# 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 D4.7**

# Releases of the Lab Owner and Cloud Services (Final) – M33

| | |
|---|---|
| Editors | Wissam Halimi (EPFL) |
| | Sten Govaerts (EPFL) |
| Date | 30th July, 2015 |
| 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 |
|---|---|
| Wissam Halimi, Sten Govaerts, Christophe Salzmann, Denis Gillet | EPFL |
| Pablo Orduña | UDEUSTO |
| Danilo Garbi Zutin | CUAS |
| Irene Lequerica | UNED |
| Eleftheria Tsourlidaki (Internal Reviewer) | EA |
| Lars Bollen (Internal Reviewer) | UT |

## Legal Notices

## *Executive Summary*

This deliverable is the corresponding companion of D4.5 Specifications of the Lab Owner and Cloud Services. The latter deliverable has been submitted at M30, and is the final document detailing the solution devised by Go-Lab for integrating new and legacy remote laboratories (labs) in its infrastructure. Go-Lab provides two approaches to integrate remote laboratories: the Smart Device and the Smart Gateway approaches. While the first corresponds to the development and deployment of new labs, the second is destined for existing labs. Yet, both implement the Smart Device Paradigm as presented in D4.5, and briefly presented in the Introduction of this document. Starting from the specifications presented in D4.5, lab owners are expected to be able to create/adapt and smoothly deploy their labs in the Go-Lab platform.

In this deliverable we provide templates for lab owners to use in order to interface their new physical labs with embedded devices (the Smart Devices). The templates provide the architecture skeleton for the laboratory server application following the Smart Device paradigm. The templates implement a set of specifications to ensure that they are scalable, readable, and maintainable; insuring the ease and augmented use of these templates. Using the templates, lab providers are not concerned about the architecture of their lab servers. This task is alleviated by the templates. The targeted platforms are Node.js on BeagleBone Black (BBB), and LabVIEW on myRIO . We also provide example labs implemented with the produced packages, as well as with other platforms and technologies.

We also provide support for existing online laboratory systems with the Smart Gateway. The Smart Gateway supports both real laboratories (e.g. Remlabnet) and simulation (e.g. Concord, QuVis). The integration of these labs in the Go-Lab infrastructure with the Smart Gateway means that all current and future laboratories provided by these systems are and will be automatically available for Go-Lab consumption. Each of these systems offers more than 20 laboratories at the moment, which is a good gain for the project. Additionally, new developed features of the Smart Gateway since D4.3 are presented, such as the support of internationalization of external laboratories, and new mechanisms for the integration of laboratories (HTTP plug-in examples).

## Table of Contents

# 1 Introduction

Remote experimentation consists of operating and observing a physical laboratory at distance over the Internet, by communicating with sensors and actuators. A common abstraction of the architecture enabling such manipulations consists of two major components: a client application and a lab server. Figure 1 depicts this architecture:

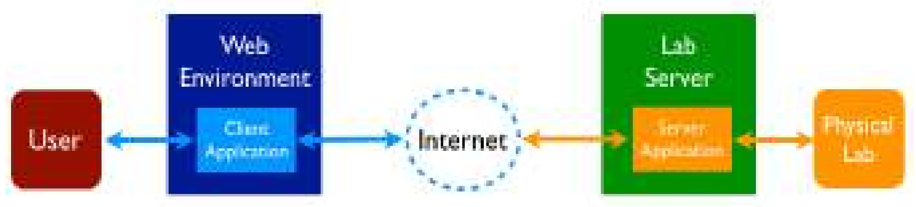


**Figure 1: Common Architecture for Remote Laboratories**

Interaction with the *Physical Lab* begins with the *User* using the *Client Application* which transmits *User's* commands over the Internet to the *Server Application*, which in turn controls the *Physical Lab*. We assume that the *Client Application* and the *Server Application* communicate through the Internet, and that the *User Application* is running in a *Web Environment*.

## 1.1 Smart Device Overview

Following the architecture presented in Figure 1, the *Client Application* and the *Lab Server* are tightly coupled, in the sense that proprietary technology governs the communication between them (Salzmann, Govaerts, Halimi, & Gillet, 2015). The Smart Device Paradigm as presented in D4.5 aims at decoupling the Client and Lab server applications. This is done by redesigning the *Lab Server* to expose it through standardized interfaces. In other words, the Smart Device Paradigm specifies the *Lab Server* as a set of well-described services through an API that complies to the Specifications of the Lab Owner Services presented in D4.5. This enables the separation between the two previously tightly connected user client, and lab server sides.

The figure below describes the above:

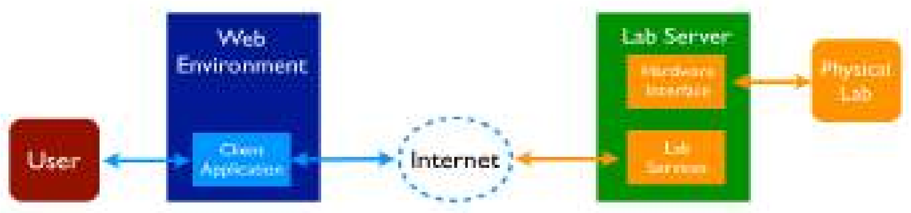


**Figure 2: Remote Labs Architecture with the Smart Device Paradigm**

## 1.2 Smart Gateway Overview

The Smart Gateway paradigm aims at making existing labs compatible with the Smart Device specifications. Due to the large technological variety of existing remote labs, the Smart Gateway offers 3 different compatibility levels with the Go-Lab infrastructure as presented in D4.5 through its different adaptation mechanisms. The Smart Gateway paradigm conceptualizes and embeds the cloud services.

The figure below shows how the Smart Gateway fits in the remote lab architecture:

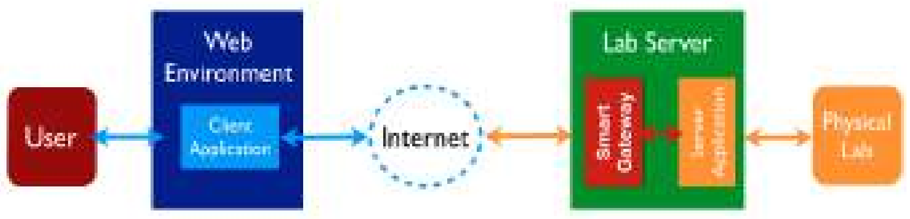


**Figure 3: Smart Gateway in the Remote Lab Architecture**

## 1.3 Repositories of Templates and Examples

The work invested in this deliverable results in a number of packages (templates) for deploying new labs following the Smart Device Paradigm, corresponding implemented remote labs, in addition to packages for legacy labs such as ViSH, PhET, iLabs, WebLab-Deusto, and QuVis to ensure integration in the Go-Lab infrastructure.

The template packages for the releases of the lab owner services can be found here:

- `https://github.com/go-lab/smart-device/tree/master/templates/`.

And the corresponding links for the implemented examples for the different platforms are:

- BeagleBone Black: `https://github.com/go-lab/smart-device/tree/master/BeagleBoneBlack`
- Desktop Computer: `https://github.com/go-lab/smart-device/tree/master/Desktop`
- myRIO: `https://github.com/go-lab/smart-device/tree/master/myRIO`
- Raspberry Pi: `https://github.com/go-lab/smart-device/tree/master/RaspberryPi`

The packages for the releases of the Smart Gateway can be found here:

- `https://github.com/gateway4labs`

# 2 Releases of Lab Owner Services

The releases of the the lab owner services are template software packages following the Smart Device Paradigm. The templates are meant to be used for developing and deploying new remote laboratories. Examples of implemented labs corresponding to the templates are also provided. This chapter first compares the Releases of the Lab Owner Services in this deliverable with those in D4.3, then details the Releases of Lab Owner Services through 3 main sections: *Overview* which presents the software requirements and specifications of the packages, *Templates* which details the packages developed for targeted platforms, and *Examples* which presents the implemented labs using the templates.

## 2.1 Lab Owner Services Final Release Compared to the Initial Release

In D4.3, the Releases of the Lab Owner Services were example-based templates implemented with different combinations of selected software and hardware technologies. Lab owners are then provided with instructions in order to reuse the examples, by modifying its code to make it respond to their hardware and software requirements.

In this deliverable, we follow a different approach for the Releases of the Lab Owner Services: we provide generalized templates with adequate instructions and recommendations to start from the provided packages and build a complete application. The architecture of each of the templates is thoroughly detailed. We also provide a concise explanation for the hardware and software technology choices made for the respective templates.

Additionally, the examples presented in D4.3 underwent testing and bug fixing, all updated code is available on the github repository (links to each lab are provided in Section 2.4). As a proof of concept, new example labs have been implemented using the templates of this deliverable: 2 wind turbine labs presented in Section 2.4.3 and 2.4.4.

At last, new appendices are added to provide more technical information aiding lab owners in selecting the adequate board for their labs. Appendix A describes the boards, Appendix B provides information of a programming paradigm adopted in one of the templates, Appendix C provides lab owners with recommendations to enhance user experience.

## 2.2 Overview

The software packages presented in this deliverable follow the Smart Device Paradigm. They set up the skeleton of the lab server application for lab owners. The software templates implement a set of requirements to ensure their ease of use and the efficient build up upon them.

In this section, we will present the software requirements of the templates provided for lab owners (the party using the templates), and the corresponding specifications we (the party developing the templates) abide by in order to implement those requirements.

### 2.2.1 Templates Requirements

Three requirements are respected in order to provide lab owners with a promise of good usability:

1. Scalability: The template should be easy to extend to handle more work. In other terms, if in the template data acquisition is implemented for one sensor, it should be easy to extend the same functionality to any number of extra sensors.
2. Readability: It should be easy to inspect the code and structure of the application, understand its functionality, and the connection amongst its components.
3. Maintainability: It should be easy to add new features to the code without affecting the original implemented features.

### 2.2.2 Templates Specifications

Two specifications for the templates are devised in order to implement the requirements presented in 2.2.1:

1. Data Specifications: The template assumes that incoming requests are in JSON format, and outgoing responses are also in JSON. The template consumes and produces JSON encoded data.This JSON data follows the 'metadata' specifications as detailed in D4.5.
2. Behavioral Specifications: Two main components comprise the software package: an **API Server** and a **Process Controller** as depicted in figure 4. The API[1] (Application Programming Interface) Server is the interface between the lab server and the outside world. It takes care of receiving requests, decoding them, and forwarding them to the Process Controller. It also receives data from the Process Controller, adequately encodes them into responses. ***The API Server sends and receives Interaction Data from the Process Controller***. The Process Controller senses the physical environment (through sensors) of the lab and makes changes to it (through actuators). It is the hardware interfacing module that receives data from the API Server and transforms them to commands to the Physical Lab. It also senses the Physical Lab and forwards the collected data to the API Server.***The Process Controller receives/sends experimental data to API server***.

[1] https://en.wikipedia.org/wiki/Application$_{programming_interface}$

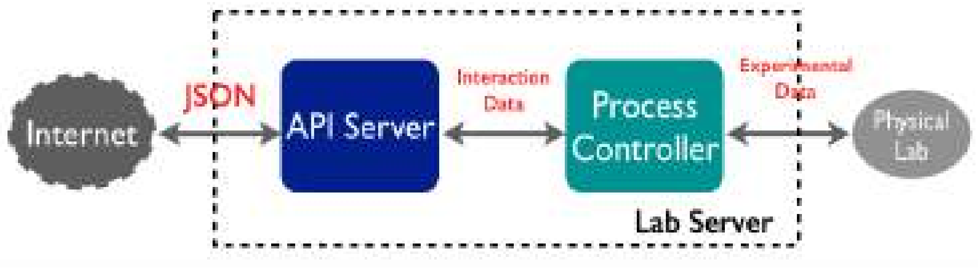


**Figure 4: Software Template Architecture**

## 2.3 Templates

The releases of the lab owner services provide templates for 2 platforms: a Node.js platform and a LabVIEW platform. In this section we discuss the technological choices made for the platforms, the structure of the respective templates, and how to use them.

### 2.3.1 Platforms Choices

#### *Node.js with BeagleBone Black*

Currently there is a shift in hardware computing towards Javascript and supported platforms such as the Beagle Family[2]. This is gaining popularity as mini-computers are getting smaller in size and more powerful in computing and communication capabilities. And so it is enabling the migration towards high-level languages such as Javascript for physical computing from the classic low-level languages such as C. Since it utilizes the Linux operating system, it combines the capability of a traditional embedded platform with the power of open-source Linux software (Molloy, 2014).

The BeagleBone Black[3] is a single-board open hardware and software mini-computer produced by Texas Instruments in association with a number of smaller companies. Its hardware and software specifications provide support for a large spectrum of applications:

- Processor: It's an ARM Cortex processor with DDR3 RAMs, on-board eMMC flash storage, 3D graphics accelerator, floating-point accelerator, and 2 PRU microcontrollers. This translates to having different types of on-board storage with fast access to memory, and support of optimized Digital Signal Processing (DSP) applications.
- Software Compatibility: The system is Debian, Android, Ubuntu, and more. Which means that developers have access to open source communities and support, in addition to support for event-based physical programming instead of C.
- Connectivity: The board disposed of a USB client for power and commu-

[2]http://beagleboard.org/
[3]http://beagleboard.org/black

nication, a USB host, Ethernet plug, HDMI plug, and 2x46 pin-headers. This provides a large set of GPIOs and peripherals to interface. Also, the headers can be differently configured to respond to applications needs.

The core strength of the Beagle family is its BoneScript. BoneScript is a Node.js library specifically optimised for the Beagle family, and features familiar Arduino function calls. It provides several functions useful for interacting with hardware using JavaScript, alleviating the task of hardware interfacing by changing the classic way of doing it with low-level languages.

Performing physical computing with JavaScript is different than performing it with C; which might impose a learning curve upon migration to the BoneScript way. C executes sequentially, and interrupts modify order of execution. JavaScript & Node.js execute asynchronously using callbacks launched on events, listened to by a continuous event loop [4]. This new physical computing paradigm imposes an another way of thinking an application and tackling it, but it is a move forward for those who wish to make full use of all the BeagleBone Black software and hardware powers.

***Example Applications with BBB***

1. Client-side RFID authentication (and general access control) for Pumping Station: the system is developed on a BBB with a Sparkfun's USB board for RFID readers for hardware, and PS1Auth's server-side RFID authentication on the backend. The client-side software is written in Go (Kridner, 2015).
2. Podtique: an antique podcast player. It is a modern-day MP3 player with the look and feel of an antique radio. The prototype is built in the enclosure of a 1936 Goldentone radio. The internals were removed, and replaced with a BeagleBone Black-based system, and the necessary mechanical components to reproduce the tuning dial movement. Tuning the radio selects the playlist (or podcast) to be heard. Between stations pink noise is mixed in to simulate static (Rick, 2015).
3. Internet Speedometer: using a BBB and its two Programable Realtime Units (PRUs), the Internet Speedometer tests download times, and outputs the speed results visually to a tricolor LED strip (Karve & Worman, 2014).
4. Dirty Dish Detector: the project utilizes the BBB and a Logitech webcam. An email and/or MMS are sent when an unclean sink is detected. Notifications will occur on every ‘status change’ in the sink. Thus when the sink goes from ‘clean’ to ‘dirty’ (Normal & Kridner, 2014)

***LabVIEW with myRIO***

LabVIEW (Laboratory Virtual Instrument Engineering Workbench) is a system-design platform and development environment for a visual programming lan-

[4] http://beagleboard.org/Support/BoneScript

guage from National Instruments. The graphical language is named "G" (not the same as G-code). LabVIEW is commonly used for data acquisition, instrument control, and industrial automation applications on a variety of platforms [5]. With this graphical programming syntax, it is simple to visualize, create, and code engineering systems (NI, 2015a). With LabVIEW, it is quick and easy to create GUIs to interact with applications and visualize data using included charts, graphs, thermometers, and 2D and 3D visualization tools (NI, 2015b). A LabVIEW program is based on a basic unit called the VI (Virtual Instrument) which is composed of 3 components: the front panel which is the user interface, the block diagram which fosters the G code, and the connector panel which provides a dynamic logo of the the created VI with wired inputs and outputs. The last is used to represent the VI in the block diagrams of other, calling VIs. The front panel is built using controls and indicators. Controls are inputs they allow a user to supply information to the VI. Indicators are outputs they indicate, or display the results.

The myRIO1900 board adopted in this document as the device for templates is a portable reconfigurable I/O (RIO) device, manufactured in a way to make it easy to LabVIEW beginners to develop and deploy a wide range of applications. The main hardware and software specifications of myRIO1900 are:

- Processor: The board is equipped with an ARM Cortex-A9 processor, a XilinxZ-7010 customizable FPGA, 667MHz, Dualcore, 512MB NVMemory, and 512DDR3; which translates to benefiting from a powerful processor and FPGA for DSP and real-time applications.
- Software Compatibility: LabVIEW is the main support programming language, and the processor is running a real-time OS. This provides support for event-based physical programming with the G programming language.
- Connectivity: The board has a USB client for power, a USB host, 10 AI (Analog Input) channels, 6 AO (Analog Output) channels, 40 DI/Os (Digital Input/Output), AudioJacks, Accelerometer, LEDs, a push button, and expansion port connectors (MXP). This wide range of IOs provides support for interfacing different peripherals.

LabVIEW adopts a data-flow programming model as opposed to the sequential order of execution of commands in most classical physical computing languages. With LabVIEW as well, programmers used to the C-way of physical computing will face a learning curve to restructure the way code execution is thought of[6]. In this case, parallelism defined by data capturing governs.

***Example Applications with myRIO***

1. myRIO FPGA Audio Pitch Changer: it changes the pitch of an audio signal passing through the myRIO audio input and output ports. For example, the signal connected to the "Audio In" port from an MP3 player is transformed to a higher or lower pitched audio, which is connected to "Audio Out" port

[5]https://en.wikipedia.org/wiki/LabVIEW#Graphical_programming
[6]https://en.wikipedia.org/wiki/LabVIEW#Graphical_programming

on headphones. The pitch-shifting algorithm was developed using the frequency domain pitch shifting method (Hughes, 2011) (Frobinson, 2014).

2. myRio Control and Telemetry system for Formula-Hybrid Racecar: Bulldogs Racing at Yale University designed and built a fully functioning race car to compete at the 2014 Formula Hybrid International Competition. One myRIO1900 was utilized to control all the vehicle's drive systems, and another to collect and transmit all data collected from the sensor suite via WiFi to both an iOS mobile application and a LabVIEW base station (Belter, Piper, & Durkin, 2014).
3. The myExplorer remote controlled vehicle: myExplorer is a remote controlled vehicle that uses the myRIO device. myExplorer communicates with the user's laptop or desktop via WiFi, so their are no spatial limitations to where the car and user are. myExplorer executes control commands for its motion, while streaming video back to the computer, so that the user knows where it is headed to (Petru, 2014).

### 2.3.2 Templates Structure

As previously mentioned, the templates set the skeleton for the software application of the lab server. While some components of the application can be commonly used as a start for its development, others are specific to the application. More specifically, the templates are composed of 3 components:

1. A Process Controller Module: which is specific to the connected devices, and cannot be included in the general template
2. An API Server Module: which is general to all labs implemented with the Smart Device Paradigm. It is the module that serves the "metadata files" of the labs. Given that the paradigm standardizes the "metadata" amongst all connected labs in Go-Lab, this module is part of the template.
3. Communication between the Hardware Process Controller and the API Server Modules: which is the vital part of the template and the most important part to have working when starting with the application. Even though the Process Controller Module cannot be generalized among the labs, it is created with basic features such as Turn ON/OFF, and hooks are provided with the API Server Module.

