IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 3, 2019, accepted October 22, 2019, date of publication November 7, 2019, date of current version November 21, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2952321

Improving the Scalability and Replicability of Embedded Systems Remote Laboratories Through a Cost-Effective Architecture

AITOR VILLAR-MARTÍNEZ[1,2], LUIS RODRÍGUEZ-GIL[2], IGNACIO ANGULO[1], PABLO ORDUÑA[2,3], (Member, IEEE), JAVIER GARCÍA-ZUBÍA[1], (Senior Member, IEEE), AND DIEGO LÓPEZ-DE-IPIÑA[3], (Member, IEEE)

[1]Faculty of Engineering, University of Deusto, 48007 Bilbao, Spain
[2]LabsLand, 48002 Bilbao, Spain
[3]DeustoTech-Deusto Foundation, 48007 Bilbao, Spain

Corresponding author: Aitor Villar-Martínez (aitor.v@deusto.es)

ABSTRACT Online remote laboratories are a particularly promising tool for effective STEM education. They offer online universal access to different hardware devices in which students can experiment and can test and improve their knowledge. However, most of them have two significant limitations. First, given that most of them are developed as, or evolve from single-user proofs of concept, they have no scalability provisions other than full laboratory replication. And second, when this is done, cost efficiency is often neglected. This paper presents the requirements for the creation of a novel remote laboratory architecture focused on, but not limited to, embedded systems experimentation. An architecture, based on Redis (an open source, in-memory data structure store, which is often used as database, cache or message broker), a modular design, and hardware-sharing techniques, is proposed in order to achieve the combined requirements of high scalability and cost efficiency. This mixed hardware-software architecture serves as a basis for the development of remote laboratories, especially those focused on microcontroller-based systems experimentation and embedded devices experimentation. From a user perspective the architecture is web-based, and has provisions to be easily adaptable to different Learning Management Systems and different hardware embedded devices. A new microcontroller-oriented remote laboratory based on the architecture has been developed, with the aim of providing valid evaluation data, and has been used in a real environment. The architecture and the resulting remote laboratory have been compared with other state of the art remote laboratories and their architectures. Results suggest that the proposed architecture does indeed meet the main requirements, which are scalability through replicability and cost efficiency. Furthermore, similarly to previous architectures, it promotes usability, universal access, modularity and reliability.

INDEX TERMS Remote laboratory, scalability, embedded system, online experimentation, architecture.

I. INTRODUCTION

Throughout the last years several initiatives have been undertaken that promote the practical use of remote laboratories [1], [2] in distant learning scenarios. For example, the European Go-Lab project, with 18 partners from 12 different countries, aimed to enrich the classroom experience with Inquiry Learning [3] and the use of virtual and remote labs [4]–[7] in schools.

Another example is the LabsLand[1] network, which is a commercial initiative that aims to establish a large global network of remote laboratories from different providing institutions, allowing schools and universities to access them through a unified and reliable platform [8], [9].

These initiatives can build upon the many advancements on remote labs that are now found in the literature. Remote labs today can be fully-based on the web, and can thus be easily accessible to most users [10]–[13]. Remote Laboratory

The associate editor coordinating the review of this manuscript and approving it for publication was Martin Reisslein.

[1]https://labsland.com

Management Systems (RLMS) are available, which can provide common features and ease the development of new labs [14]–[17]. Labs can be shared across institutions [18]. They can be accessible from any kind of device, including mobile ones and tablets [19]–[21]. Architectures have been created to make the scaling of laboratories and adapting laboratories to new *remotizable equipment* easier [22].

Based on the classification of [23], a remote laboratory offers an internet-mediated experiment. There are various types of remote experiments [24] depending on their discipline, level, complexity, or other characteristics. One type of remote laboratory is called *controlled laboratories*, in which the user is able to design and implement a controller system that controls a process. In *control engineering remote labs* the student typically designs or parametrizes a PID controller, which often controls a complex process, like water tanks management, coupled motors control, pendulum systems, ball and beam systems, etc. The most representative example of such laboratories according to [25] is UNILabs [26], [27]. NCSLab [28] is another interesting example of control engineering remote lab. In the previous cases, the learning outcome is focused on the design or parametrization of the controller, leaving aside its implementation, a task that is usually carried out by the laboratory itself. However, there is another type of remote laboratories with a different approach. Sometimes, the main learning outcome is focused on the design of a complete control algorithm, rather than in the parametrization of it. In this case, the control algorithm is developed using programming languages such as C, Python or VHDL. In order to successfully test the developed algorithm, the code is compiled and programmed in a real testing platform, or remotizable object. This platform usually comprises a microcontroller, a FPGA, or a PLC, among other devices, and a series of output peripherals. In this case, the learning outcome is focused on the programming competence. RELLE [29], RELDES [30] and GOLDi [31] are interesting examples of this type of laboratories, which also coincide with the laboratories that the architecture proposed in this paper tries to support.

However, the practical and massive use of remote labs across different institutions and by many students, as promoted by the aforementioned initiatives, poses some significant additional challenges. First, it should be possible to provide service to many concurrent users. Teachers will often wish to have their whole class, either individually or in small groups, using the labs at the same time. Likewise, different institutions might need to use the lab at the same time. Second, the service should be reliable. Teachers need to trust that the service will be available when they need it, as they need it.

These challenges can be partially alleviated by allowing calendar reservations. However, effectively, for many labs, the only practical way to actually overcome them is by replicating it a number of times. The LabsLand Arduino Robotics remote laboratory [32] allows students to learn robotics and Arduino programming with real hardware. Teachers need to be able to use it in-class, and they need to prepare their classes in advance. Several instances of the same robot are deployed across several institutions, and several students can thus be served concurrently.

Replicating a laboratory can solve both challenges. By having several identical copies, the load can be distributed across them, thus being able to serve several students concurrently. Furthermore, it can also provide reliability. Ensuring that a laboratory with many not particularly reliable consumer-grade moving parts is always working is far from trivial. A high downtime is to be expected, especially under heavy use. By having several instances of the same laboratory available, however, it is possible to automatically detect which ones are not working, and redirect the load to the working ones. This way, it is possible to reduce the effective downtime to almost zero percent, despite the inherent unreliability of the individual instances.

The main goal of this work is to propose a new mixed hardware-software architecture, based upon existing ones (e.g., [22]), which aims to promote the development of practical laboratories that are suitable for the aforementioned real-world usage. The architecture should serve as a basis for the development of multi-instance remote laboratories, especially those focused on microcontroller-based systems experimentation and embedded systems experimentation. In order to offset the previously exposed hurdles, this architecture has two main, interlinked goals. The first one is to promote the development of remote labs that can be replicated easily, and that contemplate replication in their initial design. The second one is to focus on the cost efficiency of the laboratory, which is a key aspect for labs that are meant to be replicated and which are meant to scale to a large number of users.

To evaluate the architecture, an Arduino remote laboratory has been implemented as an example. This laboratory has been designed from the ground-up to be easily replicated and to support several concurrent users and to be low-cost. It is used to verify that the technical requirements of the architecture are met, and a cost comparison and analysis with related works has been provided. Furthermore, the laboratory has been replicated as intended, and a real experience has been conducted with it with a class of school students.

The paper is organized as follows: Section II describes in more detail the relevant state of the art on remote labs and the relevant architectures. Section III analyzes the goals and requirements for the proposed architecture. Section IV presents an overview of the proposed architecture. Section V describes each layer in more detail. Section VI describes the Arduino remote laboratory that has been created to evaluate the proposed architecture. Section VII evaluates the architecture from a technical perspective. Section VIII compares the costs with those of related works in the literature. Section IX describes the pilot experience of the implemented laboratory with a school classroom. Section X summarizes the conclusions and Section XI describes future lines of work.

II. RELATED WORK

Remote laboratories are a fundamental tool for the education of embedded systems, allowing access to experimentation with the latest devices without the need for a continuous update of traditional hands-on laboratories. The Arduino platform is considered by experts to be the perfect tool for approaching the development of embedded systems [33], [34]. Despite the low cost of development systems based on this platform, having a remote laboratory provides the educator with an approachable tool that offers total abstraction of the hardware and facilitates access to the software development cycle required by the technology [35].

A set of remote labs are currently available providing real experimentation with Arduino over the Internet. Some of these systems [36]–[38] focus on allowing a single simultaneous access to the lab. This, however, cannot be used by the teacher during classes, because several simultaneous accesses are expected. Other laboratories are focused on the development of low-cost remote experimentation [39]–[41], but are not integrated into a RLMS that provides features such as authentication, LMS integration, federation or queue and reserve management, that guarantee scalability. All these systems focus on reducing the infrastructure required for laboratory deployment, and sometimes even provide the capability of balancing load among several instances of the same laboratory. However, they do not provide any particular means of facilitating the physical replication of the laboratory instances. Some labs, also, require the student to have the hardware to develop their own experimentation at home [42], [43]. These laboratories, known as *pocket labs*, give the users the opportunity to experiment with often more complex hardware devices, at home, solving the physical access barrier. These laboratories, while still being a good alternative for providing physical decoupling, are not extremely cost-efficient, because the laboratory hardware to user relationship is one to one. From an architectural point of view, laboratories [36]–[38] present a centralized distribution focused solely on the specific experiment itself, while laboratories [39]–[41] have been designed on a distributed architecture focused on cost reduction to favor scalability. This work proposes an architecture focused on scalability that allows the deployment of the desired number of instances of experimentation at a low cost, in order to provide simultaneous access to large student numbers and not hinder the experimentation capability.

In this work, two of the previously mentioned state of the art remote laboratories and its architectures [37], [38] have been thoroughly analyzed with the aim of creating comparative data. These laboratories and its architectures are representative of the main trends in microcontroller-based and embedded systems remote laboratories. This comparative data illustrates how the proposed remote laboratory architecture builds upon the state of the art to provide improvements to certain characteristics. These laboratories have been selected due to various reasons. Firstly, the objective of both laboratories is very similar, allowing the user to program an Arduino board remotely, which matches with the proposed architecture objective. Secondly, both laboratories and their architectures employ Linux-compatible single-board computers as a sort of control server, which also matches part of our proposed architecture control system. Thirdly, both of them are developed with cost efficiency in mind. In the following subsections, both of these similar Arduino-experimentation-capable remote laboratories are presented. Technical details about their architecture and their implementations are given, in order to establish the comparison basis. These laboratories will be analyzed and compared with this work's implementation in following Sections VIII and IX from cost-efficiency and scalability points of view, respectively.

A. RELDES ARDUINO REMOTE LABORATORY

The RELDES Arduino remote laboratory [30], [38] is a remote laboratory designed specifically for embedded systems design and experimentation using Arduino UNO boards as a remotizable object. It has been developed by the Software Tools Department of the Zaporizhzhya National Technical University, in Ukraine. From an architectural point of view, the RELDES laboratory only has two main components, a main laboratory server and the remotizable objects, based on Arduino boards.

