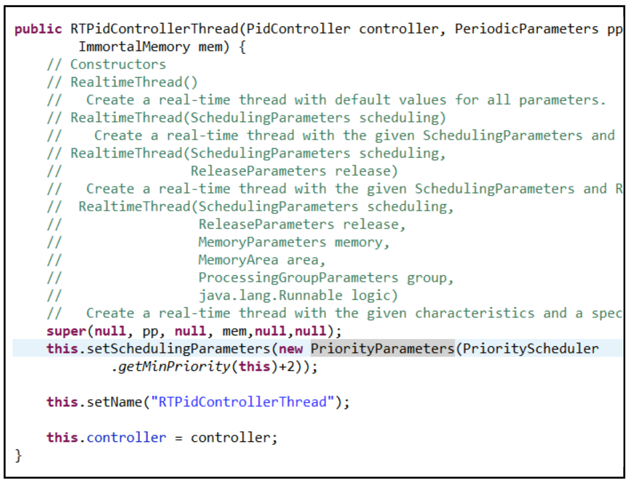
Once the communication interface is added, the process that represents the real-time PID is ready for running in the RTOS.

V. RELATED REAL TIME PID MODULE

Once the real time process representing the PID controller (running in a RTOS system) is ready and running, it is time to develop the RLAB module (mandatory to integrate the real-time PID process with an RELATED module representing simple tank simulation or real time equipment, like the DC motor). In this case, the RELATED module acts as an RMI client to consume information from the real-time PID process.

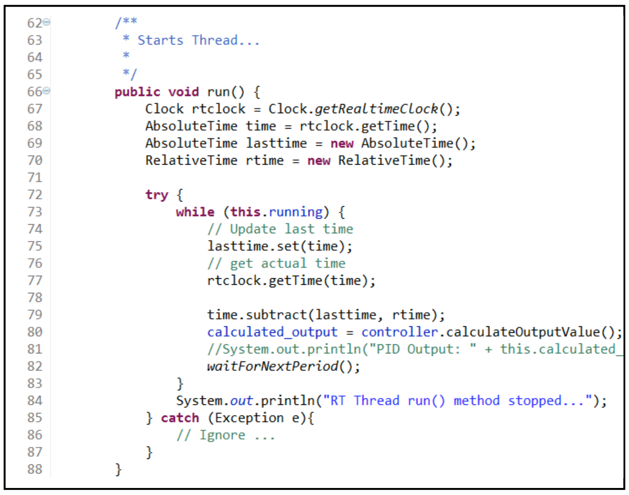
Having in mind the explained specifications, the main class of the RELeATED PID module must implement several specific methods that are defined in the `es.uned.scc.related.modules.IRLABModule` interface (detailed in section 2). This interface is located in the distribution file of the RLAB Server Component (RLABCServerProject.jar file). Students are provided with a copy of the source code, the RLABCServerProject file and also an example of the implementation of a simple random number generator emodule.

To get the RELATED module working, several steps must be followed. To summarize, they are the following:



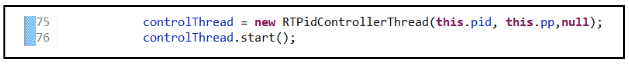
```
public RTPidControllerThread(PidController controller, PeriodicParameters pp
        ImmortalMemory mem) {
    // Constructors
    // RealtimeThread()
    //   Create a real-time thread with default values for all parameters.
    // RealtimeThread(SchedulingParameters scheduling)
    //    Create a real-time thread with the given SchedulingParameters and
    // RealtimeThread(SchedulingParameters scheduling,
    //                ReleaseParameters release)
    //   Create a real-time thread with the given SchedulingParameters and R
    //  RealtimeThread(SchedulingParameters scheduling,
    //                 ReleaseParameters release,
    //                 MemoryParameters memory,
    //                 MemoryArea area,
    //                 ProcessingGroupParameters group,
    //                 java.lang.Runnable logic)
    //   Create a real-time thread with the given characteristics and a spec
    super(null, pp, null, mem,null,null);
    this.setSchedulingParameters(new PriorityParameters(PriorityScheduler
            .getMinPriority(this)+2));

    this.setName("RTPidControllerThread");

    this.controller = controller;
}
```

