Specifications of the lab-owner services and cloud services - Initial

Elio San Cristobal Ruiz, Christophe Salzmann

Go-Lab

Global Online Science Labs for Inquiry Learning at School

Collaborative Project in European Union's Seventh Framework Programme

Grant Agreement no. 317601

Deliverable 4.1

Specifications of the lab-owner services and cloud services - Initial

Editors	Elio San Cristobal (UNED) Christophe Salzmann (EPFL)
Date	29 October 2013
Dissemination Level	Public
Status	Final

The Go-Lab Consortium

Beneficiary Number	Beneficiary name	Beneficiary short name	Country
1	University Twente	UT	The Netherlands
2	Ellinogermaniki Agogi Scholi Panagea Savva AE	EA	Greece
3	École Polytechnique Fédérale de Lausanne	EPFL	Switzerland
4	EUN Partnership AISBL	EUN	Belgium
5	IMC AG	IMC	Germany
6	Reseau Menon E.E.I.G.	MENON	Belgium
7	Universidad Nacional de Educación a Distancia	UNED	Spain
8	University of Leicester	ULEIC	United Kingdom
9	University of Cyprus	UCY	Cyprus
10	Universität Duisburg-Essen	UDE	Germany
11	Centre for Research and Technology Hellas	CERTH	Greece
12	Universidad de la Iglesia de Deusto	UDEUSTO	Spain
13	Fachhochschule Kärnten – Gemeinnützige Privatstiftung	CUAS	Austria
14	Tartu Ulikool	UTE	Estonia
15	European Organization for Nuclear Research	CERN	Switzerland
16	European Space Agency	ESA	France
17	University of Glamorgan	UoG	United Kingdom
18	Institute of Accelerating Systems and Applications	IASA	Greece
19	Núcleo Interactivo de Astronomia	NUCLIO	Portugal

Contributors

Name	Institution
Christophe Salzmann, Sten Govaerts, Denis Gillet	EPFL
Elio San Cristóbal, Irene Lequerica, German Carro, Miguel Latorre, Agustín Caminero, Antonio Robles, Gabriel Díaz, Manuel Castro	UNED
Pablo Orduña	UDEUSTO
Danilo Garbi Zutin	CUAS
Anjo Anjewierden	UT
Mavromanolakis Georgios	EA

Legal Notices

Executive Summary

This deliverable is the initial version of the lab owner and cloud service specifications. It is divided into three major Sections. The first Section introduces online labs and related definitions. Then, the smart device paradigm is defined and presented as a metaphor to specify the services that a remote lab should provide to be seamlessly integrated in the Go-Lab infrastructure. Finally, the smart gateway is presented as an extension of the smart device to specify cloud services that enable the reuse of existing online labs.

Table of Contents

1 Introduction

Existing online labs usually rely on ad-hoc solutions developed for specific scenarios. They are providing dedicated functionalities supported by diverse IT architectures [1]. As a consequence, their reusability is limited. In addition, the vast majority of the existing online labs are not integrated in learning environments.

In order to face these challenges, the Go-Lab project integrates a set of technical tasks whose goal is to provide a set of specifications and guidelines enabling lab owners to seamlessly integrate their remote labs (task 4.1) and a set of specifications to ease the sharing of existing labs (task 4.2) into the Go-Lab infrastructure (see D5.2).

This deliverable is an initial version of the lab owner and cloud service specifications. It is divided into three major Sections. The first Section introduces the online lab concepts and related definitions. Then, the smart device paradigm is defined and presented as a metaphor to specify the services that a remote lab should provide to be seamlessly integrated in the Go-Lab infrastructure. Finally, the smart gateway is presented as an extension of the smart device to specify cloud services that enable the reuse of existing online labs.

2 Online Labs

Within the Go-Lab project, online laboratories (referred hereafter as online labs) have been divided into three general categories (see deliverable 2.1):

- **Virtual labs** are simulations of real labs. Their strength is that they can be used in all educational fields, from chemistry to electronics [2-3]. They however rely on complex dynamic models and animations.

- **Remote labs** are real labs remotely accessible [4]. Remote labs are exploited in a wide range of scientific fields in which phenomena are fast enough for live observation, such as computer science, physics, electronics, automation and control [5-10].

- **Data sets** are measurements gathered using real scientific instruments such as telescopes. Recorded data sets can be exploited directly when access to such instruments is limited.

Task 4.1 and task 4.2 are focusing on remote labs, which is the category for which there are significant challenges in terms of plugging and sharing. This Chapter presents a definition of remote labs, the problems that throughout their evolution have arisen, the steps taken to overcome them and the new concepts introduced within the Go-Lab project framework.

2.1 Definition of remote laboratories

Remote laboratories are physical laboratories that can be operated at distance and that offer to students the ability to conduct real experiments and collect real data (see deliverable 2.1).

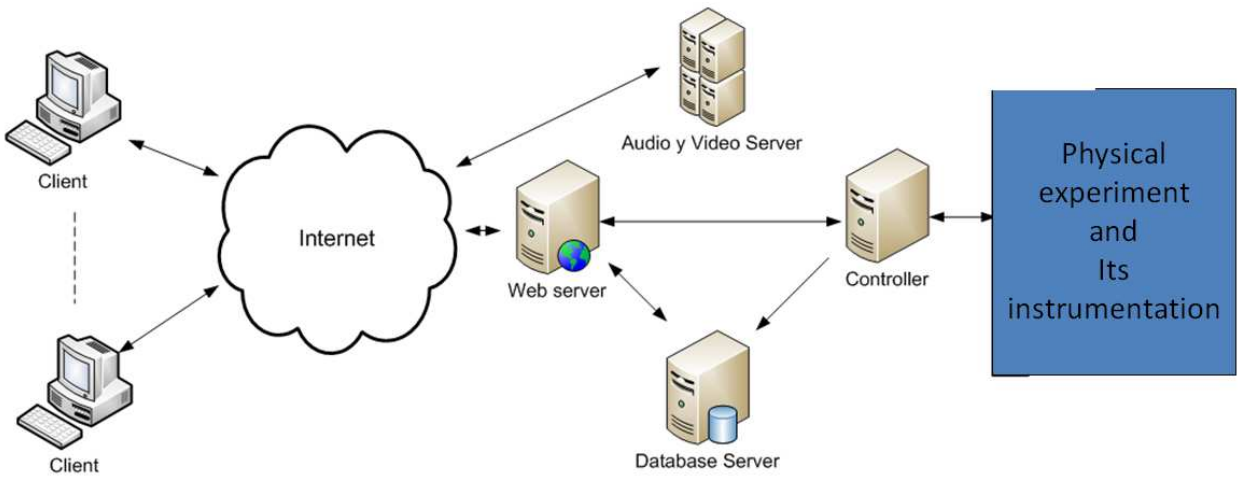

Figure 1. General architecture of a remote lab.

Currently, the typical architecture of a remote lab (Figure 1) is composed of:

- A **Physical experiment and its instrumentation**, which is the physical object the students are interacting with and the instrumentation enabling such interaction.

- A **client** application, usually a user interface application running on a Web browser, that allows users to interact with the experiment by setting or changing configuration and operation parameters and observing their effects through measurements delivered by the instrumentation.

- A **Web server,** which is in charge of delivering Web content to the client, interacting with a Database server and sending parameters to the Controller.

- The **Controller** is connected to the experiment. It receives parameters from the Web server, translates them, and resends them to the experiment. It also interacts with the Database Server in order to store measurements.

- The **Database** server stores information such as user credentials, experiment configurations, or measurements.

- **Audio and Video Server** sends video and audio signals. For example, the live video of the experiment or the ambient audio.

The Controller, the Web server, the Database server and the Audio and Video Servers can be running on a single computer or on several ones as depicted in Fig.1.

Currently, a large number of remote labs can be found on the Web following the architecture described in this Section or a slight modification of it, such as the Faulkes telescope or the ELVIS / OP - AMP Labs (for more information, see deliverable 2.1).

2.2 Reusing remote labs and integrating them in learning environments

The generic architecture for remote labs described in Figure 1 can be implemented using different designs, technologies and software solutions. Therefore, most remote labs rely on ad hoc solutions that are difficult to share or reuse. To tackle this problem, several initiatives have been launched to provide schools, universities and organizations with common management architecture to implement their labs and then share them broadly (for more information see Appendix A). This architecture is known as a Remote Laboratory Management System (RLMS). The main RLMS initiatives are:

- **LabShare**. Led by the University of Technology, Sydney, is a joint initiative of the Australian Technology Network: Curtin University of Technology, Queensland University of Technology, RMIT University, University of South Australia, and the University of Technology, Sydney (http://www.labshare.edu.au/). This project aims at creating a

national network of shared remotely accessible laboratories. To do this, they have developed a software architecture called SAHARA [11-13].

- The **iLab** Shared Architecture (ISA). Implemented by the MIT to facilitate the rapid development of new Web labs and to provide a mechanism for students from one university to use the experiments and the instruments of another university (https://wikis.mit.edu/confluence/display/ILAB2/Home) [14-16].

- The **WebLab-Deusto** project. It is an open source project providing a Web-based, experiment-agnostic, scalable software infrastructure, which permits the University of Deusto to offer several labs to its students through the Internet (https://www.weblab.deusto.es/web/weblab-deusto.html) [17-19].

- **LiLa** (Library of Labs). An European eContentPlus project that promotes a portal of Online Labs resources and fosters exchanges of experiments among institutions (http://www.lila-project.org/) [20-22].

Despite of these initiatives, there is a lack of management integration among them and therefore sharing labs is still difficult. This is recognized as the main problem for a wide dissemination of remote labs.

In the Go-Lab framework (task 4.1 and 4.2) two new concepts are defined and applied for developing, standardizing and reusing new or already existing remote labs. The first one is the smart device paradigm which focuses on specifications for developing new remote labs (see Section 3). It revisits the traditional remote lab's client-server approach and re-engineers the server-side components to add more agility and adaptability. This paradigm enables the definition of services that permits a complete decoupling between the server and the client components. The second one is the concept of smart gateway (see Section 4), which enhances existing solutions such as iLab or LabShare, to make them behave as smart devices.

Another important issue with remote labs and RLMS, is the lack of learning environment integration where students can carry out experiments and other e-learning activities. This is an added value in the Go-Lab project to enable remote labs implemented as smart devices or interfaced through the smart gateway to be easily integrated in inquiry learning phases and to be supported by learning analytics and scaffolds (deliverable 1.1 and 5.2).

The following Sections, 3 and 4, describe the development of new remote labs to be integrated within the Go-Lab infrastructure (as smart devices), and the integration of existing remote labs into the Go-Lab infrastructure (through a smart gateway). Section 5 describes briefly how these two technical solutions can be integrated in the Go-lab Portal.

3 Lab-Owner Services - Plug Technology

There are many remote labs available on the Internet. Unfortunately, the necessary effort to make them accessible by other users or organizations is a strong inhibitor for reusing them. This Section presents specifications to develop and enable an easy plug of new remote labs. These initial specifications will be refined in upcoming deliverables following evaluation in pilot phases. The core idea is to rely on the smart device paradigm to provide a consistent frontend to new remote labs.

3.1 Introduction

The Internet has experimented a tremendous growth in the last decade, evolving from a network of a few hundred hosts to a platform linking billions of “things” such as sensors, computerized devices, RFID tags, computers and their associated applications [23]. This growth drives previous Internet developments into a new pervasive paradigm in computing and communication. This new paradigm enhances the traditional Internet into a smart Internet of Things (IoT) created around intercommunications of diverse objects in the physical world [24-25]. Some of these physical objects are hardware devices with the ability to manage themselves intelligently, they are referred as smart devices.

This Section introduces the smart device paradigm to elicit the initial specifications for the creation of new remote labs. The smart device paradigm is used as a metaphor to define a software component that can easily be integrated in the Go-Lab infrastructure. The deliverable 4.5 in month 30 will presents the final specifications and complete examples.

3.2 Smart device paradigm

Smart devices find their roots in smart sensors. A smart sensor is a sensor that is capable of transmitting its state or data representing measurements. These sensors can be combined to compose a sensor network. While smart sensors have limited “intelligence”, i.e., they only broadcast measured values; smart devices have some onboard “intelligence”. The interconnection of smart devices sketches the Internet of Things (IoT). Thomson [35] gives the following requirement for a smart device:

I. A smart device has sensors and actuators,
II. A smart device has identity and kind,
III. A smart device has memory and status tracking,
IV. A smart device has communication capabilities,
V. A smart device is capable of reasoning and learning.

3.2.1 Description

The smart device paradigm defines an ideal device that is capable of understanding any type of existing or future requests and taking the appropriate actions autonomously according to receive requests, this using any type of existing or future protocols and standards. While this is unrealistic, the internal architecture of the smart device should be agile [30] enough to easily be adapted to new requests, standards and protocols. The smart device paradigm revisits the traditional client-server approach used to implement ad-hoc remote labs solutions. It enables

the definition of services which fully decouple the server from the client. In addition, it suggests the re-engineering of the server-side components to add agility and adaptability. A clear client and server decoupling through services enables an effortless integration into learning environments and social media platforms.

The Go-Lab smart device specifications aim at defining external services to access the real world through actuators and sensors, and to manage the device using internal services (Fig. 2.). The smart device services differ from traditional solutions that often provide a monolithic interface without the possibility to access a specific service [31]. There is no assumption regarding the communication channels for smart devices. The Internet is the obvious choice, but other ad-hoc channels may be used instead to fulfil specific criteria.

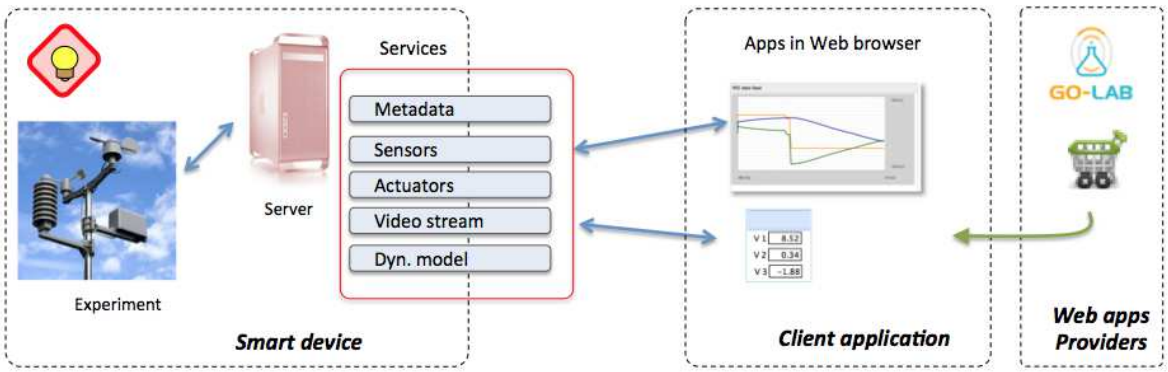

Figure 2. The smart device can be accessed through services. The client application relies on Web apps that render data transmitted by the various services.