#### ***The BBB Template***

Since the BeagleBone Black template is a Node.js application, it naturally follows the conventional Node.js application structure. It is as follows:

1. node_modules directory: contains dependency modules for the Node.js project. Commonly used modules are already included: *binaryjs* for binary streaming with WebSockets, *jsonfile* to read and write JSON files, *moment* to capture the current date and time, *q* for creating and composing asynchronous promises in JavaScript, and *ws* the RFC-6455 WebSocket implementation for Node.js.

2. metadata directory: contains template "metadata" files for the lab and sensors/actuators. It has the required metadata files: metadata.json, getClients.json, getSensorData.json, getSensorMetadata.json, getActuatorMetadata.json, sendActuatorData.json. The purpose of each of these files is further explained in later sections.
3. bbb-process-controller.js file: contains the controller code in which functions are defined to sense and control the connected lab to the board. In this file, BoneScript is used in order to interface with the hardware. A matter detailed in a coming section.
4. websocket-server.js file: contains the code for the server written in JavaScript, accepts and responds to incoming requests using the WebSocket protocol. It ensures communication between the API (metadata) and *bbb-process-controller.js*

The template can be found here: `https://github.com/go-lab/smart-device/tree/master/templates/bbb`

***The myRIO Template***

This template has a very packed structure composed of many VIs, libraries, and folders. It follows the LabVIEW Object Oriented[7] (LVOOP) paradigm, and implements the Queued Message Handler (QMH) approach for managing events. QMH is biefly detailed in Appendix B. Not all files in this structure are of interest to the lab owner reusing the G code. In this section, we will only detail the essentials of this template:

1. Test_full_comm4.vi: This VI is the equivalent known Main.vi in a LabVIEW project. It is composed of 3 main parts. The first initialises the services and the hardware used in a sequence structure. The second starts the application and services for logging, user client, and lab server. The third and last part handles quitting the application by quitting and killing all running 'workers' and stopping all called subVIs.
2. WebSocket directory: This directory gathers all VIs that take care of implementing the WebSocket standard in LabVIEW. It contains the code for acknowledging requests, getting request headers, encoding, and streaming requests.
3. JSON directory: All needed VIs for encoding and decoding JSON-encoded requests and replies according the Smart Device Specifications are found in this directory.
4. services directory: This directory has 2 sub-directories: PID and Video. The PID sub-directory includes the code that takes care of configuring, initializing, and controlling the connected hardware of the lab. The Video sub-directory has the code for handling video streaming through a USB-connected webcam to the board.

[7] http://www.ni.com/white-paper/3574/en/

5. HTML directory: It holds the VIs taking care of serving the 'metadata' as per the adopted specifications in Go-Lab, in addition to servicing user client as OpenSocial widgets or HTML-JavaScript web pages.

The template can be found here: `https://github.com/go-lab/smart-device/tree/master/templates/myRIO`

### 2.3.3 How to Use

This section provides a detailed tutorial for the lab owner to get started with the provided templates. There are 2 tutorials, for the BBB and the myRIO respectively.

#### *BBB Template*

Referring to the section 2.3.2 as reference for the application structure, this tutorial shows how to get started on the BBB template for the development and deployment of new labs.

#### Step 1: Setting Up the Development Environment

*This first step for setting up the development environment is written for Macs as target platforms. For Linux based system, this procedure is very similar, with some obvious difference to Linux users. As it is for Windows users, it is recommended that they install Cygwin*[8] *which provides a Linux-like environment on Windows.*

First, 2 drivers need to be installed:

1. Network Driver
2. Serial Driver

You can find the suitable distributions for your system on this webiste: `https://learn.adafruit.com/ssh-to-beaglebone-black-over-usb/overview`.

Once the BeagleBone Black shows as a connection in your Network settings, you can proceed to ssh-ing your BBB. To do so, you simply need to type the following in your terminal:

```
ssh 192.186.7.2 -l root
```

You can also find some additional information, on the previously mentioned website.

Second, you need to make sure that the board has the latest Angstrom distribution installed:

```
cat /media/BEAGLEBONE/ID.txt
```

[8] https://www.cygwin.com/

In case of a needed update, these links are useful for:

- Updating the software: `http://elinux.org/Beagleboard:Updating_The_Software`
- Installing operating systems: `https://learn.adafruit.com/beaglebone-black-installing-operating-systems/`

Third, an Internet connection needs to be shared with the board in order to download the template code. We will refer to this video tutorial for setting up the internet connection in the BBB: `http://www.youtube.com/watch?v=Cf9hnscbSK8`

*Please note that you need to make a distinction between BBB arriving before and after November 2013.*

Those are the steps to follow:

1. Connect your BBB to your computer using the USB cable coming with the board. To know to which socket on the board you need to plug the cable, refer to the one-page catalogue coming with the board.
2. Once your BBB is recognized by the system, and is present on the network, eject it from the Finder; but keep the board connected to the computer with the cable.
3. Go to System Preferences → Sharing. On the left column in the window, check "Internet Sharing" and in the right column, check BeagleBoneBlack. This should allow sharing the internet of your computer with the BeagleBone Black.
4. Open the terminal and:

   ```
    screen /dev/tty.usb* 115200
   ```

   if you are using a BBB shippped after November 2013. Otherwise:

   ```
   screen /dev/tty.usb*B 115200
   ```

   *Your BBB's USB might not be connected to tty.usb. Go to the /dev directory and check which tty.usbXXX is corresponding the connection with your BBB. For example, in my case it was tty.usb133.*
5. Use *root* for the login to the beaglebone
6. Next you need to:

   ```
   udhcpc -i usb0
   ```

   After this step, if all is alright, you should be able to ping `www.google.com` for example.
7. Next you need to:

   ```
   ifconfig
   ```

8. Kill the screen. The shortcut is **Ctrl+a+**\.

At this stage, you are connected to the Internet and you are able to do whatever you wish to do over the Internet. To have access again the memory of you beaglebone, you need to:

```
ssh root@beaglebone.local
```

**Step 2: Cloning the Github Repository on the BBB**

After making sure that git is installed, the code can be cloned from the repository. For more information on how to install git, you can refer to this website: `https://git-scm.com/book/en/v2/Getting-Started-Installing-Git`

To clone the repository, execute the command:

```
git clone https://github.com/go-lab/smart-device.git
```

The template resides in the directory: smart-device/templates/bbb

**Step 3: Getting Started with the Template**

The BBB is specifically conceptualized to be dynamic, enabling developers to create their own custom configurations and extensions known as “Capes”. The board can change its hardware configuration at runtime using its in-kernel mechanism : the “Cape Manager” with the help of the “Device Tree Overlays”.

The Device Tree is made of a set of human readble files of the type DTS (Device Tree Source), ending with the extension “.dts”. The DTS files are editable with a simple text editor such as VI, to set the configurations for pins. The DTS files are source files compiled into DTB files (binary files) ending with “.dtbo”. Those files are knows as device tree fragments of overlays. The Cape Manager then dynamically loads and unloads the DTB files on boot, and on runtime to set the hardware configuration of the board. More information can be found in Adafruit’s introduction to the BBB Device Tree : `https://learn.adafruit.com/downloads/pdf/introduction-to-the-beaglebone-black-device-tree.pdf` and on this web page: `http://elinux.org/BeagleBone_and_the_3.8_Kernel#Device_Tree.27s_data_driven_model` (Parvizi, 2013).

Having concisely introduced the hardware configuration of the BBB, we can get started on the code of the template. First, let’s start with the file bbb-process-controller.js:

**Listing 2.1: Initialization of Process Controller**

```
function bbbProcess(processPath, param) {
    // param to be replaced with the process param to control/sense.
    // e.g. period, duty cycle...
    bbbProcess.PARAM = param;
    bbbProcess.RUN_PATH = processPath + ‘run’;
    bbbProcess.PARAM_PATH = processPath + ‘param’;
    this.configureDevice();
}
```

This function is to initialize the Process Controller, it dynamically provides the path to the configuration files that the lab owner will take care of creating and populating. Regardless of the chosen path, 2 functions are provided in this controller code: turn on/off the process and configure one parameter. Respectively, the files are given directory locations. Once the Process Controller is initialized, the configuration function is called in line 7 of listing 7.1:

**Listing 2.2: Device Configuration**

```
//process initialization/configuration
bbbProcess.prototype.configureDevice = function () {
    var _this = this;

    this.writeFile(bbbProcess.RUN_PATH, ’1’).then(function () {
        //process initialization code
        // param to be replaced with the process param to control/sense.
        // e.g. period, duty cycle...
        return _this.writeFile(bbbProcess.PARAM_PATH, bbbProcess.PARAM);
    }).then(function () {
            console.log(’Process Configured...’);
        }, _this.errorHandler).done();
};
```

The configuration function uses the 2 available configuration data: turn on the process by writing the value 1 to the file responsible for turning on/off the process, and configure the general parameter param by writing the value bbbProcess.PARAM to the file responsible for configuring parameter param.

Next come 4 general purpose functions: writeFile, turnOn, turnOff, and setParam.

**Listing 2.3: Write to Pin Configuration File**

```
bbbProcess.prototype.writeFile = function (file, content) {
    var deferred = Q.defer();
    fs.writeFile(file, content, function (error) {
        if (error) {
            deferred.reject(error);
        }
        else {
            console.log(’writeFile complete: ’ + file);
            deferred.resolve();
        }
    });
    return deferred.promise;
};
```

This function takes 2 arguments: the path to the file, and the value to write to. The lab owner reuses this function for each pin s/he wishes to write to.

Listing 2.4: Turn Off Process

```
//turns OFF process
bbbProcess.prototype.turnOff = function () {
    this.writeFile(bbbPWM.RUN_PATH, '0');
};
```

Listing 2.5: Turn On Process

```
//turns ON process
bbbProcess.prototype.turnOn = function () {
    this.writeFile(bbbPWM.RUN_PATH, '1');
};
```

The functions turnOff and turnOn in Listings 2.4 and 2.5 already make use of the writeFile function of Listing 2.3 by writing respectively 0 and 1 to the file located at bbbPWM.RUN_PATH.

Listing 2.6: Set Process Parameter

```
bbbProcess.prototype.setParam = function (param) {
    try {
        bbbPWM.PARAM = param;
        fs.writeFile(bbbPRocess.PARAM_PATH, bbbPWM.PARAM );
    }
    catch (e) {
        console.log('setParam error: ' + e);
    }
};
```

Just like the turnOff and turnOn functions, the setParam function of Listing 2.7 writes the value param to the file located at bbbProcess.PARAM_PATH.

A last function is an error handling function:

Listing 2.7: Set Process Parameter

```
bbbProcess.errorHandler = function (error) {
    console.log('Error: ' + error.message);
};
```

The Q module of Node.js is used to take care of the case where a function cannot return a value or throws an exception without blocking. It then returns a promise instead. A promise is an object that represents the return value or the thrown exception that the function may eventually provide. The promise is then used as a proxy for a remote object to overcome latency [9]. More information can be found here: `http://documentup.com/kriskowal/q/`. Using this modality in in handling response times is good practice for guaranteeing the determinism of an application.

***myRIO Template***

Referring to the section 2.3.2 as reference for the application structure, this tutorial shows how to get started on the myRIO template for the development and deployment of new labs.

**Step 1: Setting Up the Development Environment**

All needed software for developing and deploying lab applications with LabVIEW and myRIO are:

1. LabVIEW[10]
2. LabVIEW Real-Time Module[11] in case the lab implements RT applications
3. LabVIEW myRIO Toolkit[12]

There are available installations for Mac, Linux, and Windows systems. The installation process is pretty straight forward with installing guides, where typically the user is asked to choose installation directories and going to the next steps.

In order to be able to directly work with the myRIO as a target device, it needs to be connected all the time to the development computer.

**Step 2: Cloning the Github Repository on the Development Computer**

After making sure that all needed installations are done as per Step 1 , the code can be cloned from the repository and used.

To clone the repository, execute the command:

```
git clone https://github.com/go-lab/smart-device.git
```

The template resides in the directory: smart-device/templates/myRIO

**Step 3: Getting Started with the Template**

Basically, to adapt this template to the desired application, the VIs of interest are:

[9] http://documentup.com/kriskowal/q/
[10] http://www.ni.com/download/labview-development-system-2014/4735/en/
[11] http://www.ni.com/download/labview-real-time-module-2014/4832/en/
[12] http://www.ni.com/download/labview-myrio-toolkit-2014/4854/en/

1. PID_real.vi: The Core Loop of this VI encloses 5 cases:
   a) idle (default): it's the case where the process is in idle state, and it is the default case to fall back on. This case is general to labs and need little or no modification for different lab setups.
   b) init: the case which starts hardware initialization and leads to the 'run' case. This case as well is general to labs and need little or no modification for different lab setups.
   c) stop: stops the process by sending and stop command to the modules responsible for stopping the process. This case is general to labs and need little or no modification for different lab setups.
   d) quit: quits the lab server process responsible for running the controller. This case is general to labs and need little or no modification for different lab setups.
   e) run: in this case lies the main dynamics of the lab's process. The modules and sub-called VIs for hardware interfacing and operating the hardware which are specific to a lab's setup need to be changed. In the figure below, the needed modules to replace are in the red rectangle:

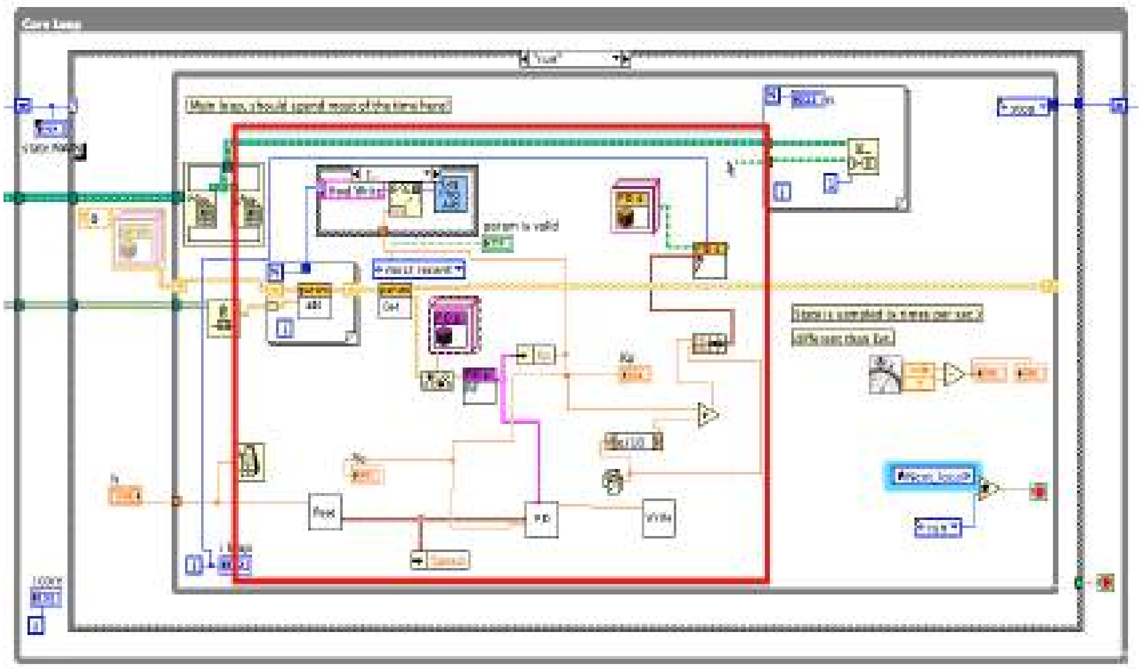


**Figure 5: PID_real.vi Snapshot**

2. HTMLReplies.vi: In this VI, the main case structure dispatches the incoming requests for user clients, metadata, and log files. The code in this VI for each of the cases is a string constant. The lab owners need to modify this code according to their requirements. The snapshot below shows how those string constants are linked to the main case structure:

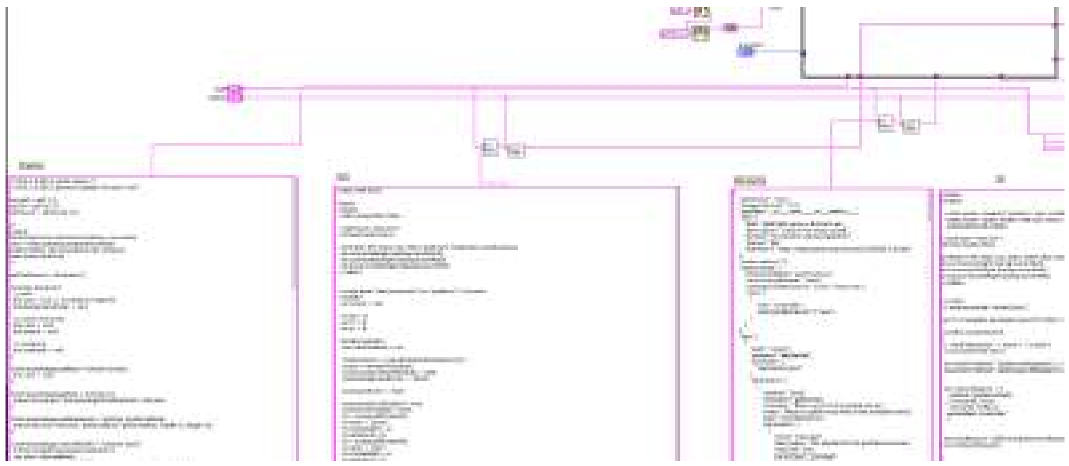


**Figure 6: HTMLReplies.vi Snapshot**

## *2.4 Examples*

### 2.4.1 Angular Position Control of an Electrical Drive: a LabVIEW Example on Desktop

#### *General Information*

**Source code**: `https://github.com/go-lab/smart-device/tree/master/Desktop`
**Documentation Wiki**: `https://github.com/go-lab/smart-device/wiki/Desktop-wiki`
**Hardware**: Desktop computer (Mac, PC) with DAQ card (NI PCIe 6259)
**Software**: LabVIEW (2013)
**Demo**: `https://github.com/go-lab/smart-device/wiki/Desktop-wiki`

#### *Short Description*

The example lab provides an experiment for controlling the angular position of an electrical drive. The motor's axle position is sensed and the motor's voltage is controlled to reach a reference position. It also uses a USB Video Class (UVC) webcam.

#### *Services and Functionalities*

This lab implements:

- Sensor and actuator access (including video access)
- A combined web and WebSocket server that handles the smart device services
- A mechanism to link the sensor/actuator to their respective services
- Internal functionalities management (request validation, controller)
- Concurrent user access with race policy for the controller mode (refer to D 4.5 to get more information about concurrency in the scope of Go-Lab)

The lab allows the manipulation of an electrical drive (RED Servo). The user can set and read the angular position of the axle. The client application implements the following:

- Displaying the axle position (sensor value display)
- Displaying the live video feed showing the disk connected to the drive's axle
- Setting the axle position (setting actuator value)
- Simple web apps (HTML client applications) for the above sensors/actuators
- Tracking of the desired axle position with an internal controller

#### *Guidelines to Reuse the Example*

To adapt the example for different physical instrumentation, the lab owner needs to:

- Provide new VIs or modify existing ones to read sensor values and add them to the queue
- Provide new VIs or modify existing ones to write actuator values dequeued from the queue
- Extend the existing VIs which validate user requests
- Enable/disable the internal controller or adapt it for the respective application
- Update and extend the 'metadata'

### 2.4.2 Angular Position Control of an Electrical Drive: a LabVIEW Example on myRIO

#### *General Information*

**Source code**: `https://github.com/go-lab/smart-device/tree/master/myRIO`
**Documentation Wiki**: `https://github.com/go-lab/smart-device/wiki/myRIO-wiki`
**Hardware**: NI myRIO1900
**Software**: LabVIEW (2013)
**Demo**: `https://github.com/go-lab/smart-device/wiki/myRIO-wiki`

#### *Short Description*

The example lab provides a different implementation of the experiment presented in 2.4.1. This example is adapted from the latter in 2.4.1 to fix the hardware interfacing differences between the DAQ module and the myRIO board.