Figure 4. Implementation of the Real Time Thread



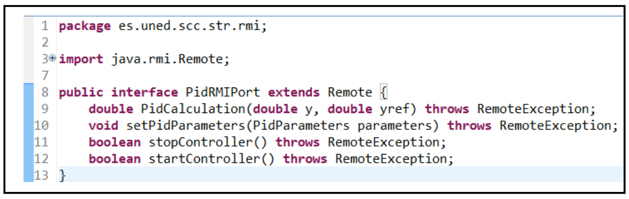
```
        /**
         * Starts Thread...
         *
         */
        public void run() {
            Clock rtclock = Clock.getRealtimeClock();
            AbsoluteTime time = rtclock.getTime();
            AbsoluteTime lasttime = new AbsoluteTime();
            RelativeTime rtime = new RelativeTime();

            try {
                while (this.running) {
                    // Update last time
                    lasttime.set(time);
                    // get actual time
                    rtclock.getTime(time);

                    time.subtract(lasttime, rtime);
                    calculated_output = controller.calculateOutputValue();
                    //System.out.println("PID Output: " + this.calculated_
                    waitForNextPeriod();
                }
                System.out.println("RT Thread run() method stopped...");
            } catch (Exception e){
                // Ignore ...
            }
        }
```

Figure 5. Run method for the PID thread



```
            controlThread = new RTPidControllerThread(this.pid, this.pp,null);
            controlThread.start();
```

Figure 6. Starting the PID thread



```
package es.uned.scc.str.rmi;

import java.rmi.Remote;

public interface PidRMIPort extends Remote {
    double PidCalculation(double y, double yref) throws RemoteException;
    void setPidParameters(PidParameters parameters) throws RemoteException;
    boolean stopController() throws RemoteException;
    boolean startController() throws RemoteException;
}
```

Figure 7. RMI communications interface

I. Modify the XML specification provided with the lab distribution.

II. Develop the main class implementing the IRLABModule interface.

III. Implement every method in the interface, and those methods that could be necessary to complement the laboratory required functionality.

IV. Pack the main class (and the additional classes, if it is the case) in a .jar file.

Fig. 8 shows the XML tags (an XML fragment) needed to define the RT PID module in order to run it in RELATED. A more detailed explanation of LEDML and how to define components by XML fragments is given in

[6] but basically it can be seen that lab developer must specify several basic parameters along with needed variables to get the laboratory working.

In this case, there are three parameters used to communicate with the RTOS host and the Realtime thread running the PID controller: *host* (ip or dns name for the host with the RTOS), *port* (port number for RMI registry needed to get an object reference to the remote interface) and *rmiName* (an string representing the name of the object which is registered in the RMI registry facility). These three parameters allow to the RELATED module get a reference to the remote interface (with RMI) and control/communicate with the RTOS Realtime Thread (basically, the module acts an RMI client).

The rest of variables represent the PID parameters of the controller (Kp, Ti, Td, Beta, Tr, N and h) and the values for the setpoint variable (yref), the calculated value (u) and the manipulated variable (y).

Regarding to programming issues, students must take in account the next issues to properly get the module running and connected to the RTOS host:

- The `init()` method in the RELATED module must establish and init the RMI communication with the RTOS host, getting the proxy and starting the Realtime Thread (invoking the `startController()` method)
- The `run()` method in the RELATED module will call the RMI remote method PidCalculation() using the values of the above variables (yref and y) and it will get the calculated value (u) from the RTOS Realtime PID Thread.
- The `stop()` method in the RELATED module must stop the Realtime PID Thread. To get this, the code will make a method invocation to the `stopController()` method of the RMI remote interface.

VI. TANK MODEL MODULE REUSING: TESTING THE REAL TIME IMPLEMENTATION

Tank model virtual laboratory is composed by two components/entities, one module (representing the model) and a view (GUI with information about evolution of data model). Again, a XML representation for both of them is available (with the jar files implementing them). The module and view definition is shown in figures 9 & 10.

The module and view for the tank model is developed using EJS (Easy Java Simulations) [15], [16]. EJS generate a set of classes which can be reused in the developed code for the module and the view. Only minor modifications must be set in the code before include these classes. Following this procedure, it is very simple to develop virtual laboratories (modules and views) and include them as graphical interfaces in a remote laboratory, not only in this case but in so many more.

EJS allows to lecturers an easy definition of physical models based on ordinary differential equations (ODEs) (EJS allows the use of pure Java code also). One of the main advantages of using EJS is that also use the MVC paradigm, so the model can be decoupled from the java generated code by EJS and used as a model component (module) in RELATED with little effort.

