Towards an Extensible WebLab Architecture

J. García-Zubia [1], P. Orduña [2], J. Irurzun [2], U. Hernández [1], E. Sancristobal [3], S. Martín [3], M. Castro [3], D. López-de-Ipiña [1], I. Angulo [1]
[1] Faculty of Engineering, University of Deusto, Bilbao, Spain
[2] DeustoTech – Tecnológico Fundación Deusto, Bilbao, Spain
[3] IEECD – Spanish University for Distance Education (UNED) – Spain
zubia@eside.deusto.es

Abstract- **Remote Laboratories have traditionally been focused on specific solutions for specific problems. We can find a wide range of Remote Laboratories in the literature, assisting different types of subjects but commonly bound to a restricted set of requirements. Due to this, little attention has been paid on working on a maintainable, scalable, secure architecture that addresses the requirements of a wider set of experiments, and that could be open enough to support or adopt new experiments, developed using different technologies in both client and server side. In this paper, we describe several aspects that might be taken into account when designing Remote Laboratories, describing the XWL Architecture and comparing it with other existing architectures.**

I. Introduction

A Remote Laboratory is composed of a client-side software and a server-side software.

The decisions taken for choosing among different client-side technologies (AJAX, simple HTML, Adobe Flash, Java Applets, Microsoft Silverlight, etc.) are critical [1][2][3] to allow or not the final users to use the Remote Laboratory on different browsers and different operating systems, as they are critical to enable certain degree of security, accessibility and user interface capabilities (3D, video, sound). These decisions will have an impact on the Remote Laboratory architecture [2] because depending on the client technology used, the server side will be able to interact with the client using different types of protocols (SOAP, REST, JSON, RMI, sockets, etc.).

However, other required factors (i.e. scalability, maintainability of the system, most of the security aspects, quality of service, etc.) of Remote Laboratories are bound to the server design and independent of the client side.

In this paper we illustrate the different requirements of a Remote Laboratory and the approach taken by our Remote Laboratory (WebLab-Deusto) to match these requirements, and the XWL Architecture is described.

II. Defining Requirements of Remote Laboratories

Examining the requirements of Remote Laboratories, there are important aspects that have to be taken into account when designing an architecture for Remote Laboratories. We have summarized these aspects in a set of questions that the designer of a Remote Laboratory architecture might consider, and grouped them into the following six categories:

A. *Dependence on the type of experiment and flexibility*

a) How dependent on the nature of the experiment is the architecture?

b) Can experiments be shared with different users at the same time by time-division multiplexing? Most experiments can not be multiplexed (those that have some kind of state -a program in a device, a robot in a position, etc.-), but there other types of experiments that can be multiplexed. Can this technique be used in some experiments in the architecture?

c) Can't the experiments be shared and thus the users need time-based sessions? Does the Remote Laboratory assume a magnitude of time for each session (milliseconds, seconds, minutes, hours...)? Does the experiment have a scheduling system based on this magnitude (i.e. no waiting in the case of milliseconds, waiting in a queue for seconds or minutes, and having to schedule it for hours and days)?

d) How generic are the commands used to interact with the device? Does the Remote Laboratory assume some types of experiments which receive certain type of information? Does the Remote Laboratory assume a size or frequency for these commands?

e) Does the Remote Laboratory use a video stream (i.e. from a webcam)? Does the Remote Laboratory assume a video quality for it?

B. *Scalability*

a) How many users does the architecture support in the highest peaks? How affordable is to increase that number?

b) Does the Remote Laboratory scale vertically (adding more resources -memory, CPUs- to a single node, the system supports proportionally more connections)?

c) Does the Remote Laboratory scale horizontally (adding more nodes, the system supports proportionally more connections)? Can the application be distributed along those different nodes? It is easier to add more nodes than to add more resources to a single node, but it is also usually more complex to implement a Remote Laboratory that scales horizontally that one which does not.

C. Maintainability

a) Does the architecture assume a single schema for the integration of the Remote Laboratory?

b) Can it be integrated in the IT Services of different universities? Can it use different schemas for this integration (supporting SSPI, LDAP, different database providers, etc.)? Different entities tend to use different solutions for storing credentials and personal information of the users. Providing a pluggable authentication system is useful to integrate the Remote Laboratory in different schemas.

c) Does it support an advanced user management - involving multiple types of users with different privileges: users, professors, laboratories administrators, system administrator, etc.-? As the Remote Laboratory user and experiment base grows up, the administration tasks gets more complex. Not supporting different roles centralizes these tasks into a single role -administrator-, consuming time doing tasks that could be done by other maintainers that can't have so many privileges.

