A Componentizable Server-Side Framework for Building Remote and Virtual Laboratories.

http://dx.doi.org/10.3991/ijoe.v8iS4.2278

Jesus L. Muros-Cobos, Juan A. Holgado-Terriza
University of Granada, Granada, Spain

***Abstract*—Currently, virtual/remotes laboratories are often being built to improve learning and researching capabilities in some areas of knowledge. Generally these virtual/remotes laboratories are built from scratch again and again, instead of reusing software and hardware infrastructures. This paper presents a new framework, RVLab, to help developers building flexible and robust server-side virtual and remotes laboratories quickly. RVLab affords support for the basic requirements of these systems such as the user management or the resources (instruments and devices) reservation. Unlike other lab systems, RVLab is adapted to devices and instruments of any real laboratory due to a secure and robust mechanism that allows the remote execution of lab programs. Moreover, it improves the user interaction with real labs, providing a real-time visualization of experiments and lab instruments by means of the control of video camera placed into lab, and the transmission of video streaming with different quality to users.**

Index Terms—Server-side framework, remote laboratory, virtual laboratory

I. INTRODUCTION AND BACKGROUND

In university environment, especially in technical and scientific degrees, it is usually necessary to manage several sophisticated lab instruments for student formation, in many cases with a restrictive use, due to the high costs associated with respect to the deployment, startup and maintenance of these systems. A frequent strategy to improve the utilization of these expensive labs, reducing at the same time the overall costs, is the inclusion of (some kind of) remote support or simulation capability. Those inclusions do not significantly affect the effectiveness and educational objectives achieved by a physical laboratory [1]. However, the integration of hardware and software technologies into a remote or virtual laboratory supposes a very specific design, not reusable in general for others labs.

Many confusing terms such as on-line, web-based, remote, distance are mostly employed to describe and to identify a remote laboratory in contrast with the traditional one. Eick [2], for example, takes the term web lab to highlight the web nature of the used tools and technologies, while Garcia-Zubia et al. [3] focus the importance on the distance between the laboratory and the computer used to manage the lab. However, the system architecture of both proposals is very similar. Couter et al. [4] carried out a study analyzing the terms used to characterize a remote lab in a wide set of papers from the bibliography, and he found that the differences basically depends on the available features, the technologies used, the purpose or scientific interest of the lab and, finally, the functionality or possible activities performed by the lab.

In order to avoid confusion and misunderstanding [5], we define a laboratory computing system (LCS) as a set of hardware-software components and equipment necessary to perform any researches, experiments, scientific or technical work. The LCS can be considered a Virtual Lab when the lab carried out simulations or a partial emulation of the equipment available on a real lab. Instead, a Remote Lab (RL) refers to the set of hardware-software components which allows the access and control of real instruments from any location in the world through a network (e.g. Internet). Therefore a virtual/remote lab (VRL) denotes a combined LCS which can operate with simulation and real instruments. Similar definitions are found in other paper [6]. In some examples of LCS are the following: RLs for measuring instrument [7][8], VRLs for program microcontroller [1], RLs for networking [9], general RLs[10][11]. An inspection of the features and capabilities of VRL reveals that there are some basic services and applications shared by all the LCS such as the management of users and lab resources, lab reservation, the scheduling, control and monitoring of experiments or experimental sessions with laboratory system. Most of the applications are developed specifically for every LCS, instead of developing generic components that should be integrated into laboratory system, making difficult the reusability of LCS [12].

Despite the advances in software and hardware platforms (or infrastructures), there are some aspects to be improved in LCS. A user rich experience with the lab in learning environments requires a continuous feedback of students in order to enforce the learned concepts in experimental sessions. An active way to achieve that reinforcement is by enabling the interaction with instructor and other students during lab sessions [13], for example, with the use of collaborative tools such as discussion boards, online conferencing system or concurrent live chat. Another option is adding video-camera management in order to improve the perception of user interaction with the lab.

The laboratory must organize the data generated (experimental data, measurements, video streaming, images, etc.) during an experimental session, and then transmit it to the users in order to give them the perception of a real interaction with the lab. Couteur [4] pointed “Seeing is believing – using a camera to see the experiment is important to the student.” in order to incise that real-time visualization of the laboratory with cameras is an important data source to be managed by the LCS. Thus, an effective coordination and synchronization should be performed with heterogeneous data sources provided to the user, and not individual unrelated components such system with virtual desktop [14] or the use of webcam used only with VNC [5].

The framework, RVLab, is proposed to implement server side LCS systems to control remotely real and virtual labs in an easy and free fashion. RVLab provides a set of components adaptable to any LCS in two ways. Firstly, it delivers a set of common basic pluggable modules for the general management (users, lab resources, reserve time) of any LCS, deployed on a server, not necessary coupled with the LCS. The server hides the localization of lab resources to users enabling the security of system and provides a common endpoint to the startup of client applications.

Secondly, RVLab includes an extensible component, Instrument Management Subsystem, for the handling of specific data sources (experimental data or images) from physical instruments and the injection of commands from client to instruments. RVLab supplies a set of communicating channels adaptable to several communication protocols between the instruments and the clients. Furthermore, RVLab acts as a bridge between instruments and the clients, monitoring and supervising the data and commands between counterparts. The mechanism is very flexible and not intrusive in the sense that it releases to developers the way that client applications can handle the instruments of a RVL lab system. Therefore, developers must program the client applications within the context of physical lab, and RVLab extents its remote execution deriving the data and commands through secure communication channels.