The smart device paradigm does not include a User Interface (UI) application, but for convenience it often proposes a minimal set of UI components that can be rendered at the client side. This client-server decoupling implies that the client application (UI) can be designed by different providers. Web browsers are the preferred environments to render the UI at the client side, and several Web apps can be aggregated to provide a complete UI. There is often a direct relationship between a smart device service and the app that renders the information in the client UI, for example an oscilloscope app that displays temperature evolution measured by a smart device sensor.

3.2.2 Initial specifications for remote labs as smart devices

The smart device paradigm defines an ideal autonomous device capable of adaptation that can be accessed through well-defined services. This paradigm is used to define the specifications for the Go-Lab remote labs. To define a smart device for a remote lab, additional information and service definition should be added to the Thomson's initial definition. A generic smart device could already be seen as an autonomous remote lab. On the other hand, it is not targeted for any specific purpose and therefore the expected outcomes may not be satisfactory. The principal aim of remote labs is to represent, at the client side, its partial or full state and to enable real-time interaction. It could be implemented in the form of a simple oscilloscope window depicting the temporal evolution of a given sensor or a full 3D representation of the system combined with multi-camera streams. Interacting with the physical lab by directly controlling actuators or indirectly through a supervision stage (local controller or other logic) should also be possible. When considering online labs, the client side that renders the server

information is also to be taken into account. Client applications that connect to online labs are typically, but not necessarily, a Web browser. This specific choice of open Web technologies enables a broader compatibility and favours adaptation as well as adoption. Proprietary technologies (java, flash, etc.) should specifically be avoided since they remove the solution ubiquity. The smart device paradigm enables the rethinking of such interface into a Web 2.0 interface.

The smart device for a remote lab is seen from the client side through services. A fine definition of these services permits the trustworthy decoupling between the client and the server. Some of these services are dedicated to serve the client application while others are only meant for the smart device itself. The smart device additional intelligence and agility mainly comes from these internal services. The service definition enables any user to design his/her own interface to the smart devices for remote labs. Smart devices for remote labs should provide the following services (Table 1).

Services	***Required***	***Internal/External***
Metadata	**Yes**	external
Sensor, instrument and actuator access	**Yes**	external
Self and known state	**Yes**	internal
Alarms and logs	**Yes**	internal
Local control	No	external
Data/video streaming	No	external
Authentication	No	external
Serving client apps	No	external
Graphical model	No	external
Mathematical model	No	external
Model parameters	No	external
Simulation engine	No	internal

Table 1. Smart device services.

Metadata: this service returns a structured list of services offered by the smart device. The service description (capabilities, format, etc.) returned by this service is sufficient to exploit the other services offered by the smart device.

Sensor, instrument and actuator access: this service interfaces the available sensors, actuators or instruments part of the lab. The smart device's sensors and actuators allow to measure and act on the physical equipment or on complex instrument, like a telescope.

Self and known state: the service ensures that the smart device is able to come back to a fully defined and safe state after a user completes his/her experimentation or after a power failure or a regular start up/power on.

Alarms and logs: detected errors and anomalies are handled by this service. Wrong usage or network attacks also triggers alarm and appropriate actions. Much information will be logged on the smart device and provided as needed, such as user activity, network activity or internal activity.

Local control: The physical lab connected to the smart device may require a local controller to operate safely or according to certain requirements. The controller algorithm, the parameters and the internal state can be accessed via the parameters service. The physical security of the physical system *must* be ensured at *all time*.

Data / video streaming: Other sources of data and information may not be related to actuators or sensors. Typically a video stream, a live parameters stream or an activity stream can be handled by this service.

Authentication: This service ensures that the current client is correctly identified and has the right to talk to the smart device at this given time. If not, the access will not be granted and an alarm should be issued. The reservation and booking mechanism may be partially handled by external entities, in this case the smart device only verifying correct authentication.

Serving client apps: The smart device may provide a basic user interface in the form of apps.

Graphical model: this model describes the physical equipment such that it can be rendered/drawn at the client side in real time. The model can be 2D or 3D.

Mathematical model: this model describes the dynamics of the physical equipment as mathematical equations. These equations can be integrated at the client side to perform a simulation.

Model parameters: represents the parameters of the mathematical model. They may have been identified on the physical system.

Simulation: simulation engine that computes the state of the smart device and mimic the sensor and actuator. It can be used when there is no concurrent access. Also simulation can run slower or faster that real-time.

3.2.3 Smart device services protocols

The previous Section defines the smart device services for remote labs. At this point, the protocol used by the service needs to be specified. While the smart device paradigm suggest the support of any protocols, the Go-Lab infrastructure, the considered remote labs and the targeted audience impose Web 2.0 protocols combined with efficient real-time transmission. Nowadays Websocket [33] is the most efficient transmission mean at the browser level. Websocket are similar to TCP/IP sockets but at the browser level.

To simplify the communication handling at both the smart device and the client application (Web apps) the JSON encoding [34] has been chosen by default for the Go-Lab smart device. The JSON protocol has the advantage of being both readable by a human and processable (parsed/generated) by a machine. For example to specify a sensor service that broadcast temperature, the metadata would contain:

```
{
  "name":"Temperature1",
  "unit": "Celsius",
  "range": "min: -20 max:60",
  “mode”: push,
  “rate”: 1,
  “wsURL”:“ws://server:port/T1”
}
```

By parsing the above information, the Web app has the needed information to connect to the temperature sensor via the given Websocket. The next step for the Web app is to open a connection to the provided wsURL and wait for the temperature measurements to arrive at a rate of 1 per second. The receive information will look like the following:

```
{
  "name": "Temperature1",
  “value” : 23,
}
```

A JSON schema could be provided by the lab owner to validate the exchanged information [34]. Enforcing such schema is under evaluation.

With the above information, the data exchange between the smart device and the web apps are defined. The metadata service provides a list of available services and the associated information. The other services provide/consume the data according to the metadata service. Only the main URL is needed to access the smart device, it will be provided through the Go-Lab portal.

3.3 Scenarios

The Section presents two examples of remote labs that are built following the smart device specifications. This ensures an efficient and easy integration with the Go-Lab infrastructure. On the software side, these two examples are built from scratch. Section 4 sketches how to extend an existing remote lab following the smart device specifications.

In the first scenario the remote lab runs on a dedicated embedded hardware (Arduino) in the second example the remote laboratory runs on a desktop computer.

3.3.1 RGB LED: Smart device for remote labs on embedded hardware

Nowadays, there are many alternative hardware and software solutions which are able to manage themselves and manage other devices connected to them, such as Raspberry pi and Arduino. This subsection describes briefly one of the well-known devices exploited in Go-Lab to implement the smart device specifications by combining an integrated hardware and software solution.

The RGB LED lab demonstrates the required steps to integrate a simple physical device (the RGB LED) into the Go-Lab infrastructure. It first describes the pedagogical context, then explain how to interface the physical LED using embedded hardware. The next step is the development of the software according to the smart device paradigm. Finally the integration to the Go-Lab infrastructure is presented.

Pedagogical Context

The RGB LED allows teachers to discuss and explain the process of perception of the human eye and brain, the different perception of colours that each person can have, and could be used for other purposes, such as:

- The formation of human vision: What is the mechanism that causes light passing through the optic nerve and it is shown in our brain as sharp image.

- The physics of colour: How we can see some colours and not others, what frequencies are perceptible by the human eye and which ones are not, how animals perceive colours, or how the insects see.

- Primary colours: What primary colours there are, how their combination can lead to other colours, what types of combinations there are.

- Representation of colours: How colours are represented in the real world, in digital format, on TV, on canvas, or on paper.

- Effect of colours on the mind of people: Why do certain colours convey certain feelings to the observer?, what mechanism articulates these reactions in the brain?, why some people like some colours and others like others?

This lab can be integrated into inquiry learning spaces, as described in the deliverable D.1.1, and complemented with other guidance tools to support scenarios mentioned above.

Arduino Development

Arduino is a flexible microcontroller and development environment that can not only be used to control devices, but also to read data from all kind of sensors. Its simplicity and extensibility, in addition to its great success and adoption by users has led to the development of a variety of hardware extensions and software libraries that enable wired and wireless communication between Arduino and the Internet [26][27].

The microcontroller on the board is programmed using the Arduino programming language, based on Wiring [28] and implemented in C/C++. It also provides an Arduino development environment which consists of a text editor for writing code, a message area, a text console, a toolbar with buttons for common functions, and a series of menus. The development environment connects to the Arduino hardware in order to upload applications and interact with them.

This microcontroller can be connected to the Internet through hardware extension which are called Shields. The most well-known Shields are:

- The Arduino Ethernet Shield connects the Arduino to the Internet in a few minutes.
- The Arduino WiFi Shield provides a wireless connection between the Arduino and the Internet.
- The Arduino GSM Shield connects the Arduino to the internet using the GPRS wireless network.

Therefore, Arduino is a microcontroller which can interact with the environment by receiving input from a variety of sensors and can perform actions as controlling lights, motors, and other actuators (for more information, see Appendix B).

Development

This subsection describes the development carried out and the draft specifications of services proposed in 3.2.2.

Once the pedagogical purposes of this lab have been defined, it is necessary to select the hardware components needed to fulfil the requirements. RGB LED lab is composed by an Arduino One, the Ethernet shield, a protoboard which implements the circuit that modifies the red, blue and green channels of the LED, (see Figure 3), and an IP camera. The Ethernet shield can be replaced by a RPi or PC connected to Arduino by a USB connector (For more information, see Appendix B).

The Arduino has been programmed to modify the Red, Green and Blue channels, this are LED actuators, that are provided as services by the Ethernet shield to Internet. The camera provides the video for teachers and students.

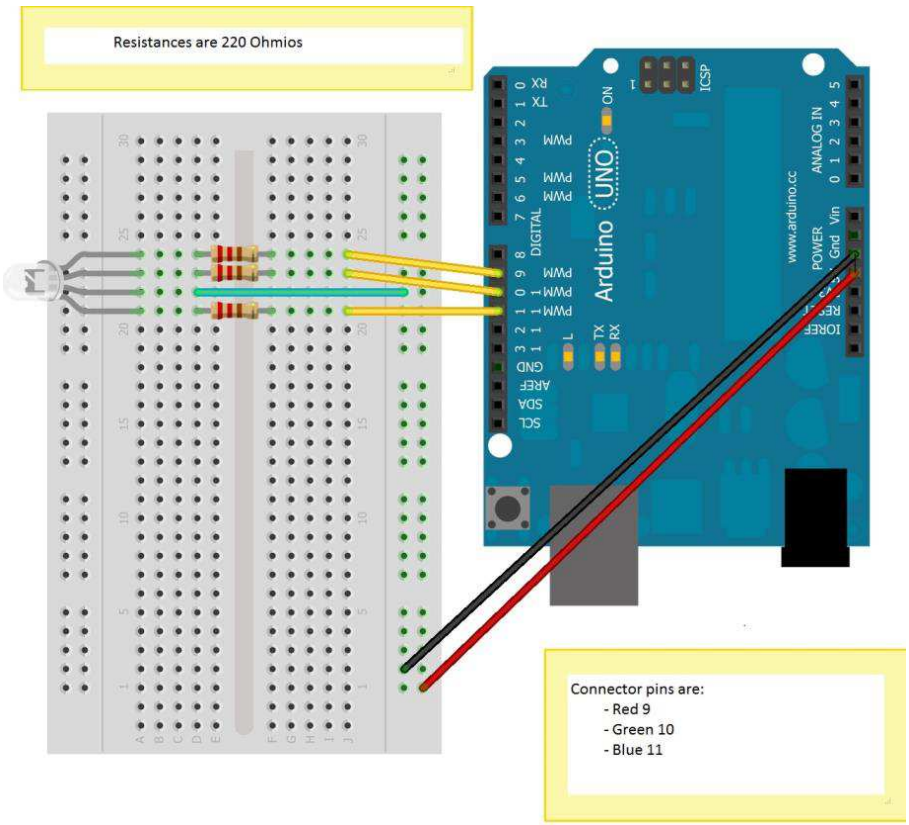

Figure 3. Hardware implementation of RGB LED lab.

Once the basic schema has been implemented, a set of services must be designed and implemented. These services provide the desired additional intelligence, see Section 3.2.2.

- **Metadata (required, external).** This service provides a description of the smart device, its components and a brief description about their features. An example of the *JSON pseudo code* for the RGB LED is:

{
metadata version: September 24 2013 - 1432
authors: Chris, Elio, Pablo
institution: EPFL, UNED, UD
format:
Available smart services: (1)
sensors: (0)
actuators: (1)
1: name: RGB LED parameters (3)
Param
1: name: RED
unit: percent

range: min: 0 max:100

2: name: GREEN

unit: percent

range: min: 0 max:100

3: name: BLUE

unit: percent

range: min: 0 max:100

experiment: This experiment illustrates the additive colour model. In the client interface the users will be able to adjust three sliders that represent the three primary colours. Red, Green and Blue (RGB). These three sliders will control an RGB LED located in a protoboard.

Access: free access, no booking possible, first come first served

Streaming device (1)

1: name: camera1

format: RGB, w:640, h:480

compression: JPEG, adaptive

refresh rate: 10 Hz

access right: no

...

Physical model: none

Mathematical model: none

Apps (2)

1: name: RGB Sliders

format: W3C

URI: /RGBSliders.xml

2: name: Webcam Streaming

format: W3C

URI: /WebCam.xml

}

- **Sensor, instrument and actuator access (required, external).** This service allows receiving data from sensors and sending data to the actuators. In our RGB lab that would be changing the colour of the LED through the actuators RED, GREEN and BLUE

-> {RED:100, GREEN:100, BLUE:0 } // this should give a yellow colour

<- {100,100, 0 } // the server acknowledge the commands

- **Self and known state (required, internal).** The device smart device should be able to come back to a fully defined and safe state. The RGB LED should be able to turn back a safe state when the experiment ends.

-> {experiment ended } // experiment done

<- ok // Initial state

- **Alarms and logs (required, internal).** This service should store information as user access, user activities, etc. and should send alarms if there are attempts of unauthorized accesses, etc. I.e.: the RGB LED lab should store the use of the actuators by students.

- **Local control (optional, external).** RGB LED lab controls a LED, therefore it does not need to control that the LED operate safely or according to certain requirements. E.g., if

the smart device controls a pendulum then this service should control the oscillation period and other issues to avoid problems with the physical pendulum.

- **Data/video streaming (optional, external).** This service is focused on video stream and others like stream activities. An example of this for our RGB LED lab could be:

  -> {video: source: camera1, resolution: 640 x 480, compression: JPEG}
  <- {header:..., payload: JFIF 6 06 07 06 05 08 ...}

- **Authentication (optional, external).** Basic authentication could be required for some labs. The smart device will check that the "client" has the right to interact with it. In our example, as is described by the metadata service, the access to RGB LED lab is free.