Also, EJS allows the definition of views (like the one it is shown in Fig. 12 (a)) using a built-in editor. EJS allows creating the view providing a set of advanced graphical

```
<module name="RT PID module">
    RT PID execution module
    <param type="long" name="ExecutionTime" value="100">Non RT Thread time</param>
    <param type="String" name="host" value="62.204.199.126">RMI Remote object for RT
        Thread: host ip/dns name</param>
    <param type="int" name="port" value="9900">RMI Remote object for RT Thread: port
        valuefor RMI Registry</param>
    <param type="String" name="rmiName" value="PidRMIPort">RMI Remote object for RT
        Thread: rmi name bound in remote host:port</param>
    <var type="double" name="y" units="seconds" min="0" max="1000" initial="0.2">Output
        variable</var>
    <var type="double" name="u" units="seconds" min="0" max="100" initial="50">Control
        variable</var>
    <var type="double" name="yref" units="m" min="0" max="0.8" initial="0.2">Reference
        variable</var>
    <var type="double" name="Kp" units="N/A" min="0" max="10000" initial="850">Proportional
        Gain</var>
    <var type="double" name="Ti" units="seconds" min="0" max="10000" initial="10.0">Integral
        Time</var>
    <var type="double" name="Td" units="seconds" min="0" max="10000" initial="0.8">Derivative
        Time</var>
    <var type="double" name="Beta" units="N/A" min="0" max="1.0" initial="1.0">Beta</var>
    <var type="double" name="Tr" units="seconds" min="0" max="10000" initial="10.0">Tr</var>
    <var type="double" name="N" units="N/A" min="0" max="10000.0" initial="5">N</var>
    <var type="double" name="h" units="ms" min="0.01" max="10.0" initial="0.01">h</var>
    <implementation type="JAVA"
        classname="es.uned.scc.related.modules.rt.RTPidControllerModule"
        jarfile="../examples/rtpid/code/RTModule.jar">RT PID, Java
        implementation</implementation>
</module>
```

Figure 8. XML definition for RT PID module

```
<module description="EJS tank model" name="Tank">
    <param type="long" name="ExecutionTime" value="100">Thread time</param>
    <param type="double" name="A" value="0.00785">Tank area</param>
    <param type="double" name="a" value="0.000314">Flow out area</param>
    <param type="double" name="g" value="9.82">Gravity constant</param>
    <param type="double" name="ku" value="0.0001">Pump constant</param>
    <var type="double" name="level" units="m" min="0" max="0.5" initial="0.2">Level</var>
    <var type="double" name="pump" units="seconds" min="0" max="100" initial="50">Pump value
        (0-100%)</var>
    <var type="double" name="qin" units="m3/s" min="0" max="0.001" initial="0.0006">In flow</var>
    <var type="double" name="qout" units="m3/s" min="0" max="0.001" initial="0.000622">Out flow</var>
    <implementation type="JAVA" classname="es.uned.scc.related.modules.models.simpletank.SimpleTank"
        jarfile="../examples/tank_model/code/EJSSimpleTankModelModule.jar">EJS
        simulation</implementation>
</module>
```

Figure 9. XML definition for Tank model

```
<view description="Virtual Tank developed with EJS" name="Virtual Tank"
classname="es.uned.scc.related.views.tank.TankEJSView"
jarfile="../examples/tank_model/code/TankView.jar"
helperUrls="../examples/tank_model/code/ejs_tank_view2.jar">
    <param type="boolean" name="allow_manual" value="true">Allows pump changes from view</param>
    <use name="level" as="level" module="Tank"/>
    <use name="pump" as="pump" module="Tank"/>
    <use name="qin" as="Flow_in" module="Tank"/>
    <use name="qout" as="Flow_out" module="Tank"/>
</view>
```

Figure 10. XML definition for virtual view of tank model

elements that build on top of both standard Java Swing components (containers, buttons, text fields, sliders, combo boxes,…) and on specific scientific two- and three-dimensional visualization classes from the Open Source Physics project (particles, vectors, images, vector and scalar fields,…). These elements are used in a simple drag-and-drop way to build the interface

The views components of EJS can be easily added as a RELATED view in the same way as a module.