#### *Services and Functionalities*

This lab implements:

- Sensor and actuators access
- A combined web and WebSocket server that handles the smart device services
- A mechanism to link the sensor/actuator to their respective services
- Internal functionalities management (request validation, controller, etc)
- Concurrent users access with race policy for the controller mode (refer to D 4.5 to get more information about concurrency in the scope of Go-Lab)

The lab allows the manipulation of an electrical drive (RED Servo). The user can set and read the angular position of the drive axle. The client application implements the following:

- Displaying the axle speed (sensor value display)
- Setting the axle position (setting actuator value)
- Simple web apps (HTML client applications) for the above sensors/actuators
- Tracking of the desired axle position with an internal controller

#### *Guidelines to Developers*

To adapt the example for different physical instrumentation, the lab owner needs to:

- Provide new VIs or modify existing ones to read sensor values and add them to the queue
- Provide new VIs or modify existing ones to write actuator values dequeued from the queue
- Extend the existing VIs which validate user requests
- Enable/disable the internal controller or adapt it for the respective application
- Update and extend the 'metadata'

### 2.4.3 Savonius VA Wind Turbine Control: a LabVIEW Example on myRIO

#### *General Information*

**Source code**: `https://github.com/go-lab/smart-device/tree/master/myRIO/wind-turbine-interplay`
**Documentation Wiki**: `https://github.com/go-lab/smart-device/blob/master/myRIO/wind-turbine-interplay/README.md`
**Hardware**: NI myRIO1900
**Software**: LabVIEW (2014)
**Demo**: `http://shindig2.epfl.ch/windturbine_myrio.htm`

### *Short Description*

The example lab provides an experiment for controlling the operation of a Vertical Axis (VA) wind turbine, also known as the Savonius[13] turbine. A wind source is turned ON/OFF to operate the turbine and cause electrical current generation. The voltage is graphed in a real-time interactive display.

### *Services and Functionalities*

This lab implements:

- Sensor and actuators access
- A combined web and WebSocket server that handles the smart device services
- A mechanism to link the sensor/actuator to their respective services
- Internal functionalities management (request validation, controller, etc)
- Concurrent users access with race policy for the controller mode (refer to D 4.5 to get more information about concurrency in the scope of Go-Lab

The lab allows the manipulation of a Savonius wind turbine. The user can operate the wind source to provoke electrical current generation. The client application implements the following:

- Displaying the value of generated voltage in an interactive graph (sensor value display)
- Operating the wind source (setting actuators value)
- Providing live video feed
- Showing if user is controller/observer and remaining time for control/watch mode

### *Guidelines to Developers*

To adapt the example for different physical instrumentation, the lab owner needs to:

- Provide new VIs or modify existing ones to read sensor values and add them to the queue
- Provide new VIs or modify existing ones to write actuator values dequeued from the queue
- Extend the existing VIs which validate user requests
- Enable/disable the internal controller or adapt it for the respective application
- Update and extend the ‘metadata’

[13]https://en.wikipedia.org/wiki/Savonius_wind_turbine

### 2.4.4 Savonius VA Wind Turbine Control: a Node.js Example on BBB

#### *General Information*

**Source code**: `https://github.com/go-lab/smart-device/tree/master/BeagleBoneBlack/vertical-turbine-bbb`
**Documentation Wiki**: `https://github.com/go-lab/smart-device/blob/master/BeagleBoneBlack/vertical-turbine-bbb/README.md`
**Hardware**: BeagleBone Black
**Software**: Node.js
**Demo**: `http://shindig2.epfl.ch/windturbine_bbb.htm`

#### *Short Description*

The example lab provides a different implementation of the experiment presented in 2.4.3. This example uses different technologies, and provides extra features in the user client: saving experimental data in text files, real-time lab status update, scrolling back to old data, and different user interface.

#### *Services and Functionalities*

This lab implements:

- Sensor and actuators access
- A WebSocket server that handles the smart device services
- A mechanism to link the sensor/actuator to their respective services
- A metadata service as per the Smart Device specifications in D4.7 for lab identification and interfacing

The lab allows the manipulation of a Savonius wind turbine. The user can operate the wind source to provoke electrical current generation. The client application implements the following:

- Displaying the value of generated voltage in an interactive graph (sensor value display)
- Operating the wind source (setting actuators value)
- Providing live video feed
- Providing real-time lab status update
- Saving experimental data
- Scrolling back to old data

#### *Guidelines to Developers*

To adapt the example for different physical instrumentation, the lab owner needs to:

- Install the necessary node modules using the Node Packaged Modules[14].

[14]https://www.npmjs.org/

Lab owners can find node modules necessary for their specific applications on the mentioned website.

- Modify the package.json file to include the newly installed modules or delete the unnecessary ones
- Modify the README.md file to better describe the specific application.
- Modify the bbbADC.js file to correctly interface the specific instrumentation connected to the BeagleBone Black. Lab owners can use the JavaScript libraries on this Beagle website.[15]
- Modify the websocket-server.js file to adequately service the user client.

### 2.4.5 Angular Position Control of a Mini-Servo Motor: a Node.js Example on BBB

***General Information***

**Source code**: `https://github.com/go-lab/smart-device/tree/master/BeagleBoneBlack/servo-beaglebone-black`
**Documentation Wiki**: `https://github.com/go-lab/smart-device/tree/master/BeagleBoneBlack/servo-beaglebone-black#servo-beaglebone-black`
**Hardware**: BeagleBone Black
**Software**: Node.js application
**Demo**: `https://github.com/go-lab/smart-device/wiki/BeagleBone-wiki`

***Short Description***

The example lab provides an experiment for controlling the angular position of a mini-servo motor with PWM (Pulse Width Modulation). The user is provided with a slider to increase/decrease the duty cycle of the period, and hence controlling the angular position of the motor's shaft. Sensor and actuator values are displayed, and live video feed is provided, with real-time system status updates.

***Services and Functionalities***

This lab implements:

- Sensor and actuators access
- A WebSocket server that handles the Smart Device services
- A mechanism to link the sensor/actuator to their respective services
- A metadata service as per the Smart Device specifications in D4.7 for lab identification and interfacing

The lab allows the manipulation of a mini-servo motor. The user can set and read the angular position of the motor's shaft. The client application implements the following:

- Displaying the duty cycle corresponding to the angular position (sensor

[15]http://beagleboard.org/Support/BoneScript

value display)

- Setting the duty cycle (setting actuator value)

***Guidelines to Developers***

To adapt the example for different physical instrumentation, the lab owner needs to:

- Install the necessary node modules using the Node Packaged Modules[16]. Lab owners can find node modules necessary for their specific applications on the mentioned website.
- Modify the package.json file to include the newly installed modules or delete the unnecessary ones
- Modify the README.md file to better describe the specific application.
- Modify the bbb-pwm.js file to correctly interface the specific instrumentation connected to the BeagleBone Black. Lab owners can use the JavaScript libraries on this Beagle website.[17]
- Modify the websocket-server.js file to adequately service the user client.

### 2.4.6 Robot Arm: a Node.js Example on Raspberry Pi

***General Information***

**Source code**: `https://github.com/go-lab/smart-device/tree/master/RaspberryPi/robotic_arm-raspberry-pi`
**Documentation Wiki**: `https://github.com/go-lab/smart-device/wiki/Robotic-Arm-Laboratory`
**Hardware**: Raspberry Pi B, and Arduino UNO
**Software**: Raspbian O.S., Node.js, Socket.io, serialport, Apache
**Demo**: `http://graasp.eu/applications/54d8ad0f17cf888ac8d6af04`

***Short Description***

The example lab provides an experiment for controlling a robotic arm which simulates a set of the human arm movements: moving right/left backwards/onwards, grabbing an object, etc.

***Services and Functionalities***

This lab implements:

- A live video feed
- Setting the control for each of the robotic arm's motors for the clamp, wrist, elbow, shoulder and base (setting actuator value)

[16]https://www.npmjs.org/
[17]http://beagleboard.org/Support/BoneScript

***Guidelines to Developers***

To adapt the example for different physical instrumentation, the lab owner needs to:

- Modify the use of the serialport library in case of switching from USB to another protocol for interfacing the Raspberry Pi and Arduino boards
- Modify the HTML files for the corresponding UI requirements
- The hardware interfacing modules for different instrumentation

# 3 Releases of Cloud Services

As defined in D4.5, in the scope of this document, Cloud Services are a set of services designed to extend the functionalities of the Go-Lab infrastructure to lab owners of legacy lab systems, by providing them with means to plug their systems into the Go-Lab ecosystem. Legacy laboratories are all online lab platforms not designed in compliance with the Smart Device specifications as defined in D4.5. This chapter is an updated version of chapter 3 of D4.3 (Releases of the Lab Owner and Cloud Services - Initial). In the following sections details about the Smart Gateway architecture and implementation are presented, in addition to the software releases.

## *3.1 Cloud Services Final Release Compared to the Initial Release*

As previously mentioned, this deliverable is an updated version of D4.3 (Releases of Lab Owner and Cloud Services - Initial). This section describes the timeline of the cloud services since the beginning of the project, with a focus on the developments made since the release of D4.3. The main features implemented since then are:

- **Internationalisation support:** This feature is described in detail in section 3.4.4. It provides legacy lab owners with the possibility to get their labs translated by the Go-Lab community using the App Composer. Translations have to comply with one of the supported formats and it is an additional benefit for lab owners that sharing labs with Go-Lab.
- **Additional HTTP plug-in templates:** New templates were developed to be offered as as tool for lab owners wishing to share labs with Go-Lab. These templates were developed for PHP and LabVIEW. More details about the HTTP plug-ins is available in section 3.4.3.
- **Integration with Remlabnet RLMS:** A new remote laboratory management system (Remlabnet) was integrated into the Go-Lab ecosystem with an HTTP plug-in. More details follow in section 3.5.6
- **Implementation of a protocol translator:** A protocol translator that translates legacy requests to smart device compliant requests was implemented as a proof of concept for ISA and WebLab Deusto RLMSs. A detailed description can be found in section 3.4.6.

## *3.2 Introduction*

In D4.5, the requirements and the design of the Smart Gateway are presented. The Smart Gateway is presented as an approach to integrate legacy laboratories. The methodology is composed of:

- gateway4labs: a management tool for integrating the reservation process of the legacy laboratories, which might require wrapping authentication, authorization, scheduling and providing additional services (such as metadata or booking on the Go-Lab level).

- Action Logger: an optional mechanism to support the logging of user actions with the laboratory.
- Protocol translator: an optional component, to be implemented by the lab owner for translating the existing communication mechanisms to Smart Device compliant services.

As already discussed in D4.3, there is a trade-off between the development effort put on components and the features provided.

## 3.3 The Smart Gateway Architecture

The main purpose of the Smart Gateway is to allow the integration of legacy labs in the Go-Lab Platform, by making them fully or partially compliant with the Smart Device specifications. The level of compatibility depends on the implementation strategy adopted (see D4.5 for a description of the different integration levels). Figure 7 below shows the Smart Gateway architecture in detail.

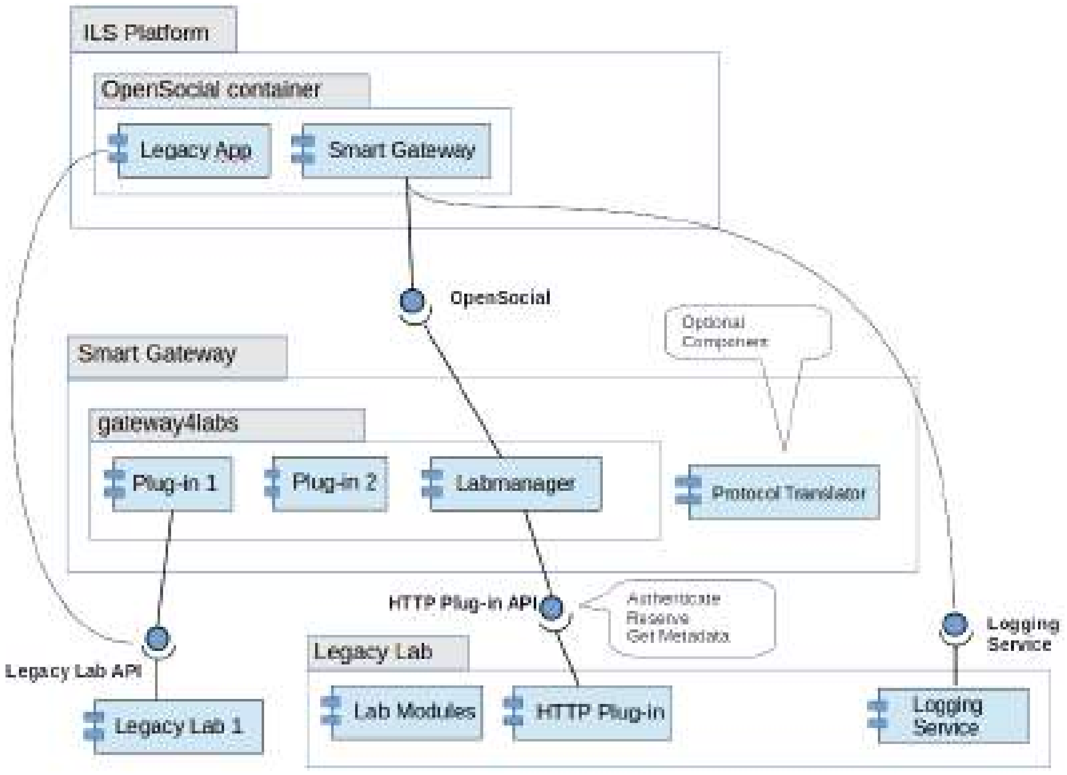


**Figure 7: Smart Gateway Architecture**

The *gateway4labs* is the core of the architecture. It provides a core component (a web application called LabManager), and different approaches for including external resources. It provides metadata services and exports the legacy lab clients as OpenSocial gadgets that can be added to an Inquiry Learning Space (ILS). The Smart Gateway supports different legacy lab systems via different plug-ins. In this architecture, the plug-ins are responsible for bridging the communication between the LabManager and the legacy lab systems. The functionalities that a plug-in should implement depend on the level of integration desired (see D4.1 for details). For online labs managed by a Remote Laboratory Management System (RLMS), (an RLMS is a system that provides a management layer for multiple remote laboratories, optionally including authentication, authorization, scheduling or user tracking), a single plug-in is necessary to integrate all labs managed by the same RLMS. Plug-ins for some well-known RLMSs like WebLab-Deusto and iLabs Shared Architecture are provided. The detailed implementation of each one is discussed in the following sections.

Lab owners can choose from four different options to plug their systems into the Smart Gateway that will affect the development of plug-ins and the supported features. Therefore there is a trade-off between the supported features and the implementation efforts. These options were described in detail in D4.1, and briefly recapitulated below:

1. iFrame in Smart Gateway: this can be implemented if authentication is not required by the legacy lab system. This option consists of providing the legacy client as an OpenSocial application. Hence technically, it is an enhanced iFrame. Additionally, by including a lab, the Smart Gateway administrator has the possibility to author some metadata content for the lab that the Smart Gateway will use to provide the metadata services (part of the Smart Device specifications).
2. A simple version of the plug-in: if the legacy lab requires authentication the plug-in can be implemented in such a way that it will log into the legacy lab system with a fixed user account. In this approach the legacy lab administrator will not be able to uniquely identify the user, so some features like the possibility of tracking used actions in the context of an experiment (lab level) would be unavailable. Of course the logging of user actions at ILS level is always available.
3. A full version of the plug-in: in this case all reservation features provided by the legacy system are bridged by the plug-in. Users can be uniquely identified. Parts of these features also depend on implementations at the legacy lab side. It should return a URL that will be loaded by the client.
4. A full version of the plug-in plus a protocol translator: Additionally to implementing a full version of the plug-in this option requires all messages exchanged between client and server to be translated according to the Smart Device specification. It requires potentially a large implementation effort and an individual solution for each legacy system.

## 3.4 The Smart Gateway Software

The Smart Gateway (`http://gateway.golabz.eu/`) consists of gateway4labs (which manages the authentication, scheduling mechanisms), the protocol translator and a lightweight implementation of the protocol translator, that supports the logging of user actions in the context of a lab reservation. More background about gateway4labs is available in chapter 8. The following diagram describes the overview of the gateway4labs software components.

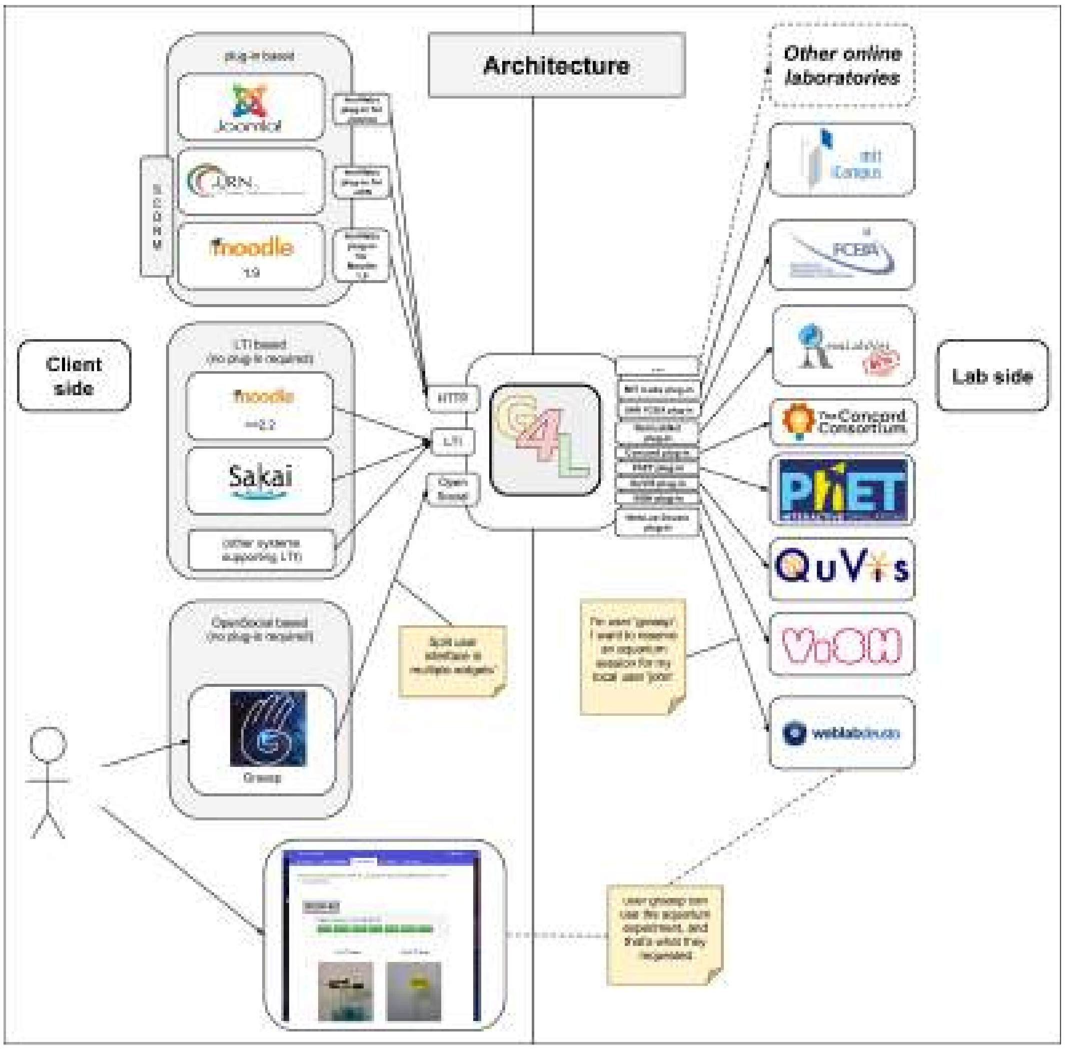


Figure 8: Gateway4Labs overall architecture

One can identify two main layers on Figure 8: the left side is the client side and the right side is the lab side. On the left side, different learning tools can be integrated through the support of existing standards by the core component of gateway4labs, which is called Labmanager. On the right side, different remote laboratories can be integrated through a set of plug-ins. In the middle, the Labmanager performs all the management operations and conversions between the different actors of the left side and the right side.