The main server of this laboratory setup is in charge of managing all of the remote laboratory tasks, which are described below:

1) Access server, for managing authorization and queue control.
2) Web Server, supporting both the web-client and an integrated IDE.
3) Compiler.
4) Programmer.
5) Live-streaming server.

This laboratory has four remotizable objects, based on Arduino boards. These remotizable objects are not aimed towards scalability, but towards offering different experimentation setups. Each remotizable object is based on the same Arduino UNO boards and each of them has a different output peripheral. The laboratory offers the remotizable objects shown below in order to support different experimentation setups.

1) Arduino UNO + 4 PWM LED diodes
2) Arduino UNO + Servomotor
3) Arduino UNO + HD44780 display
4) Arduino UNO + HC-SR04 ultrasonic sensor

The user of this remote laboratory is capable of programming an Arduino UNO board with one of the previously described output peripherals. Once programmed in the integrated IDE, the user monitors the experimentation results via a real-time stream. It is noteworthy that once it has been programmed, the user has no means of controlling the experimentation setup. There are no connections between the server and the remotizable object, other than a USB cable to perform programming tasks only.

B. RELLE REXLAB ARDUINO REMOTE LABORATORY

The RELLE RExLab Arduino remote laboratory [29], [37] is an Arduino remote experimentation system developed by the Federal University of Santa Catarina, in Brazil.

The laboratory, focused also on the experimentation with Arduino-based remotizable objects is slightly more advanced, from an architectural point of view than its Ukrainian counterpart. This laboratory is formed by three key components, which are the RLMS, the laboratory server, and the remotizable objects. In this case, all the user management tasks are done by the RLMS, and only the controlling, programming and compiling tasks are done by the laboratory server. The remote laboratory offers two equal experimentation instances, achieved by total replication of the laboratory server and remotizable object, which are connected with the RLMS server via Ethernet, through a local access network. In this case, each remotizable object is formed by an Arduino UNO board, and the components shown below:

1) 4x LED diodes
2) 1x Servomotor
3) 1x HD44780 display
4) 1x Hardware potentiometer
5) 1x Humidity sensor
6) 1x Temperature sensor

Users, after accessing the laboratory through a RLMS system, which manages queue control, access and authorization, are presented with an integrated IDE that allows the programming of the remotizable object both in Arduino language or a visual blocks-based language.

Once programmed, users are able to see the experimentation instance through a live-stream. The user, in this case, is able to interact with the programmed remotizable object through a serial console or via some virtual buttons, controlled directly through GPIO pins in the laboratory server.

III. ARCHITECTURE GOALS AND REQUIREMENTS

The main goal of the remote laboratory architecture proposed in this work is to satisfy the previously analyzed needs that are present in the state of the art of remote laboratory architectures, especially in those designed for embedded systems experimentation.

The main goal of this work has been divided in several goals and requirements, which are explained below. The general goals are:

- **Scalability for multiple concurrent user support:** The architecture should be designed in a way that supports a high number of users accessing the laboratory devices concurrently. This way, multiple users can experiment with embedded devices at the same time.
- **Adaptability:** The architecture should provide the means to easily interconnect with a broad range of embedded devices.
- **Cost efficiency:** The architecture should be developed in a way that minimizes the deployment costs of a remote laboratory, especially since it is designed to produce multi-instance laboratories, where most of the hardware is replicated many times.

The main requirements are the following:

- **Embedded systems experimentation:** The architecture should, as previously described, be capable of providing a real and remote experimentation experience over a variety of embedded platforms.
- **Modularity:** The architecture should be reconfigurable in a way that adapts successfully to different deployment scenarios.
- **Universality:** The architecture should, from a user perspective, be accessible with independence of platform, device or operating system without the needs of any network reconfiguration or any software or plugins dependency.
- **Reliability:** The architecture should improve the reliability of the laboratory as a whole, as a side effect of scalability and replicability. Easing the deployment of multiple equal laboratory instances helps maintaining a pool of available instances, even when one or some of them fail.

In the following subsections, the previously exposed goals and requirements are analyzed in a more thorough way.

A. HIGH SCALABILITY FOR MULTIPLE USER SUPPORT

Embedded systems experimentation requires an embedded device to be physically programmed, per user. Those embedded devices are finite devices. In order to support a high number of concurrent users, such as those in a classroom environment, it is necessary to increase the number of available experimentation instances, thus creating a multi-instance laboratory. The proposed architecture should be designed in a way that can successfully support a high number of equal experimentation instances.

B. ADAPTABILITY

In the last years, a broad range of embedded devices and embedded development platforms [44] has emerged, and each of them can have different communication protocols and connectors. This increases the needs for easy adaptability. The architecture should provide the necessary hardware and software techniques to be able to withstand this variety of interconnection systems. This matter is also important from another point of view. Embedded devices and development platforms are constantly updated, meaning that in order to develop an easily-updatable remote laboratory, the architecture should provide the means to easily adopt new embedded devices and systems with minimum reconfiguration, and, when possible, avoiding the need to re-implement the laboratory from scratch.

C. COST EFFICIENCY

Traditionally, remote laboratories have required the use of expensive servers and several different hardware components in order to be developed. With the rise of single-board

computers [45] and other low cost hardware devices, the deployment cost of remote laboratories has been reduced. This architecture, in addition to offering support for being deployed over low cost hardware devices, should also reduce the costs even further by sharing different hardware resources among the multiple instances, with the aim of significantly reducing the cost per instance.

D. EMBEDDED SYSTEMS EXPERIMENTATION & USABILITY

The architecture should provide the means to facilitate a real remote experimentation over any embedded device or system. In order to ensure that the experimentation experience is successful and is not hindered, it is a requirement that the architecture provides a number of ways to interact with the embedded system once it has been programmed. Users should be able to program their scripts on the device, monitor them through a live stream and interact with them via different means, which can include digital and analog inputs, digital and analog outputs, input and output peripherals, and bidirectional serial communication via a console.

E. MODULARITY

In order to better adapt to different deployment scenarios, the architecture should be designed in a modular way. To achieve this modular design, the core of the architecture is subdivided into smaller modules or entities, which handle simpler tasks, and are interconnected via industry standard interfaces and communication protocols, such as Ethernet or SPI. This way, laboratory owners can follow different topology designs to better suit their laboratory setup requirements.

F. UNIVERSALITY

The meaning of *universality* varies among different contexts. In this work, the universality requirement means that the remote laboratories created upon this architecture should be as broadly compatible as possible [10]. These remote laboratories should be accessible with independence of the platform, device or operating system, opening the possibility of accessing to the lab from not only PCs and laptops, but from tablets and smartphones as well. To achieve this universality requirement, the remote laboratories should thus be fully web-based, and should not require any different access ports than the common HTTP 80 and HTTPS 443 ports, in order to eliminate firewall access problems. Also, in order to enhance universality, it is advisable that the architecture should be integrable in different Learning Management Systems or LMSs, such as Moodle, in order to be merged seamlessly with other web-based tools that educational institutions may have.

G. RELIABILITY

Due to the fact that a remote laboratory based on this architecture could have multiple instances of the same remotizable object, the reliability of the laboratory as a whole is directly related to the number of these instances being in an available state.

The RLMS or Remote Laboratory Management System, a core component of the architecture in charge of the laboratory administration tasks, should be able to properly manage both active and failed laboratory instances in order to maintain the largest possible number of experimentation instances available for use. In case a laboratory instance fails, the user load would be distributed among the remaining working instances. The waiting times for accessing the laboratory could be increased depending on the number of functional instances remaining, but the service would stay online.

IV. ARCHITECTURE OVERVIEW

As described in Section I, the proposed architecture has been designed with the goal of improving the scalability of embedded systems remote laboratories, in respect to the number of end users, with low cost and high flexibility in terms of supported devices.

As described in Section I, the proposed architecture aims to improve scalability through replication, in order to allow concurrent access to multiple users. Furthermore, it aims to do so with a low-cost, and it aims to be adaptable to different types of remotizable objects. In order to achieve close to horizontal scalability and flexibility while keeping costs reduced, the architecture has been divided in four main layers, which are highly decoupled in order to promote easiness of development and to better separate the concerns of each layer. Section V analyzes these layers from a low-level, more in-depth perspective.

Figure 1 shows an overview of the proposed architecture, which is decoupled in four main layers, and its main components.

In the first place, the lowest level layer in the architecture, which is the Hardware Layer, encompasses all of the hardware devices required to properly manage the remotizable object of the remote laboratory. Usually, a single-board computer with a series of hardware components is capable of interfacing correctly with the remotizable object, in order to replicate the physical actions that the user would perform on a hands-on laboratory. This remotizable object, which is the focus of the remote laboratory is also part of the hardware layer.

In the second place, the Interface Server Layer, is an entity in charge of abstracting all of the control and hardware-related idiosyncrasies via a simplified REST API. Its goal is to store and distribute all of the laboratory control tasks, that represent actions to be performed over the remotizable object, to the existing hardware drivers. These tasks are received through a web server that exposes a REST API, via GET and POST requests, and are saved in a FIFO data structure. Then, the tasks are then distributed sequentially to the available Hardware Drivers in order to be performed. Both the storing and distributing actions are performed by an in-memory data store/message broker mixed component. As can be seen on Figure 2, this entity along with the Hardware Layer, can be