In order to reuse both of them (module and view for tank model), students only have to copy the XML fragments in Fig. 9 and 10 for the module and the view, and get access to the implementation files. Finally, they must complete their laboratory description defining an experiment which uses their developed module and tank laboratory components (module and view). Students only have to modify a pre-defined XML template which is distributed along the rest of jar files and other additional software. The experiment definition is it shown in Fig. 11. In this definition, the two modules will be run (see the <run> tags in the figure) in order to produce its associated data, and the data needed by the PID module (control variable, u, and measured variable, y) will be connected with its corresponding tank modules variables (pump value and tank level). To achieve this, there are two tags called <in> and <out> which allows to RELATED to do the work automatically, setting the values before (in) and after (out) computing the module code for an experiment sampling.

Once the laboratory description is completed, it can be running using the RLAB application, which provides facilities to run the experiment. Fig. 12 shows an experimental session. This experimental session is what the student see when he/she is manipulating the laboratory. There are two graphical components: one corresponding to the virtual view of tank model (as it's defined in Fig. 10) and another corresponding to the experimentation panel.

In this experimentation panel (Fig.12 (b)), students can change the PID parameters (selecting "RT PID Module" tab at right) using the integrated editors (provided by RELATED facilities) and view the level evolution on the trend graphics tabs at the left. Using this visual representation, students can validate the behavior of their PID implementations.

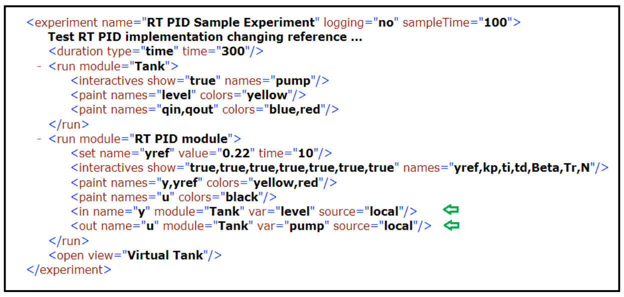
VII. REMOTE LAB DC MODULE REUSING: TESTING THE REAL TIME IMPLEMENTATION

This is the final part and it is no need to develop anything. At this time, every necessary module is available and, like in the case of the tank model, it is only necessary to modify the XML configuration file of the laboratory. The difference in this case is that the module that allows the interaction with the DC motor is located in a different lab/computer, then, it is necessary to identify it inside the student's laboratory configuration file.

As it is a remote module there is no need to specify it as a module in the student's laboratory XML file, but with the purpose of knowing its structure, students are provided with the original DC motor laboratory XML configuration file, that could be seen in Fig. 13.

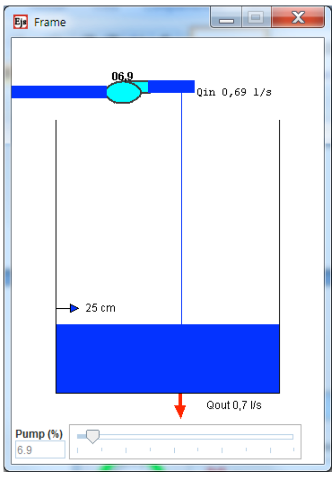
So, in this experiment students will reuse the code associated to the "Manual Module" module of the motor (module named "motorRemotoMan"). In this case, the PID module variables (u,y) will be "connected" with the motor modules variables (voltage, speed), so an speed control experiment will be defined. In Fig. 14 it is show the XML fragment representation of this experiment.

To include the remote execution of motor module, the <run> tag must include in the "type" attribute the "remote" value to indicate RELATED to run the module in the "source" laboratory. The "source" attribute identifies uniquely a RELATED lab using its IP, its communication port and a GUID (Globally Unique Identifier) generated by RELATED. Also, it is possible to get information of the remote module to paint the evolution of modules variables (<paint> tag) and change its values (<interactive> tag). For example, in this case, students will be able to turn off/on the illumination system associated to the lab (setLigth variable).