In this way, we can split the functionality of the gateway4labs in three parts:

1. The support for standards, and OpenSocial in particular since it is the one

being supported by the ILS platform. This is detailed in section 3.4.1.

2. The support for remote laboratories, through the plug-in mechanism, detailed in Section 3.4.3.
3. The management side of the core component (Labmanager).

### 3.4.1 Support for Standards

In gateway4labs, the core component (labmanager) natively supports three different systems:

1. **OpenSocial:** described in this section.
2. **IMS LTI:** Learning Tools Interoperability, it is a standard supported by the main LMS environments, which supports integrating external tools natively. Since the labmanager implements this standard, every laboratory supported can be automatically integrated in any LMS, which supports it.
3. **HTTP-based plug-ins:** A custom HTTP interface is provided, so external tools not supporting IMS LTI or OpenSocial can consume this interface to connect to gateway4labs. This includes plug-ins in systems such as Joomla[1] or an LMS not supporting LTI, such as dotLRN[2].

Since the interface between gateway4labs and the ILS platform is based on OpenSocial, this section is focused on this implementation.

#### *Context Information Management*

When a user (teacher or student) is using the ILS platform, there is certain context information which could be useful for gateway4labs, such as: who is the user (if identified), where is the user accessing from (ILS) or what language is selected by the teacher for this ILS (e.g., French, German, Spanish, etc.). This contextual information is useful to provide feedback to the laboratory about who is using the laboratory. The context information is particularly important when the labmanager requests a reservation on the legacy lab system.

#### *Public Laboratories*

Once the administrator registers a laboratory, by default gateway4labs does not make it publicly available for unregistered users. This is the default behaviour in remote laboratory management systems. However, the administrator can configure that a particular laboratory is openly available for everyone.

This way, in the OpenSocial version it is possible to provide the laboratory to certain ILSs, or it can be available for every space, as it will be the common case.

[1] https://github.com/gateway4labs/cms_joomla
[2] https://github.com/gateway4labs/lms4labs/tree/master/lms/dotLRN

### 3.4.2 Existing Features

Additionally, gateway4labs supports other optional features, such as enabling the laboratory to restart the lab app whenever it is required. In WebLab-Deusto, for example, whenever the user performs a reservation, the user will be able to use a single reservation. Whenever the reservation is over (and the student is not using the laboratory anymore), a message is submitted to the labmanager to restart the lab app and be able to perform a new reservation if desired.

So as to support this optional mechanism, gateway4labs provides a URL that will force a reload of all the lab apps of the same laboratory. This URL is passed to the laboratory plug-in, so it can optionally load this URL when finished. Certain laboratories (like the iLab radioactivity laboratory, where the user can perform more operations once using it) do not require these features.

### 3.4.3 Support for remote laboratories: The plug-in system

The core functionality of gateway4labs is to provide an API to support a wide range of laboratories. This API aims to be simple so as to encourage laboratory owners to develop their plug-ins, by avoiding strict requirements other than a link to the final laboratory crossing the reservation processes of the final laboratory. This API is described in detail in D4.5.

***Implementations of the plug-in system for different legacy platforms***

The base of the plug-in system is that the laboratory must support some mechanism to provide a link that identifies the current reservation. There are different ways to approach this:

- Using federation protocols: in the case of WebLab-Deusto (`http://weblab.deusto.es/`), a federation mechanism is used to access a WebLab-Deusto server as if it was another WebLab-Deusto server requesting a reservation. WebLab-Deusto returns a URL which includes a reservation identifier, and the labmanager can use this link to redirect the user to that location. From that point, all the communications do not cross the gateway4labs infrastructure.
- Creating users dynamically: in the case of the iLab Shared Architecture (`http://icampus.mit.edu/projects/ilabs/`), the plug-in creates a new user (if it was not previously made) and grants this user permission to use that laboratory (if he did not have that permission), and finally the user is redirected to the final system. There, the user can start using the laboratory he has access to.
- Using encryption to sign messages: in the case of the UNR (`http://labremf4a.fceia.unr.edu.ar/about/` - see section 3.5.5), a secret key is stored in the plug-in, and a message that includes the username, the current timestamp, and the laboratory is signed. The message and the signature are returned to the labmanager in the form of a URL. When the user is redirected to that URL, the laboratory at UNR verifies whether this

message is valid and if it is, it enables the user to access the laboratory until the timestamp expires.

- Simple redirection: in the case of PhET ( `https://phet.colorado.edu/` - see section 3.5.3), the plug-in just generates the public link to the PhET simulation. This way, the user is redirected to the public link directly.
- Federated search: in the case of ViSH ( `http://vishub.org/` - see section 3.5.4), the plug-in must forward the query provided by the user to the ViSH repository, and then generate a link.

  Other mechanisms could be employed, since this depends completely on how the final laboratory enables users to access. Additionally, the plug-in must provide a way to define the initial configuration. In the case of WebLab-Deusto or iLab, this includes the particular server and credentials of this server. In the case of UNR, the secret key must be configured.

***The Plug-in: Python version***

Since gateway4labs is developed in Python, the basic plug-in structure is based on Python. The developer only needs to develop a module or a package called g4l_rlms_{{ name }}, and then in the configuration file of the Labmanager, the developer must add the name of the plug-in to the RLMS variable (which is a list, as shown below). For example, if we create a plug-in called "foo", we just need to create a file called g4l_rlms_foo.py (or a package called g4l_rlms_foo, with different files inside), and in the config.py add:

**RLMS = ['weblabdeusto','ilabs','foo']**

Then, it can provide a function called get_module(version), so it could support different versions through having different modules. For example, iLab could have two independent modules that fulfill the Python API in the same plug-in, namely "g4l_rlms_ilab.py", "ilab_3_1.py" and "ilab_3_5.py", and in the first one it could provide the get_module(version) function. Whenever version was 3.1, it could rely on one module, and whenever the version was 3.5, it could rely on the other one.

The module used must provide an instance FORM_CREATOR, which creates the required forms. In the case of the Python version, the configuration is managed by subclassing a set of classes based on WTForms[3]. In these classes, the developer can select how to convert these values into a single JSON document. This JSON document will be stored in the database, and it will be used with all the instances of the plug-in (e.g., different instances of WebLab-Deusto). This process is detailed in Figure 9.

[3]http://wtforms.readthedocs.org/

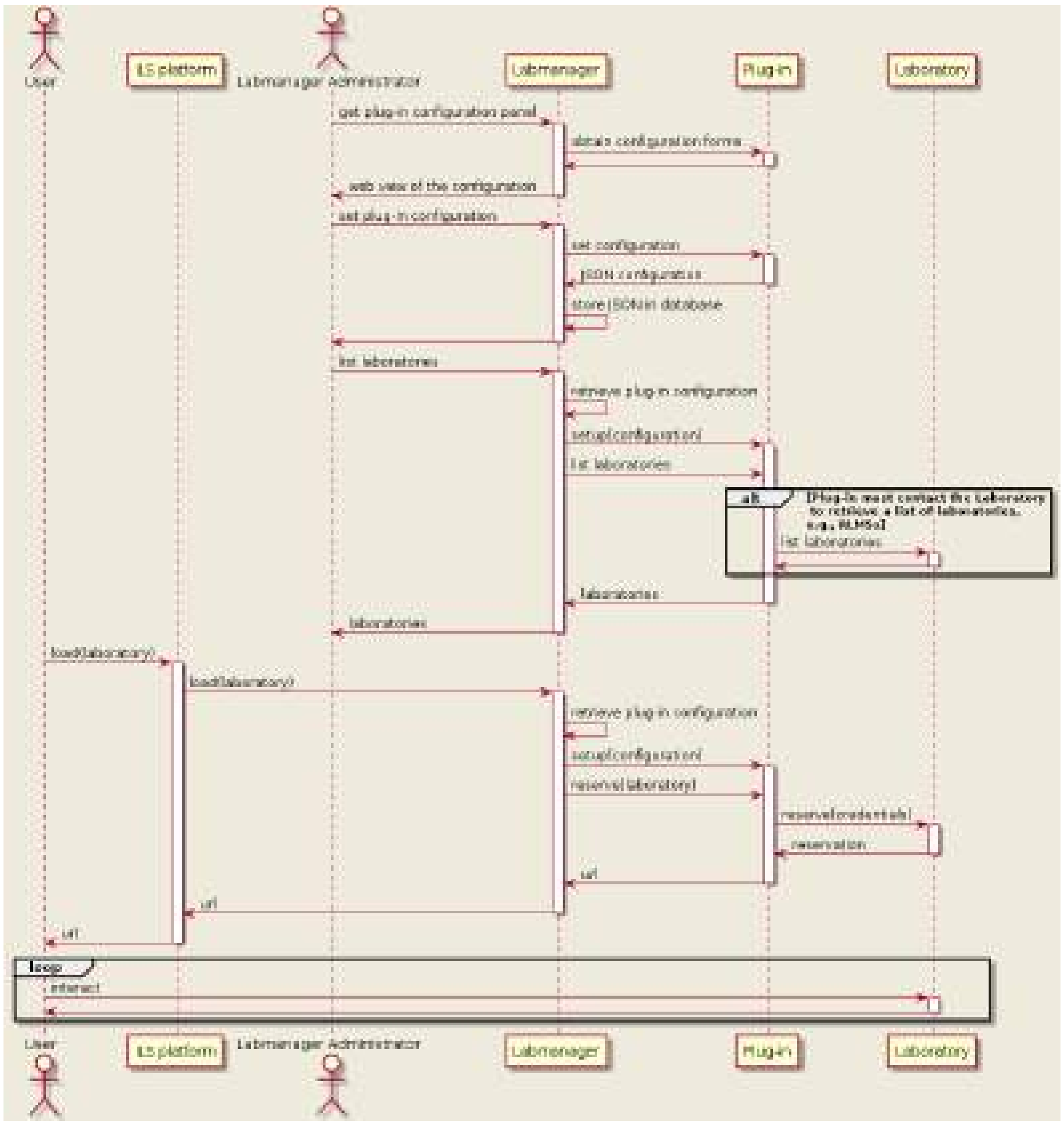


**Figure 9: Configuration process in Python plug-ins**

Finally, the module must create a class called RLMS. This class will be instantiated with the JSON configuration, and it is expected to use that configuration to perform each of the tasks mentioned in the previous subsection.

The following are examples of Python plug-ins:

- WebLab-Deusto plug-in.
  - Repository: `https://github.com/gateway4labs/rlms_weblabdeusto`
  - Original: `http://weblab.deusto.es`
  - Notes: three files are used in a single package g4l_rlms_weblabdeusto. Two of them (weblabdeusto_client.py and weblabdeusto_data.py) are taken from WebLab-Deusto directly,

while the third one (__init__.py) specifies the rest.

- iLab Shared Architecture plug-in.
  - Repository: `https://github.com/gateway4labs/rlms_ilabs`
  - Original: `http://ilab.mit.edu/wiki`
  - Notes: there is a single g4l_rlms_ilabs.py file which matches the specification explained above. There is also a ASPX file to be optionally added to the iLab server.
- UNR-FCEIA plug-in.
  - Repository: `https://github.com/gateway4labs/rlms_unr`
  - Original: `http://labremf4a.fceia.unr.edu.ar/`
  - Notes: there is a single g4l_rlms_unr.py file. In this case, there is no communication between the plug-in and the final server, since it's based on a cryptographic solution where the plug-in generates and signs tokens that the user will forward to the final server.
- PhET plug-in.
  - Repository: `https://github.com/gateway4labs/rlms_phet`
  - Original: `http://phet.colorado.edu`
  - Notes: this represents a set of simulations. There is no reservation process, so the reserve method is focused on generating links to the simulation.
- ViSH plug-in.
  - Repository: `https://github.com/gateway4labs/rlms_vish`
  - Original: `http://vishub.org/`
  - Notes: this represents a set of simulations. There is no reservation process, so the reserve method is focused on generating links to the simulation.
- QuVis plug-in
  - Repository: `https://github.com/gateway4labs/rlms_quvis`
  - Original: `http://www.st-andrews.ac.uk/physics/quvis/`
  - Notes: this represents a set of simulations. There is no reservation process, so the reserve method is focused on generating links to the simulation.
- Concord plug-in.
  - Repository: `https://github.com/gateway4labs/rlms_concord`
  - Original: `http://concord.org/`
  - Notes: this represents a set of simulations. There is no reservation

process, so the reserve method is focused on generating links to the simulation.

***The Plug-in: HTTP version***

The HTTP version relies on a RESTful interface that can be implemented in any language to support the development of plug-ins in other technologies. It also makes it possible to decouple the Labmanager maintenance and the plug-in maintenance, since the plug-in could be deployed in another institution, as discussed in D4.1. Furthermore, a Python server supporting this RESTful interface will be implemented to make it possible to distribute the existing plug-ins in other institutions if desired. In the approach selected (and explained in D4.1), the plug-in stores information of the final system such as credentials or URLs. To configure the plug-in and add this information, the Smart Gateway administrator will be redirected to a website provided by the plug-in. This workflow is described in Figure 10.

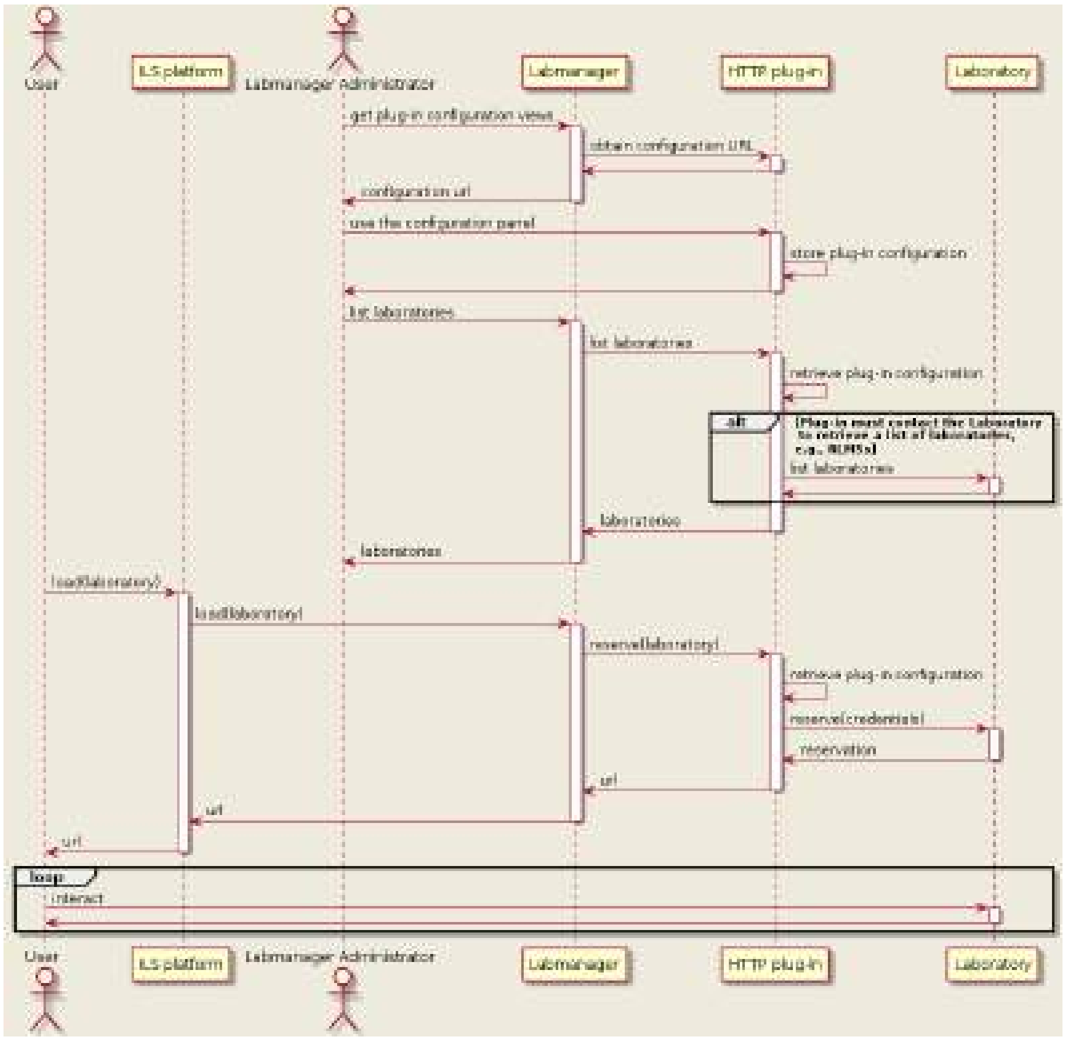


**Figure 10: HTTP plug-in configuration stored in the plug-in side**

This approach provides a considerably wider flexibility to the HTTP plug-in developer, since he can provide and upgrade in the future the configuration variables without interacting with the Labmanager. The panel can be very simple or very sophisticated, depending on the complexity of the remote laboratory. The model becomes much simpler, since it does not need to agree the available constraints. Additionally, the configuration secrets are kept on the HTTP plug-in side. On the other hand, this requires the HTTP plug-in to be publicly available to the Internet, while previously it could be just available in the Labmanager. It also requires the HTTP plug-in to be responsible of more roles, such as storing the configuration information.

***HTTP plug-in implementations since D4.3***

Different implementations of the HTTP plug-in were developed that can be used by lab owners as templates for the integration of legacy lab with the HTTP interface. These templates are available for different platforms:

- Python (Flask)
- Java
- PHP (Slim)
- .NET (ASP.NET with C)
- LabVIEW

A list of examples is provided in the following repository:

- **Repository:** https://github.com/gateway4labs/labmanager/tree/master/examples/http_plugins/

By the time deliverable D4.3 (M21) was released the available implementations were:

- **Repository:** https://github.com/gateway4labs/labmanager/tree/b2dfb7a395d1bd7d2d1ac1bd0c426d0309197ea6/examples/http_plugins

### 3.4.4 New features

In this deliverable (D4.7), new features were added, including support for internationalisation, automatic resizing of the application ((as requested through participatory design by WP1 and WP3), a caching mechanism and a repeating task scheduler, explained below:

***Internationalisation support***

Go-Lab resources must support a wide range of languages to reach its audience. Go-Lab applications rely on the OpenSocial standard, which provides an internationalisation mechanism. In WP5, the App Composer uses this mechanism to automatically translate applications and laboratories developed in OpenSocial.