The rest of the paper is organized as follows. Section 2 introduces presents the architecture and components of RVLab. Section 3 describes how add new instruments to RVLab step by step and how communicate the instrument with clients. Section 4 details the proposed method to add new cameras and how the system can be changed to storage user data. Section 5 explains a study case of how it is applied on a practical domain. Section 6 presents the performance evaluation of RVLab achieved in the study case. Section 7 describes some related works. Finally, Section 8 exposes the conclusions of the paper.Architecture

RVLab is a componentizable framework which builds online lab systems, remote and virtual, adaptable to any instruments of the lab, reducing the implementation tasks only with respect to the instrument to be virtualized or to be accessible remotely. RVLab produces lab systems with three-tier architecture as it is shown in figure 1. Each tier represents a logical piece of the lab system placed on a computer or device into the network with a specific responsibility with respect to overall system. For instance, instrument-side applications are in charge of virtualizing the real instruments of the lab, and give to server-side applications, an interface to access and control the instruments. Server-side applications arrange data from instruments and hide the management of instrument resources to client applications, and finally client-side applications is responsible to show to an user (or several ones) the state of the controlled environment where the experiment is occurred in base of data obtained from instruments, and the corresponding user's interface with panels in order to allow the supervision and control of the lab system. The insertion of a server-side tier improves the security of the

TABLE I.
COMMANDS OF THE API OF SERVER-SIDE COMPONENTS

Call	Input	Output
User management		
addUser	String:id, String:pass, String:name, String:surname, String:group, String:type, String:adminId, String adminPass.	"success" or descriptive error.
modifyUser	String:id, String:pass, String:name, String:surname, String:group, String:type, String:adminId, String adminPass.	"success" or descriptive error.
addType	String:name, String:priority, String:year, String:web,String:group, String:professor, String:adminId, String:adminPass	"success" or descriptive error.
changePass	String:id, String:oldPass, String:newPass	"success" or descriptive error.
removeUser	String:id, String:adminId, String:adminPass	"success" or descriptive error.
removeType	String:type, String:adminId, String:adminPass	"success" or descriptive error.
removeType-ByYear	String:year, String:adminId, String:adminPass	"success" or descriptive error.
clearByType	String:type, String:adminId, String:adminPass	"success" or descriptive error..
clear	String:adminId, String:adminPass	"success" or descriptive error..
listUser	String:value, String:filter, String:adminId, String:adminPass	Array with all data.
listMyType	String:id, String:pass	Array with all types of the user.
login	String:id, String:pass	success" or descriptive error..
Turns management		
order-Ticket<number>	String:id, String:pass	Boolean true if user has been added to queue, false if there was a error
giveTurn<number>	String:id, String:pass	void
Reserves management		
addReserve	String:beginDay, String:endDay, String:beginHour, String:endHour, String:User, String:repetitions, String:user, String:pass	"success" or descriptive error..
modifyReserve	array:data	"success" or descriptive error..
deleteReserve	array:data	String "success" if ok, else descriptive error.
getReserves	String:user, String:pass	String with dat in xml format
Video management		
addCamera	array:data	int code, 0 if success
modifyCamera	array:data	int code, 0 if success
removeCamera	int:number String:adminId, String:adminPass	int code, 0 if success
moveCamera	int:number, int:direction, String:dni, QString:pass	int code, 0 if success
moveSpeed-Camera	int:number, int:velocity, String:dni, String:pass	int code, 0 if success
zoomCamera	int:number, bool:in, String:dni, String:pass	int code, 0 if success
Instrument management		
addInstrument	array:data	int code, 0 if success
modifyInstru-ment	array:data	int code, 0 if success
removeInstru-ment	int:number String:adminId, String:adminPass	int code, 0 if success

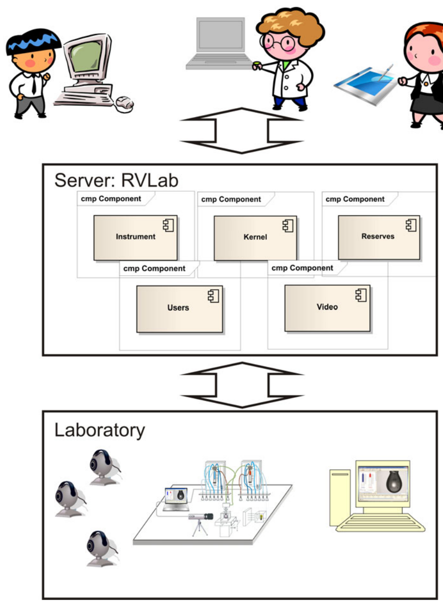

Figure 1. Componets of RVLab framework

lab system, hiding the physical address of all instrument resources. In contrast, the exposition of any instrument resources directly to user (e.g., a webcam [5]) can be dangerous with respect to security and it should be avoid [10] [11].