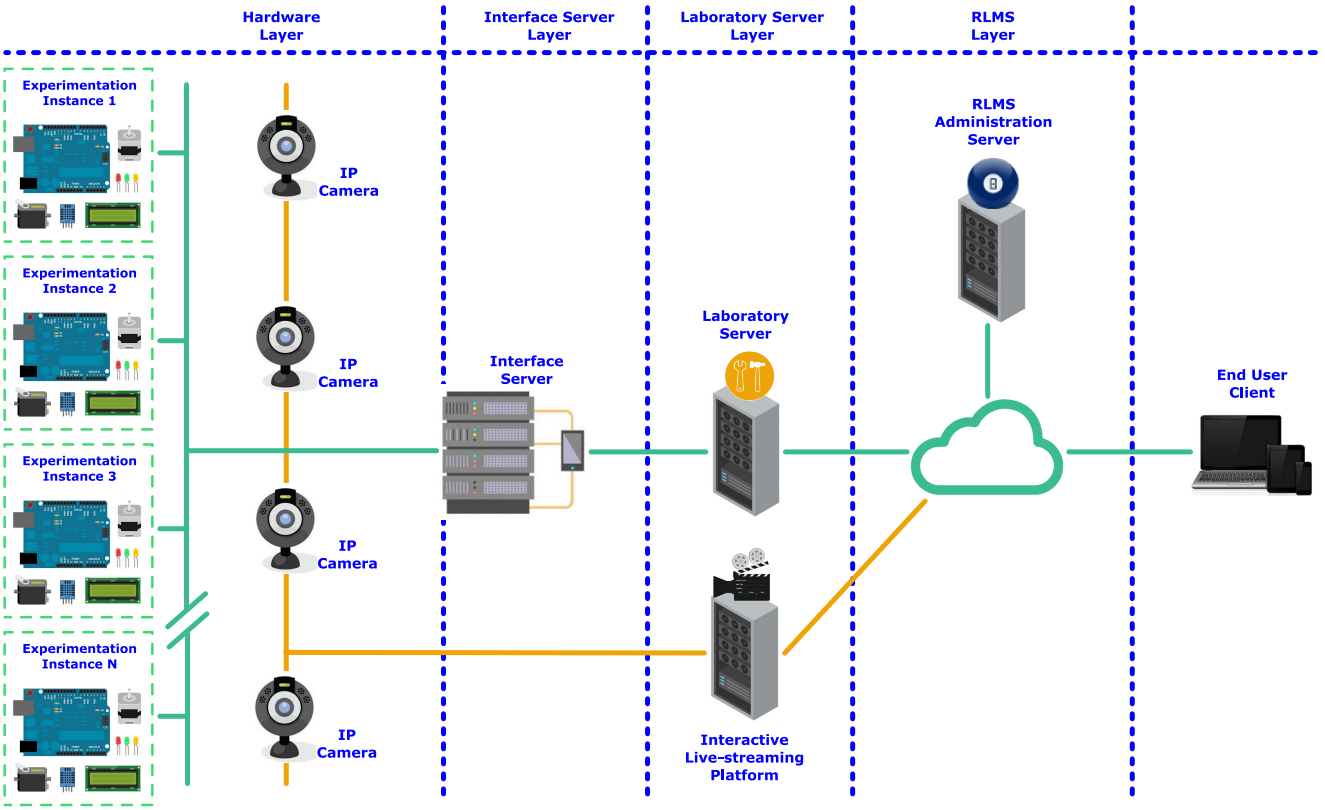

FIGURE 1. Overview of the proposed architecture split into several high-level layers.

divided in three minor subsystems each. These subsystems are explained in a more exhaustive way in Section V.

In the third place, the Laboratory Server Layer, is a subsystem in charge of supporting the required interaction between user and experiment. This component supports the web-based client that is shown to the user, which includes a real time stream of the experiment and the different virtual laboratory controls. Also, this entity, communicates with the Interface Server Layer in order to propagate downstream the user actions and to retrieve the actual state and information of the experiment.

In the fourth place, the RLMS Layer or Remote Laboratory Management System Layer is an entity in charge of controlling all of the administration tasks required to properly manage the different concurrent laboratory sessions that may arise. These tasks include user authorization, booking, queuing or load balancing, among others. It also serves as an integration tool, acting as a bridge between the remote laboratory and the different learning management systems or LMSs from where the students gain access.

Finally, an Interactive Live-streaming Platform is deployed in a parallel way to the other four subsystems. This component, which is also formed by various layers to ensure scalability and reliability, is in charge of providing the user of the remote laboratory with a real time stream, which is captured on-site by a camera. This platform abstracts out specific camera peculiarities and reliably provides these functionalities.

In the remote laboratory spectrum we can differentiate at least two types of laboratories. Those that require visual feedback and those that not. Some remote laboratories, like batch-processing ones for example, offer their users the ability to queue their jobs and retrieve the laboratory results at a deferred time. In these cases, laboratory outputs are usually captured by different hardware systems, and therefore, there is no need to stream a live image of the laboratory setup. On the other hand some remote laboratory setups require some sort of visual feedback, often in real time. Some of these labs use output hardware peripherals that are by nature, visual, like LED diodes, 7-segment displays, screens, servomotors or actuators among others. In these cases, users need to be able to observe the behaviour of these devices in order to check if the result of the experiment is correct or not. These types of labs are usually interactive, meaning that the user is able to control those peripherals in real time, In this case, the lab architecture needs to be capable of delivering the user a real time stream capture of the laboratory setup. The Arduino remote laboratory, which is explained in detail in Section VI is a case of remote laboratory where real time visual feedback is imperative, due to the use of visual peripherals which are controllable through the Arduino board in real time.

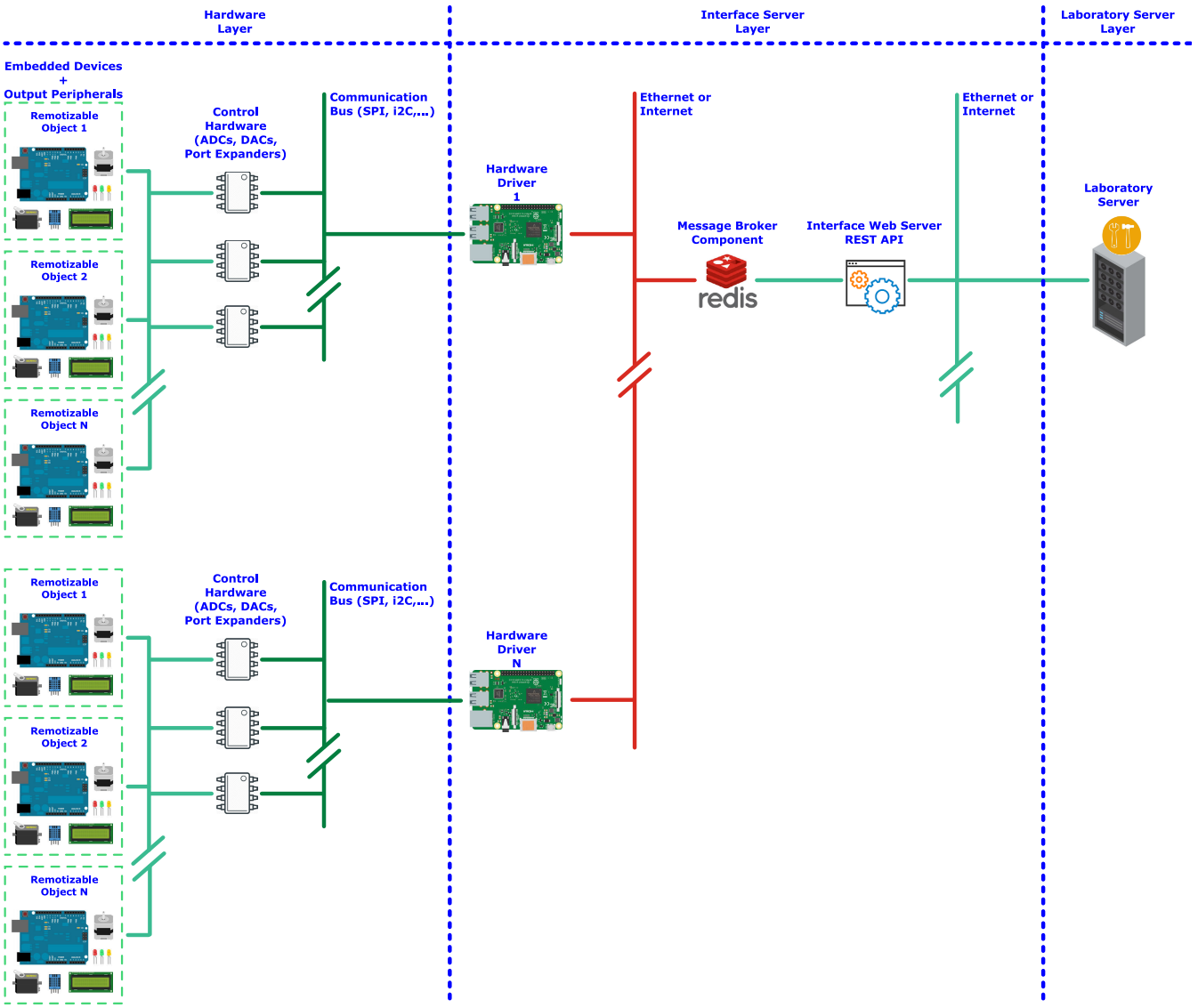

FIGURE 2. Exploded view of the Hardware and Interface Server layers, with all of the related decoupled entities.

As it can be observed, the information generated by the end user flows from right to left until it reaches the remotized embedded system, and the information generated or processed by this, flows from left to right until it is shown or provided to the end user. The architecture serves as a medium of communication between the user and the embedded device, providing physical decoupling and maintaining at the same time a high quality interaction capability.

V. ARCHITECTURE LAYERS

In this section, each of the layers and components that form the architecture, and which were briefly introduced in Section IV, will be described in more detail. All of the exposed entities can be handled as decoupled items, in order to create the topology that better suits the deployment scenario.

A. HARDWARE LAYER

The Hardware Layer of the architecture encompasses three main architectural entities.

1) **Output Peripherals:** They are at the lowest level of the architecture. These peripherals are controlled exclusively by the embedded device and therefore, by the user's own testing script. These peripherals are optional, because in some cases, they are already included in embedded development boards, and can range from simple LEDs or 7-segment displays to more complex peripherals, such as screens or servomotors.
2) **Embedded Device:** The previously described Output Peripherals are controlled directly by the Embedded Device. This embedded device can be changed for an embedded development board to suit different user requirements. Embedded devices can vary enormously

in order to suit different experimentation and learning requirements, ranging from microprocessor boards to FPGAs. These embedded devices, together with the output peripherals, form the remotizable object. The remotizable object forms an entity which can be replicated N times, in order to serve concurrent users.

3) **Control Hardware Layer:** In order to control, interact and communicate with the remotizable object, analog and digital signals management is needed. The Control Hardware Layer is an entity, formed by various devices, which is in charge of interacting electrically with the remotizable object. This layer includes different ICs, such as port expanders, digital-to-analog converters, JTAG programmers, CAN-Bus interfaces, in order to input and capture signals to and from the remotizable object.

B. INTERFACE SERVER LAYER

All of the remotizable objects are controlled by the Interface Server Layer, a layer which settles on top of the Hardware Layer, which is also formed by three independent entities. The boundary that separates these two layers is an interconnection bus, such as SPI, i2C or CAN-Bus among others.

1) **Hardware Driver:** The Hardware Driver is a software entity, that is usually located in a single-board computer. Its specific purpose is to convert the user petitions that arrive through the message broker component, into control commands that manage the Control Hardware Layer devices. These commands perform changes to the remotizable object. For example, when a user wants to change some digital input signal, in a specific instance, a task that describes this action arrives through the message broker component. The Hardware Driver interprets this task, and creates the necessary SPI control commands to change the behaviour of a determined input pin. Any number of Hardware Drivers can be deployed in order to manage different interconnection buses or any number of instances. In case of upwards communication, from the remotizable object to the users, the Hardware Driver works in a reverse way, interpreting the different signals and transmitting them via the message broker component.
2) **Message Broker Component:** On top of the Hardware Drivers there is the Message Broker Component. This entity is formed by an open-source in-memory data store that, in this case, acts as a FIFO data structure and as a message broker. The user commands are received, through the REST API, as tasks. For example, ''programming a binary'', ''changing output of a pin to high level'', or ''reading the value of a digital input''. These tasks are located in the FIFO data structure, which is accessible through Ethernet in a Local Access Network as a horizontal layer, thanks to the message broker capabilities of the in-memory data store. The tasks are then transmitted to the Hardware Drivers and the required actions are then performed. The system also works in the opposite direction for the laboratory information that is needed to be transferred to the user.
3) **Interface Web Server:** In order to provide a unified access through HTTP commands, another entity is needed. The Interface Web Server, which is located over the Message Broker Component, is another software entity that acts as a converter between the in-memory data store and the HTTP world. The Interface Web Server exposes a REST API in order to offer a control method based on simple HTTP commands. This web server, through the use of GET and POST methods, interprets and loads into the FIFO data structure the tasks to be done, and reads the output statuses in order to propagate this information through the upper layers. Different endpoints are offered for each action, such as: ''/program'', for programming tasks, ''/serial'' for serial communication tasks, or ''/potentiometer'' for virtual potentiometer-related tasks. Each endpoint is accessed through GET and POST commands in order to command a change or to retrieve information, respectively.