- **Serving client apps (optional, external).** The smart device may provide a basic user interface in the form of apps. The Ethernet Arduino shield provides a publishing facility that allows publishing web pages in the Web to modify Red, Blue and Red channels. The client application service uses basic Web apps, see example:

  -> JSON {app name: Color Sliders}
  <- JSON {W3C Opensocial: URI://.....xml }
  XML or any other format

- **Graphical model (optional, external).** This model represents the physical equipment such that it can be rendered/drawn at the client side. The model can be 2D or 3D. It is not currently developed for our RGB LED lab.

- **Mathematical model (optional, external).** This model describes the dynamic of the physical equipment as mathematical equations. These equations can be integrated at the client side to perform a simulation. The RGB LED lab uses a triad of colours (Red value, Green value and Blue value) and upon this triad a mathematical model could be used in order to represent another colour model such as CMY:

  // RGB -> CMY
  //RGB values = From 0 to 255
  // CMY values = from 0 to 1

  $C = 1 - (R / 255)$
  $M = 1 - (G / 255)$
  $Y = 1 - (B / 255)$

- **Model parameters (optional, external).** Represents the parameters of the mathematical model. They may have been identified on the physical system. The RGB LED lab has the parameters Red, Green and Blue.

- **Simulation (optional, external).** Simulation engine that represents the smart device and mimic the sensor and actuator interaction. This possibility is quite useful when there is no real access to the experiment. Also simulation can be used to run slower or faster

than the real-time experiments. This simulation uses the mathematical model and parameters from smart devices.

Go-Lab Integration

The RGB LED lab example provides two basic Web apps through “Serving client apps” service. The first one is a set of sliders to modify the Red, Green and Blue channels, and a second one streams the video from the camera.

These Web applications will be stored in the Go-Lab Portal and will be integrated in inquiry learning spaces, Figure 4, as a set ready to be used by students and teachers.

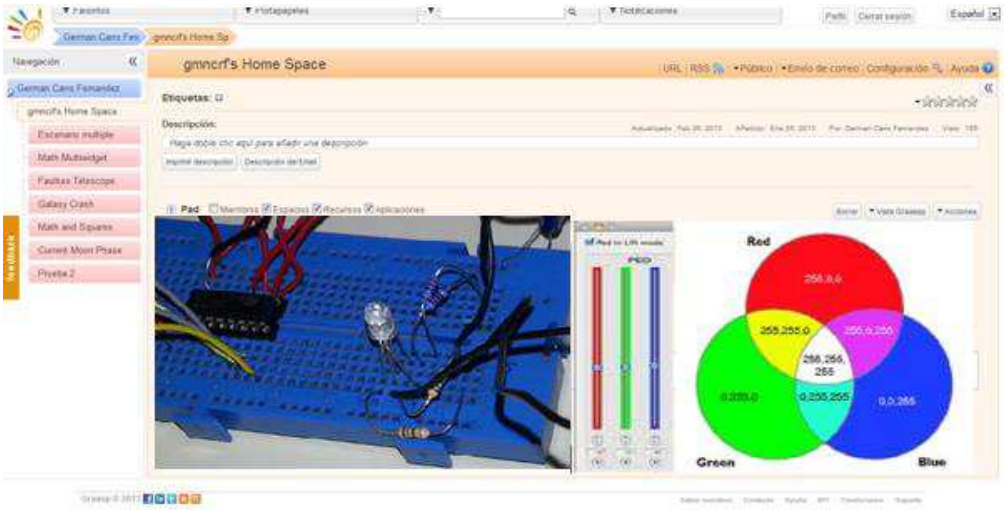

Figure 4. The RGB LED lab integrated in an inquiry learning space. The various Web apps connect to the specified services (video stream, actuators).

3.3.2 RED: Smart device for remote labs on desktop computer

The RED lab scenario presents the integration of a more complex experiment. In addition, this example shows briefly how an existing server has been redesign to follow the smart device specifications. It first describes the pedagogical context, and then explains the choice of smart device services that will be implemented. The in-depth server implementation will be provided as a software package ready to use by other lab owners. Finally, the integration to the Go-Lab infrastructure is presented. This scenario, especially the pedagogical context, takes advantage of the initial server described in [31]. The server hardware is kept the same while the software has been rewritten to follow the smart device specifications. The original server software was not designed to handle an evolving client application; it assumed an ad-hoc monolithic client interface written in Java. In addition, the technological choices concerning the communications did not permit a pure Web based interface.

Pedagogical context

Laboratory experiments remotely controlled are often mechatronic devices with moving parts as they exhibit visually observable dynamical behaviors. For example the laboratory-scale electrical drive (the experiment in Figure 5) is used in many textbooks and courses to illustrate motion control approaches. This setup consists of a DC motor equipped with a digital encoder.

The angular position is measured with a digital encoder connected to the motor axle. A rotating disk permits the visualization of the angular motion that is captured by a video camera. The whole hardware has been designed in such a way that it is secure, fully controllable and can be diagnosed from the connected computer. The electrical drive is connected to the computer through an AD/DA card.

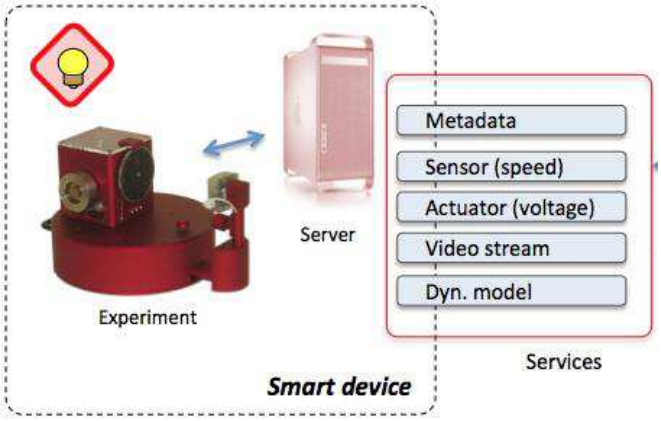

Figure 5. The RED smart device.

The software manages the electrical drive on one side and provides on the other side services to client applications and Go-Lab infrastructure. The smart device software implements a real-time controller for the electrical drive, a video acquisition component for the camera, a custom Web server to handle client requests and the needed components to manage internally the services.

RED services

The RED smart device implements the four required services. In addition, it implements the following services: *local controller*, *video stream, parameters stream* and *basic client app* (Web apps).

The built-in custom Web server is serving the Web apps source code (javascript) to remote users. In parallel, it handles websocket connections issued by the Web apps. These websocket connections refer to the services offered by the smart device. The websocket connections related to sensors and actuators are linked internally to the real-time data acquisition (AD/DA in Figure 7). The proposed services are:

- **Metedata:** A JSON encoded list if available services and related information.
- **Sensor, instrument and actuator access (required, external). T**here are 2 sensors, one for the disc angular position and one for the disc speed. In this example we focus on the position. Note that there is no direct access to the actuator (the motor voltage), the actuator is controlled via local controller and the parameter stream services.
- **Self and known state:** at power up, or when the investigation phase is completed, the server goes back in a predefined state, i.e., the motor voltage is set to 0.

- **Alarm and logs:** internally the server logs all access, it also checks that received packet is valid and raises an alarm if needed.

- **Data/video streaming:** a camera films the RED smart device disc (see centre of Figure 7). The frames are streamed as a succession of jpeg images.

- **Local controller:** the desired position/speed of the RED disc is enforced by a PID controller. The controller computes the voltage to apply to the motor according to both the current and the desired values of the sensor. The controller parameters are given by the Parameter stream.

- **Parameters:** 3 parameters are defined for this example: reference position, PID proportional gain and PID integral term. The controller will behave according to these parameters. The user will change these parameters in the user interface and see its the effect.

- **Serving client apps:** a basic user interface is provided as 3 Web apps, see center of Figure 7. The sensor, actuator, measurement are rendered is an oscilloscope window. The video is directly rendered by the provided Web apps. The 3 parameters can be changed using 3 sliders. Alternatively these Web apps can be provided by other entities.

All the needed information to connect and render the various smart device services is provided by the metadata service. It is the matter of entering a URL to use the Web app proposed by other as interface.

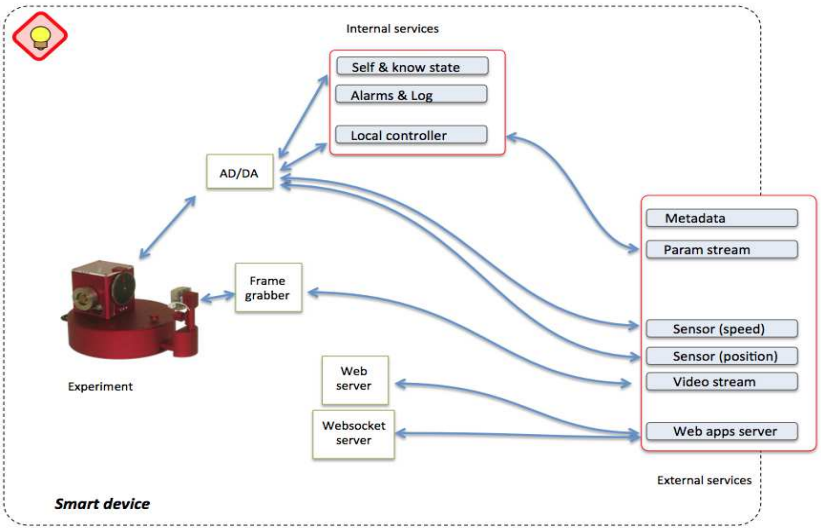

Figure 6. The RED smart device internal connection scheme.

Server implementation

The server software implemented as a smart device (https://graasp.epfl.ch/#item=asset_13490). It is written in LabVIEW, a graphical language designed especially but not only for data acquisition and control. A LabVIEW software package implementing the smart device specifications is in preparation and will be provided to lab owners who do not want to write their

remote lab server from scratch. By using this package they will be ensured of a seamless integration into Go-Lab infrastructure.

Go-Lab integration

The RED smart device user interface is visible in Graasp using the following URL: https://graasp.epfl.ch/#item=space_2185.

The RED smart device provides 3 basic Web apps to display the video stream, the measurement stream and to enter the internal controller parameters. There is another Web app not provided by the smart device to save current measurements.

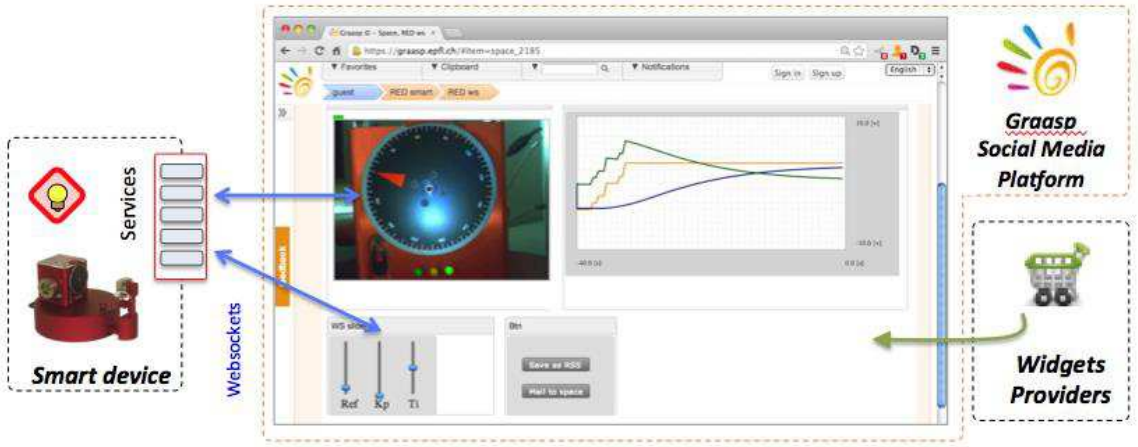

Figure 7. The smart device interface for the electrical drive rendered within an inquiry learning space. The various Web apps connect to the specified services (video stream, sensors, actuators, parameters).

3.4 Remarks

The RGB LED lab prototype provides an easy example to turn a physical lab into a remote lab accessible as a smart device. Both the hardware and the software necessary to plug the physical lab are provided. This is especially helpful if a physical lab is available, but is not connected yet online.

The RED prototype shows how a more complex smart device can be implemented. This prototype has been used to help defining the smart device specifications and services. Its internal structure is left open to the implementer (lab owner) but it should be easy to add sensors or implement a new release of the communication protocol. A LabVIEW package that implements an initial smart device will be provided freely. In this initial demonstration the authentication/reservation services are not implemented. They will be in a forthcoming release.

4 Cloud services

The next Section presents the smart gateway concept as an extension of the smart device to specify cloud services enabling the reuse of existing online labs.

4.1 Introduction

Cloud Services are services necessary to ensure compatibility and allow different laboratory providers to plug their systems in the Go-Lab infrastructure.

If a legacy remote lab cannot be updated to be in compliance with the smart device specifications, a smart gateway can be added as a layer between the legacy lab and the Go-Lab infrastructure to ensure this compatibility. One of the tasks of the smart gateway is the "translation" of legacy lab requests/protocols to smart device services.

Due to the differences between Remote Lab Management Systems (RLMS) and the fact that each system has different design goals, it was decided that the smart gateway should be as flexible and generic as possible to support a variety of these RLMS. Following these requirements, a plug-in architecture was chosen. Thus, the smart gateway relies on plug-ins to implement the translation required by native legacy RLMS. Plug-ins for well-known RLMS such as WebLab-Deusto or iLab Shared Architecture will be provided. Plug-ins for specific RLMS will be developed by lab owners according to the proposed specifications and guidelines.

The Cloud services rely on Lab Metadata. The initial specifications of these Metadata are used to describe the different lab resources in a machine processable way. The main purposes of the metadata are:

- To allow systems (RLMS, Lab repositories, etc) to share information about labs
- To provide mechanisms for discovery of labs
- To structure data in laboratories and RLMS databases
- To provide requirements for plugging/sharing labs with third party systems (e.g., Go-Lab Portal) and for the exploitation of labs
- To provide easy exchange of lab data and allow for advanced search mechanisms
- To support dissemination of labs

The metadata schema used to describe the resources in the Go-Lab is based on a combination and extension (based on Go-Lab requirements) of the ROLE Ontology and the GOLC specification [32]. The metadata schema is described in detail in deliverable D5.2.

4.2 Smart Gateway

The Smart Gateway is the component in the cloud services that enable lab owners to plug their systems into the Go-Lab infrastructure. It targets lab owners in the sense that it is transparent for teachers and students. The smart gateway implements the services specifications defined in Section 3 to expose existing legacy labs as smart devices.

Figure 8 shows the smart gateway connecting different labs to the Go-Lab infrastructure.