For instance, a professor who owns an experiment should be able to check the logs of the use of that experiment, but not the logs of other experiments. If there is no such role, the professor will need to contact the administrator to get those logs in a manual way so no automatic response is obtained, and if there are many experiments the administrator will be wasting too much time in maintainability tasks that could be automated. However, all those professors should not have administrator privileges, and although only certain privileges could be provided to each professor user, the creation of roles and groups can speed up the maintainability of the system.

d) Does it support an advanced log management? Can professors easily know how much do their students use the experiments they handle?

D. Security

a) Does the architecture take into consideration security in its design? It is important to take into account security issues during the whole process of software development, including its design. A vulnerable design can become difficult to secure in the latter stages of the development.

b) Does the Remote Laboratory avoid security flaws in the different modules of the system? Does the Remote Laboratory support secure communication protocols? Does the Remote Laboratory count with systems to avoid code injection (such as SQL/LDAP/XPath injection)? Does the Remote Laboratory store the passwords in a secure way?

c) Have security policies been established in the Remote Laboratory development?

E. Dependence on the protocol

a) Does the architecture assume a certain topology and bases the protocol decisions on that topology?

b) Does the Remote Laboratory assume that the different experiments and the application servers are in the same computer/room/building/city?

c) Does the Remote Laboratory support multiple protocols for different types of experiments, depending on the requirements of these protocols? For instance, an optimized binary protocol is more suitable for experiments which require real time feedback from the device, while it might have problems when dealing with firewalls or proxies. The decision depends on if real time feedback really is such a requirement.

d) Does the Remote Laboratory support multiple protocols depending on the security needed given the topology (using IPC -i.e. UNIX sockets-, or a dedicated network, a university private network, a public network)? Are authentication and encryption considered depending on the type of network?

F. SOA-compliant

a) Does the architecture match the Service Oriented Architecture?

b) Is the Remote Laboratory deployed as a service using a well known transport that can be consumed by other applications such as SOAP, REST or JSON? Or does it only support its own client (i.e. it only supports a web client)?

c) Can other services be built on top of the Remote Laboratory using a public interface?

III. ANALYSIS OF EXISTING REMOTE LABORATORIES

An analysis of two existing Remote Laboratories is provided using the previously detailed aspects:

A. MIT iLab

The iLab [4] is a Remote Laboratory developed by the MIT. The iLab has a advanced architecture which supports powerful cross-university deployments, where an experiment can be shared with other universities. The Architecture is scalable and has been demonstrated with its usage [5] in different organizations.

In terms of dependence on type of experiment, iLab provide both batch and interactive experiments, treated different by the architecture. Since the interactive experiments used in iLab take a long time (20 mins – 1 hour), the users must schedule their uses instead of using a queue. While iLab embraces the Service Oriented Architecture in the batched experiments adopting SOAP, it becomes flexible with the interactive experiments, delegating the decision to the Remote Laboratory developer in order to use a more optimized protocol [6].

The Architecture of iLab reflects that security has been considered at the design level, and the available source shows that it has also been considered in the implementation. However, the current servers where the ServiceBrokers are installed seem to lack secure connections support, so sensible information sent through the web interface by the end user might be read by an attacker. Furthermore, most clients are developed using Java Applets and some are self-signed, so the student using the experiment will accept that the Java Applet downloaded from a not secure connection to the server has access to her computer as any Desktop Application. These are low-level issues that do not affect the architecture itself, and the architecture itself doesn't demand a concrete client technology. Actually, other client technologies have been tested [5].

Finally, in terms of SOA-compliance, the iLab offers a public interface based on standard Web Services and the architecture itself is designed based on Web Services.

B. *BTH VISIR*

BTH provides the VISIR Project [3], which provides a Remote Laboratory where the student creates circuits using an advanced Adobe Flash based user interface, and then this circuit is sent to the server and tested there. The Remote Laboratory is interactive so the user will receive real-time information from the experiment itself. The powerful board where the circuits are built supports multiple concurrent usages by multiplexing them in the time. This way, multiple students can use the same hardware simultaneously while testing different experiments without noticing it.

The VISIR Project uses a proprietary protocol based on sockets so as to improve the performance of the experiment and avoid any latency generated by using higher level protocols based on HTTP.