C. LABORATORY SERVER LAYER

Over the Interface Server Layer is the Laboratory Server Layer. This layer, which encompasses two software entities, is in charge of supporting two important features. One, it supports the laboratory web-based client, which includes the user controls, communication console and live-streaming video. And two, it abstracts all the communication idiosyncrasies between the web-based client and the Interface Server layer.

1) **Web-based Client:** The Web-based Client, which is a software entity, is in charge of transmitting visually the laboratory setup to the end user. This client can include a real-time stream, a programming utility, a communications console, virtual switches, buttons, potentiometers and led diodes and different debugger options.
2) **Laboratory Web Server:** The Laboratory Web Server is another software entity, tightly coupled with the Web-based Client. This web server, which supports the previously explained entity, also acts as a control system. It is in charge of transferring all of the users commands to the Interface Web Server, and collecting all of the output data that is generated by the embedded device, for it to be shown on the client.

D. REMOTE LABORATORY MANAGEMENT SYSTEM LAYER

The Remote Laboratory Management System Layer or RMLS Layer manages all of the administration tasks that may arise in a remote laboratory provider context. The RLMS Layer properly manages the sessions for different concurrent users that may access the remote laboratory platform, as well as manages queuing, authentication, booking, user tracking or load balancing over various instances of the same type of laboratory. After granting access to the users, and

allocating a laboratory instance, redirects the user to the Laboratory Web Server.

E. INTERACTIVE LIVE-STREAMING PLATFORM

In addition to the previously described layers, an Interactive Live-streaming Platform needs to be deployed, in order to provide the end users with a real time view of each laboratory instance. The development of this system is out of the scope of this work, which is centered in the actual embedded devices control tasks. In the successive Section VI, a more in-depth explanation about the WILSP platform, which has been used in the architecture implementation, is given.

VI. THE ARDUINO REMOTE LABORATORY

A remote laboratory, named ArduinoRL, which is based on the previously proposed architecture has been developed and deployed as a proof of concept. ArduinoRL is a highly scalable remote laboratory that provides web-based experimentation over four Arduino UNO Rev3[2] boards, equipped with Atmega 328P MCUs, low-power 8-bit AVR microcontrollers that are used with great success in both educational and do-it-yourself environments. The choice has been made due to Arduino being one of the most popular platforms used by the educational and DIY clusters around the globe [46]–[50].

Each experimentation instance is formed by the aforementioned Arduino UNO Rev3 board and a series of peripherals, which include LED diodes, potentiometers, serial communications console, switches, buttons, an OLED screen and a servomotor. These peripherals are prearranged and already connected in a specific way, because the users are unable to connect these components by themselves due to the system being a remote laboratory. A schematic, shown on Figure 3 is given to the users in advance, so they can program the Arduino board accordingly. Input peripherals, such as switches, buttons and potentiometers are controlled remotely thanks to the use of the Control Hardware Layer, which is explained below, that allows the user to replicate those electrical signals in-situ through the internet. Each Arduino experimentation instance has therefore, a series of capabilities for the user to explore with:

- **Analog input signal control (4x)**
  - Connected each to potentiometer as voltage divider.
  - Voltage range from 0V to 3.3V.
  - Remotized control through web-client.
- **Digital input signal control (4x)**
  - Connected each to a switch and a button in parallel.
  - Remotized control through web-client.
- **Digital output signal control (8x)**
  - 2x non-PWM outputs connected to 2x LED diodes.
  - 2x PWM outputs connected to 2x LED diodes.
  - 3x PWM outputs connected to 1x RGB LED diode.
  - 1x PWM output connected to 1x servomotor.
- **Serial communication control**
  - Accessible through Arduino IDE-alike console.

[2]https://store.arduino.cc/arduino-uno-rev3

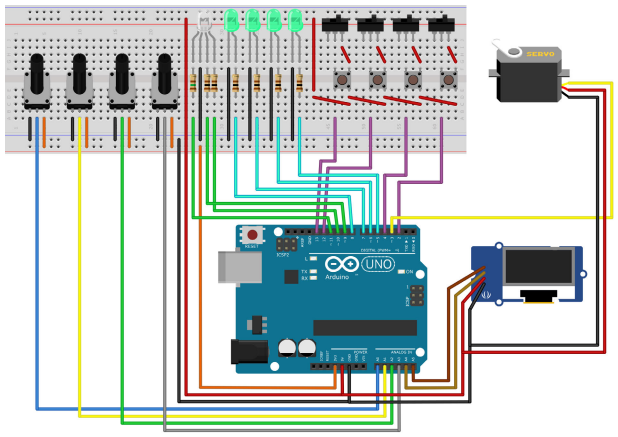

FIGURE 3. Fritzing schematic equivalent of each Arduino experimentation instance. Figure shows how the peripherals are already connected, so user coding can be correctly matched.

- **I2C communication control**
  - I2C ports connected to OLED SSD1306 display.
  - User controls the screen through an Arduino script.

These peripherals are popular in the Arduino ecosystem, and are usually encountered in the vast majority of Arduino *starter kits*. The aim of this remote laboratory is to offer users and students the ability to experiment as they would do with a hands-on Arduino starter kit, but in a remotized way, through the Internet. An increasing number of schools and universities use these starter kits for practices in different subjects, so having a remotized version which is accessible through the Internet, is a positive thing. The laboratory flexibility is on par with the hands-on Arduino version, meaning that the number of realizable exercises and practices is almost infinite, so teachers can better adapt them to the contents taught in class.

The ArduinoRL remote laboratory has been integrated in the LabsLand[3] remote laboratory platform. The laboratory is currently in production state, and can be tested for free[4].

In the next subsections, the implementation of ArduinoRL is analyzed in accordance with the layers previously exposed in Section V.

A. HARDWARE LAYER

The Hardware Layer of ArduinoRL remote laboratory is formed by three hardware entities.

1) **Output Peripherals:** Each of the four ArduinoRL instances has a fixed amount of output peripherals, which have been chosen after a thorough analysis. These peripherals maximize the connection capabilities of the Arduino UNO board, using all of the input and output ports available. These output peripherals are:
   - 1x RGB LED diode with PWM capabilities.
   - 2x LED diodes with PWM capabilities.

[3]LabsLand is a spin-off of the WebLab-Deusto research group.
[4]https://labsland.com/en/labs/arduino-board

- 2x LED diodes without PWM capabilities.
- 1x servomotor.
- 1x i2C-compatible OLED SSD1306 screen.

2) **Embedded Device:** As has been previously explained, each of the ArduinoRL instances has an Arduino UNO Rev3 board, in its SMD version, as an embedded device.
3) **Control Hardware Layer:** The Control Hardware Layer of ArduinoRL is formed by:
   - 8x Microchip. MCP4822[5] dual-output digital-to-analog converters.
   - 1x Microchip MCP23S17[6] sixteen input/output port expander.

   These components, that are connected to a SPI bus, are in charge of providing 4 analog input signals and 4 digital input/output signals to each Arduino instance. In total, they handle 32 signals. These Arduino UNO boards can handle programming and serial communication tasks through their USB connection, so no other devices are needed in this layer.

B. INTERFACE SERVER LAYER

The Interface Server Layer of ArduinoRL is formed by three software entities, which are described below. These software entities are deployed over a Raspberry Pi 3 Model B[7] single-board computer. This device has been chosen because it has four USB connections, it supports SPI communication, has enough computational power to handle all three software entities, has Ethernet connection and is low cost.

1) **Hardware Driver:** This software entity has been deployed as a Python script, that handles all of the hardware management. This Python script connects to the Redis[8] FIFO stack through the loopback connection because the Redis entity is also local. It manages the programming tasks and serial communication directly through the USB connection and also manages the SPI bus communication through Linux SPI API, in order to control all of the four laboratory instances.
2) **Message Broker Component:** In order to develop the combination of a FIFO data structure and a message broker, Redis [51] has been chosen. It is an open source, in-memory data structure store that has been used successfully in many well-known projects to provide scalability [52]–[54]. An instance of Redis has been deployed locally over the Raspberry Pi 3. This instance is accessed over the loopback connection both by the Hardware Driver and the Interface Web Server.
3) **Interface Web Server:** A Flask[9]-based web-server has been deployed also locally in the Raspberry Pi 3. This Flask web-server exposes a REST API capable of controlling different tasks for the four available instances. These tasks include, for each instance:
   - Programming the instance.
   - Resetting the instance.
   - Changing any of the 4 analog input signals.
   - Changing any of the 4 digital input signals.
   - Changing any of the 4 digital output signals.
   - Sending a serial command to Arduino.
   - Reading a serial command from Arduino.

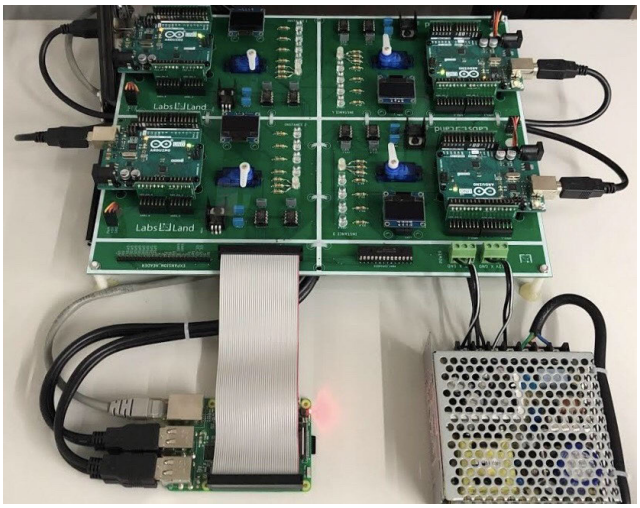

FIGURE 4. Implementation of the ArduinoRL remote laboratory, including the four experimentation instances and associated peripherals and hardware.

Figure 4 shows the implementation of the ArduinoRL remote laboratory. The figure shows a PCB, which supports and interconnects all of the components of the Hardware Layer with the Interface Server Layer, which has been implemented over a Raspberry Pi 3 single-board computer.