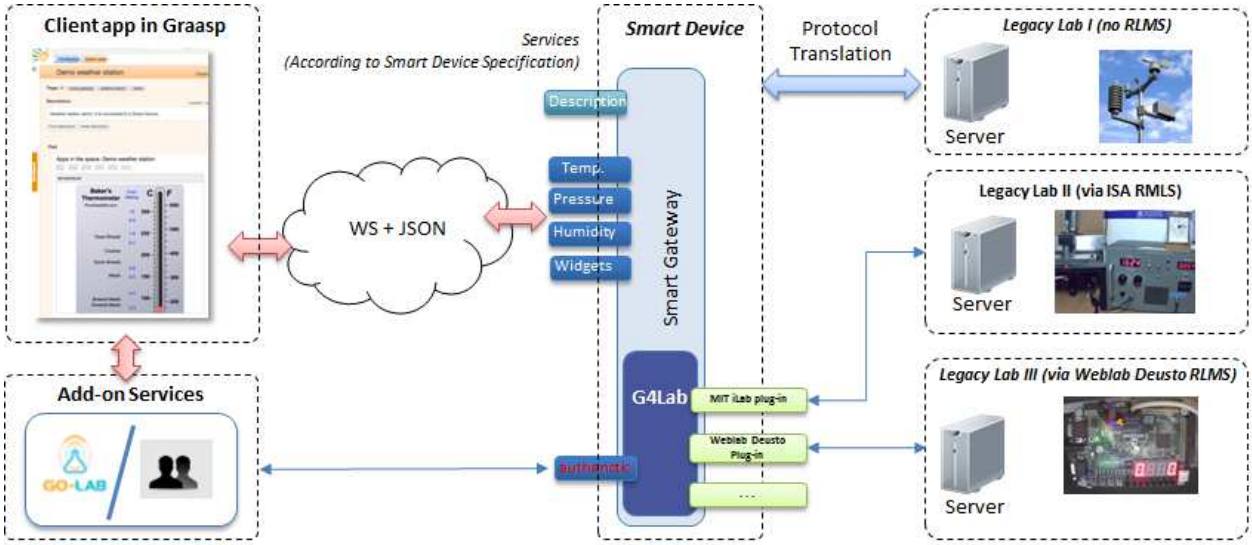

Figure 8. Legacy Labs and the Smart Gateway.

The Legacy Lab I is connected to a server that transmits data to a remote client app via the TCP/IP protocol. In this case the smart gateway mainly repackages legacy communication (TCP data) into services and provides the "translation" mechanism that converts Go-Lab authentication and reservation requests into its native legacy counterpart. This "translation" mechanism is implemented by the G4Lab component (G4Lab is described in detail in Section 4.4) of the Smart Gateway. If a legacy lab is available via a RLMS (Remote Lab Management System) a plug-in that implements the needed communication layer for a given native legacy RLMS is necessary in order to validate the user's credentials and possible reservations. Examples of such RLMS are iLab or WebLab Deusto. A more detailed description about the different RLMS mentioned here is available in appendix A.

4.2.1 Initial specifications

The Smart Gateway must provide the following functionalities:

- Act as a bridge to request reservations to the laboratory (or RLMS).
- Wrap the authentication and authorisation processes of the laboratory (or RLMS).
- Interact with the laboratory (or RLMS) to enable the proper separation of the service into different apps as requested by the user.
- Translate the data provided by the particular laboratories to the specifications defined earlier in this deliverable (WS + JSON).

Depending on the particular laboratory or RLMS, some of these features might not be provided and therefore they will have a lower compliance level with the specifications defined above, while still accessible from the Go-Lab Portal.

4.2.2 Design

As previously stated, the Smart Gateway covers different requirements. Some of them can be grouped in wrapping authentication, authorisation and scheduling, while others are focused on translating protocols of the particular laboratories. For example, two laboratories developed on the same RLMS (e.g., an aquarium and an eggs incubator developed in WebLab-Deusto) will have the same authentication, authorisation and scheduling mechanisms, but will have different variables (e.g., balls, cameras in the aquarium as opposed to an egg tester and cameras in the egg incubator). For this reason, two complementary solutions are required.

For the first requirements (dealing with authentication, authorisation and scheduling), the Smart Gateway relies on G4Labs, described in detail in Section 4.4. This solution supports the OpenSocial specifications so it acts as a bridge for different RLMSs and OpenSocial. Lab owners and especially RLMS providers are expected to develop small plug-ins for G4Labs and they will automatically benefit from the integration in the Portal. Essentially, G4Labs makes a simpler API that deals with these functionalities and provides a framework for reusing code among the different RLMSs. This way, the connector to the Go-Lab Portal used by all the laboratories of WebLab-Deusto are the same than the one used by all the laboratories of the iLab Shared Architecture. Furthermore, once one of these RLMSs is supported, automatically every other laboratory developed on top of it will work and it can be integrated in the portal.

For the communication translation requirements, ad hoc solutions are required. Every laboratory is different, with different variables, even inside a particular RLMS. The communications used by those laboratories are only understood by them and therefore it is required to make different bridges that, taking a particular laboratory and a running reservation identifier, can map it to the smart device specifications.

4.3 Scenarios

This Section describes real-world scenarios of integration in the Go-Lab portal using the Smart Gateway.

4.3.1 Introduction

Based on the initial requirements for the Smart Gateway defined above, it is possible to identify two broad conceptual scenarios for lab owners to plug their legacy systems in the Go-Lab infrastructure. The first one is when the smart gateway acts as a protocol translator, repackaging legacy lab communication and exposing their functionalities according to the smart device services specifications.

However, repackaging legacy lab communication might not always be feasible, especially if the lab is owned by third party institutions or uses a specific and/or proprietary protocol. Beyond that, it would require a redesign of all existing legacy client applications that communicate with the legacy labs. In these cases, the smart gateway will provide all necessary mechanisms to authenticate users coming from Go-Lab against the legacy lab systems and to package legacy clients into Web apps. As the majority of legacy lab systems are grouped around RLMSs and for other reasons already outlined in Sections 4.1 and 4.2, the Smart Gateway uses a plug-in architecture to provide support for these different communities.

The scenarios described in this Section are the aquarium (using WebLab-Deusto) and the radioactivity laboratories (using iLab Shared Architecture). Both labs are available in Go-Lab via the Smart Gateway. In both cases, the authentication, authorisation and scheduling processes are supported, while the translation of the protocols has not been implemented. As previously stated, that would require new independent components to be implemented (these initial releases will be available at M21 in D4.3).

4.3.2 Aquarium laboratory

This Section describes the migration of the existing aquarium laboratory of WebLab-Deusto through the smart gateway

Pedagogical Context

The Aquarium laboratory was developed using the WebLab-Deusto RLMS. For this remote lab, the user interface shows a set of balls which contain different liquids. The user can drop them into the aquarium and observe how they float. In order to be integrated in the Go-Lab Portal, some changes in this lab were required. The user interface was divided into different apps, there is communication between the different components, and it complies better with the specifications defined in this deliverable. This Section explains the initial state of this lab and provides an overview about the migration of this lab to the Go-Lab infrastructure.

URL: https://graasp.epfl.ch/#url=SmartGatewayAquarium2013

Initial state

The aquarium laboratory is developed using the WebLab-Deusto RLMS. The original user interface is shown in Figure 9. It can be seen that there are two real time cameras on the left side, and two pictures showing how much under water are the dropped balls.

As any WebLab-Deusto laboratory, the user can use it for a particular slot, so there is a fixed time for that slot (3 minutes 45 seconds) in the picture. Since this laboratory is collaborative (i.e., multiple students can access it and interact with the laboratory at the same time), every 3 seconds the Web browser asks the server if anything changed. If it is the case (e.g., a ball was moved up), it updates the user interface showing that the yellow ball is up.

Migration

The migration from the legacy lab to a Go-Lab compliant version consists in bridging the authentication and authorisation from the inquiry learning spaces (ILS) to the WebLab-Deusto server. So as to do this, G4Labs was used, with a custom plug-in for WebLab-Deusto. This way, WebLab-Deusto may know which user in the ILS platform used it (as long as the user was logged in, otherwise it will only know that it was a Guest user) and it may know from which space the user is coming. Furthermore, it consists in the separation of the user interface in different independent apps. Each app can run on its own, and will contact the space to coordinate with other apps if they aim to reserve the same laboratory (the aquarium). If it is the case, they coordinate to make sure that only one of them requests the reservation.

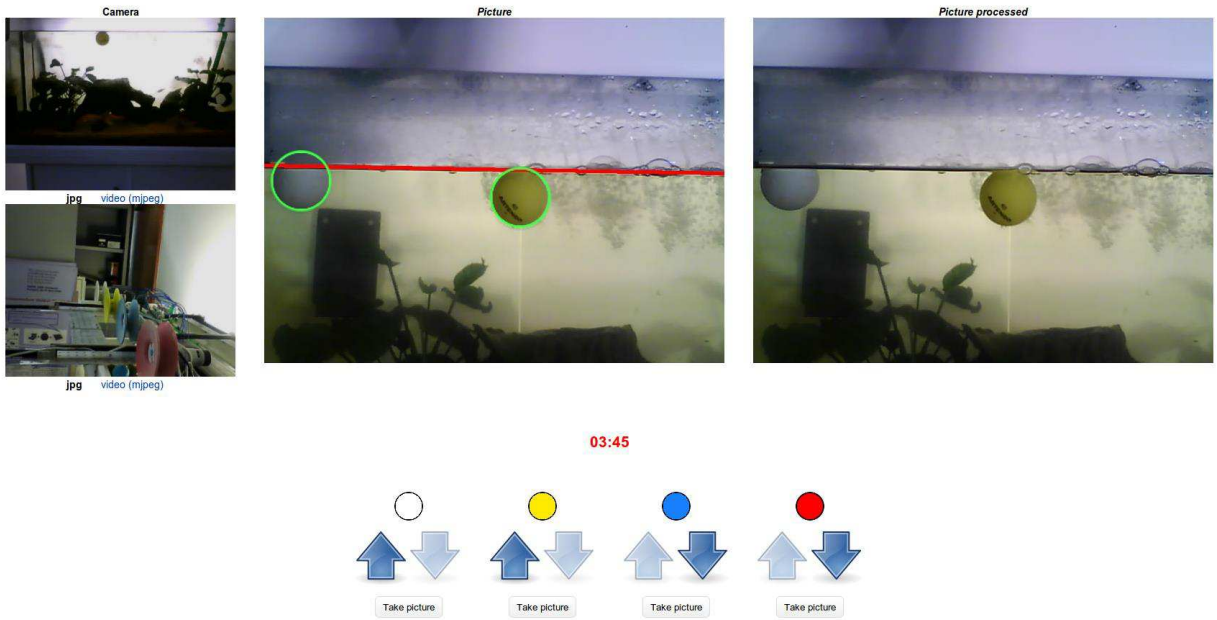

Figure 9. Original user interface of the Aquarium laboratory in WebLab-Deusto.

From an architectural point of view, the WebLab-Deusto architecture supports two types of laboratories: managed and unmanaged. The aquarium is a managed laboratory. The WebLab-Deusto architecture for managed laboratories is described in Figure 10, as well as the plug-in architecture for G4Labs. In the managed architecture, the clients connect to a main core server, which forwards (and stores) the requests to the assigned "experiment server" (using the WebLab-Deusto nomenclature). In the Figure, three different laboratories go to three different "experiment servers" located in two different locations inside the same institution.

Regarding the integration, as shown in this Figure, the client connects to the Go-Lab Portal. The portal provides the aquarium apps. Then, the client loads them, loading resources from G4Labs. Each of these apps knows what part of the user interface must be rendered (e.g., the blue ball or the second camera). The user clicks on the "Reserve" button in any of the apps, G4Labs contacts through a proper RLMS plug-in that understands WebLab-Deusto for a reservation identifier. Then, that reservation identifier is returned to the app that made the request. This app propagates this reservation identifier to the rest of the apps in the ILS. This way, all these apps contact the G4Labs server asking to display a particular part of the user interface with that reservation identifier. The G4Labs server translates that to the way WebLab-Deusto separates its user interface, and from that moment all the communications are between the client and the WebLab-Deusto server. From this point, the communications are the same as in the WebLab-Deusto managed architecture, adding no latency.

Complete migration

The complete migration, which will be completed as part of D4.3, will require two major changes: protocol translation and optimization of the shared communications.

The first change, i.e., the protocol translation, will require a new component to be deployed in the WebLab-Deusto side that will wrap the communications with the WebLab-Deusto server to provide a unified protocol and be compliant with the specifications explained in this deliverable. That will require reimplementing the user interface to use those specifications instead of the

WebLab-Deusto protocol with the data of the aquarium. It will also add some latency since the communications go through other layers.

The second change, i.e., the optimization of the shared communications, will require client side coordination. In the original aquarium laboratory, every 3 seconds a request to the server is performed to retrieve the current status (e.g., somebody else has moved a ball). In the current migration, if there are 4 Web apps showing 4 balls, they will all perform these requests every 3 seconds. This way, there will be 4 times more requests for information, while it would not really be required. The Web apps at client side could select that one of the Web apps will maintain the shared information, so it will be the only one performing requests to the server and it will communicate the results to the rest in the client.

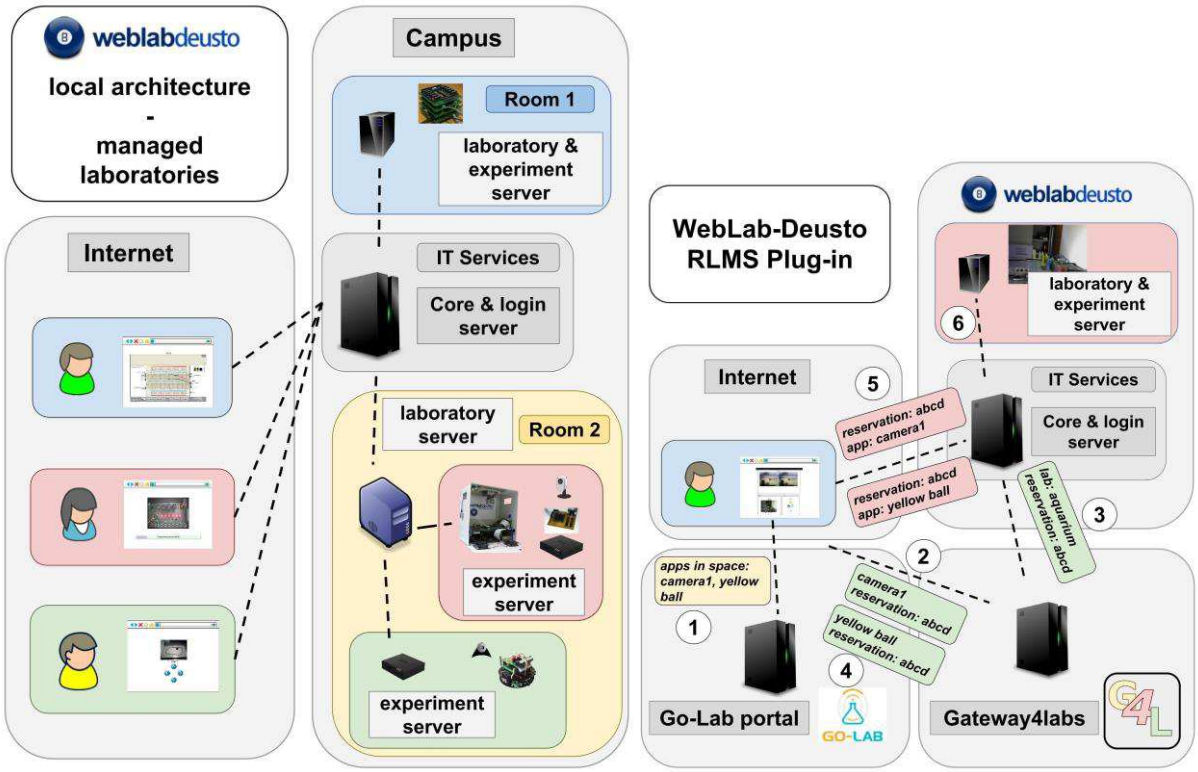

Figure 10. WebLab-Deusto architecture for managed laboratories (left), and the plug-in architecture (right).