However, the Smart Gateway integrates external laboratories, developed in other technologies. So it needs to support a variety of internationalisation formats. To simplify this process, the Smart Gateway provides an additional optional method called `getTranslations`, which should be implemented by the lab owner. It returns the translations in a common JSON format, such as the one available in 3.1. On it, the lab owner indicates:

- Contact e-mail addresses: whenever somebody provides a translation of the laboratory, these e-mail addresses will be contacted saying who translated what.
- Translations in each language. Identified by a key message, they contain a value (the text in the specified language) and optionally a namespace. The namespace identifies unique phrases through different laboratories. For example, laboratories developed by the same author or using the same system will typically have common code with common phrases. Identifying them as shared among independent laboratories enable the translation engines to use existing translations.

By obtaining these translations, the Smart Gateway later exports them as OpenSocial translations. This way, the App Composer (WP5) can later take those translations, and contact the authors providing the translations in a wide range of formats for developers (e.g., in jQuery i18n, .properties file, a generic JSON file, OpenSocial).

**Listing 3.1: Internationalisation format used in the Smart Gateway**

```
{
    "mails": [
        "labowner1@university.edu",
        "labowner2@university.edu",
    ],
    "translations": {
        "en": {
            "experimenthostedby": {
                "namespace": "http://labsystem.university.edu/#",
                "value": "Experiment hosted by"
            },
            "sensorsHelpTitle": {
                "namespace":
                    "http://labsystem.university.edu/experiment/#",
                "value": "Sensor information"
            },
        },
        "es": {
            "experimenthostedby": {
                "namespace": "http://labsystem.university.edu/#",
                "value": "Experimento hospedado por"
            },
            "sensorsHelpTitle": {
                "namespace":
                    "http://labsystem.university.edu/experiment/#",
                "value": "Informaci\u00f3n del sensor"
            },
        },
    }
}
```

### *Caching mechanisms*

The Smart Gateway provides OpenSocial representations of all the labs it manages. The OpenSocial xml file must be obtained quickly, and it must contain the translation messages. Additionally, many tasks (such as retrieving the list of labs in an external repository, or obtaining the URL from an external service), can be cached to provide a faster feedback to users. To this end, the Smart Gateway Python plug-in API provides a caching mechanism which stores in the local database whatever is requested, using a key/value basis. If necessary, in the future it could be implemented with other faster database or memory database without affecting existing plug-ins.

In the case of PhET, QuVis and Concord, the Smart Gateway caches all the labs to the final laboratories. This way, once cached, it will be stored and the original servers will not be contacted. This cache is stored for 24 hours, and it can be forced to be removed using the web browser tools for forcing a cache reload. In the case of iLab, WebLab-Deusto and HTTP plug-ins (such as Remlabnet), it is used for the list of labs (but it is maintained for only one hour) as well as for the translations (so the method is called only from time to time). In the case of HTTP plug-ins, it is also used for common methods such as "getCapabilities()" or "getVersion()". Further fine-grained cache mechanisms in the HTTP plug-ins would require the plug-in developer to manage it inside the plug-in.

Additionally, the Smart Gateway provides a special HTTP client which stores a common cache for all the plug-ins. If a URL is retrieved with HTTP caching mechanisms (e.g., standard etag / Last-Modified headers), it is stored and in the future it will act accordingly, obtaining the information from the shared cache.

### *Repeating task scheduler*

Another feature provided by the Smart Gateway is a scheduler so the plug-in can explicitly provide tasks to be run every given amount of time. For example, the Smart Gateway lets the plug-in developers to provide functions that populate the whole cache so as to keep always the data in the cache and therefore avoid that somebody ever needs to wait for the data. This way, it is guaranteed that while the plug-in code is called and is kept simple, if properly configured it will never take time populating a cache when a user is requesting it. So in the case of the plug-ins developed, it typically forces in background a cache load every few minutes before it expires.

***Automatic height resize***

The Smart Gateway laboratories typically embeds external resources located in other domains. A common problem with this is that, while the ILS manages the app width using a responsive design, there is no way to know the height of this resource in real time due to security constraints. For this reason, two complementary approaches have been selected:

- Enable teachers taking the laboratory to select the height of the lab app manually with a slider, as shown in figure 11.
- Enable plug-in developers to embed an external JavaScript library called iframe-resizer[4] which reports the container (Smart Gateway) what is the height of the current frame. This approach is used in WebLab-Deusto and iLab.

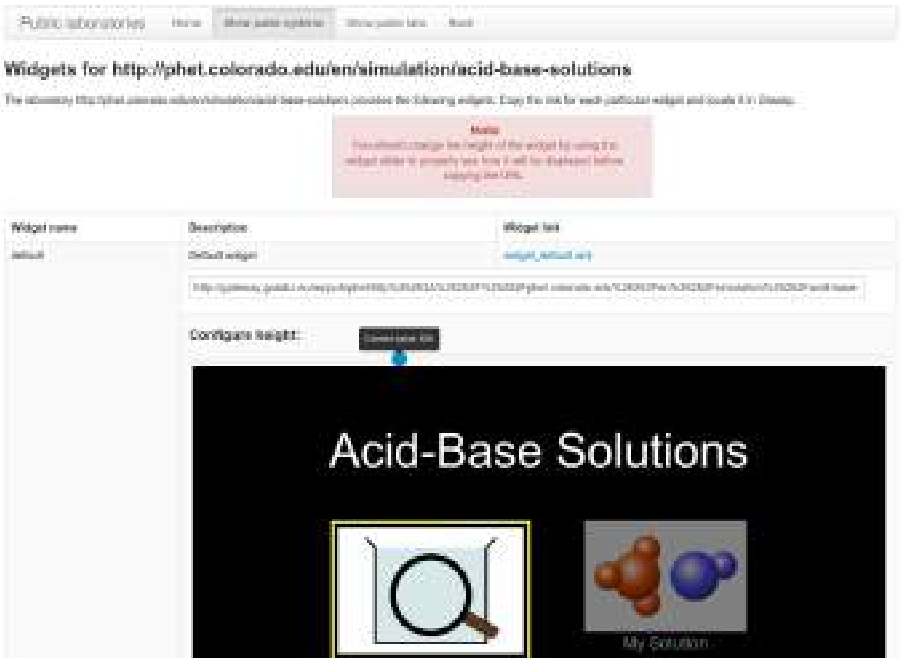


**Figure 11: Slider to select the height of the labs integrated in the Smart Gateway**

### 3.4.5 Demo and Software Repository

The production system is located in the following URL, and it is regularly used in Go-Lab:

- `http://gateway.golabz.eu`

If you click on "See public labs", you may explore the active laboratories available in the Smart Gateway.

However, the administrative actions are not available to the general public. So as to see these options, a sandboxed environment has been created for this

[4]https://github.com/davidjbradshaw/iframe-resizer

deliverable:

- URL: `http://weblab.deusto.es/golab/labmanager_d47/`
- Username: `admin`
- Password: `password4reviewers`

Additionally, all the source code of the labmanager is in the following GitHub repository and documented in the following readthedocs site:

- `http://github.com/gateway4labs/labmanager/`
- `http://gateway4labs.readthedocs.org/`

### 3.4.6 A prototype of the protocol translator

The protocol translator is an additional and optional component of the Smart Gateway that mainly translates legacy communication (legacy lab specific) and expose them as services compliant with the Smart Device specifications (using WebSockets & JSON) The concept of the protocol translator is explained in detail in deliverable D4.5. Its implementation is lab specific and will usually serve one single legacy lab since the messages exchanged by these systems are often different and domain-specific. In the scope of this deliverable, two prototypes of the protocol translator were developed as a proof of concept. The developed prototypes are for the Radioactivity laboratory (`http://www.golabz.eu/lab/radioactivity-lab` and the Archimedes' Principle laboratory (`http://www.golabz.eu/lab/archimedes-principle`). Both labs are available in the Go-Lab Repository.

#### *The protocol translator for the Radioactivity Laboratory (iLab Shared Architecture)*

The Radioactivity laboratory is owned by the University of Queensland, Australia. This lab is a batched laboratory (asynchronous), which means that the user does not control the lab equipment in real time. This characteristics of this lab poses several challenges when implementing a mechanism that translates all legacy messages to smart device call (see D4.5 for more details on the smart device specification). Furthermore, the radioactivity lab was developed for the iLab Shared Architecture (ISA). The translation process, in the particular case of ISA means translating smart device compliant calls to Web Service SOAP calls. The protocol translator has to abstract the whole process involved in a batched experiment execution. Figure 12 depicts this process.

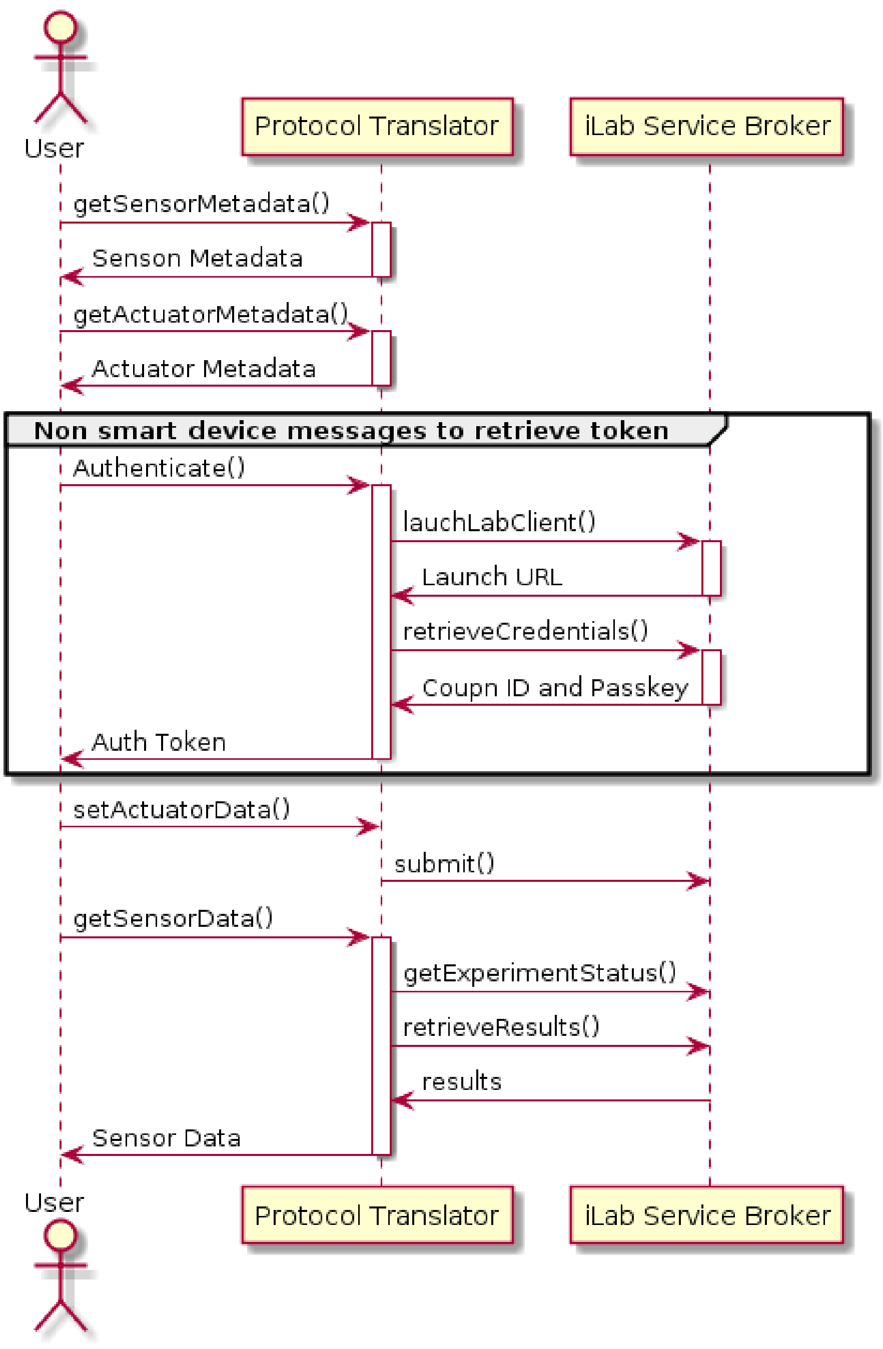


**Figure 12: Sequence diagram protocol translator ISA**

Repository:
`https://github.com/gateway4labs/protocol_translator_radioactivityLab`

For this proof of concept, the following smart device methods were implemented:

- *getSensorMetadata()*
- *getActuatorMetadata()*

- *getSensorData()*
- *setActuatorData()*

The methods *getSensorMetadata()* and *getActuatorMetadata()* do return the metadata of the smart device, which is stored in the protocol translator and contains basically static data. Therefore it does not need to contact the legacy lab to retrieve it. The smart device message schema and protocol are described in detail in deliverable D4.5. Both methods previously mentioned do not require an authentication token. The methods *getSensorData()* and *setActuatorData()* do require an authentication token to be provided along with the request and this token has to be retrieve by a separate process. Assuming the token was retrieved, the user can call the services and interact with the legacy lab. In the particular case of the iLab Shared Architecture batched labs, every request to *setActuatorData()* will cause the protocol translator to call a *submit()* method on the iLab Service Broker and an experiment execution will begin. This method returns an experiment ID to the user. By calling the *getSensorData()* method (with requires the user to provide an experiment ID), the protocol translator will query the Service Broker for the status of the experiment. If this is completed the results will be returned to the user according to the smart device specifications. If the experiment is not complete the current status will be returned. In this way, this legacy lab behaves as close as possible to a native smart device.

This prototype of the protocol translator was developed to serve as a proof of concept. It is however not realistic to assume that it will be developed to translate request to all legacy labs in the Go-Lab ecosystem, since this requires an specific implementation for each legacy system. Furthermore, in this prototype only the protocol translator was implemented. In order to be able to use this lab a client app has to be re-written. The protocol translator is an optional component, and we do recommend other more effective integration methods to share legacy labs withing Go-Lab, like implementing a gateway4labs plug-in.

***The protocol translator for the Archimedes laboratory (WebLab-Deusto)***

The archimedes laboratory is developed by the University of Deusto, using WebLab-Deusto. WebLab-Deusto uses a set of HTTP calls for pushing information and retrieving results during the session. For its conversion to the Protocol Translator, none of the client code can be reused, and an example HTML was created to replace it which displays the results from the laboratory, but does not push any information. A WebLab-Deusto communications wrapper was created, so once a reservation had been done, the reservation identifier could be used to start the communications using the Smart Device protocol to first map the existing communications as sensors and then convert each communication into a native WebLab-Deusto request.

Repository:
`https://github.com/gateway4labs/protocol_translator_weblabdeusto`

For this proof of concept, the following smart device methods were implemented:

- *getSensorMetadata()*
- *getSensorData()*

As in the previous scenario, the prototype was developed as a proof of concept, and a full implementation even for a single laboratory (like the Archimedes laboratory) would have required a different magnitude of effort than the original development. For this reason, it was discontinued and its deployment in production is using a gateway4labs plug-in.

### 3.4.7 Summary of the benefits for integrated remote laboratories

The focus of gateway4labs is attracting laboratory owners to the Go-Lab ecosystem, so they can share their labs in Go-Lab. To achieve this goal, the following incentives are provided:

1. **Easy integration:** A flexible, pragmatic approach to integrate existing remote laboratories into gateway4labs through the plug-in mechanism and the management panels. The amount of code required is not relevant, since it only acts as an initial bridge, and two interfaces (native Python API and HTTP API) are provided. Multiple deployment schemas are supported, as specified in the Architecture section of D4.5.
2. **Additional Go-Lab incentives:** Go-Lab will provide the laboratory owners with several benefits. The most important one is the visibility of the laboratories. Thousands of students and teachers will be able to easily find the federated laboratories. Other benefits include the support of Go-Lab Add-on services, such as the booking system. Certain remote laboratories might have a queue for managing students, which does not scale well. However, if the laboratory is integrated in gateway4labs and gateway4labs supports the booking mechanism of the Go-Lab portal, then the laboratory will be only available to students of groups which have booked the laboratory, reducing the amount of concurrent students.
3. **Keep control of the laboratories:** The Go-Lab project requires that the laboratories are open and no registration is needed, and the Smart Gateway will encourage developers to keep their laboratories open. However, gateway4labs provides mechanisms to enable that a remote laboratory is only available to a particular ILS. This would require those interested schools to register in the gateway4labs server used by that laboratory owner, and then the laboratory owner could provide the laboratory only to that school. The visibility of the laboratory would be consequently decreased. However, guaranteeing this type of control to the laboratory owner, it is possible that certain laboratory owners could have fewer rejections to participate in the integration, and once integrated; they can consider if they finally open their laboratories.
4. **Use a federated approach:** gateway4labs plug-ins support federation mechanisms if these are provided by the integrated systems. For ex-

ample, WebLab-Deusto provides a federation protocol so one WebLab-Deusto system with 4 laboratories can share a subset of them to other WebLab-Deusto system. The plug-in of this WebLab-Deusto benefits of this feature so it translates requests from gateway4labs (which come from the ILS platform) as if it was an external WebLab-Deusto system requesting a laboratory for a local user. This federated approach enables remote laboratories to be also provided through their original portals or be integrated in other tools, while increasing their visibility by sharing them with Go-Lab.

5. **Incentives outside Go-Lab:** gateway4labs does not only support OpenSocial, but also other specifications, such as IMS LTI, and a HTTP interface to be deployed in custom systems (such as CMS as Joomla). If a laboratory owner aims to integrate a remote laboratory in a Moodle LMS or a Joomla CMS, then gateway4labs is a tool that makes this process easy, so the laboratory owner only needs to develop the plug-in for the remote laboratory. Once this plug-in is developed, intended for supporting an LMS or a CMS, this remote laboratory is, from a technical perspective, available and compatible for the Go-Lab context.

## *3.5 Smart Gateway Plug-in Releases*

This section describes the Smart Gateway plug-ins released for well-known remote lab management systems and other legacy online lab systems. For any legacy remote lab managed by one of these RLMS the Smart Gateway will offer out of the box support for its integration in the Go-Lab infrastructure. Depending on the RLMS and the integration level desired (see D4.5 for details on the different integration levels) the support for the plug-in might require some implementation also at the RLMS or lab owner's side. In cases like this, not all versions of the RLMS will be supported.

### 3.5.1 WebLab-Deusto

WebLab-Deusto is an Open Source remote laboratory management system. It supports the development and administration of remote laboratories. In WebLab-Deusto, remote laboratories are usually managed through a priority queue. A key feature of WebLab-Deusto in this context is its federation model. A simple scenario where two WebLab-Deusto instances are using it is presented in Figure 13. On it, a user reserves a laboratory in the University A, which forwards the request to University B without requiring the user to be registered in University B. Essentially, the plug-in consists of a client of WebLab-Deusto (extracted from WebLab-Deusto, since it is also developed in Python). The client only requires a URL, a username and a password. This username and password represent a federated node, and each federated node can use it to proxy all their requests.