RVLab facilitates the design of lab systems providing a set of server-side pluggable components which simplify the management of lab system. These components (figure 2) can be selected and parameterized at runtime using program configuration. By default RVLab supports the following modules:

a) User management subsystem. It checks the user's identity and controls his or her permission levels of any lab resources. This subsystem can manage an individual user or multiple users organized in groups. The user's list can be stored in XML files or a database depending on the number of users to be controlled. RVLab can execute secure scripts for the creation, modification or deletion of multiple users in a secure way. The scripts can be stored on a special secure ftp, and are executed by admin user.

b) Camera management subsystem is a basic parameterized component of RVLab to install, control and configure remotely the cameras placed on a physical laboratory. It is also responsible to capture a video signal of the environment and the further transmission of video- streaming with different quality to users. RVLab includes natively support for commercial IP cameras from Axis and Vivotek, fixed and PTZ. Opposite to a direct access to an IP camera (not recommended in a secure lab), RVLab addresses transparently video-streaming, removing even the users limit usually imposed by some IP cameras to control the video-streaming bandwidth.

c) The Time Reservation subsystem provides a common flexible procedure to reserve and assign equitable time access to laboratory (and lab resources). An user can reserve an instrument the time necessary for his or her experiment, and he or she can know the time limits to carry out the experiment and the actual user reserves. Then, RVLab schedule the lab time into a FIFO queue.

d) Instrument Management subsystem. RVlab has implemented a flexible and adaptable module to administer directly the instruments or instrument-based devices. The instruments can be implemented by developers using the preferred programming language, distributed paradigm or network protocol. However, some restriction and rules must be imposed in order to be controlled by the lab system.

e) Lab Resources Management subsystem is a module responsible of administering any other lab resources such as manuals, tutorials, and so on, in general, necessary for the training of users. Lab resources in some cases can be also user logs, reserve lists or stored partial results. RVLab includes a common way to deal with lab resources, enabling commands to upload them to server, to download them or to assign time use and user groups. Each lab resource uploaded by a user is stored on a secure ftp server after checking the nature of the uploaded files.

f) Chat Subsystem. RVLab includes a concurrent live chat system to provide active interaction between users (e.g. student-student and instructor-student in an education domain). The subsystem opens a socket for each user once they have been registered to broadcast all the messages to user opened sessions.

Figure 2 shows the deployment diagram of the system architecture for an online lab system, which indicates the three main blocks, client-side, server-side and instrument-side, and how they are connected. RVLab adds pluggable server-side components into the server, which decouples the instruments from clients using the same or different communication protocols and gives a secure access between clients and instrument resources. RVLab does not provide code for instrument-side and client-side blocks, only for server-side block, leaving to developers the mission to implement the rest of blocks. Furthermore, RVLab components impose a model to manage instruments with some slight constraints (examined in next section), and give an interface with a well-defined API to help the coding of client-side applications.

Client-side applications can access to server by using XML-RPC [15], a distributed paradigm based on remote procedure call using XLM for data format and HTTP as transport protocol. The selection of XML-RPC is because it defines a simple RPC mechanism over usual HTTP transport, which is language and platform independent; there are multiple libraries in many programming languages such as c, c++, java or php, covering the requirements of any developer. RVLab provides an API to server-side components in terms of commands to be invoked by client-side applications through XML-RPC; Table I shows a list of these commands, but developers may add more commands for their instrument components (see next section). To design RVLab we are focused on three main aspects that become design goals: versatility/reusability, security and instruments ubiquity.

One of the major shortcomings of a VRL or an online lab system is the reusability [12]. The most of VRLs are developed for a specific type of laboratory instruments, requiring a new development from scratch for every new lab system. Instead of reinventing the wheel in the development of a lab system, reusing common components give to developers a reusable base to reduce the software efforts and costs, achieving a new lab system in shorter delivery time. Besides to reusability, the versatility is a strong feature desirable for any real laboratory, and also a key factor for the design of RVLab framework

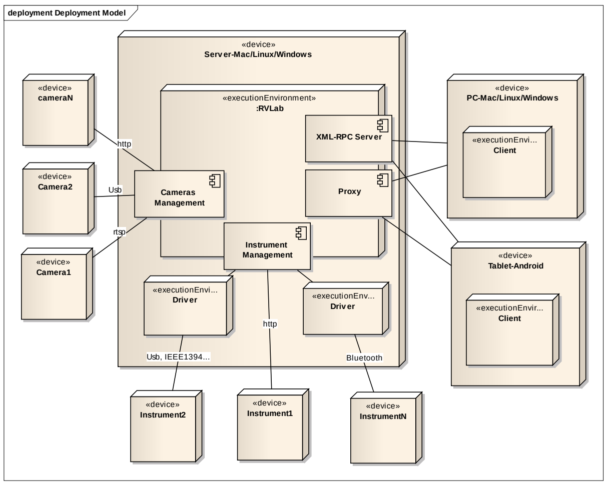

Figure 2. Deployment diagram of an online lab system.

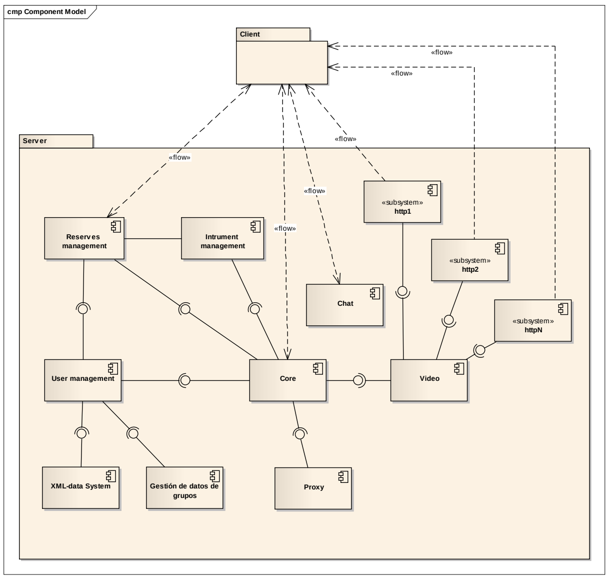

Figure 3. Component diagram of server-side components

Sommerville recommends the use of some techniques to ensure the reusability of the designed program [16], listed in Table II. RVLab makes use of some of them to improve the code reusability of server-side components.