4.3.3 Radioactivity laboratory

This Section describes the migration of the existing radioactivity lab of iLab through the smart gateway.

Pedagogical Context

The radioactivity laboratory was developed and deployed at the University of Queensland in Australia and it has been used by secondary schools in the USA through the Northwestern University. This lab allows students to explore how radioactive radiation changes as a function of distance. The intensity of radiation emitted by a Strontium -90 source is measured by a Geiger counter at different distances set by the students.

URL: https://graasp.epfl.ch/#item=widget_4282

Initial state

This lab has been developed in compliance with the iLab Shared Architecture (ISA), a Remote Laboratory Management System from the MIT in use by several institutions worldwide. ISA defines two types of laboratories, batch and interactive. In Figure 11 the batch case is illustrated. In these model labs, clients communicate with lab servers via a Web service API based on SOAP and this process is always mediated by a Service Broker. The Radioactivity laboratory falls into these type of laboratories.

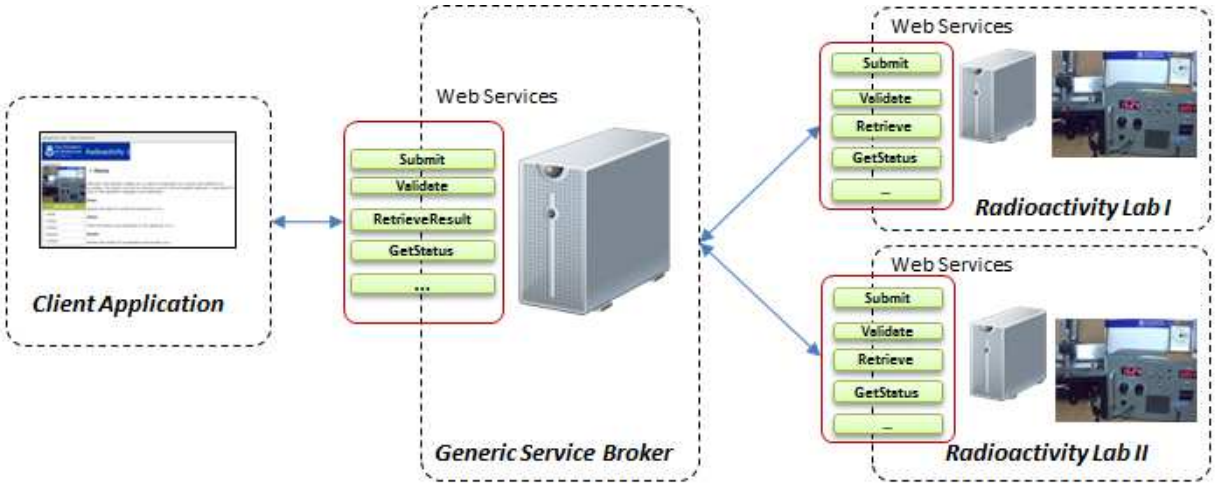

Figure 11. iLab Shared Architecture: Batch laboratories.

Migration

In the context of Go-Lab, the Carinthia University of Applied Sciences (CUAS) deploys a MIT iLab Service Broker, and the Radioactivity laboratory is registered on it. Therefore, using the G4Labs plug-in for ISA, it is possible to have this lab available in the Go-Lab infrastructure.

Figure 12 shows the process flow for the communication with the radioactivity lab after the integration in Go-Lab.

1. The user connects to the Go-Lab Portal and downloads the apps to access the lab and includes it into an ILS.
2. The user will access the lab via the downloaded app. When the user clicks on “reserve” the app connects to the G4Labs requesting the initiation of a lab session.
3. The G4Labs acts as a trusted authority that initiates a lab session in the service broker on behalf of the user.
4. Assuming the credentials of the G4Labs are successfully verified by the service Broker, a lab session is created and a URL to launch the client application is returned to the *G4Labs* and later on to the user.
5. The app contacts the Service Broker (at CUAS in this case) with the URL provided in the previous step.

6. That Service Broker will connect to the Radioactivity Lab Server hosted at University of Queensland, Australia.

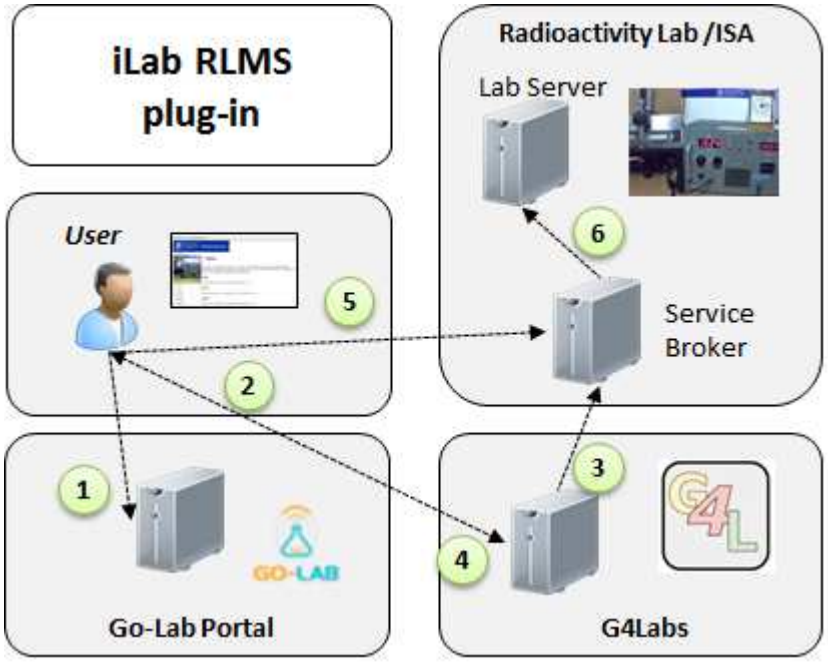

Figure 12. Migration of the Radioactivity Lab to Go-Lab with the Smart Gateway.

In this approach no modification in the original Lab Servers is necessary. From their perspective all experiment execution requests from Go-Lab users will come from the Service Broker at CUAS. The laboratory will be available in ILS in a transparent way through the connection with *G4Labs*.

Complete Migration

A complete migration in this context is requiring the implementation of a protocol translation that converts client requests (according to smart device specifications) made to the Smart Gateway to the SOAP based Web Service API supported by an ISA Service Broker. At this moment this feature is not implemented.

Similar as in the case of the aquarium lab, this process might add some additional latency to the requests made to the lab. However, as batch experiments run asynchronously this should not interfere on the user experience. In the case of the batch labs of ISA RLMS the API for client - Service Broker communication is well specified and unique for any batch lab compliant with ISA. However client apps would also need to be redesigned to follow the new service specifications.

4.4 G4Labs

G4Labs[1] is a software system that fosters interoperability between learning environments (and other systems), such as the Go-Lab Portal, with different remote laboratory management systems (RLMS, such as WebLab-Deusto or MIT iLabs). It has been extended to support the

[1] https://github.com/gateway4labs/

Go-Lab Portal, as well as to support the apps-based approach where the lab owner may split the user interface of the laboratory in different parts.

4.4.1 Architecture

The architecture is described in Figure 13. It connects two different types of components: the left side represents the consumer systems (Go-Lab Portal in this case), and the right side represents the RLMS connector. Every RLMS connector installed on the right side will be supported by all the consumer systems (including the Go-Lab Portal).

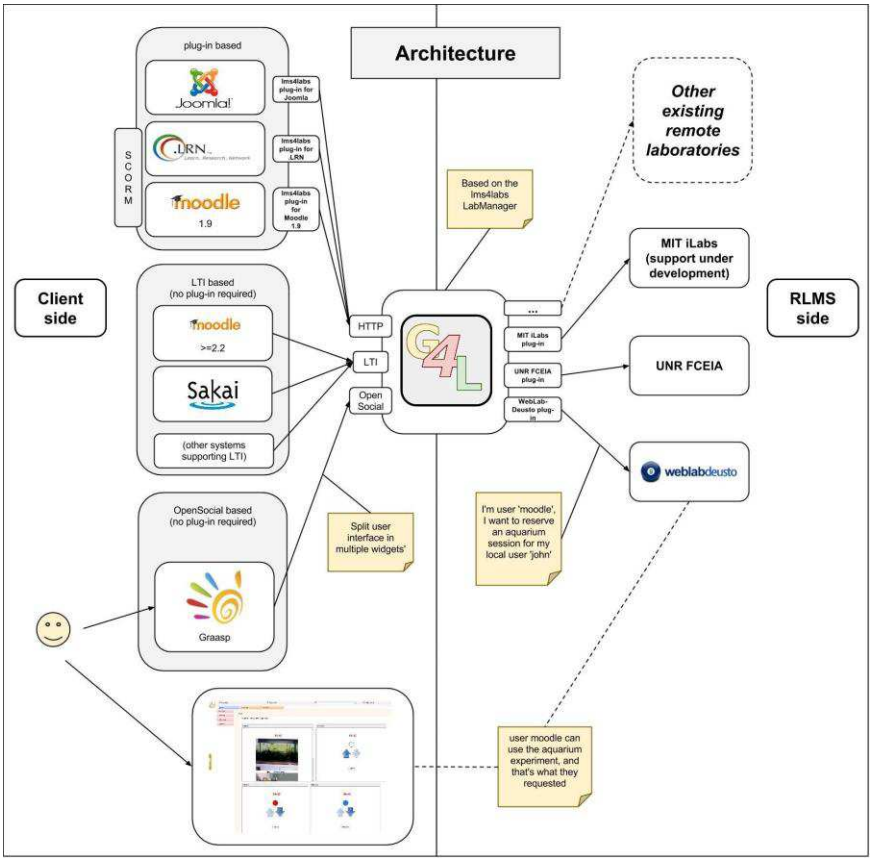

Figure 13. G4Labs architecture.

The RLMS connectors work with a plug-in system. Each one of this plug-in implements a common interface, mapping the concepts to a particular RLMS. For example, the plug-in for WebLab-Deusto will provide a set of method such as “reserve laboratory ‘Aquarium’ for Go-Lab Portal user ‘122’, and provide the URL for the ‘Blue ball’ app, as well as the reservation identifier”, or “with this reservation identifier, provide the URL for the ‘Yellow Ball’ app in that existing ongoing reservation”.

The following Section describes the changes to adapt this architecture to the Go-Lab Portal. The Scenarios Section describes the integration of particular RLMSs.

4.4.2 RLMS Plug-in Architecture and Design

So as to support new laboratories, each lab owner must create a bridge between G4Labs and the particular lab. In order to do this, G4Labs provides an API for developing new plug-ins. This process is not especially complex, since most of the time this only implies writing the required code for performing a reservation and sharing a secret with the final user. However, this might become more difficult if there are limitations in the laboratory code, such as not being able to securely pass a URL for identifying a booking.

The plug-in system, at the time of this writing, requires that a developer write a plug-in in the programming language used by the rest of G4Labs (Python). However, ongoing work is focused on providing a REST API so that lab owners can use any other programming language.

The lab owner must provide through the plug-in:

- A method to retrieve the capabilities of the laboratory. If it supports being separated in apps, for instance. Otherwise it will consider the whole Web application a single big app.
- A method to retrieve what apps the laboratory provides. For instance, the aquarium laboratory will return a set of components (e.g., camera1, camera2, red ball, etc.).
- A method to request a reservation. The G4Labs main component will provide information such as the full name of the user ("Guest" if it is not authenticated), user agent (what web browser is using), the origin IP address, etc. The plug-in must contact the laboratory server and return a reservation identifier and a URL to be loaded. If the laboratory supports being separated in different apps, the plug-in must also provide a method to load each app with the provided reservation identifier, to guarantee that all the apps are using the same reservation, instead of creating different reservations.
- A method to request a Web form for configuring the laboratory details (different laboratories require different information, different credentials forms or different identifiers).

From the side of the RLMS it should be able to respond to the requests sent by its plug-in. So the RLMS should be able to:

- provide and external URL and other necessary credentials for consumption by the G4Labs
- Support the dynamic creation of new users (or manage external users properly or be stateless)

At the time of this writing, there are three prototypal plug-ins implemented: one for WebLab-Deusto, other for the iLab Shared Architecture, and other for the physics lab of the Facultad de Ciencias Exactas Ingeniería y Agrimensura of the National University of Rosario (Argentina).

4.5 Remarks

As previously mentioned, the Cloud Services include all services necessary to ensure compatibility of legacy systems with the Go-Lab platform and therefore allow different laboratory providers to plug their systems in the Go-Lab infrastructure.

The Smart Gateway is the core component of the cloud services and, as mentioned in Section 4.2.1, it acts as a bridge between the Go-Lab infrastructure and the legacy lab systems. In the ideal scenario, it should provide a protocol translation between legacy lab systems and the services specified in Section 3 of this document. However, it should also ensure the compatibility of those systems for which the implementation of such protocol translation is not feasible.

Furthermore due to the fact that most legacy lab systems are grouped around RLMSs (Remote Lab Management Systems) it was decided that the Smart Gateway should include support for those RLMS. The Smart Gateway includes this support via its component G4Labs and this is achieved by means of a plug-in architecture. However, work towards the implementation of a REST Web Service API is underway. It will allow lab owners to implement support for their systems using their preferred development platform.

5 Conclusion

This deliverable presents the initial specifications for the lab owners to implement remote labs. It present the smart device paradigm which revisits the traditional client-server approach for implementing remote laboratories by adding more agility at the server side and by relying on Web solutions at the client side. This paradigm is used to define the specifications for remote labs where a clear definition of services proposed by the smart device enables a complete decoupling between the client and the server. The decoupling combined with the proposed specifications greatly simplifies the integration of new remote labs within the Go-Lab infrastructure.