In order to extend the VISIR Project, the experiments must fit in this type of experiment, so adding experiments of a completely different type might be considered outside the scope of this project.

IV. WEBLAB-DEUSTO ARCHITECTURE

The software architecture of our Remote Laboratory has taken into account the previously explained.

A. *Software architecture evolution*

WebLab-Deusto had previously gone through two main iterations:

- Version 1.0: The first approach of WebLab-Deusto that students actually used. The client was developed as a Jython applet (the user had to install the Java Runtime Environment in order to use the Remote Laboratory), while the server was a single Python application that could manage a single experiment. The communication between client and server was a proprietary socket-based protocol; this way the user could not be behind a HTTP proxy server and problems would arise if the user dealt with firewalls.
- Version 2.0: The second approach of WebLab-Deusto improved the client side of the project. The client side was rewritten using AJAX, and in the server side a new layer was written on top of the previous server so it managed SOAP messages instead of low level socket based messages. The user now could use the experiments even behind a HTTP proxy server and a firewall. Furthermore, since no software installation was required (since the client was purely written in AJAX), the user could use the Remote Laboratory from certain mobile devices [7]. Other enhancements, such as a cross-platform version of the laboratory, administrative applications to manage the Remote Laboratory, etc. were achieved in this branch through the different 2.x versions.

B. *WebLab-Deusto 3 Requirements*

As a result of the success of these previous iterations, new requirements came up and required a new Remote Laboratory architecture.

In terms of flexibility, WebLab-Deusto 2.0 still managed a single experiment. This way, in order to provide multiple experiments to the students it was necessary to deploy multiple instances of the Remote Laboratory, and maintain them independently.

The number of students accessing the laboratory has also increased, so scalability had to be taken in the new design. Since the hardware was distributed along different laboratories of the Faculty of Engineering, the Remote Laboratory needs to support a complex deployment that will deal with heterogeneous networks that might include elements such as HTTP proxies, firewalls and untrusted networks between the different servers.

C. *WebLab-Deusto 3 Architecture*

In order to match the requirements explained above, the WebLab-Deusto 3 is based on a distributed architecture as shown in Fig.1.

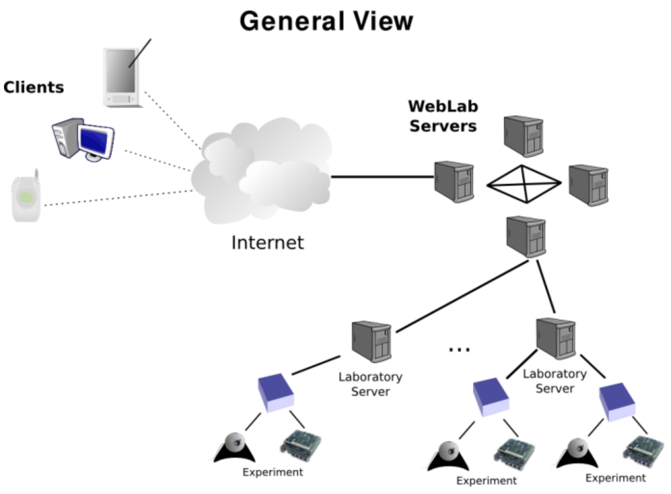

Fig. 1. WebLab-Deusto 3 Architecture

1. *Logical architecture*

In this architecture, the clients connect to servers located in a server farm, maintained by the IT services of the University.

A multitier architecture is applied, where the presentation tier is found in the client side, and the logic and database tiers are physically placed in this server farm.

The project currently supports MySQL 5 [8] for the database tier, Python is used in the logic tier, and an AJAX script (written with the Google Web Toolkit framework [9]) is used in the presentation tier.

The servers in the logic tier will communicate with the authentication services found in other tier. Currently, these authentication services support three ways to verify the user credentials:

1. Using a username and a password in the MySQL database. This is intended for local users.
2. Through a remote LDAP server. The users of WebLab-Deusto can use the credentials they use in the Universities that support LDAP.
3. Based on the IP address of the client who logs in. This way, a WebLab-Deusto instance can trust in a set of servers of a third-party entity (such as other University) for a set of users. This third-party entity will sign up these users in the WebLab-Deusto instance, without a password.