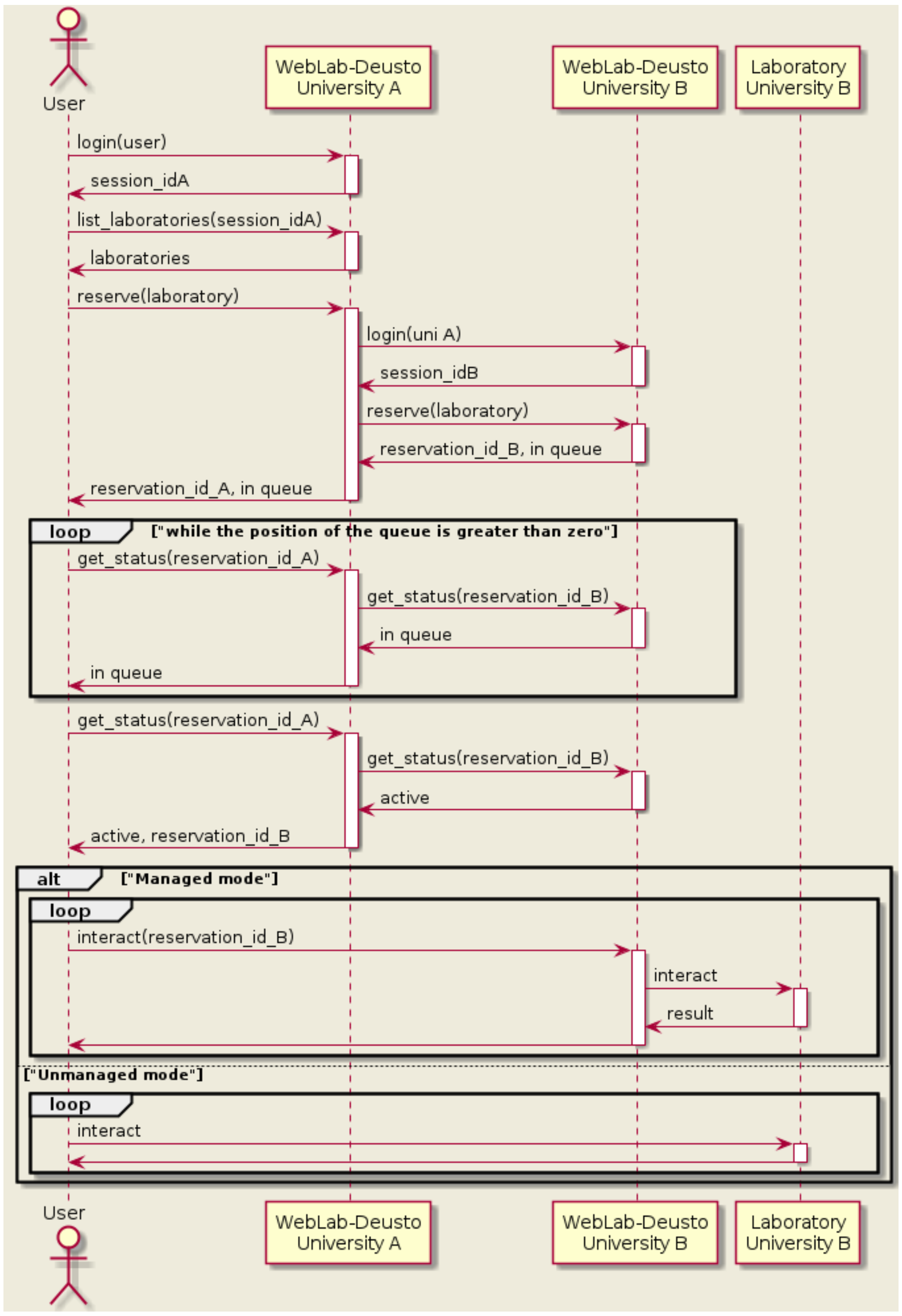
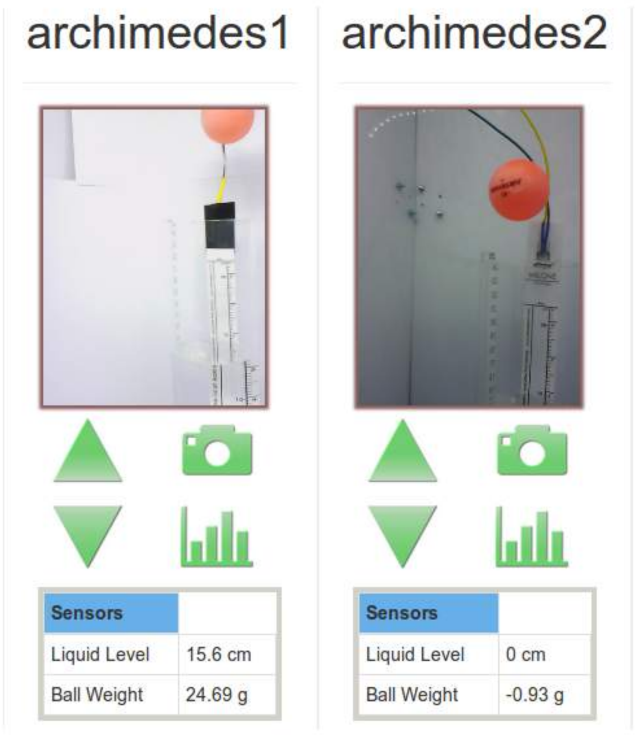

User
WebLab-Deusto University A
WebLab-Deusto University B
Laboratory University B
login(user)
session_idA
list_laboratories(session_idA)
laboratories
reserve(laboratory)
login(uni A)
session_idB
reserve(laboratory)
reservation_id_B, in queue
reservation_id_A, in queue
loop ["while the position of the queue is greater than zero"]
get_status(reservation_id_A)
get_status(reservation_id_B)
in queue
in queue
get_status(reservation_id_A)
get_status(reservation_id_B)
active
active, reservation_id_B
alt ["Managed mode"]
loop
interact(reservation_id_B)
interact
result
["Unmanaged mode"]
loop
interact

**Figure 13: Federation algorithm in WebLab-Deusto**

While WebLab-Deusto through the different institutions that compound it provides different remote laboratories for engineering studies, some of them are suitable for secondary schools. In particular in Go-Lab the Archimedes laboratory is being used. An excerpt of the Archimedes lab app can be seen in Figure 14.



| Sensors | |
|---|---|
| Liquid Level | 15.6 cm |
| Ball Weight | 24.69 g |

| Sensors | |
|---|---|
| Liquid Level | 0 cm |
| Ball Weight | -0.93 g |

**Figure 14: Two of the four balls in the Archimedes laboratory**

The Archimedes Lab can be accessed and tested directly via the preview lab app page of the Smart Gateway:

- `http://gateway.golabz.eu/public/labs/public/widgets/archimedes/`

### 3.5.2 iLab Shared Architecture (ISA)

The iLab Shared Architecture (ISA) is a distributed software architecture developed at the Massachusetts Institute of Technology (MIT) that offers online lab developers and users a common framework to use and share Online Labs (Harward et al., 2008). It is an Remote Laboratory Management System that separates the experiment logic from the management part like managing users' accounts, user authentication and other tasks that follow a lab session. It is a middleware architecture with a service broker providing common shared services for lab servers and lab clients.

According to the specifications defined in deliverable D4.5 a full version of the plug-in for the iLab shared architecture was developed (see levels of integration in D4.5). Since ISA requires a user to be uniquely identified within a service broker to run an experiment, the developed plug-in bridges the legacy system authentication. The interaction of the plug-in with the legacy lab system (iLab service broker) is described in Figure 15. In this diagram it was skipped the interactions that take place before between user, ILS Platform and Smart Gateway since it was described in detail in deliverable D4.5.

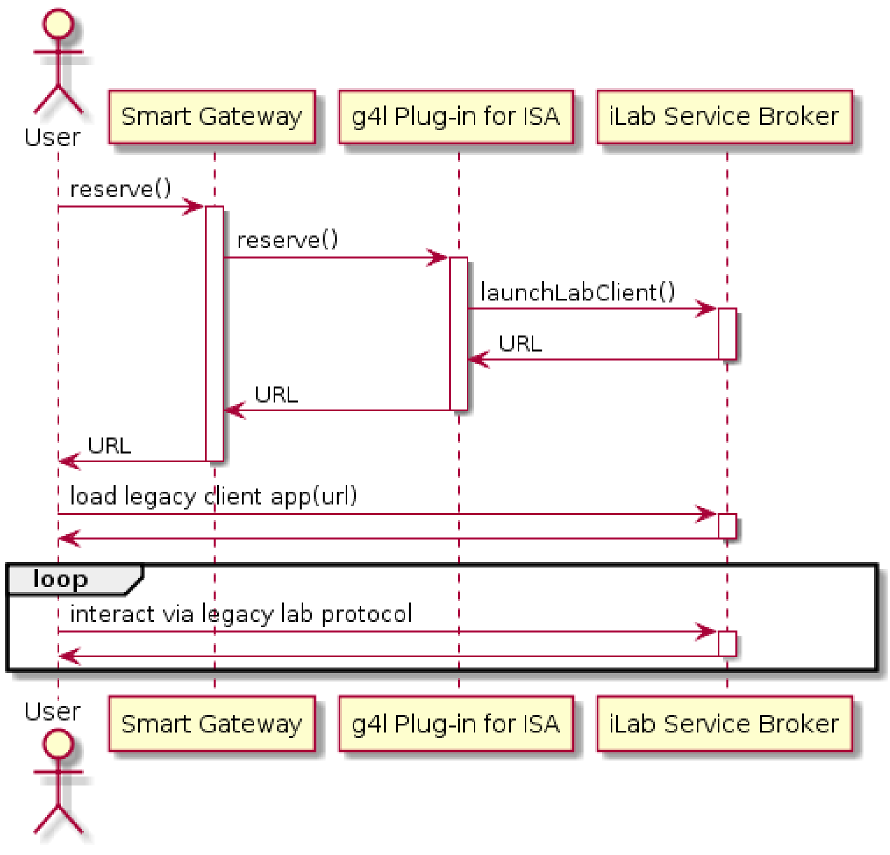


**Figure 15: ISA plug-in sequence diagram**

This diagram assumes that the lab gadget was successfully loaded and that Smart Gateway and iLab Service Broker administrators exchanged the necessary credentials and that Smart Gateway and iLab RMLS are registered respectively. ISA exposes its functionalities to external clients and lab servers via a SOAP Web services API. When a request to reserve a lab is started by the user, the plug-in contacts the iLab service broker by calling a Web method launchLabClient(). The parameters provided are described in Table 1.

This service will return a URL that will launch the particular lab client requested. At the user's side the lab client will be launched in the OpenSocial container. The interaction that follows between lab client and server is carried out using the legacy lab's.

As previously mentioned, the plug-in for ISA is available by the Smart Gateway as an out of the box support. It can be deployed with ISA versions 4.3.1 or higher. Earlier versions of ISA will have to be updated in order to support the integration with the Go-Lab ecosystem.

**Table 1: Description of the launchLabClient service**

| Name | Type | Description |
|---|---|---|
| clientGuid | string | Unique ID of the client to be launched |
| groupName | string | Name of the group it belongs to |
| userName | string | User requesting access to it |
| authorityKey | string | Unique ID of the Smart Gateway (assigned by service broker). This credential should be exchanged beforehand with system administrators |
| start | string | Reservation starting time |
| duration | long | Duration of reservation in seconds |

An example of a lab available to Go-Lab via the ISA plug-in can be accessed via the URL below:

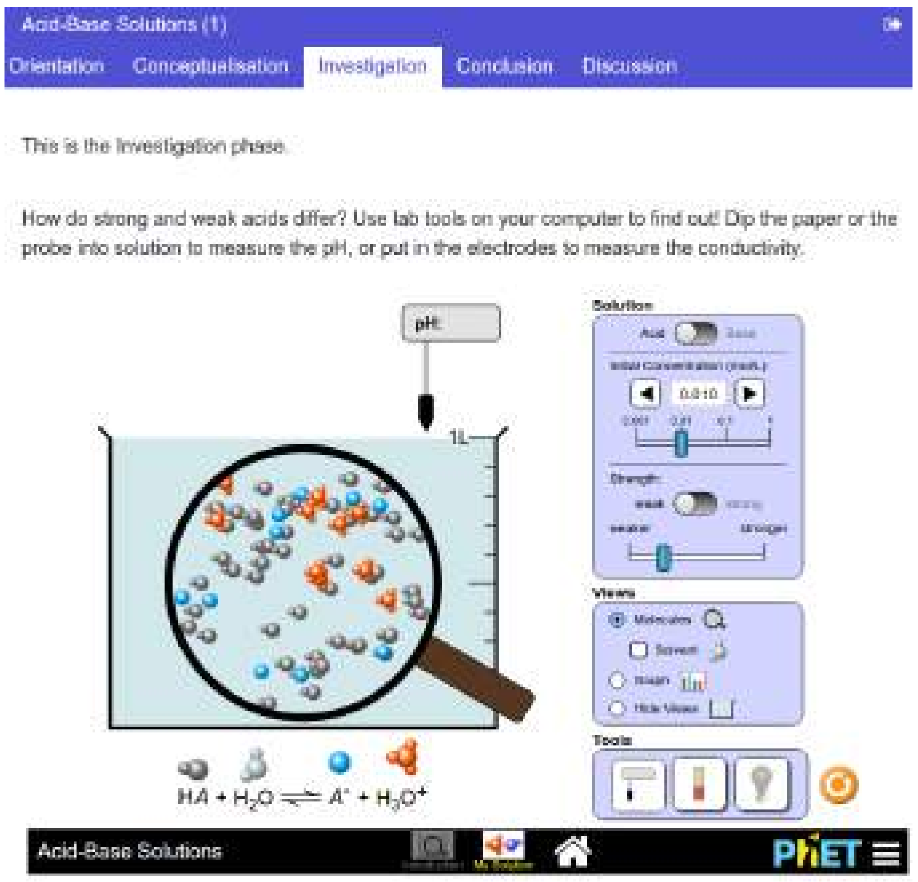
- `http://gateway.golabz.eu/public/labs/public/widgets/radiolab1/`

### 3.5.3 PhET

The PhET Project (Physics Education Technology) created useful simulations for teaching and learning physics and makes them freely available from the PhET website (http://phet.colorado.edu). The simulations (sims) are animated, interactive, and game-like environments in which students learn through exploration (Perkins et al., 2006). Many of the simulations cover introductory high school and college physics, while others introduce more advanced topics, e.g., lasers, semiconductors, greenhouse effect, radioactivity, nuclear weapons, and Fourier analysis. Users, however, have included students from grade school through graduate school. On the website, the sims are organized under nine loose categories: Motion; Work, Energy Power; Sound Waves; Heat Thermo; Electricity Circuits; Light Radiation; Quantum Phenomena; Chemistry; Math Tools; and Cutting Edge Research (Perkins et al., 2006).

PhET laboratories are already included in the Go-Lab repository and can be used by teachers to be embedded into an ILS. Example of the Acid-base Solutions lab: `http://www.golabz.eu/lab/acid-base-solutions`. Figure 16 shows a PhET simulation accessed from an ILS.

Internally, the Smart Gateway regularly downloads the list of simulations from PhET in each language, and generates an OpenSocial app that embeds the laboratory as an iFrame. This way, if a new simulation is created in PhET, it is automatically available in the Smart Gateway, and therefore it can be added to the Go-Lab repository. Furthermore, if a new translation of a simulation is available at PhET (e.g., one simulation available in the Go-Lab repository is suddenly provided in Spanish or Dutch), it will automatically be available to those students accessing in that language without requiring anybody to perform any change.



This is the Investigation phase.

How do strong and weak acids differ? Use lab tools on your computer to find out! Dip the paper or the probe into solution to measure the pH, or put in the electrodes to measure the conductivity.

**Figure 16: PhET app in an ILS**

Preview of a PhET lab (Acid-base Solutions lab) via Smart Gateway:

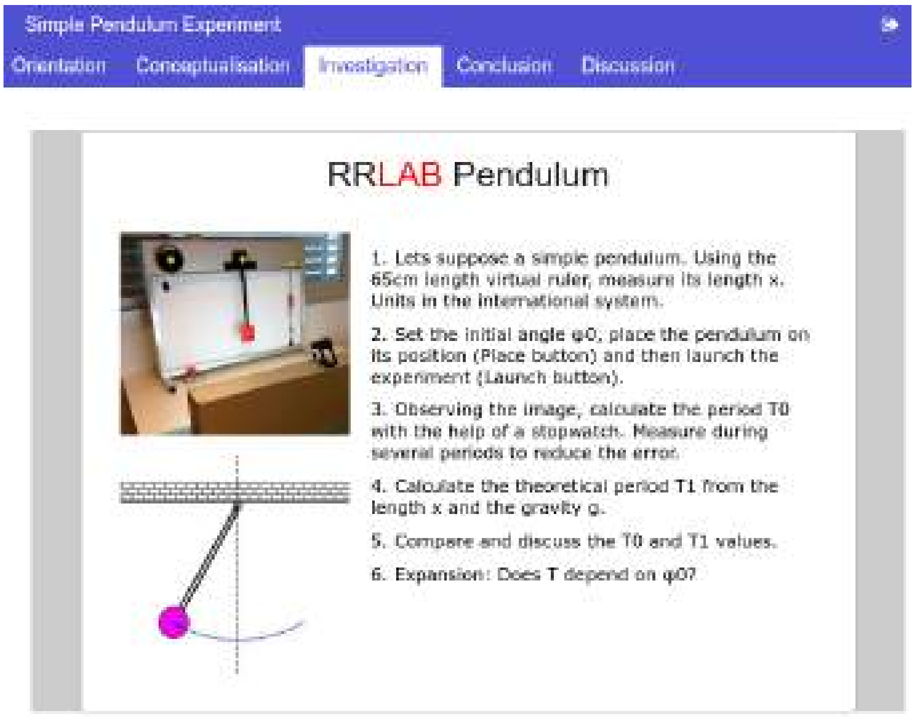
- `http://gateway.golabz.eu/public/labs/public/widgets/acid-base_solutions/`

### 3.5.4 ViSH

The ViSH project[5] is part of the FP7 Global Excursion Project[6], which provides a portal that lists, organizes and displays a wide set of open educational resources including simulations and remote laboratories among others. All the contents must be open and they are publicly available, and HTTP APIs are provided for searching resources. The resources can be listed alone, but they are commonly organized in "Excursions". An Excursion is similar to a slideshow, but where each slide is a resource. Resources include embedded pages (e.g., wikipedia), tests, Adobe Flash objects, laboratories and other rich contents.

[5]http://vishub.org/
[6]http://www.globalexcursion-project.eu/

The plug-in developed to support ViSH laboratories acts as a federation proxy. It uses the search API (which is a JSON HTTP interface) to enable the Smart Gateway administrator searching in the ViSH repository. This way, the gateway4labs plug-in enables the Smart Gateway administrator to search for resources, and if it is an excursion, it splits the excursion into multiple apps that can be exported through gateway4labs to the ILS. Figure 17 shows a slide of an existing excursion[7] displayed in a ILS.



RRLAB Pendulum

1. Lets suppose a simple pendulum. Using the 65cm length virtual ruler, measure its length x. Units in the international system.

2. Set the initial angle φ0, place the pendulum on its position (Place button) and then launch the experiment (Launch button).

3. Observing the image, calculate the period T0 with the help of a stopwatch. Measure during several periods to reduce the error.