Legacy remote labs that cannot be directly implemented as smart devices can be interfaced through a Smart Gateway. The Smart Gateway acts as a bridge between the Go-Lab infrastructure and the legacy remote labs. It provides protocols translation when feasible and support for Remote Lab Management Systems. The Smart Gateway includes this support via its G4Labs component and this is achieved by means of a plug-in architecture.

This deliverable introduces the smart device specifications and provides guidelines and specifications about the services handled by the smart device. Besides, it introduces the Smart gateway concept which describes the way of interfacing existing labs as smart devices. All these concepts and initial specifications will be refined during the course of the project and will be updated in future deliverables (G4.3 at M27 and D4.5 at M30).

Currently, prototypal implementations of the above specifications are under development and have been described in this document. Links to these open solutions will be provided on the Go-Lab Project Web site once the first stable versions will be released.

Accessing smart devices through well-defined services gives a total freedom to develop their client applications.

6 Appendix A: Remote laboratory Management Systems (RLMS)

This Appendix describes in more detail the RLMS from Section 2.2 and the solution adopted in each case.

6.1 Labshare

LabShare is led by the University of Technology, Sydney, and is a joint initiative of the Australian Technology Network: Curtin University of Technology, Queensland University of Technology, RMIT University, University of South Australia, and the University of Technology, Sydney (http://www.labshare.edu.au/). This project aims at creating a national network of shared remotely accessible laboratories. To do this, they have developed a framework for setting up a heterogeneous remote laboratory of physical apparatuses containing many labs of many types called SAHARA [11-13].

SAHARA, Figure 14, is composed by a set of elements:

- The Web Interface is the user interface to SAHARA Labs. Its roles include authenticating a user, providing the interface for users to request access to a resource and providing the interface to control a rig. It provides programmatic interfaces to implement aspects such as how users are authenticated and presentation. It also includes a suite of components for implementing the web interface associated with different rig types. A rig is a physical item of laboratory apparatus.

- The Scheduling Server is the middleware component that manages the scheduling processes of the remote laboratory rigs, including tracking the state of rigs and assigning them to users based on either queued access or time-based reservations. It is also responsible for managing the running sessions according to the allocated times as well as logging all events and activities. The Scheduling Server is agnostic with regard to the specific design of a rig and relies on a rig client application to turn scheduling requests into rig specific behaviour. In SAHARA, a rig is a physical item of laboratory apparatus.

- The Rig Client provides a software abstraction of each rig and converts abstract requests from the scheduling server into rig specific actions. If a user is being assigned to the rig, its Rig Client provides the behaviour to actually allow the user to access the rig. The Rig Client provides a programmatic interface to allow this behaviour to be defined as a set of Java interfaces. The Rig Client manages the rig and sends status updates to the Scheduling Server. If the Rig Client is not operating, from the perspective of the Scheduling Server, the rig is not operational and no user will be assigned to it. The Rig Client also provides a control channel to directly interface with the rig or as an auxiliary to an external control program.

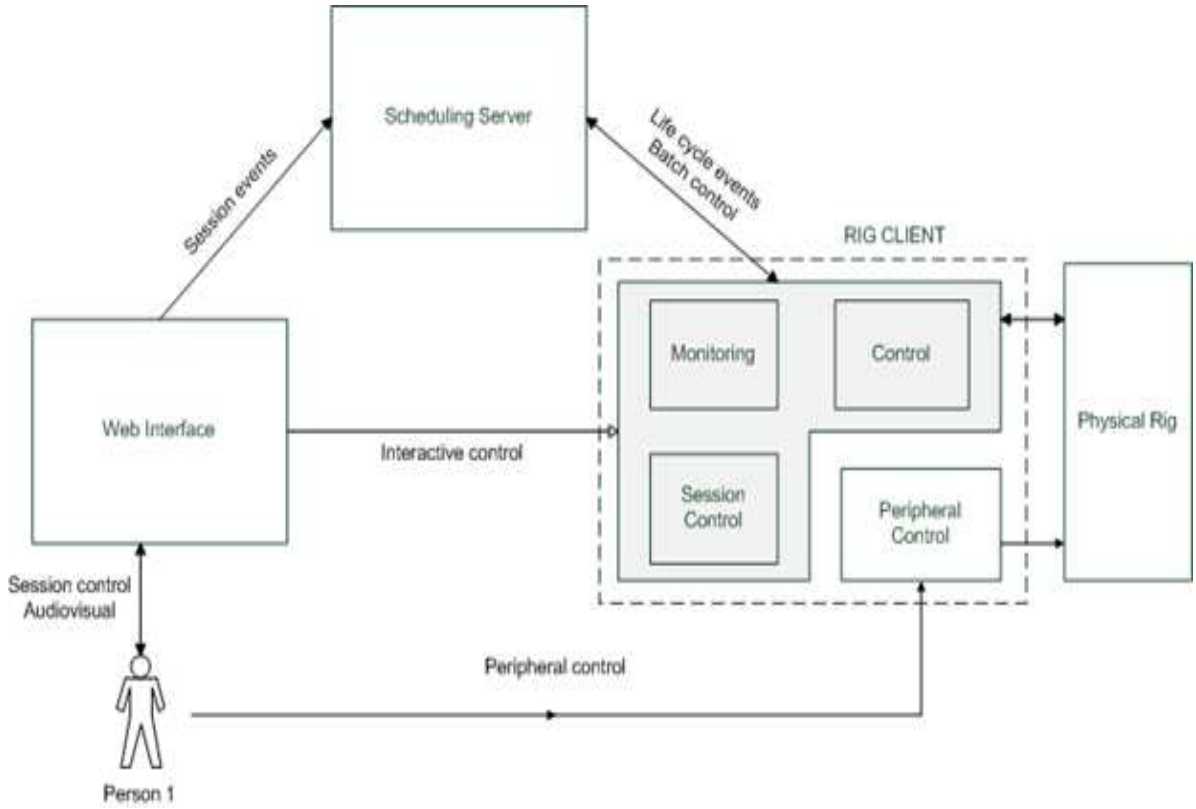

Figure 14. Sahara architecture.

6.2 iLab Project

The iLab project started at MIT in 1998, with the goal of developing and implementing a distributed software toolkit and middleware service infrastructure to support online laboratories, and promote their sharing among schools and universities on a worldwide scale [14-16]. Therefore, MIT implemented the iLab Shared Architecture (ISA), focusing on fast platform-independent laboratory development, scalable access for students, and efficient management for laboratories providers, while preserving the autonomy of the faculty actually teaching the students.

From the perspective of the ISA, at the present time, two types of online experiments can be supported:

- Batch experiments. These are laboratories where experiments are completely specified prior to submission and execution without human intervention. I.e.: the use of MIT's Microelectronics iLab, where the student specifies the entire course of the experiment before the experiment begins.

  Therefore, the ISA batched architecture in some way resembles the typical three-tier web business architecture, Figure 15.

- Interactive experiments. These experiments are fundamentally different from their batch counterparts. Primarily, interactive experiments require taking control of the laboratory

hardware while the user sets the parameters and observes the results. This is exactly in contrast to the batched model where experiments are queued and run when the laboratory hardware is available. Therefore, an interactive experiment must commit the laboratory hardware to a single user for the duration of the session - typically 20 minutes to an hour – and may require scheduling.

Another main difference between interactive and batch laboratories involves the role of the Service Broker. Interactive experiments require real-time control and potentially much greater bandwidth between the laboratory client and the laboratory server. Due to it, the batch notion of a Service Broker that uses web services to route all communications between the laboratory client and laboratory server will not work effectively in an interactive experiment.

An interactive iLab topology is detailed in the following, Figure 16. New elements are added to the laboratory client, Service Broker, and laboratory server, such as:

a. The Experiment Storage Service.
b. The Scheduling Services.
c. Laboratory manager.

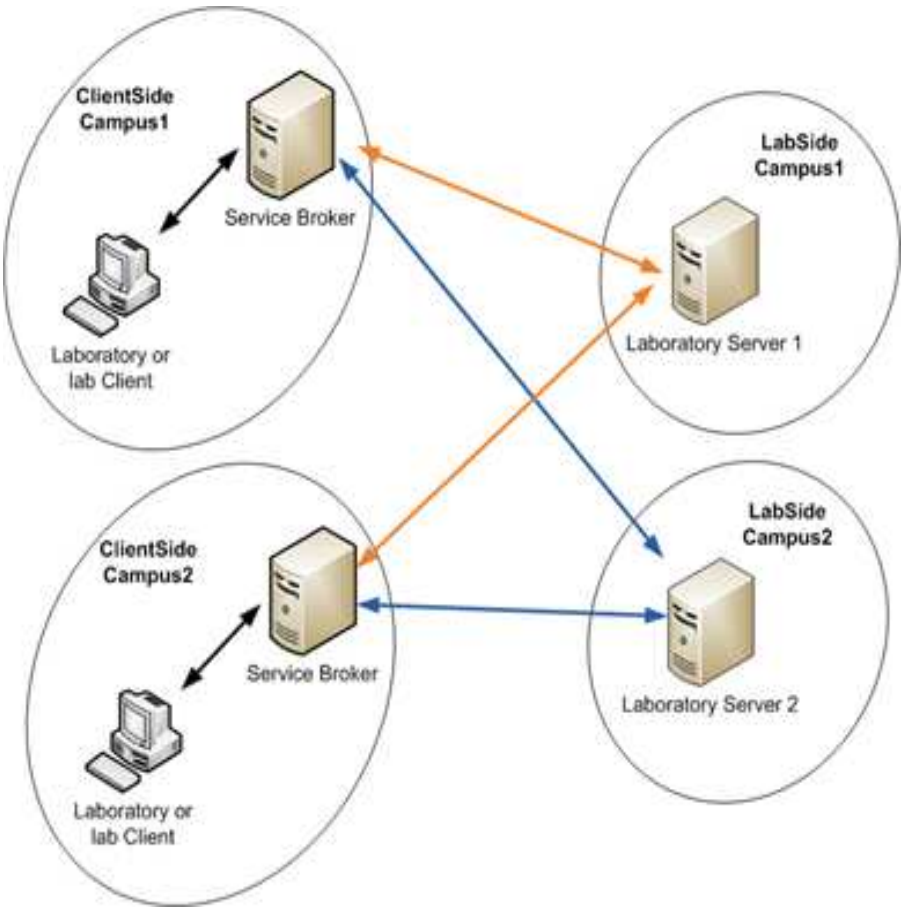

Figure 15. ILab architecture for batched experiments.

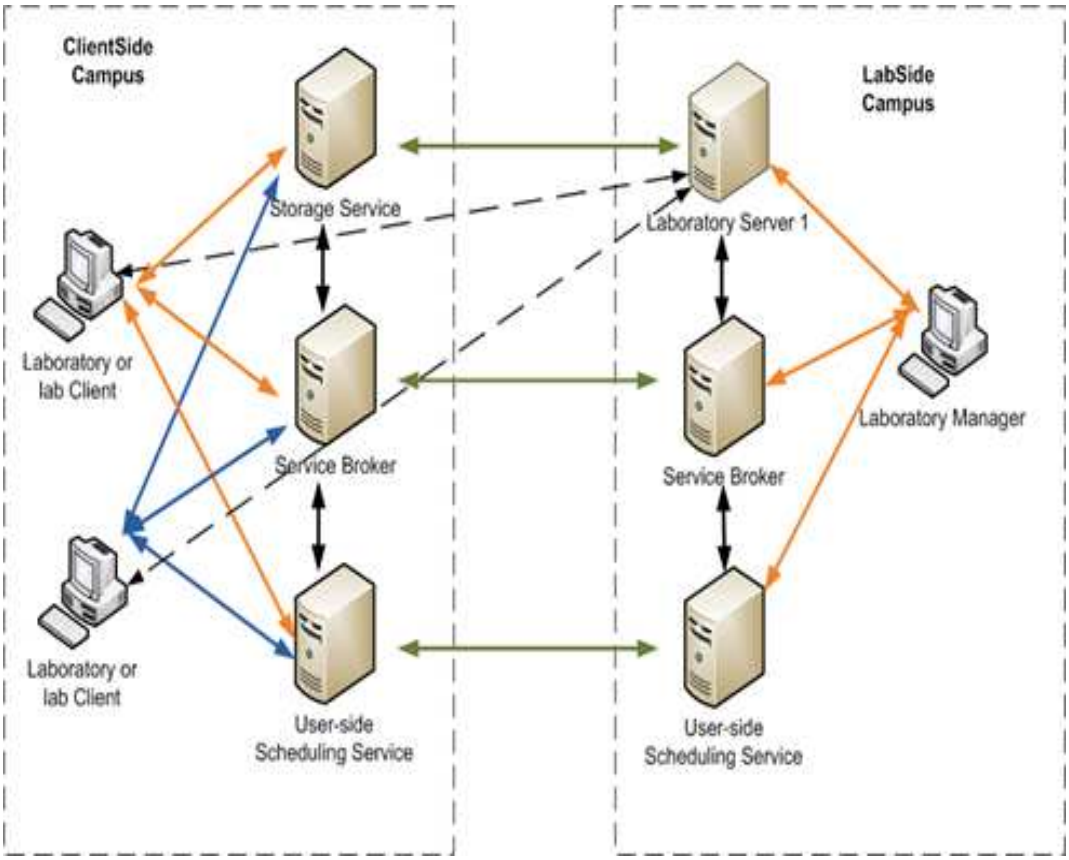

Figure 16. ILab architecture for interactive experiments.

6.3 WebLab-Deusto

WebLab-Deusto provides an open source, scalable, distributed software architecture that makes easy to integrate new experiments, Figure 17. There are two types of experiments [17-19]:

- Managed: the developer of the experiment software interaction must use any of the common web technologies for the client side (JavaScript, Flash, Java applets), and any technology for the server side (WebLab-Deusto comes with libraries for C/C++, .NET, LabVIEW, Java and Python).

- Unmanaged: the experiment developer places an application in a Virtual Machine, and WebLab-Deusto controls the access to the Virtual Machine.