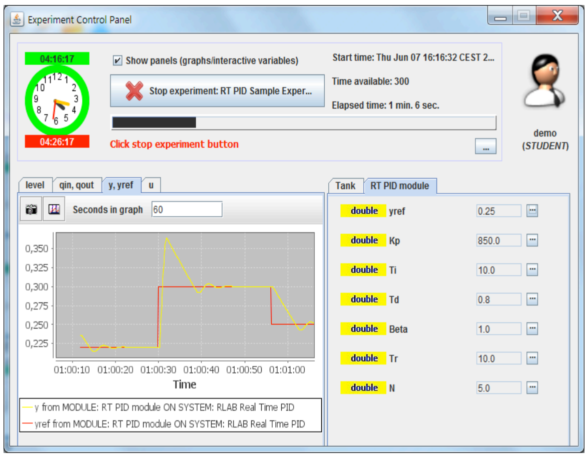


```
<experiment name="RT PID Sample Experiment" logging="no" sampleTime="100">
    Test RT PID implementation changing reference ...
    <duration type="time" time="300"/>
  - <run module="Tank">
        <interactives show="true" names="pump"/>
        <paint names="level" colors="yellow"/>
        <paint names="qin,qout" colors="blue,red"/>
    </run>
  - <run module="RT PID module">
        <set name="yref" value="0.22" time="10"/>
        <interactives show="true,true,true,true,true,true,true" names="yref,kp,ti,td,Beta,Tr,N"/>
        <paint names="y,yref" colors="yellow,red"/>
        <paint names="u" colors="black"/>
        <in name="y" module="Tank" var="level" source="local"/>
        <out name="u" module="Tank" var="pump" source="local"/>
    </run>
    <open view="Virtual Tank"/>
</experiment>
```

Figure 11. Tank experiment definition



(a) Virtual view of Tank model



Start time: Thu Jun 07 16:16:32 CEST 2...
Time available: 300
Elapsed time: 1 min. 6 sec.

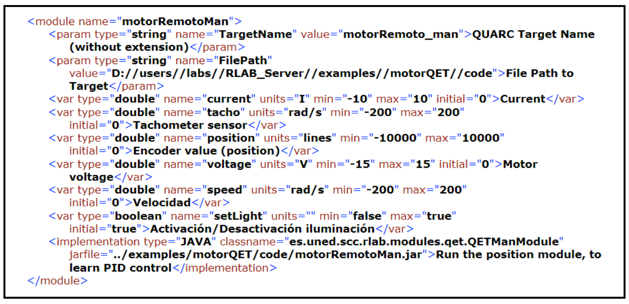
Seconds in graph 60

— y from MODULE: RT PID module ON SYSTEM: RLAB Real Time PID
— yref from MODULE: RT PID module ON SYSTEM: RLAB Real Time PID

double	yref	0.25
double	Kp	850.0
double	Ti	10.0
double	Td	0.8
double	Beta	1.0
double	Tr	10.0
double	N	5.0

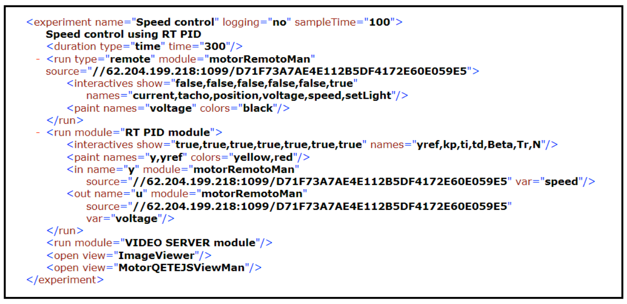
(b) Experimentation panel running the experiment

Figure 12. Experimental session with the tank simulation



```
<module name="motorRemotoMan">
    <param type="string" name="TargetName" value="motorRemoto_man">QUARC Target Name
        (without extension)</param>
    <param type="string" name="FilePath"
        value="D://users//labs//RLAB_Server//examples//motorQET//code">File Path to
        Target</param>
    <var type="double" name="current" units="I" min="-10" max="10" initial="0">Current</var>
    <var type="double" name="tacho" units="rad/s" min="-200" max="200"
        initial="0">Tachometer sensor</var>
    <var type="double" name="position" units="lines" min="-10000" max="10000"
        initial="0">Encoder value (position)</var>
    <var type="double" name="voltage" units="V" min="-15" max="15" initial="0">Motor
        voltage</var>
    <var type="double" name="speed" units="rad/s" min="-200" max="200"
        initial="0">Velocidad</var>
    <var type="boolean" name="setLight" units="" min="false" max="true"
        initial="true">Activación/Desactivación iluminación</var>
    <implementation type="JAVA" classname="es.uned.scc.rlab.modules.qet.QETManModule"
        jarfile="../examples/motorQET/code/motorRemotoMan.jar">Run the position module, to
        learn PID control</implementation>
</module>
```