4. Calculate the theoretical period T1 from the length x and the gravity g.

5. Compare and discuss the T0 and T1 values.

6. Expansion: Does T depend on φ0?

**Figure 17: A ViSH resource included in an ILS**

Preview of a ViSH lab (Pendulum Experiment) via Smart Gateway:

- `http://gateway.golabz.eu/public/systems/public/system/vishub/widgets/Excursion%25253A101%252540vishub.org/`

### 3.5.5 UNR-FCEIA

The National University of Rosario (Argentina) have developed a physics remote laboratory[8]. On it, students can create electronic circuits and test currents. The remote laboratory is a pure HTML5 application that does not require any external plug-in and internally it has an internal queue so if multiple students attempt to use the laboratory at the same time, they will be multiplexed in time. A mechanism for its inclusion in external systems is developed in the system itself, relying on a cryptography mechanism. Figure 18 shows an example of an experiment with a diode in an ILS.

[7] http://vishub.org/excursions/162.full
[8] http://labremf4a.fceia.unr.edu.ar/

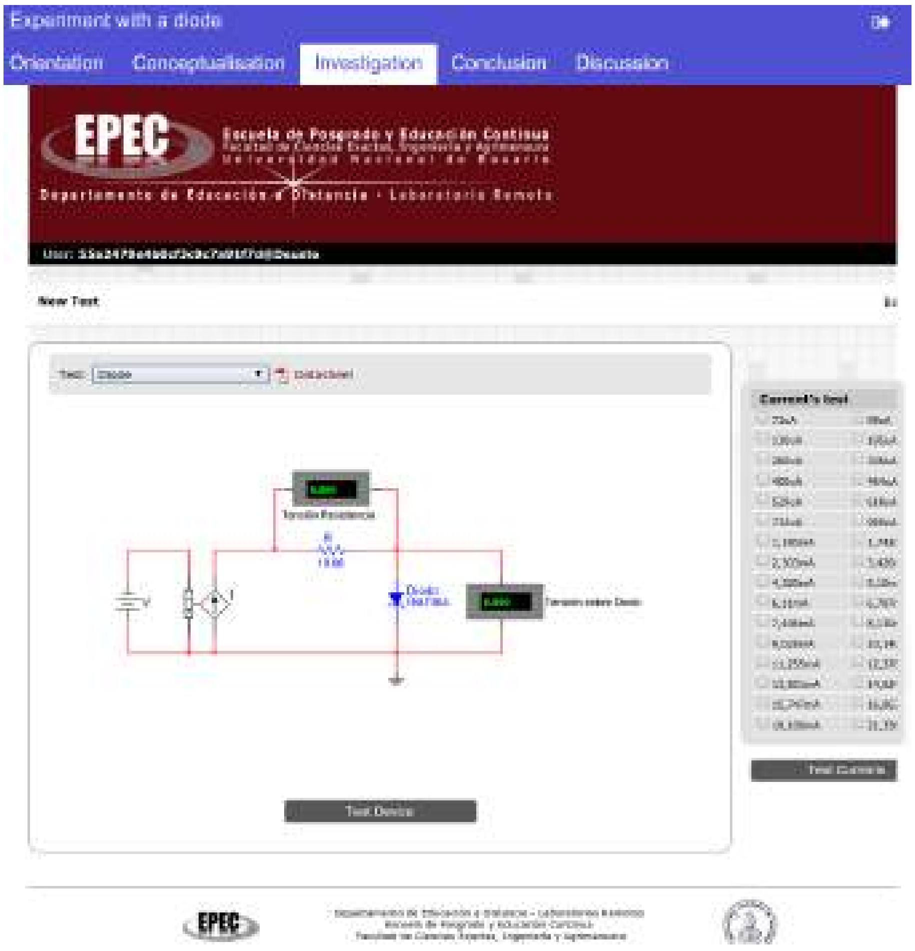


Figure 18: UNR-FCEIA lab in an ILS

The UNR-FCEIA plug-in takes advantage of this cryptography mechanism. Basically, the plug-in uses a shared secret stored in the gateway4labs database to sign a message that provides details such as a timestamp or who is accessing. Then, the user will be redirected to the remote laboratory with that message. The remote laboratory will take this message and verify that it has been signed correctly by a valid actor. This way, the plug-in never contacts the remote laboratory directly.

Preview of UNR-FCEIA Electronics lab via Smart Gateway:

- `http://gateway.golabz.eu/public/labs/public/widgets/unr/`

### 3.5.6 Remlabnet

The Remote Laboratory Management System REMLABNET delivers remote experiments for freshmen and secondary school students. It was developed in the scope of the School Experimental System (ISES) remote experiments. The RLMS is built using new components, designed for the purpose, as Measureserver, web space management, data warehouse, communication board of RLMS, etc. The communication server provides, beside connection and diagnostics services also services for the teacher's comfort as white board, IP telephony, simulation inclusion, test management and reservation management (Schauer et al., 2014). Figure 19 shows a Remlabnet app included in a Go-Lab ILS.

The Remlabnet plug-in for the Smart Gateway was implemented using the HTTP plug-in interface as described in section 3.4.3.

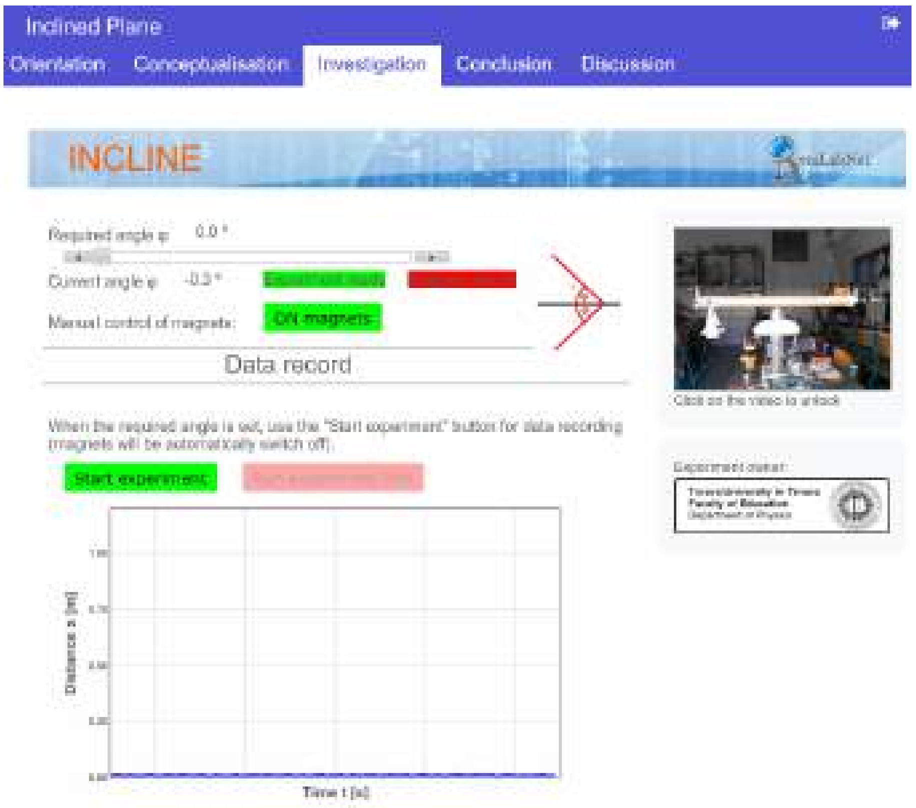


**Figure 19: Remlabnet experiment with inclined plane in an ILS**

Preview of Remlabnet Inclined Plane lab via Smart Gateway:

- `http://gateway.golabz.eu/public/systems/public/system/remlabnet/widgets/11/`

# 4 Conclusion

This deliverable is the last one in the series of Lab Owner and Cloud Services. It is the companion of D4.5, and the last update of D4.3. We presented two different types of frameworks destined for lab owners to enable them to create new labs or adapt existing ones in accordance to the Go-Lab platform requirements. Those are the ‘Releases of the Lab Owner and Cloud Services’.

Templates for the development and deployment of new remote laboratories using the Smart Device Paradigm have been generalized as opposed to the example based approach in D4.3. A number of new lab examples were added as well to the repository. The Smart Gateway includes new features (such as support for internationalisation by teachers through the App Composer, a caching mechanism or a scheduler). Additionally, support for new plug-ins corresponding to other existing online laboratories (such as concord, QuVis or Remlabnet) is implemented. Draft versions of the Protocol Translator have been implemented to keep compatibility with the Smart Device, but due to the required effort on implementing them, the use of ad-hoc plug-ins is adopted for those systems.

The impact of this work is not limited to the enrichment of the Go-Lab ecosystem. People involved in this task force are taking part of the IEEE P1876 Working Group on Networked Smart Learning Objects for Online Laboratories. The last happening of this working group was in June 2015 as part of the Exp.at’15 conference held at the Azores. Work presented in this document is also part of the proceedings of this conference. The REV conferences are also venues for exposing our work.

Collaboration with external lab owners has been a part of this work as well, in order to better enrich our repositories. Several visits to EPFL have been organised for collaborators from Remblabnet of the Czech Republic, Unilabs of Spain, and Linkare TI of Portugal. The collaboration is still ongoing, updates and additions are happening periodically.

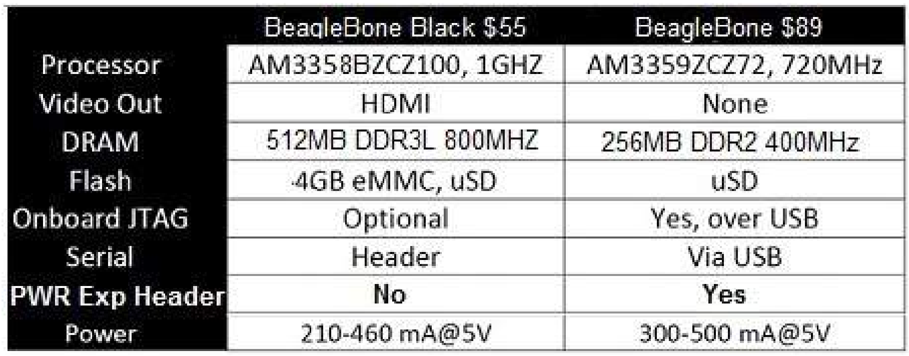
Moreover, React of EPFL got a Swiss National Fund (SNF) under the program SCOPES[1] for 3 years. This program provides one Swiss institution and eastern European institutions with funds for research-based cooperation. The Swiss institution is meant to provide the money and technology for its eastern European colleagues. In this framework, React of EPFL is cooperating with the University of Trnava, the University of Belgrade, and the University of Kragujevac in order to connect new remote labs or adjust existing ones, and support their integration in the Go-Lab infrastructure. React of EPFL is transferring the know-how to the eastern European universities as per the solutions devised in D4.5 and D4.7.

[1] http://www.snf.ch/en/funding/programmes/scopes/Pages/default.aspx

# 5 Appendix A

## 5.1 BBB Description

The BeagleBone Black is the newest member of the BeagleBoard family. It is a lower-cost, high-expansion focused BeagleBoard using a low cost Sitara XAM3359AZCZ100 Cortex A8 ARM processor from Texas Instruments. It is similar to the Beaglebone,but with some features removed and some features added. The table below gives the high points on the differences between the BeagleBone and BeagleBone Black:



| | BeagleBone Black $55 | BeagleBone $89 |
|---|---|---|
| Processor | AM3358BZCZ100, 1GHZ | AM3359ZCZ72, 720MHz |
| Video Out | HDMI | None |
| DRAM | 512MB DDR3L 800MHZ | 256MB DDR2 400MHz |
| Flash | 4GB eMMC, uSD | uSD |
| Onboard JTAG | Optional | Yes, over USB |
| Serial | Header | Via USB |
| PWR Exp Header | No | Yes |
| Power | 210-460 mA@5V | 300-500 mA@5V |

**Figure 20: Comparison between BBB and older BB**

In the box containing equipment has (1)BeagleBone Black board, (1)USB cable, and (1)catalogue that should be read.

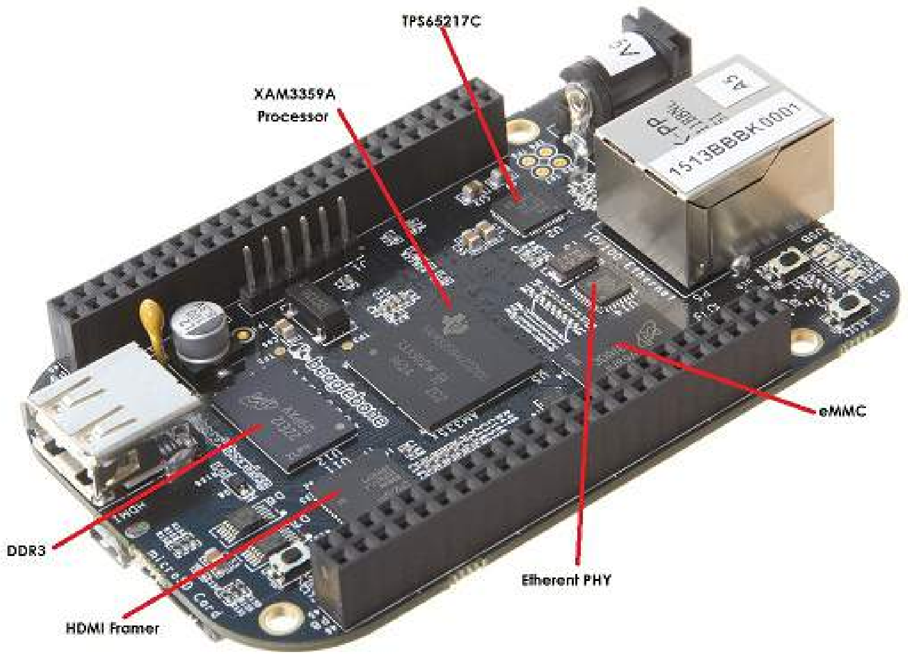


**Figure 21: BeagleBone Black Key Component Locations (Rev A5A)**

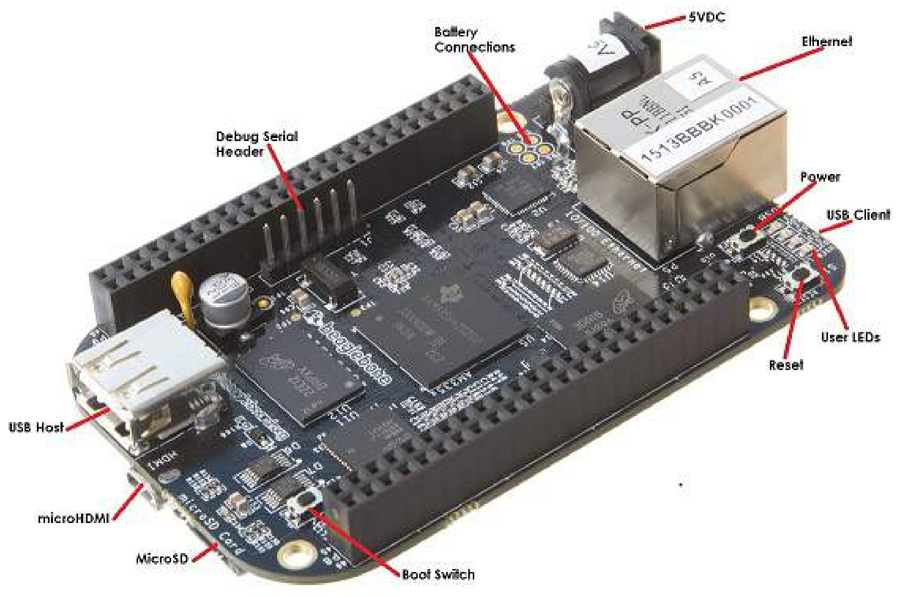


**Figure 22: BeagleBone Black Connector and Switch Locations (Rev A5A)**

For more info on what can be added to the board as accessories, you can refer to: `http://elinux.org/Beagleboard:BeagleBone_Black_Accessories`

There is no JTAG over USB support on the BeagleBone Black. JTAG is an uninstalled option as compared to BeagleBone board. To install the JTAG header, P2 needs to be installed on the back of the board. P2 is a Samtec FTR-110-03-G-D-06 connector and can be purchased from Samtec or any of their distributors.

WiFi Adapters for the BBB are listed on the table below, with compatibility requirements. It is highly recommended that DC power is used when running these dongles due to the current requirements of the dongles. It might be needed to use an extension cable to move the dongles away from the planes of the PCB. Sometimes standoffs will work. There have been instances where when placed in a metal case, there can be WiFi issues as well[1].

| **Dongle** | **Works On** |
|---|---|
| ASUS USB-N13 802.11 b/g/n | Angstrom |
| EDIMAX EW-7811UN | Angstrom |
| D-Link DWA-125 | Debian LXDE, Debian Gnome Desktop |
| D-Link DWA-121 | Angstrom Debian LXDE, Debian Gnome Desktop |
| Belkin N150 | Debian LXDE |
| TP-Link TL-WN727N | Debian LXDE, Debian Gnome Desktop |
| Netgear WNA1100 | Debian LXDE, Debian Gnome Desktop |
| Keebox W150NU | Debian LXDE, Debian Gnome Desktop |

## 5.2 myRIO Description

NI myRIO is a hardware/software platform ‘complete’ with the latest Zynq integrated system-on-a-chip (SoC) technology from Xilinx. It boasts a dual-core ARM Cortex-A9 processor and an FPGA with 28,000 programmable logic cells, 10 analog inputs, 6 analog outputs, audio I/O channels, and up to 40 lines of digital input/output (DIO). Designed and priced for the academic user, NI myRIO also includes onboard WiFi, a three-axis accelerometer, and several programmable LEDs in a durable, enclosed form factor (NI, 2014).