C. LABORATORY SERVER LAYER

The Laboratory Server Layer of ArduinoRL is formed by two software entities, which are described below. These entities are deployed over a Dell ProLiant server that was already running in our facilities. This server is also used by other previous laboratory deployments, and has been used only because it was already connected to the Local Access Network, and had available resources. In case of a massive deployment of a laboratory based on this architecture, an isolated server may be preferred. This layers entities are:

1) **Web-based Client:** A Web-based Client has been developed for this ArduinoRL remote laboratory. As can be seen on Figure 5, the client has different digital and analog controls, a serial communication console and a live-stream capture of the laboratory setup. Also, the users are offered an online IDE, in which they can create their own Arduino programming scripts using Arduino Language or a block-based programming language which uses Google's Blockly[10] library. Block-based programming languages are a new educational

[5] https://www.microchip.com/wwwproducts/en/MCP4822
[6] https://www.microchip.com/wwwproducts/en/MCP23S17
[7] https://www.raspberrypi.org/products/raspberry-pi-3-model-b/
[8] https://redis.io/
[9] http://flask.pocoo.org
[10] https://developers.google.com/blockly/

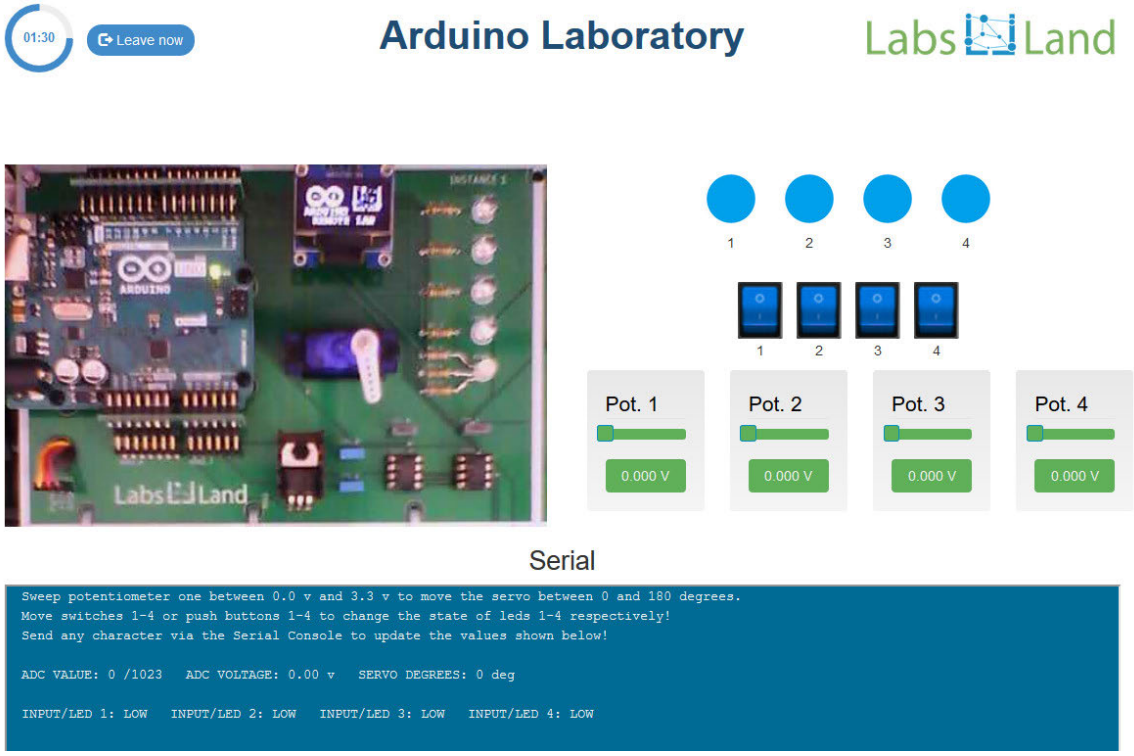

FIGURE 5. Web-based client for ArduinoRL, including the control means, a real-time stream and the serial console with an Arduino-produced example text.

trend which can improve learning gains [55]. Once the code is ready, the client handles the programming stage, where the users select the right script. Figure 6 shows the integrated, web-based IDE.

2) **Laboratory Web Server:** Another Flask-based web-server has been developed to handle each users web-based tools. This web-server supports both integrated IDEs, the laboratory client and all of the interconnection requests that happen between this layer and the Interface Server Layer. In order to manage the four available laboratory instances, four virtual instances of this web server have been deployed over the physical Dell server.

D. REMOTE LABORATORY MANAGEMENT SYSTEM LAYER

The ArduinoRL laboratory has been integrated in the LabsLand remote laboratory platform. This platform is compatible with the WebLab-Deusto Remote Laboratory Management System [56], so the integration between laboratory and platform has been straightforward. The use of the *weblablib* [57] library simplifies the integration process. This RLMS system is fully integrable in the top-tier LMS systems which are used by almost all of the educational institutions that access these laboratories. Also, WebLab-Deusto is compatible with other web-based platforms, in order to maintain universal access.

E. INTERACTIVE LIVE-STREAMING LAYER

An Interactive Live-streaming Layer, named WILSP [58], has been deployed in parallel, with four sub-instances in order to capture in real time the behaviour of each laboratory instance, and to transmit that image with the minimum delay, in a range of available formats, to the web-client. This transversal entity is explained in a thorough way below.

1) THE WILSP PLATFORM

An important component of many remote laboratories is an interactive live-stream provided by a web camera (e.g., [12], [59], [60]). While in a hands-on laboratory users observe through their own eyes, in a remote laboratory they do this through this camera and use that information to interact with the lab.

This is why to provide a satisfying user experience, this stream needs to meet certain requirements. The most important requirements are, first, that it is fully web-based (no plugins or non-standard ports required) and second, that the latency is low. Specifically, the stream needs to have a low enough latency so as to be considered interactive [61]. This latency, also known as capture-render delay, should enable users to interact with the equipment while perceiving its response in close to real-time.

To satisfy these requirements, this work makes use of the Open Source WILSP [58] Interactive Live-Streaming Server, which has been purposefully designed to be effective for remote labs.

2) WILSP AND ARDUINORL INTEGRATION

In line with the cost-saving focus of the architecture proposed in this work, additional improvements have been added on

FIGURE 6. Web-based integrated IDE for the ArduinoRL remote laboratory.

top of the standard WILSP-based architecture. The most significant of them is the ability to share a single web camera for more than a single instance of the same remote laboratory.

Stand-alone IP cameras are relatively expensive. For example, a D-Link DCS-2132L, used in the LabsLand Arduino robotics remote laboratory [9] can cost around €150. This can be a significant fraction of the total cost of the remote lab instance, considering the prices of other components (e.g., a Raspberry Pi can cost around €40, and an Arduino board around €25). See Section VIII for more details.

The WILSP platform has been extended to be able to serve *partial* streams. This way, the architecture proposed in this work can use a single camera to point at two simultaneous instances. The hardware of the Arduino remote laboratory that is proposed as an implementation example has also been purposely designed for this purpose, by arranging it in such a way so as to make use of the wide aspect ratio of modern web cameras. Thus, only two web cameras are needed to serve four different instances.

Figure 7 shows the annotated original stream of the physical IP camera. It shows that the original stream cover two different instances of the remote lab. The area around A is one of the instances, while the area around B is the other. The stream is received by the modified WILSP, which is then able to cut, rotate, re-encode and split it, to serve it as different streams for each laboratory instance.

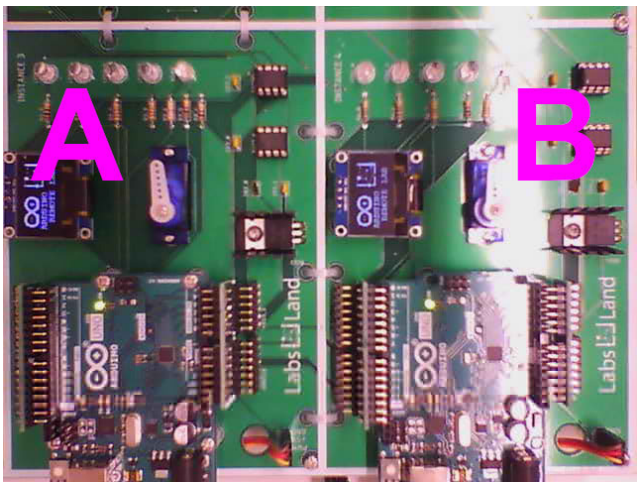

FIGURE 7. Original stream of an IP camera, covering two different remote lab instances.

This way it is possible to obtain a significant cost reduction without significantly reducing the perceived image quality of the laboratory.

VII. TECHNICAL EVALUATION

This section will use the implementation of the proof-of-concept described on Section VI to verify that the architecture meets the technical requirements initially proposed in Section III; and a comparison will be made with works and similar architectures of the state of the art taking into

account the characteristics that each architecture offers. The evaluation conducted through the proof of concept suggests that those goals and requirements have been indeed met.

A. GOALS

The goals proposed were the following:

- **High scalability for multiple concurrent users support**
- **Adaptability**
- **Cost Efficiency**

The scalability goal of the architecture has been satisfactorily fulfilled, thanks to the use of high scalability tools, like Redis, and the use of a decoupled design combined with different communication protocols, schemes and devices. A more in-depth analysis of this matter is explained in successive Section IX.

The architecture, from an interconnection point of view, can indeed be considered to provide high adaptability. Efforts have been made in order to create a Interface Server capable of connecting to different embedded systems through fully established connection buses like i2C, SPI or serial. Most of the state of the art embedded development devices can be programmed with these interfaces, because manufacturers tend to incorporate universal programming resources on the development boards. Also, the different analog or digital signals required to control the embedded system once programmed, and the selected output peripherals are CMOS and TTL tolerant, so once more, adaptability is promoted.

In terms of cost efficiency, the architecture is satisfactory as well. Thanks to a thorough design and partial hardware replication, and the possibility of developing the remote laboratory over low-cost single-board computers, the total cost of deployment is highly reduced. A more thorough analysis of the cost efficiency matter is explained in Section VIII.

B. REQUIREMENTS

The requirements stated in Section III were the following:

- **Embedded system experimentation**
- **Universality**
- **Modularity**
- **Reliability**

The architecture is indeed capable of providing a real time, remote, embedded system experimentation experience for multiple concurrent users. The end users are capable of programming, controlling, and interacting with the embedded device and are also capable of monitoring the results of their programs, all in real time, thanks to the output peripherals and the interactive live-streaming system. This novel architecture provides enhanced interacting capabilities over different state of the art remote laboratory architectures, which are explained below.

The architecture, from a client-side perspective, can indeed be considered to provide high universality. It is fully web-based, meaning that it is accessible, through standard Internet ports, from any device or platform combined with any operating system. The only access requirements are connection to the Internet and access to a web browser. From a laboratory-owner perspective, the architecture is also universal, because the integrated WebLab-Deusto remote laboratory management system or RLMS, is fully compatible with the top-tier learning management systems, or LMSs, such as Moodle, and is also integrable in any other web-based educational tools that most institutions use [62] with minor difficulties.