   Then they can deploy an application in their trusted servers that starts sessions of these users in the WebLab-Deusto instance without a password, and provide the user the session ID. This way, the verification of credentials is delegated to this third-party application (such as a Moodle or .LRN plug-in), which will redirect the user to the WebLab-Deusto instance transparently. The WebLab-Deusto instance will keep a log of the activities performed by this user so as to provide this information to the third-party application later. This type of authentication fits in architectures aiming to integrate Remote Laboratories with LMS such as [10].

   If the servers of this third-party entity became compromised, the attacker would only be able to log in as the set of users signed up by that entity.

Since the hardware is placed in laboratories that can be found in different buildings, the communication between the logic servers and the servers found in the laboratories may need to be secured, and the number of network addresses reserved by these servers should be minimized. On the other hand, in order to maximize the number of hardware experiments, it is necessary to reduce the cost of the devices that directly interact with the hardware. These devices might be micro servers, that will have direct access to the hardware but they will have limited resources. On it, a light Experiment Server -which is the only server in the WebLab-Deusto Architecture that is aware of the experiment itself- will run.

In order to combine these cheap devices with the required resources per laboratory (secure communication between the logic server and the laboratory, minimized number of network addresses), WebLab-Deusto introduces a new tier that hides the Experiment Servers to the logic servers, acting as proxies. This way, the communication between the logic server and these proxies can be secured, and each proxy will handle a number of micro servers that will directly interact with the hardware. The information communicated between the logic server and the proxy server will not contain sensitive information related to the users that are sending the information, but the information must be at least signed to avoid attackers using directly these proxy servers to interact with the hardware without credentials to do so.

2. *Communications in the WebLab 3*

The problem with this architecture is that, although it matches the logic requirements, it demands many tiers. When the client sends a message to a device, it must be sent securely to the logic servers that will redirect the message securely to a proxy in a laboratory that will redirect the message to an Experiment Server that will redirect the message to the hardware. While this is fine in big deployments, depending on the experiment it may need to be optimized. If the hardware experiment is controlled though an application run in a computer, then the Experiment Server tier should not exist. If this experiment is found in the same network where the logic servers are, the proxies tier should not exist.

Furthermore, depending on the requirements of the experiments and on the configuration of the deployment, the communication between these tiers will require to be optimized, using compressed proprietary protocols through TCP sockets instead of XML formatted data through HTTP.

To perform these optimizations, WebLab-Deusto 3 communications have been built on top of a pluggable system of protocols. Currently, five protocols have been written, and new protocols could be added. These protocols are Python-dependent SOAP messages, TCP sockets, UNIX sockets, XML-RPC and “Direct”, which calls the method name of the server in the same program instance. The decision of choosing between the different communications systems is handled through a communications broker, parameterized by the system administrator. If a server tries to connect to other server, it provides the WebLab-Deusto address of this server, and the communications broker will check what possible protocols can be used and it will automatically choose the fastest one:

- If two servers are located as different object instances in the same process, the “direct” protocol will be used, since it avoids the use of a network.
- If the two servers are located in the same machine and it is based on UNIX, it will use a UNIX socket.
- If the two servers are located in different machines but in the same network, or in the same machine in a not-UNIX environment (i.e. Microsoft Windows), it will use TCP sockets.
- If the system administrator defines that the server doesn’t support TCP sockets (i.e. if this server is behind an HTTP Proxy), it will use SOAP with Python-dependent messages.
- In order to communicate with the Experiment Servers, it can optionally use XML-RPC. The advantage of

using XML-RPC in WebLab-Deusto is that the server can be implemented in other technology (such as Java or .NET), in opposite to the other communications protocols used.

Because of this protocol-agnostic system, the Remote Laboratory can be configured in a very flexible way, supporting the avoidance of communications between different tiers if they are not necessary.

The system administrator is responsible for deploying in a secure way. If the system is deployed with a single process running the whole system using "direct", then if the Experiment Server code fails at process level, it may shut the whole server down. Or, if an attacker manages to exploit a vulnerability in a layer of the system and the Login Server is running with the same privileges, the attacker could access sensitive information such as passwords.

The communications with the client, though, only support SOAP at this moment, since sockets-based communications would not be directly supported by a pure AJAX application. Anyway we are working on provide a sockets-based alternative for performance reasons. This sockets-based alternative is only an alternative since relying exclusively on it would introduce problems with HTTP proxies and firewalls.

D. *Adding new experiments: the XWL Architecture*

The aim of the WebLab-Deusto project is to develop an architecture as independent as possible of the experiment itself, fulfilling the transversal requirements of Remote Laboratories. The developer of a new Remote Laboratory should be able to see WebLab-Deusto as a tool for developing the desired experiment without implementing the rest of the requirements.