A set of design patterns [17] such as factory pattern, singleton and observer are applied in RVLab in order to explode the capability of adapting the framework to any instruments and cameras and extend the capabilities of its block components. In addition, RVLab is implemented in QT [18], a cross-platform application framework widely used in multiple platforms, augmenting in this way the portability of the code. RVLab is completely configurable with configuration files and parameterized to change its behavior.

Security is another important factor in RVL lab system in order to maintain safe the devices, instruments, lab resources and users of the laboratory. RVLab applies security policies at two levels. First, it manages and routes all data and command connections using SSL by checking user and permissions in any request. Second, the RVLab architecture benefits the security, exposing only one access endpoint for lab clients and hiding all the devices, Recommendation for reusability [Som95]

Approach	Description
Design patterns	Generic abstractions that occur across applications are represented as design patterns that show abstract and concrete objects and interactions.
Component-based development	Systems are developed by integrating components (collections of objects) that conform to component-model standards.
Application frameworks	Collections of abstract and concrete classes that can be adapted and extended to create application systems.
Legacy system wrapping	Legacy systems that can be “wrapped" by defining a set of interfaces and providing access to these legacy systems through these interfaces.
Service-oriented systems	Systems are developed by linking shared services that may be externally provided.
Application product lines	An application type is generalized around a common architecture so that it can be adapted in different ways for different customers.
COTS integration	Systems are developed by integrating existing application systems (COTS: Commercial of the shelf).
Configurable vertical applications	A generic system is designed so that it can be configured to the needs of specific system customers.
Program libraries	Class and function libraries implementing commonly-used abstractions are available for reuse.
Program generators.	A generator system embeds knowledge of a particular type of application and can generate systems or system fragments in that domain.
Aspect-Oriented software development	Shared components are woven into an application at different places when the program is compiled.

instruments and cameras which could be connected into the same LAN or several LANs. A common weakness found on many laboratories [5][9] in our opinion is the separation of the lab management applications using different technologies; i.e., an IP camera for the control and visualization of the environment (some time directly), and a web tool based on moodle for user and content management. The technologies disaggregation makes difficult the lab configuration and the protection against security attacks. However, RVLab provides an effective and unified mechanism to cope the technological diversity required for the developing of lab tools and applications. Then, although communication connections outside RVLab control are possible and not recommended, it exposes a set of secure communication channels in order to apply security policies with XML-RPC protocol. In addition, the model favors the reconfiguration of laboratory applications, without any notification to user client application.

Instruments and cameras can be anywhere and they can be connected to server using several internet protocols or any other communication protocol. The server-side components hide the instruments and cameras and address the commands invoked by client-side applications. Insertion of an instrument into a Lab system

A RVL or online lab system developed with the support of RVLab must contain components for the management of users, cameras, time reserves, lab resources and instruments. The last one, the instruments, is the dependent part in any lab system. RVLab gives complete freedom in its design and coding with respect to the programming language, distributed paradigm and communication protocol. But it is imposed some rules and restrictions that must be satisfied by the component to be added in lab system.

RVLab applied some design patterns to simplify the way that a developer could build a new instrument into the lab system. For instance, abstract factory pattern facilitates the adaptation of any instrument to RVLab vision in order to manage them.

To design a component for an instrument with RVLab we must distinguish several stages:

a) Implement the instrument component.

b) Register the instrument component into the server.

c) Prepare a configuration file to dynamically load the instrument component by an instrument manager.

d) Design of an API for the new instrument component.

First we need to implement the instrument component as a derived class which inherits from *InstrumentResource* abstract class, and can be managed by the *Instrument Management subsystem*. The derived class must have an implementation of the instrument which supposes the communication with the instrument device or the software that controls the instrument using a specific communication protocol whether it is specified by the instrument, or any other ones defined by the developer. In study cases we have used TCP based communication protocols. Figure 4 shows the logic representation of any instrument as a subclass of *InstrumentResource*.

In order to simplify the management of InstrumentResource derived classes, a dynamic subclass registering mechanism has been implemented. This mechanism allows selecting dynamically the concrete subclass of the abstract factory at runtime from a set of registered subclass..

The dynamic subclass registering mechanism must store into a map, an *id* of the subclass and a pointer to a constructor method of objects of this subclass. The static method *InstrumentResource::registerSubClass* can be called from anywhere, and allows registering a new subclass. For example,

```
InstrumentRe-
source::registerSubClass("injector",Injecto
r::createInstance);
```

The above function stores into the map an id “injector” with the function pointer Injector::createInstance, which is a method that returns an instantiation of a new object for Injector class. Using the above mechanism the developers can register the potential instruments to be managed by the framework.

The *Instrument manager* class is responsible to store all the controlled instrument objects in the system, and in general they might be read from configuration file. Using same *ids* in configuration files than registered classes, the instrument management subsystem can create new instrument objects. Therefore it is important that *ids* included into the configuration file will be the same *ids* of the classes already registered. Otherwise the instrument

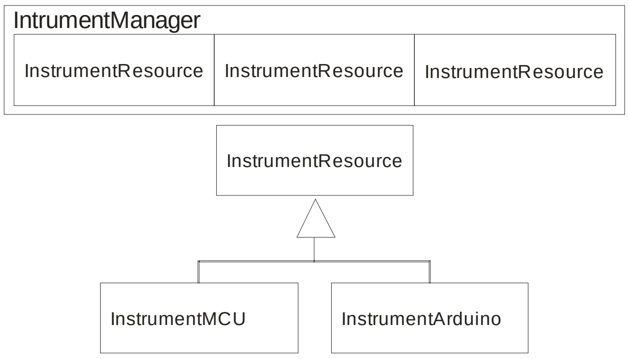

Figure 4. Logic representation of an instrument.