The modularity requirement of the architecture has indeed been achieved thanks to a mixture of cautious interconnection design based on standardized communication protocols and media, and decoupled hardware and software entities. The different software entities shown on Figure 2 can be deployed over separate physical servers or can be joined in the same machine if the physical install of the remote laboratory requires it, easing the deployment process and enhancing the modularity. Also, from a hardware point of view, the architecture uses different communication protocols, such as Ethernet or SPI, which act as layer edges, assuring that different devices, such as the Interface Server, can be interconnected with the different remotizable objects in various ways to achieve at the same time high scalability and adequate adaptation to the physical characteristics of the deployment site.

The reliability requirement has been also satisfied. The architecture is designed to manage a large number of experimentation instances. As seen in Figure 2, it is formed by three different layers, which can scale vertically in order to support an arbitrary number of users, while still keeping costs under control.

A remote laboratory with even modest number of active experimentation instances can often serve a high number of users, as it is more thoroughly explained in Section IX. In case of an experimentation instance being offline due to a software or hardware failure, the rest of the experimentation instances would still be available to the users automatically and transparently. The Remote Laboratory Management System acts as an abstraction layer for them, and automatically balances the load and adjusts the waiting time based on user load and active laboratory instances available, thus avoiding loss of service. It is noteworthy nonetheless that if shared components, such as an Interface Server, do fail, several laboratory instances could break at the same time. For example, in the case of ArduinoRL, each interface server is shared by 4 laboratory instances. Therefore, should it break, the 4 instances would fail as well. However, the components that are shared are precisely those that are not particularly prone to failure. And the architecture is designed for labs to be replicated efficiently.

C. COMPARISON BETWEEN ARDUINORL AND STATE OF THE ART LABORATORIES AND ARCHITECTURES

In this subsection, the ArduinoRL remote laboratory and its architecture will be compared, from a technical perspective, to the previously described state of the art remote laboratories

TABLE 1. Comparison of the analyzed remote laboratories and its features.

		ArduinoRL	RELDES	RELLE's RExLab
Modularity	Decoupled architecture design	✓	✗	✗
Adaptability	USB Programming	✓	✓	✓
	SPI Programming	✓	✗	✗
	Serial Programming	✓	✗	✗
	JTAG Programming	✓	✗	✗
Usability	Live streaming	✓	✓	✓
	Output peripherals	✓	✓	✓
	Output analog signals	✗	✗	✗
	Output digital signals	✓	✗	✗
	Input peripherals	✗	✓	✓
	Input analog signals	✓	✗	✗
	Input digital signals	✓	✗	✓
	Serial console	✓	✓	✓
Universality	Integrated IDE	✓	✗	✓
	Web-based	✓	✓	✓
	Integrable in LMSs	✓	✗	✓
Reliability	Multiple instances	✓	✗	✓
Positive points		**15/17**	**6/17**	**10/17**

and its architectures. A cost efficiency analysis comparing these laboratories and its architectures has been done in Section VIII.

Table 1 indicates the performance of the analyzed remote laboratories and its architectures, from a technical point of view. Positive or negative values are given for each analyzed feature. As can be seen, evaluation of these set of features suggests that the proposed architecture in this work is more advanced.

From an architectural point of view, the proposed architecture is advanced in terms of scalability, adaptability and modularity compared to the previously described remote laboratory architectures. Being a highly decoupled architecture assures that each component is in charge of managing simpler tasks and, combined with the use of well-established communication protocols among these components, achieves a high level of modularity. Both of these characteristics also allow for a high scalability, because control processes and interconnections are simpler, and makes the replication of components and branches easier. A detailed view of the RELDES remote laboratory architecture can be seen in Figure 8.

Both modularity and scalability are hindered in the previously described examples, because in both laboratories, the hardware interfaces are directly managed by the laboratory server. This means that in case of needing to support a large number of users, either a full replication of both the laboratory server and remotizable objects, which scales costs proportionally, or a full re-conversion of the laboratory architecture is needed. In the case of RELLE's RExLab Arduino laboratory, the laboratory could be scaled by replicating the full laboratory server, which is based on a low-cost, single-board computer, but even in this case, a single-board computer is needed for each instance, which increases deployment costs.

Both laboratories group their hardware and interface server layer in a single entity. This means that, generally, if the laboratory were to be modified, the change would affect the whole entity. The proposed architecture, however, is split in several layers, including decoupled hardware and interface server layers. Within those layers, entities are independent as well. Thus, if the laboratory were to be modified, the change would only involve affected entities, which will under most circumstances be only a small subset.

In the previously described laboratories, adaptability is also hindered, because both of them only support USB-based remotizable objects. The proposed architecture has been developed in a way that supports, both from a software and hardware perspective, several protocols between the remotizable object and the Worker Hardware Interface. Those include USB, JTAG, serial and SPI based interfaces, among others. This assures connectivity with most embedded systems development platforms and protection to withstand the short life cycle of those platforms.

From a usability perspective, the described remote laboratory setups can also be improved in some respects. The most noteworthy capability that these laboratories are missing is remotizable object control after programming. RELDES remote laboratory only offers a unidirectional serial console as a control means once the remotizable object is programmed, reducing the experimentation experience learning capabilities and widening the gap between real and remote experimentation. RELLE's RExLab Arduino remote laboratory setup is more advanced at this respect than its Ukrainian competitor, and it provides serial console and input digital signals. The proposed architecture in this work builds further

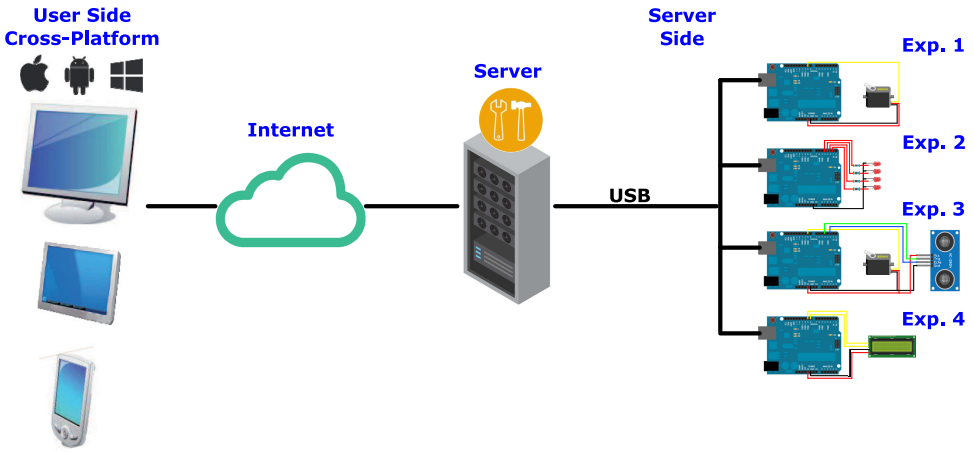

FIGURE 8. Overview of the RELDES architecture. Adapted from [38].

on this, to provide more advanced interaction capabilities. Particularly, it offers digital input and output control signals, analog input control signals, output visual peripherals and a bidirectional serial console interface in order to achieve a high-quality, comprehensive interaction between user and experimental setup.

The universality aspect of the three analyzed architectures is similarly satisfied. The three laboratories are fully web-based, which implies that they are accessible from any device (e.g., laptop, tablet, smartphone) with any operating system and from any web browser. The RELLE's RExLab Arduino laboratory offers, like the proposed architecture, both an integrated, web-based IDE, which eliminates the need for the user to develop their programs in any vendor-specific and platform-constrained IDE; and also support for integration with the main Learning Management Systems, such as Moodle or Google Classroom. The RELDES laboratory, however, does not offer those features.

In this work, the reliability requirement is heavily tied to scalability and multi-user support. Basically, every architecture that has the capability of supporting multiple users, with multiple instances that can work with independence between them in case that one of them breaks out, meets the requirement of reliability. Both ArduinoRL and RELLE's RExLab laboratories offer some degree of scalability, so both of them also match, at least partially, the reliability requirement.

VIII. COST EFFICIENCY EVALUATION

In Section VII two state of the art remote laboratories and its architectures were analyzed from a technical perspective and compared with the proposed architecture of this work. In this section, all three architectures and their resulting laboratories are analyzed from a cost efficiency point of view. The cost efficiency matter in this work is tightly linked to the number of experimentation instances supported by each laboratory. This section aims to test if an architecture focused on high scalability can increase the cost efficiency of a remote laboratory composed by multiple experimentation instances, by lowering the cost per deployed instance. Different levels of hardware sharing between instances can have a high impact in the cost per instance marker. In order to perform an analysis of the architecture's cost efficiency, the implementation costs for each remote laboratory have been estimated for four different scenarios, comprising one, four, sixteen and ninety-six laboratory instances respectively. These specific numbers are chosen to help depict the tendency of cost per instance of each architecture, and how they would scale to an ever-larger number of users. The number and type of used components have been adjusted to the specification of each architecture and their cost has been unified through the three laboratories, so they can be compared. The components have been priced according to actual prices from reputable European hardware distributors. The Remote Laboratory Management Systems are discarded from this analysis, as well as the standard servers needed by the different live streaming systems' backends, excluding the actual cameras.

In the next subsections, each laboratory setup hardware is analyzed from the Remote Laboratory Management Systems connection downwards.

A. RELDES ARDUINO REMOTE LABORATORY

The RELDES Arduino Remote Laboratory combines a Laboratory Server, Interface Server and Management Server all in one software and hardware module, which is based on a Raspberry Pi 3 single-board computer. Due to this, the cost of the Raspberry Pi single-board computer has to be added to the total laboratory expenses. The costs collected in Table 2 represent the total hardware expenses for the deployment of a RELDES Arduino laboratory with one, four, sixteen and ninety-six experimentation instances. The cost of each experimentation instance includes the cost of the four partial instances that this laboratory includes. As described in Section VII, RELDES includes four partial instances, which are similar but offer different output peripherals, and which are thus actually different laboratories from a user perspective. Thus, this laboratory and its architecture aims to

TABLE 2. Hardware cost for different number of instances of RELDES laboratory.

Units for 1 instance	Units for 4 instances	Units for 16 instances	Units for 96 instances	Item	Cost (€)
1	4	16	96	Raspberry Pi 3	38.82
4	16	64	384	Arduino UNO	22.01
1	4	16	96	IP camera	150.00
1	1	1	1	Laboratory physical structure	29.00
1	1	1	1	Power supply	25.24
1	4	16	96	Connection cables	2.92
1	4	16	96	Servomotor	15.05
1	4	16	96	HD44780 display	10.99
1	4	16	96	HC-SR04 ultrasonic sensor	3.83
4	16	64	384	LED diodes	0.46
4	16	64	384	Resistors	0.28
				Cost for 1 instance	366.85
				Cost for 4 instances	1,304.68
				Cost for 16 instances	5,056.00
				Cost for 96 instances	30,0064.80

maximize remotizable object diversity rather than scalability. For a student to perform different Arduino practises using all of the proposed hardware, four independent accesses would need to be performed. In order to satisfy the usability requirement for end users, each iteration of the remote laboratory, when scaling the laboratory, would need to include these four instances.