[1] http://elinux.org/Beagleboard:BeagleBoneBlack

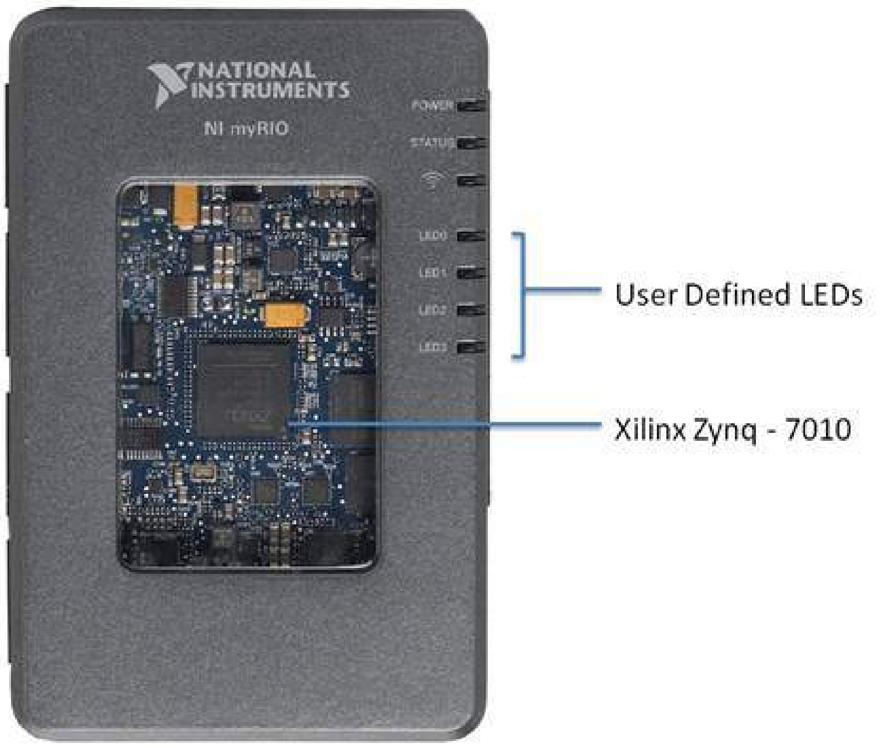


**Figure 23: NI myRIO Front View**

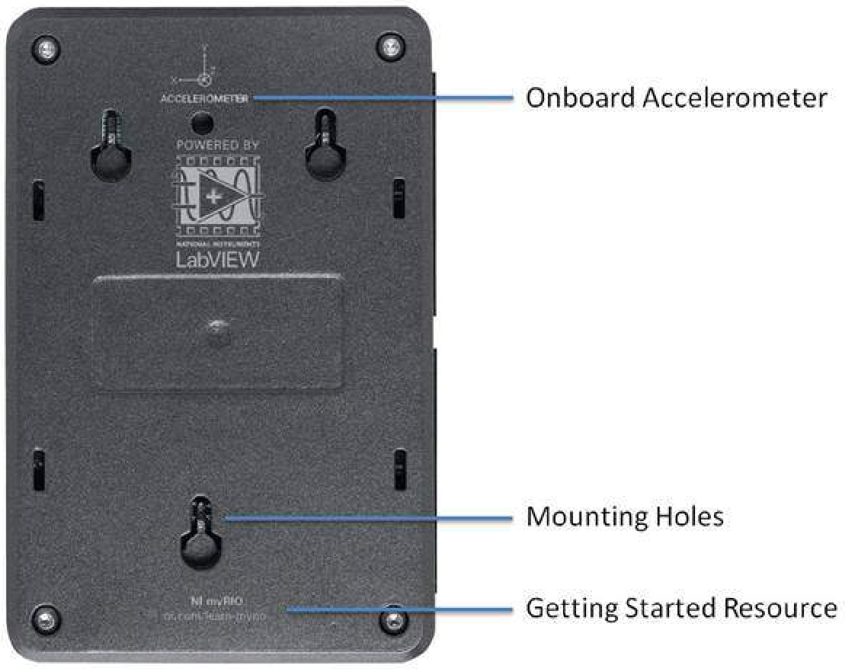


**Figure 24: NI myRIO Back View**

The default I/O configuration is shown. It is customizable with the NI LabVIEW FPGA Module. These are 0.1" pitch dual-row 34-position (17 x 2) IDC connectors.

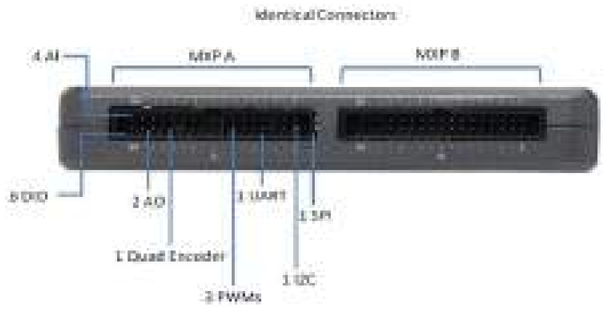


**Figure 25: NI myRIO Expansion Port (MXP) Connectors**

The default I/O configuration is shown. It is customizable with LabVIEW FPGA.

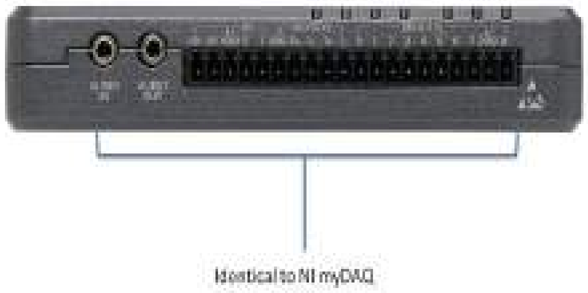


**Figure 26: NI myRIO NI miniSystems Port (MSP) Connector**

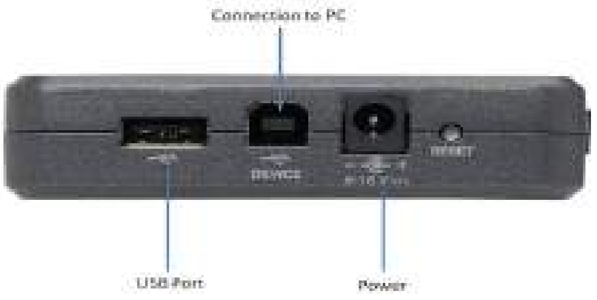


**Figure 27: NI myRIO Top View**

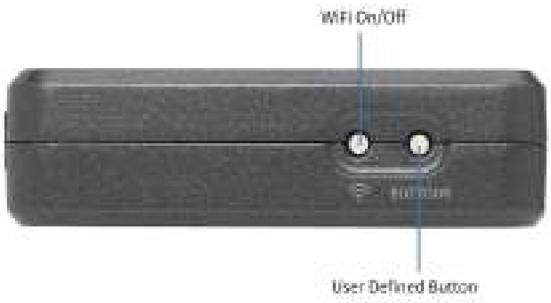


**Figure 28: NI myRIO Bottom View**

# 6 Appendix B

## 6.1 Queued Message Handler Design Pattern

The LabVIEW template presented in this deliverable, relies on the Queued Message Handler (QMH) design pattern. QMH is a version of the 'producer/consumer' design topolgy known to LabVIEW. The 'producer' and the 'consumer' are structures which respectively generate and consume messages. The 'consumer' in turn can also generate messages. A message in this context is an instruction with the data needed for its execution.
QMH uses queues to pass data between independent loops, allowing them to run at independent rates since there is no data dependency among them. Each queue can dispose of one consumer, but many producers, including the consumer. Figure 29 below illustrates the above:

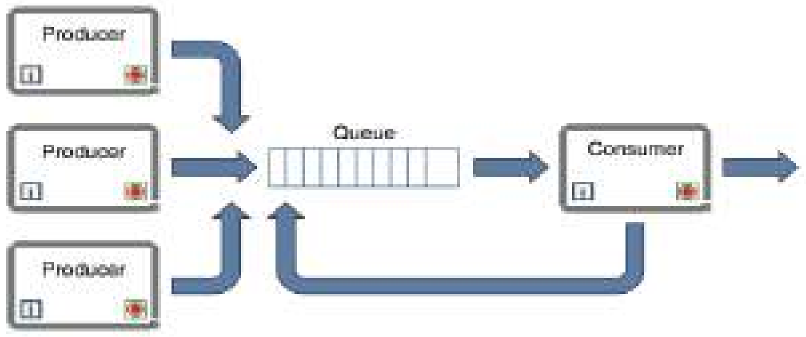


**Figure 29: A QMH Queue**

On a lower level as presented in Figure 30, the Queued Message Handler consists of an event handling loop and at least one message handling loop:

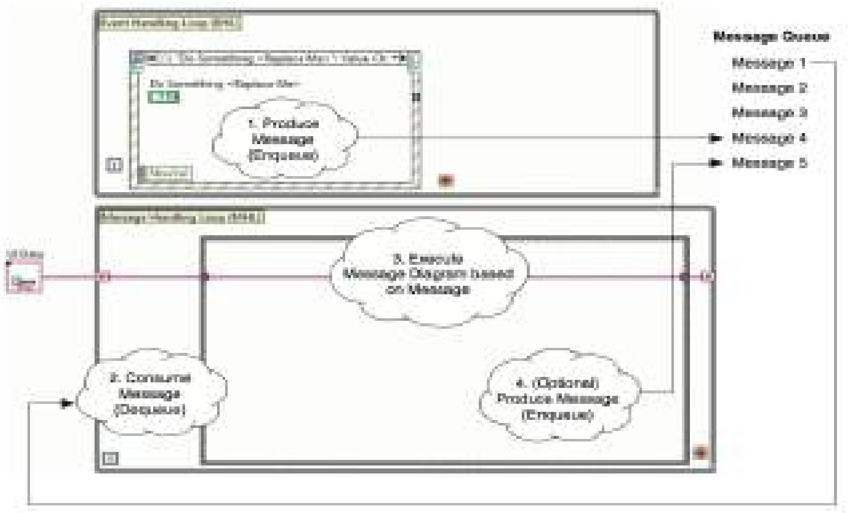

Event Handling Loop (EHL)
Message Handling Loop (MHL)
Message Queue
Message 1
Message 2
Message 3
Message 4
Message 5
1. Produce Message (Enqueue)
2. Consume Message (Dequeue)
3. Execute Message Diagram based on Message
4. (Optional) Produce Message (Enqueue)

**Figure 30: Queued Message Handler**

# 7 Appendix C

## 7.1 Recommendation #1: Running Servers on Ports 80 or 443

Educational institutions and public networks tend to usually block ports different than 80 for http, and 443 for https connections. And hence, in order to avoid that labs don't get used because of the blocked ports, it is recommended to adopt the default 80 for http, and 443 for https connections (Fielding et al., 1999).

While it might seem that this is an easy setting to configure, this is not true. Ports 80 and 443 are system ports, meaning that only processes started with the root (system) user are allowed to run on these ports. The common solution developers use is starting the processes with 'sudo', granting the process system rights; but it is not a good practice for security reasons (Hardiman, 2013).

A good practice is to use the 'ipfw'[1]. With ipwf, the programmer writes connection forwarding rules using a format dictated by ipfw. This snippet of code is an example for forwarding all calls comming on port 80 of the machine, to port 8080 (where the server's process is running):

```
add 02000 fwd 127.0.0.1,8080 tcp from any to me 80
```

## 7.2 Recommendation #2: Methods for Video Streaming

In the templates of this deliverable, 2 methods for video streaming were used with 2 different types of cameras:

1. Refreshed feed of still images at a fast rate with an IP camera
2. Video stream with the WebSocket protocol with a USB webcam

We will briefly present each of the modalities, and identify the pros and cons of choosing one over the other.

### 7.2.1 Video Streaming with IP cameras

This method is of the easiest and most trivial for streaming video and embedding it in html. The IP camera used in example 2.4.5 periodically sends still images to the cloud. The location of the images is known, which permits the developer to retrieve the images and refreshing the source at a fast rate, creating the illusion of a video feed. This code snippet illustrates this example:

**Listing 7.1: Code for Embedding Refresh Still Images in html**

```
...
<img id="img" src="http://128.178.5.183/image.jpg?cidx=2005906637"
    border="0">
<script>
window.onload = function() {
```

[1] https://www.freebsd.org/doc/handbook/firewalls-ipfw.html

```
    setInterval(updateImage, 500);
}
var image = document.getElementById("img");
function updateImage() {
    image.src = image.src.split("?")[0] + "?" + new Date().getTime();
}
</script>
...
```

With this method, the video stream has a high-resolution and isn't prone to delays caused by the limitations of the hardware (the Smart Device board). This is because the source is simply embedded in the html page as an image, and some JavaScript code takes care of refreshing it.
But using an IP camera may pose problems to lab owners since a unique IP is needed for the camera. And hence, for each lab not only one IP is needed, but two (for the Smart Device board and the camera).

### 7.2.2 Video Streaming with USB cameras

This modality follows the Smart Device Specifications by treating the camera as a sensor, and using the WebSocket protocol to transmit the video feed. In this case, binary WebSockets need to be used, which are not as trivial to handle as text WebSockets.

Moreover, the communication and computation limitations of the Smart Device board might impose some difficulties on the developer to ensure good communication with the user client. One of the causes is the limited bandwidth at which a Smart Device board can transmit data, and hence compromising either the video feed or other services if the configuration of the video is not handled properly. Another cause is the computation capabilities of the board for processing the video feed before transmission, which might take longer than the time acceptable for a real-time streaming.

For the Node.js template, binaryjs[2,3] is one option for doing it. For the LabVIEW template, a library has been written in order to handle the video streaming with binary WebSockets. The corresponding code can be found in the directory: 'services/Video' of the LabVIEW template in section 2.3.1.

[2]http://binaryjs.com/

[3]http://www.olindata.com/blog/2014/01/file-uploading-and-streaming-binaryjs

# 8 Appendix D

## 8.1 Brief history of gateway4labs

In (Orduna et al., 2012), an integration of the WebLab-Deusto Remote Laboratory Management System (RLMS) in a Learning Management System (Moodle) and a Content Management System (Joomla) is described by University of Deusto and UNED. The key concept was that a federation protocol was used to perform this integration: WebLab-Deusto did not manage the authentication or authorization of the individual students (managed by the Moodle and Joomla administrators), but only the connection with the two tools. Both plug-ins for Moodle and Joomla were developed, so they could connect to WebLab-Deusto using its federation protocol, and they provided management layers (e.g., which course could access what laboratory).

This concept, developed in September 2011, could not scale to other Remote Laboratory Management Systems: while the concept could be applied, supporting 3 LMS in 3 RLMS would require 9 plug-ins (3x3), since the plug-in for Moodle for WebLab-Deusto would not work for the iLab Share Architecture. Furthermore, each of these plug-ins dealt with their own management panels and database tables.

For this reason, in May 2012 a pet project called lms4labs started being developed between University of Deusto and UNED, with no associated funded project. Its focus was to avoid the problems presented in the previous solution, by putting in the middle a core component, called LabManager. This component supported an HTTP interface designed to be particularly small and for being consumed by LMS/CMS/PLE. It also supported a plug-in mechanism for supporting more than one RLMS. On June 2012, a first demo was available of a single LMS (Moodle) using a single RLMS (WebLab-Deusto, and all its remote laboratories). Now supporting 3 LMS and 3 RLMS would require only 6 plug-ins (3+3). Additionally the size of each plug-in is considerably smaller, since all the management was already provided by this core component. And if RLMS A developed a plug-in for LMS A, then RLMS B would benefit from that plug-in, too.

MIT (developers of the iLab Remote Laboratory Management System) was interested in this approach, so they started supporting the project by providing two developers between October 2012 and February 2013, adding support for IMS LTI. This standard is supported by many LMS systems, including Moodle (since version 2.2), so with this contribution, every RLMS would automatically support all those LMS, in addition to those supported through plug-ins. A plug-in was developed for Joomla. The results of this pet project were presented in (Orduna et al., 2013).

Within the context of Go-Lab, lms4labs was chosen to act as an initial codebase for the Smart Gateway, since it provided a plug-in mechanism for integrating remote laboratories. As such, it was renamed to gateway4labs[1]. This way,

[1] http://gateway4labs.readthedocs.org/

the project grew, with new developers from other Go-Lab partners, and major changes in the architecture for supporting OpenSocial (all the plug-ins had to be changed), with more management layers, supporting public (with no authentication) laboratories for constraints of the project and becoming more robust and scalable. Most of its current features have been implemented for supporting Go-Lab. Additionally, new laboratories were supported (iLab, RemLabNet, Concord, PhET, QuVis, ViSH, UNR FCEIA), compared to the initial WebLab-Deusto.

## *References*

Belter, J., Piper, P., & Durkin, C. (2014). *myrio control and telemetry system for formula-hybrid racecar.* https://decibel.ni.com/content/docs/DOC-37203. (Accessed: 2015-07-27)

Fielding, R., Irvine, U., Gettys, J., Mogul, J., Frystyk, F., Masinter, L., ... Berners-Lee, T. (1999). *Hypertext transfer protocol – http/1.1.* http://www.rfc-editor.org/rfc/rfc2616.txt. (Accessed: 2015-07-29)

Frobinson, J. (2014). *Myrio fpga audio pitch changer.* https://decibel.ni.com/content/docs/DOC-38990. (Accessed: 2015-07-27)

Hardiman, N. (2013). *Do you sudo? learn the basics.* http://www.techrepublic.com/blog/linux-and-open-source/do-you-sudo-learn-the-basics/. (Accessed: 2015-07-29)

Harward, V., del Alamo, J., Lerman, S., Bailey, P., Carpenter, J., DeLong, K., ... Zych, D. (2008, June). The iLab shared architecture: A web services infrastructure to build communities of internet accessible laboratories. *Proceedings of the IEEE*, *96*(6), 931–950. Retrieved 2015-01-06, from http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4527087 doi: 10.1109/JPROC.2008.921607

Hughes, D. (2011). *Real-time fpga pitch shifterr.* http://www.eeweb.com/project/dylan_hughes/real-time-fpga-pitch-shifter. (Accessed: 2015-07-27)

Karve, J., & Worman, T. (2014). *Internet speedometer.* http://makezine.com/projects/internet-speedometer/. (Accessed: 2015-07-27)

Kridner, J. (2015). *ps1rfid.* http://beagleboard.org/project/ps1rfid/. (Accessed: 2015-07-27)

Molloy, D. (2014). *The beaglebone and its application in engineering education.* http://www.ti.com/lit/ml/ssqw081/ssqw081.pdf. (Accessed: 2015-07-27)

NI. (2014). *Ni myrio hardware at a glance.* http://www.ni.com/product-documentation/14604/en/. (Accessed: 2015-07-21)

NI. (2015a). *Labview system design software.* http://www.ni.com/labview/. (Accessed: 2015-07-19)

NI. (2015b). *Ni labview for higher education (university/college).* http://www.ni.com/labview/applications/academic/. (Accessed: 2015-07-19)

Normal, D., & Kridner, J. (2014). *Dirty dish detector.* http://makezine.com/projects/dirty-dish-detector/. (Accessed: 2015-07-27)

Orduna, P., Botero Uribe, S., Hock Isaza, N., Sancristobal, E., Emaldi, M., Pesquera Martin, A., ... Garcia-Zubia, J. (2013, October). Generic integration of remote laboratories in learning and content management systems through federation protocols. In (pp. 1372–1378). IEEE. Retrieved 2015-07-12, from http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6685057 doi: 10.1109/FIE.2013.6685057

Orduna, P., Sancristobal, E., Emaldi, M., Castro, M., Lopez-de Ipina, D., & Garcia-Zubia, J. (2012, October). Modelling remote laboratories integrations in e-learning tools through remote laboratories federation protocols. In (pp. 1–6). IEEE. Retrieved 2015-07-12, from http://ieeexplore.ieee

.org/lpdocs/epic03/wrapper.htm?arnumber=6462220 doi: 10.1109/FIE.2012.6462220

Parvizi, B. (2013). *Beaglebone black and pulse width modulation (pwm), controlling a servo using html5, javascript, and node.js.* http://digital-drive.com/?p=146. (Accessed: 2015-07-15)

Perkins, K., Adams, W., Dubson, M., Finkelstein, N., Reid, S., Wieman, C., & LeMaster, R. (2006). PhET: Interactive Simulations for Teaching and Learning Physics. *The Physics Teacher*, *44*(1), 18. Retrieved 2014-07-03, from http://scitation.aip.org/content/aapt/journal/tpt/44/1/10.1119/1.2150754 doi: 10.1119/1.2150754

Petru, T. (2014). *The myexplorer remote controlled vehicle.* https://decibel.ni.com/content/docs/DOC-35956. (Accessed: 2015-07-27)

Rick. (2015). *Podtique.* http://beagleboard.org/project/podtique/. (Accessed: 2015-07-27)

Salzmann, C., Govaerts, S., Halimi, W., & Gillet, D. (2015). The smart device specification for remote labs. In *Remote engineering and virtual instrumentation (rev), 2015 12th international conference on* (pp. 199–208).

Schauer, F., Krbecek, M., Beno, P., Gerza, M., Palka, L., & Spilakova, P. (2014, February). REMLABNET - open remote laboratory management system for e-experiments. In (pp. 268–273). IEEE. Retrieved 2015-07-12, from http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6784273 doi: 10.1109/REV.2014.6784273