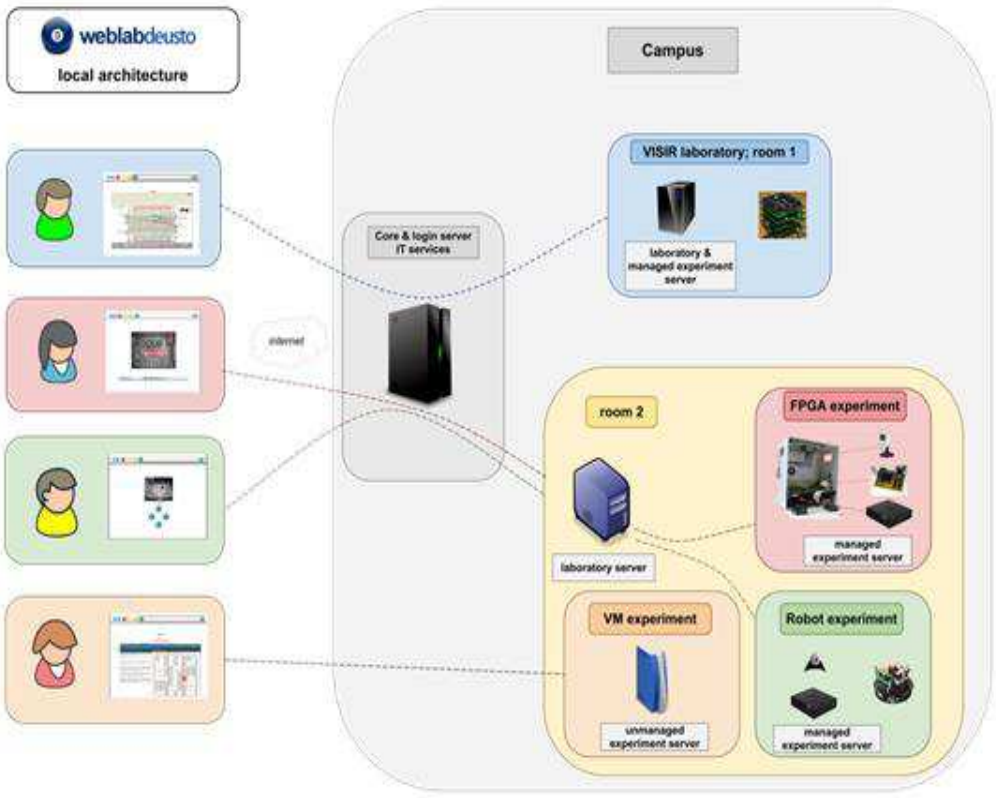

Figure 17. WebLab-Deusto architecture.

6.4 LiLa Project

LiLa is the acronym for the “Library of Labs”, an initiative of eight universities and three enterprises for the mutual exchange of and access to virtual laboratories and remote experiments (real laboratories which are remotely controlled via the internet [20-22]). To accomplish this task, the e-learning standard called SCORM has been modified to communicate with remote online labs, Figure 18.

LiLa also builds a portal through which the access to virtual labs and remote experiments is granted. It includes services like:

- a scheduling system,
- connection to library resources,
- tutoring system,
- 3D-environment for online collaboration

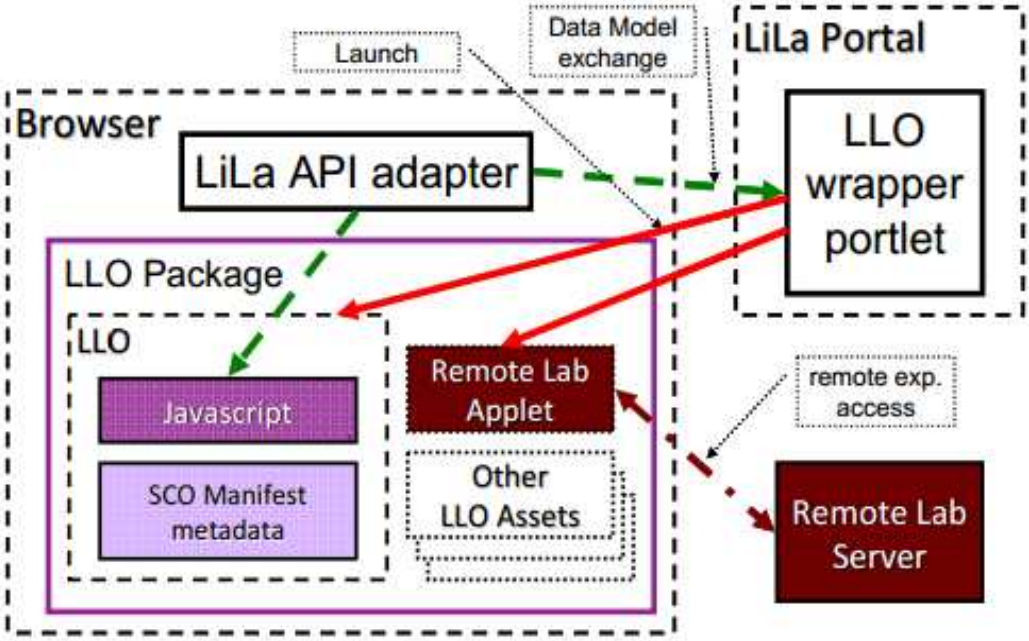

Figure 18. SCORM modified to communicate with remote online labs.

Although the LiLa project bundles labs in SCORM packages, proper interoperability among the different labs is not always possible since SCORM has not been designed for interactive labs and there is lack of support of the latest versions of SCORM (see deliverable D5.2).

7 Appendix B: Arduino

The microcontroller Arduino (http://www.arduino.cc) has a set of features that make it a strong candidate to implement the smart device specifications.

- Hardware. The versatility of the Arduino is its hardware. Technically the Arduino UNO is a microcontroller board based on the ATmega328 (datasheet). It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with an AC-to-DC adapter or battery to get started. This is the heart and the potential of the system.

  The fact of having analog and digital pins can perform functions as diverse as motor control (analog systems) or on-off emulation through them using voltage pulses LOW-HIGH (digital systems). This greatly expands the range of applications of an integrated system with arduino UNO.

- Plug & Play. The above features and physiology of the Arduino board itself, Figure 19, allows on one hand an easy connection of the board to the PC via a simple USB cable, but on the other hand also allows an easy installation of other hardware laboratory to board with wires simply stabbing the respective laboratory to analog and digital pins of the board.

Figure 19. Arduino UNO Board.

Another advantage inherent in this hardware is that it has its own software and install its own drivers that facilitate the installation of the entire system on any PC. This makes the system compatible with Windows, Linux or Mac OS X.

- Easy Programming. All Products Arduino software comes with its own installation software. The software (in its current version 1.0.5) contains everything you need to install the board, the corresponding driver, and begin programming the Arduino.

Contains examples predesigned and commented that the new users can modify and adapt to their needs easily.

Figure 20 shows a picture of the software. It clearly shows that the programming language used is C. Easy to learn and enough stability, the use of C helps further to generalize increasingly using Arduino for laboratories that wish to integrate electronics and software in a single tool.

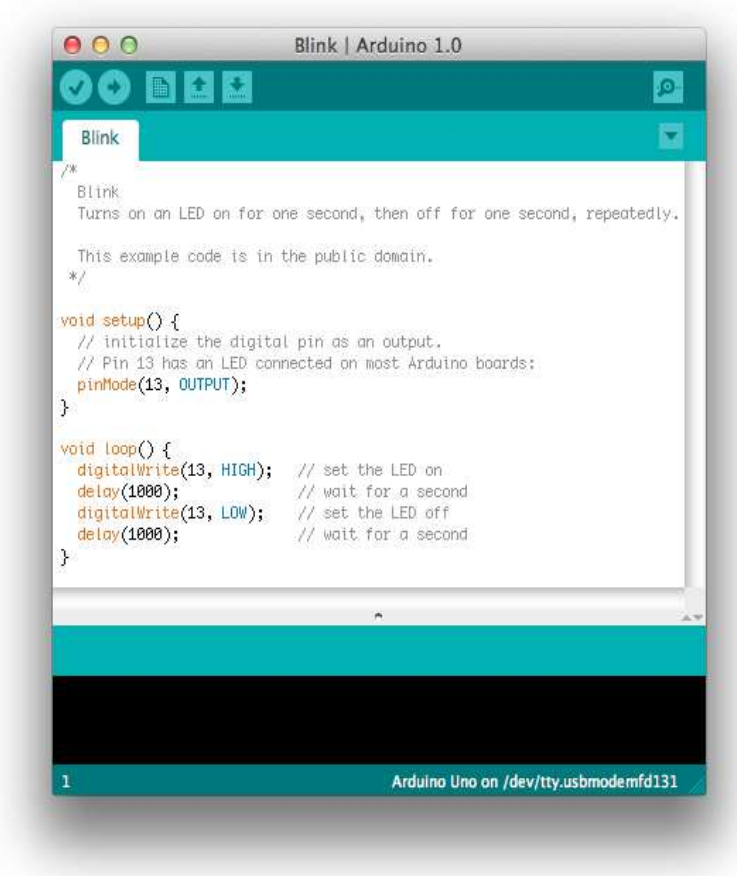

Figure 20. Arduino software in action.

8 Appendix C: G4Labs

8.1 Introduction

As described in Section 4, the Smart Gateway acts as a legacy labs converter to the specifications defined in this deliverable. This involves different steps, being one the conversion of the data exchange (i.e., the communication protocol), and other the reservation process (i.e., bypassing the authentication, authorisation and scheduling mechanisms).

This Appendix is focused on this latter step (reservation process), which is implemented on top of the G4Labs system. It is organized as follows: first, the early prototype is described, then the G4Labs system and the adaptation done to be integrated in the Go-Lab is detailed, and finally two scenarios relying on G4Labs are shown.

8.2 Initial prototype

Before using the G4Labs system, an early prototype of the integration of an external remote laboratory management system has been implemented, named wlwidget. The target of wlwidget was to act as a bridge between WebLab-Deusto (and all the labs contained in WebLab-Deusto) and the Go-Lab Portal. This bridge has been used for demonstration purposes with the Aquarium laboratory, implemented in WebLab-Deusto.

So as to explain this bridge, it's important to explain briefly what WebLab-Deusto is, and what the Aquarium laboratory is. WebLab-Deusto is a remote laboratory management system. It is an open source system that a lab owner can download, install, and develop laboratories on top. WebLab-Deusto provides authentication, authorisation, federation, user tracking, and communications management through different mechanisms. Multiple laboratories have been implemented using WebLab-Deusto in different universities. One of these laboratories is the aquarium laboratory, which allows students to drop different balls with 4 colours containing different liquids into an aquarium. Figure 21 shows how this laboratory was originally developed using WebLab-Deusto.

This way, the Aquarium lab is an example of legacy remote laboratory to be interfaced through the Smart Gateway.

The wlwidget system acts as a gateway for this laboratory (and for any laboratory developed in WebLab-Deusto). Using it, it is possible to be consumed through the Go-Lab Portal, and wlwidget would provide the WebLab-Deusto system who was the user being connected (e.g., "user 122" or "anonymous user"), and would wrap the underlying scheduling mechanism (a queue). It provided an automated system for providing URLs to define each of the different apps (being an app the red ball, another app the blue ball, another app the camera 1, etc.).

This way, it was possible to use the Go-Lab Portal to exploit it in an inquiry learning scenario using it, as shown on Figure 22. WebLab-Deusto required small modifications to support that a client requests a subset of the user interface (such as the blue ball), and the aquarium was adapted. This way, it was possible to use the Aquarium lab from an inquiry learning space and

from the WebLab-Deusto system at the same time, since in the Go-Lab portal would conceptually act as a WebLab-Deusto node when using the wlwidget bridge.

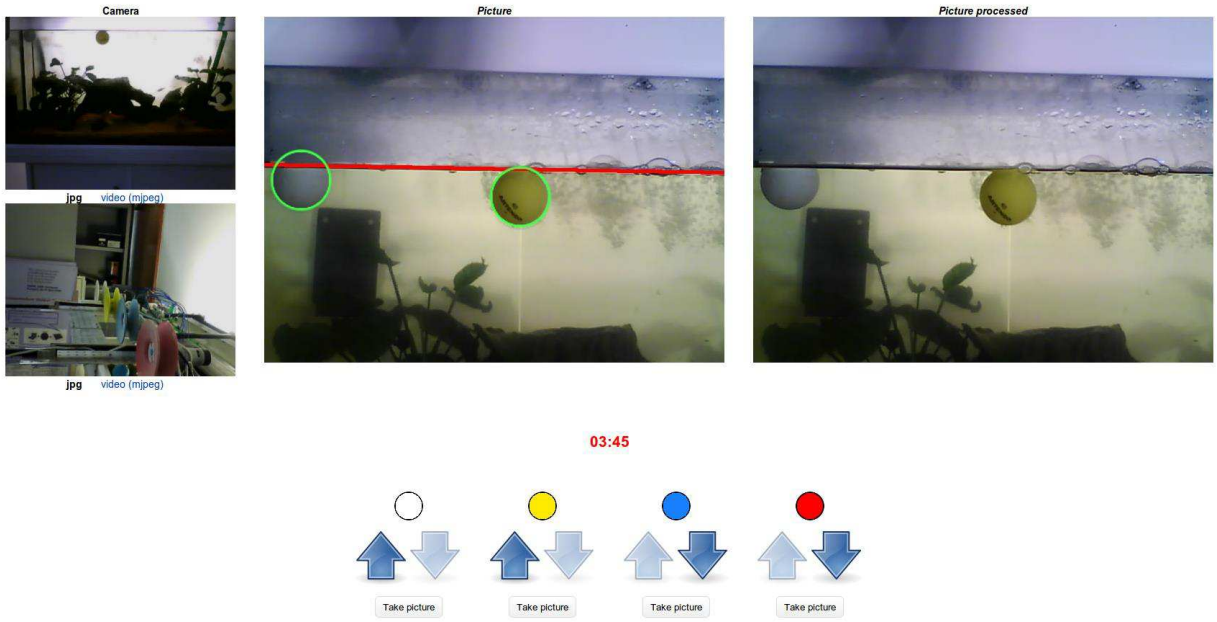

Figure 21. Original Aquarium user interface as developed in WebLab-Deusto.

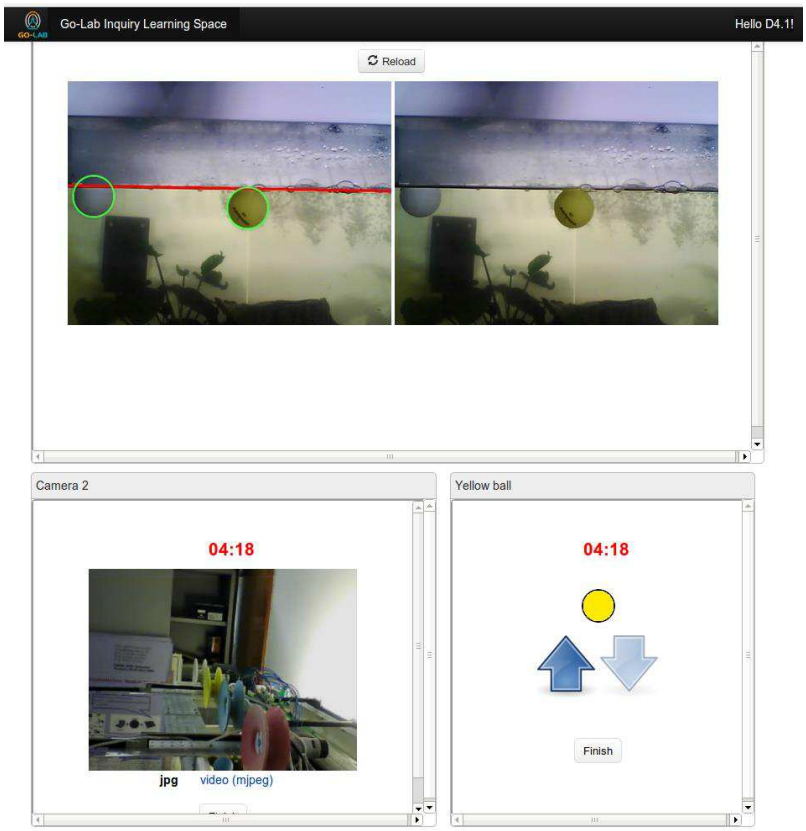

Figure 22. An aquarium lab separated in different apps in an inquiry learning space (ILS) using wlwidget.