Figure 13. XML fragment of the DC motor lab



```
<experiment name="Speed control" logging="no" sampleTime="100">
    Speed control using RT PID
    <duration type="time" time="300"/>
  - <run type="remote" module="motorRemotoMan"
    source="//62.204.199.218:1099/D71F73A7AE4E112B5DF4172E60E059E5">
        <interactives show="false,false,false,false,false,true"
            names="current,tacho,position,voltage,speed,setLight"/>
        <paint names="voltage" colors="black"/>
    </run>
  - <run module="RT PID module">
        <interactives show="true,true,true,true,true,true,true" names="yref,kp,ti,td,Beta,Tr,N"/>
        <paint names="y,yref" colors="yellow,red"/>
        <in name="y" module="motorRemotoMan"
            source="//62.204.199.218:1099/D71F73A7AE4E112B5DF4172E60E059E5" var="speed"/>
        <out name="u" module="motorRemotoMan"
            source="//62.204.199.218:1099/D71F73A7AE4E112B5DF4172E60E059E5"
            var="voltage"/>
    </run>
    <run module="VIDEO SERVER module"/>
    <open view="ImageViewer"/>
    <open view="MotorQETEJSViewMan"/>
</experiment>
```

Figure 14. RT PID Speed control

The next step in the experiment's declaration consists in the specification of variable's connection between modules. In the same way, students can use a declarative connection using the <in> and <out> tags inside a running module. For this practical experience the student's PID controller component is generating a reference signal for the voltage in the DC motor, so students have to connect the output variable of their module component ("u" value) to the voltage defined in the "motorRemotoMan" component (this module has a variable named "voltage"). In Fig. 14, it's shown this connection. The <out> tag is defined in the running "RT PID Module", so the attribute "name" must be associated with a module variable from this component. Then, the rest of attributes must be declared using the "source" attribute (remote location of remote module), the "module" attribute (name of the remote module) and "var" attribute (remote variable declared in the remote module).

Finally, to complete the experiment definition, students can define which views will be presented during the experiment (<open> tag) and services presented in the experimentation panel (graphics and interactives variables). The experimentation panel is shown in figure 15, showing the evolution of defined experiment including the remote components.

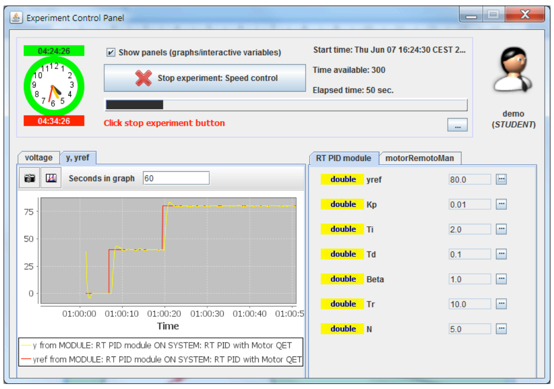
Again, once the experiment definition is finished, it can be loaded from the RLAB application in order to run the experiment. In Fig. 15, the "Speed control" experiment in an experimental session is shown. In Fig. 16, there also two additional views used in the experiment, previously developed for the motor lab which are reused using their XML fragments and jar files. One of them provides a virtual view of motor evolution and the other provides visual feedback of motor.

The visual feedback view (see Fig. 16(b)) is associated to the video streaming of images from a network camera (from AXIS Company). Really, the view is supported by the use of one specific module named "VIDEO SERVER" and really acts as a client of this module, receiving images. Also, the view has PTZ (Pan, Tilt and Zoom) features so the student can do some of these operations to get a better image. Additionally, it is possible to record an experimental session, using the corresponding icon on the right panel of view.

VIII. CONCLUSIONS

RELATED provides a standardized way of developing remote/virtual laboratories and useful services and facilities (users management, booking system, etc.). The clean separation between data and presentation (modules and views) has several advantages, but the main is the software reusing (code develop for this lab or others). In the case of the DC motor system, on-site practical experience software is reused and also, previous developed modules/views (for example, the video server and image viewer) are included with no extra development cost.