management subsystem throws an error and instrument will not be instantiated. An example of a section the configuration file is shown in the following code:

```
<instrument>
 <id>injector</id>
 <name>Injector PSD/3</name>
 <number>2</number>
 <groupInstrumentID>1</groupInstrumentID>
</instrument>
```

In above code, there are several parameters,. InstrumentManager is responsible for reading all parameter in the xml file and it will instantiate the new class with these data. The constructor of new class must accept an array of QObjects. Another advantage of our registering mechanism is that the constructor of new classes can accept on-the-fly a variable array of parameters (i.e. using QObject definition of QT) instead of a fixed list of parameters. This allows a flexible instantiation of instrument classes, but with the requirements of parsing each parameters of the array. Therefore InstrumentManager will read all parameter in xml file, it will create an array with the data and it will call to the constructor of new class using the array as parameter.

Finally, it is necessary to communicate clients with instruments. The instrument objects could create a new API for the instruments by the definition of specific commands or services, which can be invoked by a XML-RPC protocol from client-side applications. In order to register a new service or command, developers only have to register operation name, the object receptor and the method of object receptor. An example is:

```
LocalLab::addMethod("executeExperiment",
instrument1, "loadAndExecute");
```

This operation registers new XML-RPC method with name “*executeExperiment*”. When a client launches a request with *executeExperiment* id, RVLab will call the method *loadAndExecute* in object *instrument1*. The invocation of a command by a client in *executeExperiment* should have the same number of parameters with the same type; in other case the command it is not accepted.

The above procedure uses a request-response mechanism based on RPC paradigm to route the commands from the client-side applications, blocking the object receptor until a response is transmitted to client. But, sometimes we can require an asynchronous mechanism to notify data or any type of event to client. In these cases, RVLab allows the opening of secure sockets to client to send data stream or communicate client and instrument directly.

It is also possible to use the sockets to route the commands directly to instrument. But in this case, the proxy component of RVLab could be used in addition to adapt these commands between different communication protocols.

II. INSERTION OF NEW CAMERAS IN A LAB SYSTEM

It is possible to adapt or extend the functionality of default pluggable components by using a similar registering mechanism as it is describe in before section. We applied it to simplify the inclusion of new cameras into the *Camera Management* subsystem. The steps are:

a) Register the provided abstract superclass with a new derived subclass with an “id” and a pointer to constructor. An example for registering a new video camera is:

```
VideoRe-
source::registerSubClass("Logitech",VideoRe
sourceLogitech:: createInstance);
```

b) Define a configuration file from RVLab to read the cameras to be instantiated. An example of a configuration file is the following:

```
<camera>
 <vendor>Vivotek</vendor>
 <cameraIp>1.2.3.4:1026</cameraIp>
 <cgiPort>8080</cgiPort>
 <name>Camera2</name>
 <number>2</number>
 <type>normal</type>
 <streamName>live.sdp</streamName>
 <groupInstrumentID>1</groupInstrumentID>
</camera>
```

c) Use superclass “createInstance” method with subclass id to obtain new subclass object. In our case, the objects are read from the configuration file at runtime. Using the above example, we can create a new camera object by the following code:

```
vr=VideoResource::createInstance("Vivotek",
tk,recorderData, "1.2.3.4:1026", "Camera2",
2);
```

Actually RVLab can receive video data using most popular codecs (mpeg2, mpeg4, h.263, h.264, divx, xvid, vp8, theora) in several formats (mpeg, mp4, webm, ogg) and it can transcode video using webm (vp8+vorbis), ts (mpeg4+mp3 or h.264+acc), ogg (theora+vorbis).

RVLab can synchronize several video sources when they are sent using rtsp or rtp. RVlab uses marks of time included in the protocol. RVlab supports others protocols such as http. However synchronization will not be available. These protocols are not metadata to synchronize.

III. STUDY CASE: DOMOLAB

RVLab has been used for the development of an online lab system in order to verify the possibilities of this framework. The support of RVLab provides the base for the implementation of server-side application. But, in addition, it has been necessary to perform the implementation of instrument-side and client-side applications.

Domolab is an online lab system for learning the principles of concurrent, embedded and real time programming using a house-scaled model as a didactic tool [19]. The house-scaled model is equipped with a microcontroller of 8, 16 or 32 bits and a plenty of sensors and actuators to simulate home-automation systems in a home environment. Initially, the house-scaled model was placed on a laboratory and was used directly by computer science students to test their programs. In order to implement their programs, the students must use the framework JavaES (Java Embedded Systems) [20]. JavaES is a Java based middleware that abstracts the hardware interaction to sensors and actuators. Then, it provides a simplified way to implement Java programs that access to hardware devices independently of used microcontrollers.