In order to do so, the architecture must be flexible in terms of required technologies. If the architecture requires Java Applets, and the developer of a new Remote Laboratory does not count with experience in the development of Java Applets, or he wants to reuse legacy code developed using Adobe Flash, the developer will probably not use the architecture. The same applies to the server side.

1. *Developing the experiment*

The approach of WebLab-Deusto for solving this challenge is embracing the XWL (eXtensible WebLab) Architecture. In this Architecture, the developer of the Remote Laboratory can use any web-based client technology that can interact with JavaScript in the client side –which includes both Adobe Flash and Java Applets, among others–, and any technology supporting XML-RPC –which includes any widely used technology– in the server side.

Once the developer has selected which technology is most suitable for the client side, he must write a small application that provides a user interface showing the experiment –basically the inputs and outputs of the experiment–, and that consumes an simple interface for sending commands (strings) to the server and retrieving responses from it (also strings). This interface is provided by the architecture implementing the XWL Architecture, so the developer will not lead with communications. The application will be loaded by the framework as soon as the experiment is reserved, so the application will not implement any reservations management mechanism neither any authorization mechanism.

Then, the developer will need to implement the server side application that interacts directly with the hardware. The framework will call XML-RPC methods of this application to interact with it, sending programs and commands and receiving responses. This application will provide a method for starting the experiment, another for disposing its resources, and some methods to interact with the hardware. It does not need to manage reservations, concurrency or user-related information.

Both sides are communicated through the framework implementing the XWL Architecture. The framework does not understand the commands sent and received, since they are simple strings designed by the developer of the remote experiment. So it is responsibility of the developer of the remote experiment to properly serialize and deserialize those commands in both the server and the client side in order to use them.

2. *Using the XWL-enabled framework*

The framework implementing the XWL Architecture will provide the rest of requirements. It manages the authentication of the user, the communications between client and server, the administration management (including logs of the activities of the users), issues related with scalability and security, and the management of resources. In WebLab-Deusto, these features are described in Fig. 2.

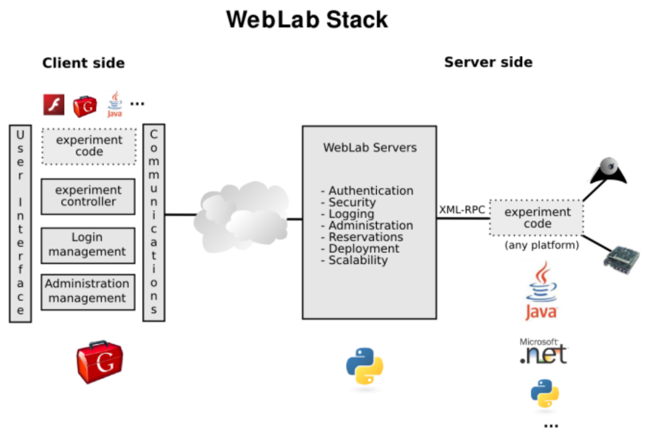

Fig. 2. The XWL Architecture: WebLab Stack

This way, when the user accesses the system through the architecture, it checks the credentials and permissions of the user, and returns the available experiments for this user. Then the user chooses what experiment he wants to use, and the system will manage the queues or the required schedules to check that there is actually a free experiment deployed that he can use.

Once the experiment is assigned to the user, in the client side the application developed by the developer of the experiment will appear, inside the XWL client (see Fig. 3). When the user presses buttons or sends files in this application, the application will delegate to the system sending them to the application in the server side. The system will do this as configured by the System Administrator, but the developer of the experiment is not aware of how.

This way, the developer of the experiment delegates most requirements to the system, and gets benefits transparently from all the new features that system implementing the XWL Architecture may add without changing the code.

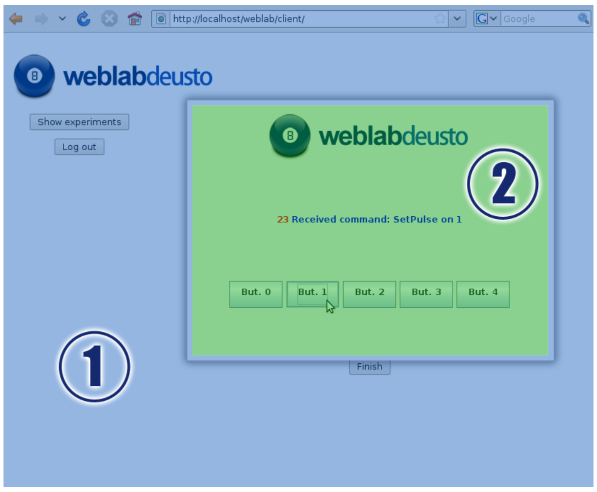

Fig. 3. "1" is the WebLab code, implemented in AJAX, and "2" is the Experiment code, implemented in Java in this case.