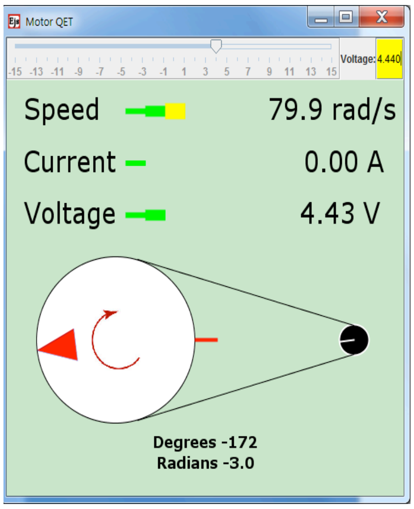
The procedure to reuse an already defined and operational remote laboratory is based on RELATED capabilities; there are definition XML fragments to reuse developed elements so it is an easy task to include them in the student's labs and provide an extensible way of building remote/virtual labs. With these features, students can develop their own virtual and remote laboratories, using the structured methodology of RELATED. The "module" paradigm allows to students develop their own code, and integrate with "real" experiments simply creating/modifying the XML specifications (fragments) which defines the behavior of systems/experiments.


Experiment Control Panel
04:24:26
04:34:26
Show panels (graphs/interactive variables) ✓
Stop experiment: Speed control
Start time: Thu Jun 07 16:24:30 CEST 2...
Time available: 300
Elapsed time: 50 sec.
Click stop experiment button
demo (STUDENT)
voltage | y, yref
Seconds in graph: 60
y from MODULE: RT PID module ON SYSTEM: RT PID with Motor QET
yref from MODULE: RT PID module ON SYSTEM: RT PID with Motor QET

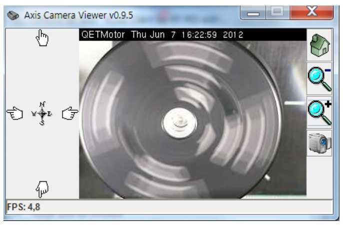
RT PID module | motorRemotoMan

double	yref	80.0
double	Kp	0.01
double	Ti	2.0
double	Td	0.1
double	Beta	1.0
double	Tr	10.0
double	N	5.0

Figure 15. Experiment control panel for DC motor


Motor QET
Voltage: 4.440
Speed 79.9 rad/s
Current 0.00 A
Voltage 4.43 V
Degrees -172
Radians -3.0

(a) Virtual view of motor


Axis Camera Viewer v0.9.5
QETMotor Thu Jun 7 16:22:59 2012
FPS: 4,8

(b) Real time image from motor

Figure 16. GUI for DC motor laboratory

ACKNOWLEDGMENT

The authors wish to thank the Spanish Ministry of Science and Innovation and the National Plan R&D TIN2008-06083-C01/TSI and TIN2008-06083-C03/TSI "s-Labs: Integration of open services for remote and distributed virtual laboratories, reusable and safe". Also authors would like to acknowledge the support of the Project 142788-2008-BG-LEONARDO-LMP mPSS – mobile Performance Support for Vocational Education and Training Project and IPLECS Project – Internet-based Performance-centered Learning Environment for Curricula Support Project ERASMUS 141944-LLP-2008-1-ES-

ERASMUS-ECDSP. Finally, the authors want to acknowledge the support provided by e-Madrid Project, S2009/TIC-1650, "Investigación y Desarrollo de tecnologías para el e-learning en la Comunidad de Madrid".

REFERENCES

[1] Carlos A. Jara, Francisco A. Candelas, Santiago T. Puente, Fernando Torres, Hands-on experiences of undergraduate students in Automatics and Robotics using a virtual and remote laboratory, Computers & Education, Volume 57, Issue 4, December 2011, Pages 2451-2461

[2] Sergio Martin, Gabriel Diaz, Elio Sancristobal, Rosario Gil, Manuel Castro, Juan Peire, New technology trends in education: Seven years of forecasts and convergence, Computers & Education, Volume 57, Issue 3, November 2011, Pages 1893-1906 http://dx.doi.org/10.1016/j.compedu.2011.04.003

[3] E. Fabregas, G. Farias, S. Dormido-Canto, S. Dormido, F. Esquembre, Developing a remote laboratory for engineering education, Computers & Education, Volume 57, Issue 2, September 2011, Pages 1686-1697 http://dx.doi.org/10.1016/j.compedu.2011.02.015

[4] Hardison, J.L.; DeLong, K.; Bailey, P.H.; Harward, V.J.; , "Deploying interactive remote labs using the iLab Shared Architecture" Frontiers in Education Conference, 2008. FIE 2008. 38th Annual , vol., no., pp.S2A-1-S2A-6, 22-25 Oct. 2008.

[5] A. Nourdine, R. Pastor, G. Vivas, "Limitations of remote laboratories in control engineering education". International Journal of Online Engineering, on vol. 6, pp. 31-33, 2010.