The wlwidget provided the following features:

- Authentication: internally, this system connected to the WebLab-Deusto system with a particular username and password (always the same), but reported the system that it was being used by a foreign user, providing the user identifier at the ILS.

- Scheduling: in WebLab-Deusto, there is a concept of reservation. The user interface in wlwidget had to be loaded in different apps, but the desired behavior was that all of them were using the same reservation. So as to do this, wlwidget applied a master/slave concept, where all the apps decide that one of them is the “master”, and display a button, and the rest assume that they are “slaves”. Whenever the user clicks on the button, a reservation is requested, and that reservation identifier is shared with the rest of the apps in the same web browser. This is represented in Figure 23.

- User tracking: the wlwidget server is periodically requesting WebLab-Deusto the status of the reservations submitted. Whenever one is finished (i.e., the user closed the window or the assigned time finished), it downloaded what the user had done and serialized to Activity Streams. However, it never submitted those to any other system.

As an early prototype, it did not implement certain features:

- The integration in ILS was simple, and did not use extensively the OpenSocial APIs.

- It only supported WebLab-Deusto laboratories, and no other remote laboratory management systems.

- It did not support any type of internationalization (so the default language was always used)

- It did not support dragging and dropping apps once the lab was running. Since the master Web app shared the reservation identifier only once, it was not stored by the rest and shared to newly reload apps.

- It did not support that in a single space there could be more than one type of laboratory. All the apps had to be part of the same particular WebLab-Deusto laboratory.

So as to deal with these features, and especially with the first one (supporting more legacy laboratories), it was decided to adapting the bridge relying on G4Labs, which has been explained in Section 4.

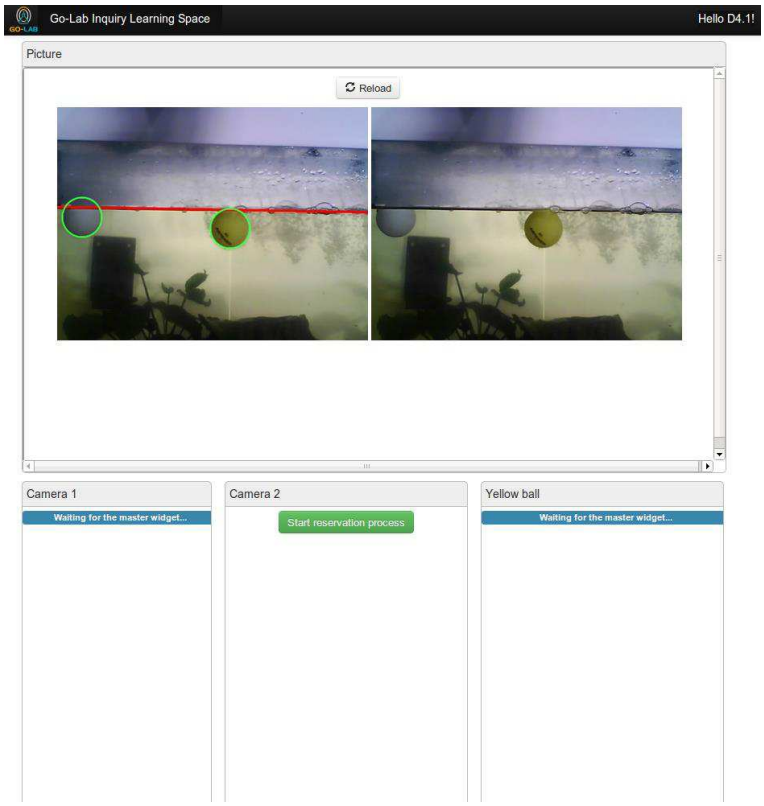

Figure 23. wlwidget in ILS before starting the interaction. Those apps requiring interaction were not loaded from the very beginning: they relied on a master (in the Figure, "Camera 2") and the user had to click on it to start the process. The other apps (e.g., "Yellow ball" in the Figure) were slave apps waiting for the master to share the single reservation identifier.

8.3 Integration of G4Labs in the Go-Lab Portal

The underlying technology of the Go-Lab Portal, Apache Shindig, provides a secure mechanism based on tokens for enabling third party services to access information about the current user, app and space. Both wlwidget and the integration of G4Labs used this feature to identify the user (as long as the user is signed in). In most scenarios, the user identification itself is not used since students are not logged in; however, it still provides statistical information regarding the context from which the laboratory is consumed.

The developed model is open to everyone, meaning that schools can use and harness Go-Lab technologies straight-away, needing no registration.

Though the main approach to Go-Lab is supporting anonymous, not registered users, the goal of the Smart Gateway is to use already existing remote labs or legacy labs. These legacy labs will be provided by the external lab owner.

The most basic interaction has been developed and will be used as the basis for the development of future remote labs requirements. What follows is the description of these functionalities. They are therefore optional and most teachers and students will not see them,

but in certain circumstances (e.g., an external lab owner who requests to know which schools are using their resources), they would be useful.

8.4 Comparison with the initial prototype

G4Labs supports the features of the initial prototype (wlwidget), explained above. Additionally, since it is not a prototype, but stable software targeting large-scale scenarios, more work has been done in polishing the result. Therefore, when compared with the early prototype:

- It supports multiple RLMS and not only WebLab-Deusto. It provides an API that can be used by the rest of the legacy RLMSs to be supported.
- It supports multiple different labs per space, and it manages that the different apps of one lab do not communicate with the different apps of other lab.
- It supports both an anonymised model, where students use the system from the Go-Lab portal with the RLMS unaware of the particular schools and users, but it also supports an authorized model where teachers register their schools in the tools and certain labs are only available to certain schools.
- It drops the master/slave decision model used in wlwidget. At the beginning, all apps are master apps, and when the user clicks on the reserve button of one of them, all the rest of the same laboratory act as slaves. This is not only more stable and simple, but it also supports that moving apps in the space is supported.

G4Labs does not provide any data conversion for the communications between the final client (the student) and the final laboratory. It does not proxy it, so it basically enables that the connection is direct. Therefore, there is no impact on performance during the usage of the laboratory.

References

[1] J. García-Zubia, P. Orduña, D. López-de-Ipiña and G.R. Alves, “Addressing Software Impact in the Design of Remote Laboratories”, IEEE Transactions on Industrial Electronics, Vol. 56, Nº. 12, pp. 4757– 4767, December 2009.

[2] M. Morozov, A. Tanakov, A. Gerasimov, D. Bystrov and E. Cvirco, “Virtual Chemistry Laboratory for School Education”, IEEE International Advanced Conference on Learning Technologies, ICALT, Joensuu (Finland), 30 Aug.-1 Sept. 2004.

[3] Z. Raud and V. Vodovozov, “Virtual lab to study power electronic converters”, International Symposium on Power Electronics Electrical Drives Automation and Motion, SPEEDAM, Pisa (Italy), 14-16 June 2010.

[4] L. Gomes, and S. Bogosyan, “Current Trends in Remote Laboratories”, IEEE Transactions on Industrial Electronics, Vol. 56, Nº. 12, pp. 4744– 4756, December 2009.

[5] M.A. Vivar and A.R. Magna, “Design, implementation and use of a remote network lab as an aid to support teaching computer network” Third International Conference on Digital Information Management, ICDIM, London (UK), 13-16 November 2008.

[6] J.M. Andújar, A. Mejías and M.A. Márquez, “Augmented Reality for the Improvement of Remote Laboratories: An Augmented Remote Laboratory” IEEE Transaction on Education, Vol. 54, Nº. 3, pp. 492–500, August 2011.

[7] A.G. Vicente, I. Bravo Muñoz, J.L. Galilea and P. A. Revenga del Toro, “Remote Automation Laboratory Using a Cluster of Virtual Machines”, IEEE Transactions on Industrial Electronics, Vol. 57, Nº. 10, pp. 3276–3283, October 2010.

[8] J. Hu, M. Haffner, S. Yoder, M. Scott, G. Reehal and M. Ismail, “Industry-Oriented Laboratory Development for Mixed-Signal IC Test Education”, IEEE Transactions on Industrial Electronics, Vol. 53, Nº. 4, pp. 662– 671, November 2010,

[9] G. Andria, A. Baccigalupi and M. Borsic, Remote Didactic Laboratory “G. Savastano,” The Italian Experience for E-Learning at the Technical Universities in the Field of Electrical and Electronic Measurements: Overview on Didactic Experiments”, IEEE Transactions on Instrumentation and Measurement, Vol. 56, Nº. 4, pp. 1135–1147, August 2007.

[10] A. Rojko, D. Hercog and K. Jezernik, “Power Engineering and Motion Control Web Laboratory: Design, Implementation, and Evaluation of Mechatronics Course”, IEEE Transactions on Industrial Electronics, Vol. 57, Nº. 10, pp. 3343–3354, October 2010.

[11] E. Lindsay and B. Stumpers. “Remote laboratories: enhancing accredited engineering degree programs,” Proceedings of the 2011 AAEE Conference, Fremantle, Western (Australia), 2011.

[12] D. Lowe, C. Berry, S. Murray and E. Lindsay. “Adapting a Remote Laboratory Architecture to Support Collaboration and Supervision”, REV 2009: 6th International Conference on Remote Engineering and Virtual Instrumentation, M. Auer, N. Gupta and J. Pallis, Eds. Bridgeport, USA: International Association of Online Engineering, pp. 103-108, June 2009.

[13] D. Lowe, S. Murray, E. Lindsay and D. Liu. “Evolving Remote Laboratory Architectures to Leverage Emerging Internet Technologies” IEEE Transactions on Learning Technologies, Vol. 2, Nº. 4, pp. 289-294, October 2009.

[14] V.J. 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.

[15] J. Hardison and D.G. Zutin. “Online Workbenches for the Deployment of Electronics Experiments,” conditionally accepted for publication in Internet Accessible Remote Laboratories, IGI Global, 2011.

[16] E. San Cristobal Ruiz, M.A. Castro Gil, V.J. Harward, P. Baley, K. DeLong, and J. Hardison. "Integration View of Web Labs and Learning Management", IEEE Educon 2010, April 14-16, Madrid (Spain), 2010.

[17] J. García Zubia, P. Orduña, D. López de Ipiña and G. Alves. “Addressing Software Impact in the Design of Remote Labs”, IEEE Transactions on Industrial Electronics, ISSN: 0278-0046; DOI: 10.1109/TIE.2009.2026368. Vol. 56, Nº 12, pp. 4757-4767, December 2009.

[18] J. García Zubia, P. Orduña, I. Angulo, J. Irurzun and U. Hernández. “Towards a Distributed Architecture for Remote Laboratories”, International Journal of Online Engineering (iJOE), ISSN: 1861-2121. Special Issue, REV 2008, Vol. 4, 2008.

[19] P. Orduña, J. Irurzun, L. Rodriguez-Gil, J. Garcia-Zubia, F. Gazzola and D. López-de-Ipiña. “Adding New Features to New and Existing Remote Experiments through their Integration in WebLab-Deusto”, International Journal of Online Engineering (iJOE), ISSN: 1861-2121, Vol. 7, 2011.

[20] L. Bellido, V. Villagrá and V. Mateos. “Federated authentication and authorization for reusable learning objects”, IEEE EDUCON Education Engineering 2010 – The Future of Global Learning Engineering Education, April 14-16, Madrid, Spain, 2010.

[21] Y. Tetour, T. Richter and D. Boehringer. “Integration of Virtual and Remote Experiments into Undergraduate Engineering Courses”, Joint International IGIP-SEFI Annual Conference 2010, Trnava, Slovakia, 19th - 22nd September, 2010.

[22] V. Mateos, A. Gallardo, T. Richter, L. Bellido, P. Debicki and V. Villagra. “LiLa Booking System: Architecture and Conceptual Model of a Rig Booking System for On-Line Laboratories”, International Journal of Online Engineering (iJOE), vol. 7, issue 4, pp. 26-35, 2012.

[23] J. Zheng, D. Simplot-Ryl, C. Bisdikian, H.T. Mouftah. “The internet of things”, Communications Magazine, IEEE, Volume: 49, Issue: 11, Page(s): 30- 31, 2011.

[24] G. Kortuem, A.K. Bandara, N. Smith, M. Richards and M. Petre. “Educating the Internet of Things Generation”, Computer, IEEE, Volume: 46, Issue: 2, Page(s): 53-61, 2013.

[25] S. Tozlu, M. Senel, M. Wei and A. Keshavarzian. “Wi-Fi enabled sensors for internet of things: A practical approach” Communications Magazine, IEEE, Volume: 50, Issue: 6, Page(s): 134-143, 2012

[26] C. Doukas. “Building Internet of Things with the Arduino (Volume 1)”. Editorial: createspace, Pages: 348, ISBN: 978-1470023430, 2012.

[27] Arduino Web page http://arduino.cc/en/Reference/HomePage accessed at 30 September 2010.

[27] Wiring Web page http://wiring.org.co/ accessed at 30 September 2010.

[28] Raspberry pi web http://www.raspberrypi.org/ accessed at 30 September 2010.

[29] Raspberry pi (Linux) http://elinux.org/RPi_Hub accessed at 30 September 2010.

[30] D. Cohen, M. Lindvall, and P. Costa. “An introduction to agile methods”. Advances in Computers. Elsevier Science, 2004.

[31] Ch. Salzmann, D. Gillet, F. Esquembre, H. Vargas, J. Sánchez and D. Sebastián. “Web 2.0 open remote and virtual laboratories in engineering education”. Book Chapter: Collaborative Learning 2.0: Open Educational Resources, by IGI Global pp.369-390, 2012.

[32] T. Richter, P. Grube and D. Zutin. “A standardized metadata set for annotation of virtual and remote laboratories”, IEEE International Symposium on Multimedia (ISM). Irvine (California), December 2012.

[33] The WebSocket Protocol. http://tools.ietf.org/html/rfc6455, accessed at 09 September 2013.

[34] JSON. http://www.json.org/, RFC 4627, accessed at 09 September 2013.

[35] C.W. Thompson. "Smart devices and soft controllers". IEEE Internet Computing, Vol. 9, Nº 1, pp. 82-85, 2005.