Domolab gives to the students the opportunity to test their programs directly from their houses, without being in the lab. Domolab has a robust self-reset system loaded onto the microcontroller that admits the upload of user programs using a remote Domolab client-side program. This is achieved by the support of RVLab framework. An operational server-side program is executed using default pluggable components, giving a control of users, lab resources, time reserves, and the microcontroller where the user program will be uploaded.Once the user program is loaded into the microcontroller through server, its execution is monitoring giving data directly to the user. The user can see what happen in the remote house model by the visualization of the laboratory with three IP cameras, which can be also controlled by user. Therefore, two static and one PTZ camera is placed on the physical

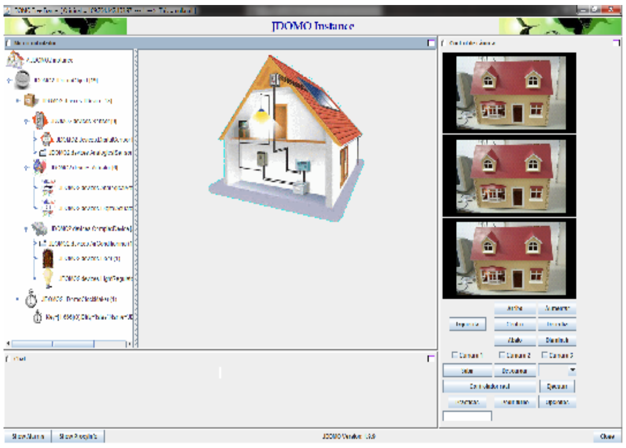

Figure 5. Remote Domolab client-side application.

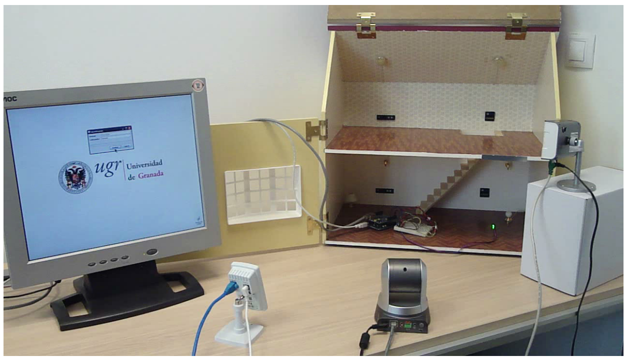

Figure 6. Instruments of the Domolab lab.

lab, that it allows users to see the consequences of their program in the remote house model (e.g. switch on a light) by transmitting video-streaming to user. For the uploading of user program into microcontroller, a script on the server is applied that store temporally the user program into a secure ftp (vsftpd) for that user, before *Instrument Management Subsystem* might send it to microcontroller of house-scaled model.

Figure 5 shows the user's interface for the remote Domolab client-side application on which user makes use of lab system. It should be implemented completely using commands from the API of server-side components that are invoked through XML-RPC. Therefore, both instrument-side and client-side application must be developed.

IV. PERFORMANCE EVALUATION

A preliminary performance evaluation is performed on the server-side components supported by RVLab for Domolab lab system.

The system has been tested with forty users connect at time. In testing session, a server with two Intel Xeon Quad Core E5405 @ 2Ghz, 4GB RAM a 100Mbits/s connection was used. To control the experiment a virtual machine with four cores and 512MB RAM was used.

The experiment consists of analyzing the registration of one hundred fifty users, belonging to four different groups at the same time to lab server. In this case there were twenty reserves available, fourteen personal and five reserves of group. During testing phase, there were two valid reserves, a personal one from 11:00 to 12:00, and a group one from12:00 to 13:00.

All the measurements are made from 11:39 to 12:19. At 12:19h we begin to disconnect the computers in the last stage. When there were forty clients connected, we sent numerous turn requests to the server, being the most expensive task, since they need to check all the reserves.

First we carried out a measurement of the CPU rate, and its variation with respect to time and connected clients. Figure 7 shows that CPU rate has a little variance in time because most of the calculation is the recoding of the three video-streaming taken by the cameras.

Figure 8 measures the memory consumption in the same time frame. As it is seen the memory consumption is very stable, since the memory load is mainly due to the reading of XML files, and especially to the recoding of the video-streaming. In our case with the encoding of three simultaneous streaming requires the use of approximately 95% of the 512 MB of RAM; this measure includes the rest of processes of OS. Therefore, the CPU burden on the connected clients is minimal, because their number has a little impact on the use of main memory.

Another critical parameter for simultaneous user connections is the bandwidth as it is shown in figure 9. This gives a comparison between data sending and data reception on the server. Received data (in blue) has a little variation around 1.37 Mb/s, because the data received from cameras had been arriving constantly with a similar bitrate, and did not depend on the number of connected users. In contrast, the red graph shows the data sent to users. As it is shown in figure 8, the mean value of sent data is 36.89MB/s, 27 times greater than received data. The value is increasing when the number of connected users grows, and is decreasing at the end of the experiment when users decrease. The reason for this fact is that