B. RELLE'S REXLAB ARDUINO REMOTE LABORATORY

The RELLEs RExLab Arduino Remote Laboratory is compatible with RLMS systems, and combines the Laboratory Server and the Interface Server all in one entity, which is also based on a Raspberry Pi 3 single-board computer. The costs collected in Table 3 represent the total hardware expenses for the deployment of a RELLEs RExLab laboratory with one, four, sixteen and ninety-six experimentation instances.

C. LABSLAND'S ARDUINORL REMOTE LABORATORY

The ArduinoRL remote laboratory, which is based on the proposed architecture by this work, is also fully compliant with RLMS systems, but its layers are deployed in a different way. In this case, the Interface Server is decoupled from the Laboratory Server, meaning that, in order to manage from one to ninety-six laboratory instances, only one Raspberry Pi 3 single-board computer is needed. Also, in order to support the Laboratory Server, which can have up to ninety-six web-server virtual instances, a dedicated general-purpose server is needed. The costs collected in Table 4 represent the total hardware expenses for the deployment of remote laboratories based on the proposed architecture with one, four, sixteen and ninety-six experimentation instances, meaning that each setup is capable of serving up to one, four, sixteen or ninety-six concurrent users, respectively.

D. COST EFFICIENCY ANALYSIS FOR STATE OF THE ART LABORATORIES AND ARCHITECTURES

In the previous subsections, the cost of deploying hardware for the three different laboratories has been analyzed. In each case, the guidelines set by each architecture have been followed. In this subsection, the three architectures are analyzed from a cost efficiency perspective. As shown in the previous subsections and its associated tables, the cost per instance when deploying only one instance of each remote laboratory is fairly matched, and the total hardware expenses range between €300 and €400. This is not a coincidence, since to support a single instance, all the analyzed architectures require almost the same basic hardware components. The differences appear when the number of required instances increases, as not all of the architectures can scale in the same manner. As seen on Tables 2, 3 and 4, the total expenses not only vary, but they diverge greatly as the number of instances increases. Both RELDES and RELLEs RExLab laboratories end with a total cost of deployment between €25,000 and €31,000, whereas ArduinoRL laboratory total cost ends below the €15,000 barrier, for a number of ninety-six supported instances. This difference is caused by how each architecture handles scalability. Since RELDES and RELLEs RExLab laboratory architectures are not designed to share hardware resources between each instance (with the exception of the power supply and physical structure), a full replication of each instances hardware is needed for each new instance, which scales costs proportionally. The architecture is designed in a way that maximizes hardware sharing between instances, which means that virtually the only hardware that is replicated when creating new instances is the remotizable object. In this case, the interface server requires no replication, and the control hardware, which includes port expanders and digital-analog converters, is cheap to replicate, and requires almost no reconfiguration of the software either.

TABLE 3. Hardware cost for a different number of instances of RELLEs RExLab laboratory.

Units for 1 instance	Units for 4 instances	Units for 16 instances	Units for 96 instances	Item	Cost (€)
1	4	16	96	Raspberry Pi 3	38.82
1	4	16	96	Arduino UNO	22.01
1	4	16	96	IP camera	150.00
1	1	1	1	Laboratory physical structure	29.00
1	1	1	1	Power supply	25.24
1	4	16	96	Protoboard	7.15
1	4	16	96	Connection cables	2.92
1	4	16	96	Servomotor	15.05
1	4	16	96	Potentiometer	2.54
1	4	16	96	HD44780 display	10.99
1	4	16	96	Humidity sensor	3.83
1	4	16	96	Temperature sensor	3.83
4	16	64	384	LED diodes	0.46
4	16	64	384	Resistors	0.28
				Cost for 1 instance	311.80
				Cost for 4 instances	1,048.48
				Cost for 16 instances	4,175.20
				Cost for 96 instances	24,780.00

TABLE 4. Hardware cost for different number of instances of ArduinoRL laboratory.

Units for 1 instance	Units for 4 instances	Units for 16 instances	Units for 96 instances	Item	Cost (€)
0	1	1	1	Laboratory Server	251.82
1	1	1	6	Raspberry Pi 3	38.82
1	4	16	96	Arduino UNO	22.01
1	2	8	48	IP camera	150.00
1	1	1	1	Laboratory physical structure	29.00
1	1	1	1	Power supply	25.24
1	1	1	1	Interconnection PCB	45.00
0	0	1	6	USB hub	9.89
1	1	4	24	SPI Port expander	2.92
2	8	32	192	SPI DA converter	2.15
1	4	16	96	Servomotor	15.05
1	4	16	96	SSD1306 i2C 0.96" display	10.99
5	20	80	480	LED diodes	0.46
7	28	112	672	Resistors	0.28
				Cost for 1 instance	347.59
				Cost for 4 instances	919.24
				Cost for 16 instances	2,517.21
				Cost for 96 instances	13,347.96

This issue greatly reduces the cost per instance of each remote laboratory. As seen in Figure 9, the cost per instance for RELDES and RELLEs RExLab remote laboratories decreases minimally between the 1 and 4 instances markers, and stabilizes in values around €315 and €260 respectively, whereas the cost per instance of ArduinoRL decreases greatly and continuously as more instances are supported, and stabilizes in a lower value of €140, which is estimated to be half the cost per instance than the other architectures.

As seen on Table 5, when deploying 96 instances, the cost per instance for ArduinoRL is approximately €140. Of these €140, approximately, €129 are destined to the purchase of the hardware that conforms the remotizable object, and only about €10 are destined to the purchase of the interconnection hardware. In this case, the cost of the interconnection hardware, which is heavily tied to the chosen architecture, only represents a 7.3% of the total instance cost, which is significantly lower than on RELDES and RELLEs RExLab

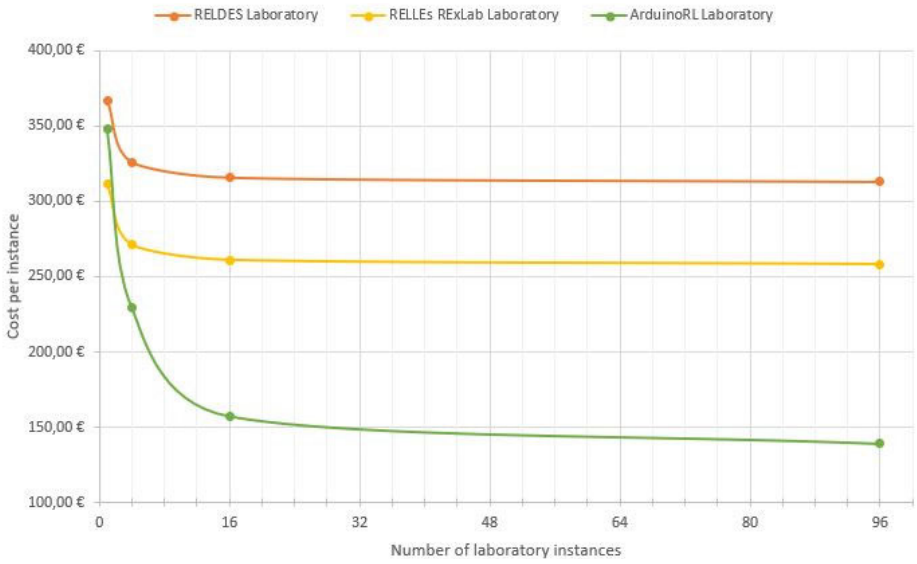

FIGURE 9. Cost per instance for 1, 4, 16 and 96 instances for each analyzed laboratory architecture.

TABLE 5. Architectural and remotizable-object related costs and percentages over the total for a 96 instances deployment.

Laboratory	Total cost (€)	Remotizable object cost (€)	Remotizable object cost percentage over total	Architecture cost (€)	Architecture cost percentage over total
RELDES	313.18	205.54	65.6 %	107.64	34.4 %
RELLEs RExLAB	258.13	216.52	83.9 %	41.61	16.1 %
ArduinoRL	139.04	128.86	92.7 %	10.18	7.3 %

laboratories. This suggests that when multi-instance remote laboratories are developed following an architecture focused on high scalability and cost efficiency, the cost of the hardware directly related to architectural needs can be significantly reduced.

Achieving a minimal cost overhead over the remotizable object's hardware cost is an important action, because this cost difference is multiplied by the number of deployed instances, making the total cost increase quickly when deploying a high number of laboratory instances.

The cost efficiency analysis suggests that the proposed architecture can achieve a minimal overhead cost over the remotizable object's cost, a marker which heavily impacts the total expenses of a multi-instance remote laboratory deployment. It also suggests that the proposed architecture is more cost efficient than the alternatives, specially when the number of deployed instances is high, due in part to the design that allows and encourages hardware sharing among groups of instances.

IX. SCALABILITY EVALUATION

This section will use the proof of concept described in Section VI to analyze the proposed architecture from a scalability point of view.

To ensure that the architecture and the implementation meets all the requirements of real-world schools, ArduinoRL has been tested in a real working environment. A group of students from a Basque secondary school has used the laboratory to perform Arduino practices both in class and at home. Due to the type of practices that they were going to develop, four ArduinoRL instances were sufficient to handle the access of up to 14 users, that were using the laboratory in hourly time brackets. In the next subsection, this in-class experience is reported and analyzed in a more detailed way.

A. IN-CLASS EXPERIENCE

This subsection gathers all the information related to a real test of the architecture that has been conducted during the period of a month. The developed remote laboratory has been used in a real production environment in order to conduct a complete test over the architecture and over the actual multi-instance Arduino remote laboratory, and to verify that it is indeed production-ready.

In the month of March 2019, for a period of 30 days, the ArduinoRL remote laboratory has been put in a monitored production state in order to test its capabilities in an extensive way and to be able to gather different parameters in a real environment.

A group of 14 users from a Basque secondary school, formed by 11 males and 3 females, with ages ranging 15 to 16, have accessed the ArduinoRL remote laboratory in order to learn Arduino-based programming in Technology class.

The main goal of the subject is to teach the principles and applications of basic programming and introduce the students to microcontroller-based device design. The students learn about programming and control routines in Arduino and block-based programming languages at the same time that they learn how to combine programming, microcontrollers and embedded devices with other electric and electronic gadgets in order to create enhanced devices.