[6] Pastor, R.; Sanchez, D.; Aliane, N.; Hernandez, R.; Mariscal, G.; Robles-Gomez, A.; Caminero, A.; Ros, S.; Tawfik, M.; Cristobal, E.S.; Diaz, G.; Castro, M.; , "Structured remote laboratory development," Technologies Applied to Electronics Teaching (TAEE), 2012 , vol., no., pp.314-319, 13-15 June 2012

[7] Xia, H.-W. "An exploratory study on specialized high school education and instruction." Technology and Vocational Education, vol. 28 p. 32–34. 1995

[8] Pastor, R; Sanchez, D; Nourdine Aliane, Roberto Hernández, Antonio Robles-Gómez, Agustín Caminero, Salvador Ros, Gabriel Diaz, Manuel Castro; "Practical experiences on building structured remote and virtual laboratories from the student's point of view" Frontiers in Education Conference, 2012.

[9] http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html, online, last access on 15/05/2013

[10] Wellings, A.; Clark, R.; Jensen, D.; Wells, D.; "A. framework for integrating the real-time specification for Java and Java's remote method invocation," Object-Oriented Real-Time Distributed Computing, 2002. (ISORC 2002). Proceedings. Fifth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, vol., no., pp.13-22, 2002. http://dx.doi.org/10.1109/ISORC.2002.1003655

[11] http://rt.wiki.kernel.org/, online, last access on 15/05/2013

[12] https://rt.wiki.kernel.org/index.php/Cyclictest, online, last access on 15/05/2013

[13] Luder, A.; Peschke, J.; Heinze, M.; "Control programming using Java," Industrial Electronics Magazine, IEEE, vol.2, no.2, pp.19-27, June 2008. http://dx.doi.org/10.1109/MIE.2008.923518

[14] http://www-01.ibm.com/software/webservers/realtime/, online, last access on 15/05/2013.

[15] http://fem.um.es/Ejs/, online, last access on 15/05/2013.

[16] S. Dormido, C. Martín, R. Pastor, J. Sánchez, F. Esquembre, "Magnetic levitation system: a virtual lab in easy Java simulation", American Control Conference, on proceedings vol. 4, pp. 3215-3220, 2004.

AUTHORS

R. Pastor-Vargas is with the Control and Communication Systems Department, Spanish University for Distance Education, UNED, Senior Lecturer, IEEE Member (e-mail: rpastor@scc.uned.es)

D. Sánchez is with the Control and Communication Systems Department, Spanish University for Distance Education, UNED, Researcher (e-mail: dsanchez@scc.uned.es)

S. Ros is with the Control and Communication Systems Department, Spanish University for Distance Education, UNED, Senior Lecturer, IEEE Senior Member (email: sros@scc.uned.es)

R. Hernández is with the Control and Communication Systems Department, Spanish University for Distance Education, UNED, Senior Lecturer, IEEE Senior Member (e-mail: roberto@scc.uned.es)

A. C. Caminero, at the Control and Communication Systems Department of Spanish University for Distance Education, UNED, Assistant Professor, IEEE Member (email: accaminero@scc.uned.es)

L. Tobarra is with the Control and Communication Systems Department, Spanish University for Distance Education, UNED, Assistant Professor, IEEE Member (email: llanos@scc.uned.es)

A. Robles-Gómez, is with the Control and Communication Systems Department, Spanish University for Distance Education, UNED, Assistant Professor, IEEE Member (email: arobles@scc.uned.es)

M. Castro is with the Electrical, Electronic and Control Department, Spanish University for Distance Education, UNED, Full Professor, IEEE Fellow Member (email: mcastro@ieec.uned.es)

Gabriel Diaz, is with the Electrical, Electronic and Control Department, Spanish University for Distance Education, UNED, Associate lecturer, IEEE Member, (email: gdiaz@ieec.uned.es)

E. San-Cristóbal, is with the Electrical, Electronic and Control Department, Spanish University for Distance Education, UNED, Assistant Professor, IEEE Gold Member (email: elio@ieec.uned.es)

M. Tawfik, is with the Electrical, Electronic and Control Department, Spanish University for Distance Education, UNED, Researcher, IEEE Member, (email: mtawfik@ieec.uned.es)

This article is an extended and modified version of a paper presented at the EDUCON2013 conference held at Technische Universität Berlin, Berlin, Germany from March 13-15, 2013. Received 15 May 2013. Published as resubmitted by the authors 27 May 2013.