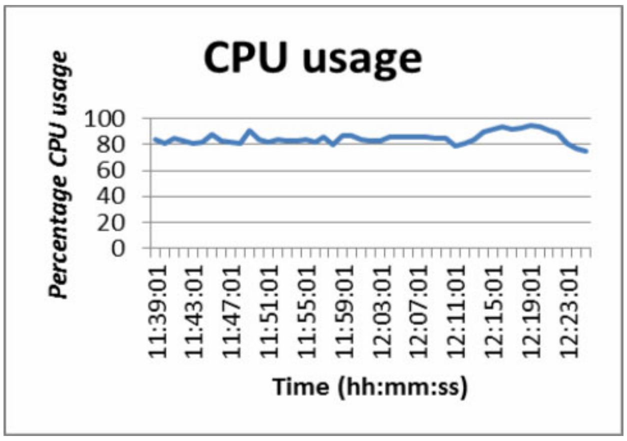

Figure 7. Domolab, CPU usage

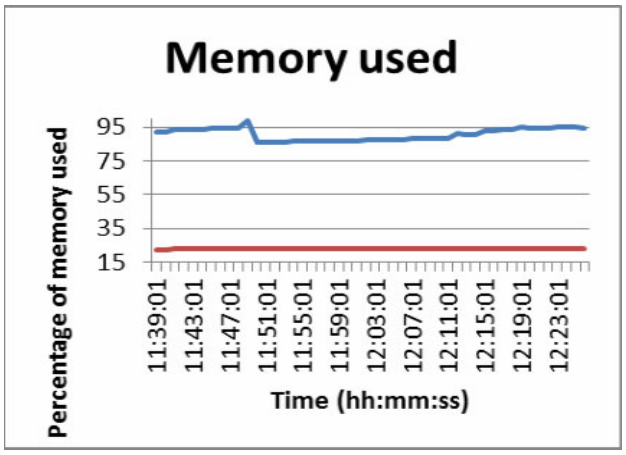

Figure 8. Domolab, Memory usage

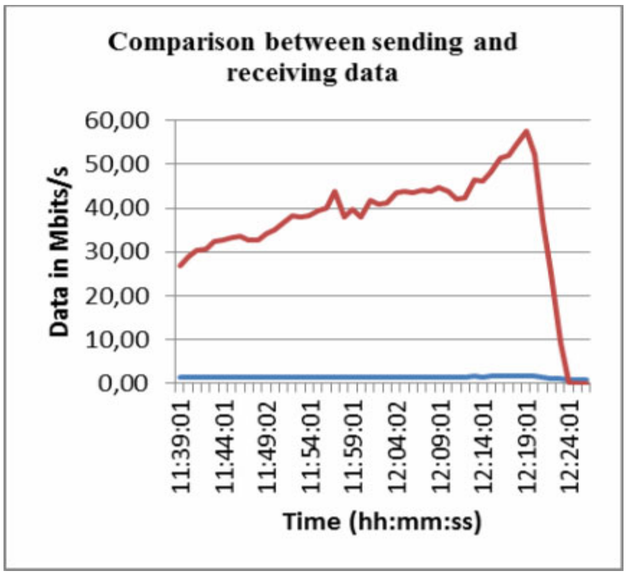

Figure 9. Domolab, Bandwight used

when the users grow, the video-streaming must be send to more users.

V. RELATED WORK

Ranaldo et al.[21] presented a similar flexible system that manages both the real-time visualization of the measurement instrumentation and data flows concerning experiments on real measurement instrumentation. But, RVLab manages any instrument, not only electronic devices, in an unified way without the need of connecting VPN or Windows environment. Moreover, RVLab can be adapted to LabView, but it is not dependent of any software and hardware platform. Harward et al. have designed iLab Shared Architecture that it is used in several universities around the world [10][22][23]. This system provides a scalable, uniform platform to access to lab systems. iLab and Ranaldo et al.[21] are focused to work with LabView, which provides good interfaces to work with several instruments in engineering and electronica areas. However there are several areas where LabView is not used and it does not offer support for video streaming.

VI. CONCLUSION

On conclusion, high costs associated to specific instruments require the adoption of software and hardware infrastructures that simplifies the development of laboratories and maximize its use at the same time. RVLab is implemented with the premise that any laboratory infrastructure (including hardware and software) should be sufficient flexible and versatile to adapt to any specific instrument without losing the performance. The *Instrument Management Subsystem* of RVLab offers a robust and secure mechanism to developers that can be adaptable and extensible for any instrument of a laboratory.

In addition, RVLab allows remote control of a set of cameras (IP camera and USB camera) and the synchronized transmission of video-streaming to user using efficient lightweight codecs (included in RVLab). The difference with other proposals is that RVLab includes natively the support for the camera and video management based on open network tools instead of working with independent devices.

RVLab is applied for the development of two RV Laboratories with good performance with respects to the number of connected users with a moderate consumption of resources. The implementation was relatively quick, requiring the coding of a few classes.

VII. REFERENCES

[1] K. Choi, S. Han, S. Kim, D. Kim, J. Lim, D. Ahn, and C. Jeon, “A combined virtual and remote laboratory for microcontroller.” in ICHL’09, 2009, pp. 66–76.

[2] S. G. Eick, A. Mockus, T. L. Graves, and A. F. Karr, “A web laboratory for software data analysis,” World Wide Web, vol. 1, pp. 55–60, 1998, http://dx.doi.org/10.1023/A:1019299211575

[3] L.-d.-I. D. O. P. Garcia-Zubia, J., “Mobile devices and remote labs in engineering education,” 2008, pp. 620–622, cited By (since 1996) 2. [Online]. Available: http://www.scopus.com/inward/-record.url?eid=2-s2.0-51849148995&partnerID=40&md5=660e1e76b41a0b7cd68066e78451c93e

[4] P. L. Couteur, “Review of literature on remote & web-based science labs,” BCCampus Articulation, p. 22, 2009.

[5] N. Pavón and J. Ferruz, “Bender 3.0, una plataforma robótica remota para aplicaciones docentes: aplicación a programación concurrente,” Revista Iberoamericana de Automática e Informática Industrial, vol. 7, no. 1, pp. 54–63, 2010.

[6] C. Maier and M. Niederstätter, “Lab2go - a repository to locate online laboratories,” iJOE, vol. 6, no. 1, pp. 12–17, 2010.