3. *Further considerations*

The XWL architecture should be implemented using AJAX, so the platform does not require any plug-in installation to the end user.

However, if the developer of an experiment selects Java Applets as client side technology, the users of that experiment will need to have the Java plug-in installed. Anyway, the users of the rest of the experiments will not need Java.

When selecting the client side technology, the developer of the Remote Experiment must be aware of this kind of problems, which have already been studied in [2]. While from the point of view of the WebLab-Deusto team AJAX is a proper technology for the Remote Laboratories development, under certain circumstances other technologies might fit better. This is the main rationale for supporting other technologies in the XWL Architecture.

E. Results

WebLab-Deusto, implementing the XWL Architecture, has been successfully used by students in the University of Deusto since February 2005, along the three different versions. WebLab-Deusto v.3 started being used by students in October 2007, and since then it has been used in five different classes with four different experiments, deployed along three different rooms of the Faculty of Engineering with four different dedicated servers. It has provided CPLD, FPGA and Microchip PIC experiments, as well as experiments regarding GPIB [11].

V. CONCLUSIONS AND FURTHER WORK

This paper has shown several software aspects that should be considered before designing the architecture of a Remote Laboratory, and the benefits of it. Consequently, we have applied these aspects to our own WebLab in order to progress to a scalable, extensible and flexible Remote Laboratory.

For future work we plan to communicate the Remote Laboratory with Learning Management Systems so as to reuse all the features they provide for online education. We also plan to increase the number of experiments and devices, and we plan to add a socket based alternative in the client side. We also plan to add more protocols to the communications engine for performance reasons.

ACKNOWLEDGMENT

The authors would like to acknowledge to the Spanish Science and Innovation Ministry for the support in the project TIN2008-06083-C03/TSI "s-Labs – Integración de Servicios Abiertos para Laboratorios Remotos y Virtuales Distribuidos"

REFERENCES

[1] C. Gravier, J. Fayolle, B. Bayard, M. Ates and J. Lardon, State of the Art About Remote Laboratories Paradigms - Foundations of Ongoing Mutations. International Journal of Online Engineering, Vol 4, No 1 (2008).
[2] J. Garcia-Zubia, P. Orduña, D. López-de-Ipiña, U. Hernández and I. Trueba. Section III - Remote labs development issues, Remote Laboratories from the Software Engineering point of view. July 2007. ISBN: 978-84-9830-077-2
[3] I. Gustavsson, J. Zackrisson, H. Åkesson, L. Håkansson, I. Claesson. and T. Lagö. Remote Operation and Control of Traditional Laboratory Equipment. International Journal of Online Engineering, Vol 2, No 1 (2006).
[4] MIT iLab. http://ilabcentral.mit.edu/
[5] Harward et al, "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
[6] Hardison et al, "Deploying Interactive Remote Labs Using the iLab Shared Architecture," Frontiers in Education (FIE) Conference, Saratoga Springs, New York, October 22-25, 2008
[7] D. López de Ipiña, J. García-Zubia and P. Orduña. Remote Control of Web 2.0-enabled Laboratories from Mobile Devices. 2nd IEEE International Conference on e-Science and Grid Computing, eScience 2006, Amsterdam (Netherlands), December 2006, ISBN 0-7695-2734-5
[8] MySQL, http://www.mysql.com/
[9] Google Web Toolkit, http://code.google.com/webtoolkit/
[10] E. Sancristobal, S. Martín, R. Gil, C. Martínez, E. López, N. Oliva, F. Mur, G. Díaz, M. Castro. Development and Interaction between LMS Services and Remote Labs. International Journal of Online Engineering (iJOE), Vol 4, No 3 (2008).
[11] J. García Zubia, P. Orduña, U. Hernández, I. Angulo, J. Irurzun. Students' review of acceptance, usability and usefulness of WebLab-Deusto. Journal of Digital Information Management (Journal of Digital Information Management. ISSN: 0972-7272. Vol. 7., no. 3.). 2009.