[7] F. Cicirelli, A. Furfaro, D. Grimaldi, L. Nigro, and F. Pupo, “Madams: A software architecture for the management of networked measurement services,” Comput. Stand. Interfaces, vol. 28, pp. 396–411, April 2006. [Online]. Available: http://dl.acm.org/citation.cfm?id=1648871.1649028

[8] H. W. S.-Z. S. Z. C. M. Zheng, S.a, “Development of gamma-ray energy spectrum remote-measurement system based on lab view,” He Jishu/Nuclear Techniques, vol. 29, no. 7, pp. 548–550, 2006, cited By (since 1996) 0. [Online]. Available: http://www.scopus.com/inward/record.url?eid=2-s2.0-33747502106&partnerID=40&md5=ce23fcce21ef996c00b85bb75ca282d3

[9] J. Genci, “The "zero cost" remote lab,” 2009, pp. 572–575, cited By (since 1996) 0. [Online]. Available: http://www.scopus.com/inward/record.url?eid=2-s2.0-67650683174&partnerID=40&md5=f6ff76d880f2da5cffcddab92e379df3

[10] V. Harward, J. del Alamo, S. Lerman, P. Bailey, J. Carpenter, K. DeLong, C. Felknor, J. Hardison, B. Harrison, I. Jabbour, P. Long, T. Mao, L. Naamani, J. Northridge, M. Schulz, D. Talavera, C. Varadharajan, S. Wang, K. Yehia, R. Zbib, and D. Zych, “The ilab shared architecture: A web services infrastructure to build communities of internet accessible laboratories,” Proceedings of the IEEE, vol. 96, no. 6, pp. 931 –950, june 2008. http://dx.doi.org/10.1109/JPROC.2008.921607

[11] E. Grosclaude, F. L. Luro, and M. L. Bertogna, “Grid virtual laboratory architecture,” in Proceedings of the 2007 conference on Parallel processing, ser. Euro-Par’07. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 164–173. [Online]. Available: http://-dl.acm.org/citation.cfm?id=1793434.1793458

[12] S. Odeh, “EnglishBuilding reusable remote labs with adaptable client user-interfaces,” EnglishJournal of Computer Science and Technology, vol. 25, no. 5, pp. 999–1015, 2010, cited By (since 1996) 0. [Online]. Available: http://www.scopus.com/inward/-record.url?eid=2-s2.0-78650214791&partnerID=40&md5=81a7860621b2e91e6b56d39e50a7548d

[13] T. A. S. Mickell, B. D. S. Danner, and O. H. College, “Virtual labs in the online biology course : Student perceptions of effectiveness and usability,” of Online Learning and Teaching, vol. 3, no. 2, pp. 105–111, 2007.

[14] F. O. F.-T. M. L. Burget, P., “Remote labs and resource sharing in control systems education,” vol. 17, no. 1 PART 1, 2008, cited By (since 1996) 0. [Online]. Available: http://www.scopus.com/inward/record.url?eid=2-s2.0-79961019331&partnerID=40&md5=8cdd5b4ae88e5fdc48e53fd80845751a

[15] S. S. Laurent, E. Dumbill, and J. Johnston, Programming Web Services with XML-RPC. Sebastopol, CA, USA: O’Reilly & Associates, Inc., 2001.

[16] I. Sommerville, Software engineering (5th ed.). Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 1995.

[17] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns. Boston, MA: Addison-Wesley, January 1995. [Online]. Available: http://www.amazon.co.uk/exec/obidos/ASIN/0201633612/citeulike-21

[18] (2011) Nokia consultado el 6 de junio de 2011. http://doc.trolltech.com/4.7/signalsandslots.html.

[19] J. H. T. J. Viudez Aivar, “Diseño y construcción de una maqueta domótica controlable a través de microcontroladores java,” Actas de V Jornadas De Enseñanza A Través De Internet/Web De La Ingeniería De Sistemas y Automática, Eiwisa™2007, pp. pags 47–52, 2007, thomson. ISBN: 978-84-9732-603-2 (2007).

[20] J. A. Holgado-Terriza and J. Viudez-Aivar, “Javaes, a flexible java framework for embedded systems,” in Distributed, Embedded and Real-time Java Systems, M. T. Higuera-Toledano and A. J. Wellings, Eds. Springer US, 2012, pp. 323–355, http://dx.doi.org/10.1007/978-1-4419-8158-5_13

[21] N. Ranaldo, S. Rapuano, M. Riccio, and F. Zoino, “Remote control and video capturing of electronic instrumentation for distance learning,” Instrumentation and Measurement, IEEE Transactions on, vol. 56, no. 4, pp. 1419 –1428, aug. 2007. http://dx.doi.org/10.1109/TIM.2007.900152

[22] J. Hardison, K. DeLong, P. Bailey, and V. Harward, “Deploying interactive remote labs using the ilab shared architecture,” in Frontiers in Education Conference, 2008. FIE 2008. 38th Annual, oct. 2008, pp. S2A–1 –S2A–6.

[23] M. Auer, D. Zutin, and C. Rajyaguru, “A labview toolkit for the development of ilab batched lab servers,” in Global Engineering Education Conference (EDUCON), 2011 IEEE, april 2011, pp. 26 –29. http://dx.doi.org/10.1109/EDUCON.2011.5773107

AUTHORS

Jesus. L. Muros-Cobos. Author, is with the software engineering department, University of Granada, Granada 18071 Spain (e-mail: jesusmuros@ugr.es).

Juan A. Holgado-Terriza Author, is associate professor in Software Engineering Department at University of Granada 18071 Spain (e-mail: jholgado@ugr.es).

This article is an extended and modified version of a paper presented at the International Conference on Remote Engineering & Virtual Instrumentation (REV2012), held at University of Deusto, Bilbao, Spain, July 4-6, 2012. Manuscript received 15 September 2012. Published as submitted by the authors 28 November 2012.