The students access the laboratory through the school's own Learning Management System, or LMS, which in this case is Google Classroom. Thanks to the architecture support, there is no need of downloading any utility or performing any reconfiguration. After accessing the integrated IDE, the students program their control routines and create their own control programs. After the development stage, they access the actual remotizable object, they program it, and they observe the programming results through a live-streaming capture. Also, in the period of time assigned to conduct these tests, the students can also interact with the programmed Arduino board by changing different input signals (both analogical and digital) and by intercommunicating with the device via the serial console. Both the programming and testing steps are done through the web-based interfaces that can be seen in Figures 6 and 5, respectively.

After students have tested their software design, the remotizable object session ends, and the students are redirected to the integrated IDE to perform new changes to the code, until it is correct. These programming and testing steps are repeated in a cyclic way as many times as needed, in order to obtain the working-as-intended Arduino program.

It is unusual for the fourteen students to try to access the Arduino instances at the same time, but it is not necessarily rare for many of them to access at the same time, either. Normally, between a quarter and half of the students try to access the instances of the remote laboratory at the same time to test their code. The ratio tends to remain low because the time it takes to write the code is generally much greater than the time it takes to test it. If more than four students try to access the Arduino instances, and these are unoccupied, the first four students will get immediate access, while the successive students are put in a queue. As the instances are released, students in the queue start accessing the Arduino instances. The time that students wait in order to access an unoccupied instance is the access time.

During the 30 day real-time testing period, the students have accessed the laboratory setup 493 times. The students performed 176 accesses to the integrated IDE in order to code different scripts and performed 317 accesses to the actual remotizable objects, in order to test their scripts. Most of these accesses have been done in short high-demanding bursts that last for an hour in most cases, because most of the uses have been done in-class. Around 20% of the accesses correspond to other time frames, because the laboratory was also used to do homework practices outside the school hours. During the high-demanding bursts, the four Arduino remote experimentation instances have maintained a relatively low access time which does not negatively affect usability and which varies in all cases between zero and four seconds, with a mean waiting time of 2.2 seconds. This means that students have only waited a maximum of 4 seconds for an Arduino instance to be released in order to test their code.

Other remote laboratory setups can have different needs, and in some occasions, the number of available experimentation instances and the number of students need to be matched. During the conducted tests, the students are given a testing time of 1 minute and 30 seconds, in which they test the previously written code. Most users spent less time than the maximum testing their programs, achieving a mean testing time of 57 seconds. This testing time can be adjusted by teachers and remote laboratory owners to better suit different student needs.

B. RESULTS

In this subsection, the results of the scalability experiment are discussed. The development of the ArduinoRL remote laboratory, based on the proposed architecture, has been conducted without any particular hiccups, and scalability has been successfully achieved thanks to its decoupled hardware and software layers. From a multi-user perspective, the laboratory has managed to support up to 14 users in a hourly time bracket, with more than 4 concurrent accesses. The users that requested access when all of the four instances were occupied, were put in an access queue. The mean access time, of 2.2 seconds, is manageable in a classroom environment, which reinforces the idea that, for mixed experimentation, where the users spend time both programming and testing, but not doing both tasks at the same time, a low number of instances can successfully handle a much higher number of users. The straightforward development of the remote laboratory, the successful instance allocation management and the low waiting time achieved suggests that the architecture has indeed met its requirements for scalability and multiple concurrent user handling.

X. CONCLUSION

This work, after analyzing the state of the art regarding remote laboratories for embedded systems experimentation and their architectures, has pursued the creation of a new mixed hardware-software architecture focused on improving the scalability of those architectures, which was an aspect beyond most other works scope. This architecture serves as a fundamental resource which facilitates the development and deployment of multi-instance microcontroller-based embedded devices remote laboratories. To better understand how it builds upon the existing architectures in the state of the art and improves some aspects of them, specific goals and requirements were proposed. These goals and requirements have served as fundamental pillars for the design and development of this enhanced remote laboratory architecture. Firstly, it has been designed to provide scalability for remote laboratories with moderated expenses, successfully supporting

multiple instances of remotizable objects in order to reach a larger set of users concurrently. Secondly, the architecture has been designed to provide easy access, both from a hardware interconnection point of view and from a user access point of view. Adaptability and interconnection to different remotizable devices is enhanced due to the high number of supported interconnection protocols. User access is universal, due to the architecture capability of being accessed through the HTTP protocol, using standard ports to avoid firewall problems. Being fully web-based means that the only requirements needed to access the laboratory are a web browser and internet connection. Thirdly, from a usability point of view, the architecture provides different input and output signals, peripherals and connection techniques, which combined with the live-streaming platform, achieve high quality experimentation sessions. Users are able to experiment with the embedded device almost as they would be doing in a hands-on laboratory. Lastly, from a purely architectural point of view, the architecture is fully decoupled, which increases modularity and gives the laboratory owner high flexibility. This allows an easy adaptation of the different hardware and software modules to the physical environment where the laboratory is being deployed. Once designed, an implementation of that architecture has been shaped as a multi-instance remote laboratory for embedded experimentation. The laboratory uses four Arduino UNO boards combined with a number of different peripherals and control means, in order to achieve a high quality remote experimentation experience. This architecture implementation has been used to experimentally evaluate the compliance of the architecture to different key aspects, such as scalability, adaptability, cost efficiency, usability, universality, modularity and reliability. The gathered results suggest that the previously described goals and requirements have been met indeed, and that the proposed architecture should be useful for new development and deployment of remote laboratories, for both academical and applied purposes.

XI. FUTURE WORK

Some work lines remain still open and can be pursued in the future in order to improve the actual architecture. Given the success of the proof of concept, the ArduinoRL remote laboratory is still online and is consired ''production-level''. It is accessible from[11] for free, for testing purposes. Every day, different students and teachers access the laboratory to improve their Arduino knowledge. This allows us to perform a continuous improvement cycle both over the architecture and over the Arduino laboratory, since new requirements and improving options arise frequently. The laboratory is expected to be replicated as needed, in order to support an increasing number of users. In the future it would be interesting to conduct new remote laboratory deployments based on the proposed architecture with other remotizable objects, such as FPGAs or other embedded devices, that could help to better evaluate the scalability and adaptability of the architecture to different hardware devices. Also, other interesting future line of work would be exploring the containerization of the software layer of the architecture, using technologies such as Docker, which could improve the deployability [63] by other third parties and work towards the standardization of the architecture as a tool for the deployment of remote laboratories. Finally, other interesting line of work that can be explored in the future is the development of a self-testing system, which could be added to the architecture, to potentially test each laboratory modules automatically in order to ensure better reliability, by conducting automated tests and notifying the laboratory owners and users for any noted errors or for expected failures through the use of predictive maintenance techniques.

[11] https://labsland.com/en/labs/arduino-board

REFERENCES

[1] J. Ma and J. V. Nickerson, ''Hands-on, simulated, and remote laboratories: A comparative literature review,'' *ACM Comput. Surv.*, vol. 38, no. 3, 2006, Art. no. 7.

[2] Z. Nedic, J. Machotka, and A. Nafalski, ''Remote laboratories versus virtual and real laboratories,'' in *Proc. 33rd Annu. Frontiers Educ.*, vol. 1, Nov. 2003, p. T3E.

[3] C. E. Hmelo-Silver, R. G. Duncan, and C. A. Chinn, ''Scaffolding and achievement in problem-based and inquiry learning: A response to Kirschner, Sweller, and Clark,'' *Educ. Psychol.*, vol. 42, no. 2, pp. 99–107, 2007.

[4] T. de Jong, S. Sotiriou, and D. Gillet, ''Innovations in STEM education: The Go-Lab federation of online labs,'' *Smart Learn. Environ.*, vol. 1, no. 1, 2014, Art. no. 3.

[5] T. de Jong, M. C. Linn, and Z. C. Zacharia, ''Physical and virtual laboratories in science and engineering education,'' *Science*, vol. 340, no. 6130, pp. 305–308, 2013.

[6] S. Govaerts, Y. Cao, A. Vozniuk, A. Holzer, D. G. Zutin, E. S. C. Ruiz, L. Bollen, S. Manske, N. Faltin, C. Salzmann, E. Tsourlidaki, and D. Gillet, ''Towards an online lab portal for inquiry-based stem learning at school,'' in *Proc. Int. Conf. Web-Based Learn.* Berlin, Germany: Springer, 2013, pp. 244–253.

[7] J. R. Brinson, ''Learning outcome achievement in non-traditional (virtual and remote) versus traditional (hands-on) laboratories: A review of the empirical research,'' *Comput. Educ.*, vol. 87, pp. 218–237, Sep. 2015.

[8] P. Orduña, L. Rodriguez-Gil, J. Garcia-Zubia, I. Angulo, U. Hernandez, and E. Azcuenaga, ''LabsLand: A sharing economy platform to promote educational remote laboratories maintainability, sustainability and adoption,'' in *Proc. IEEE Frontiers Educ. Conf. (FIE)*, Oct. 2016, pp. 1–6.

[9] P. Orduña, L. Rodriguez-Gil, J. Garcia-Zubia, I. Angulo, U. Hernandez, and E. Azcuenaga, ''Increasing the value of remote laboratory federations through an open sharing platform: LabsLand,'' in *Online Engineering & Internet of Things*, M. E. Auer and D. G. Zutin, Eds. Cham, Switzerland: Springer, 2018, pp. 859–873.

[10] J. García-Zubia, P. Orduña, D. López-de-Ipiña, and G. R. Alves, ''Addressing software impact in the design of remote laboratories,'' *IEEE Trans. Ind. Electron.*, vol. 56, no. 12, pp. 4757–4767, Dec. 2009.

[11] Z. Aydogmus and O. Aydogmus, ''A Web-based remote access laboratory using SCADA,'' *IEEE Trans. Educ.*, vol. 52, no. 1, pp. 126–132, Feb. 2009.

[12] A. Yazidi, H. Henao, G.-A. Capolino, F. Betin, and F. Filippetti, ''A Web-based remote laboratory for monitoring and diagnosis of AC electrical machines,'' *IEEE Trans. Ind. Electron.*, vol. 58, no. 10, pp. 4950–4959, Oct. 2011.

[13] J. Sáenz, J. Chacón, L. De La Torre, A. Visioli, and S. Dormido, ''Open and low-cost virtual and remote labs on control engineering,'' *IEEE Access*, vol. 3, pp. 805–814, 2015.

[14] F. Schauer, M. Krbecek, P. Beno, M. Gerza, L. Palka, and P. Spilaková, ''REMLABNET—Open remote laboratory management system for e-experiments,'' in *Proc. 11th Int. Conf. Remote Eng. Virtual Instrum. (REV)*, Feb. 2014, pp. 268–273.

[15] A. Maiti, A. D. Maxwell, and A. A. Kist, "Features, trends and characteristics of remote access laboratory management systems," *Int. J. Online Eng.*, vol. 10, no. 2, pp. 30–37, 2014.
[16] P. Orduña, J. Irurzun, L. Rodriguez-Gil, J. García-Zubia, F. Gazzola, and D. López-de Ipiña, "Adding new features to new and existing remote experiments through their integration in WebLab-Deusto," *Int. J. Online Eng.*, vol. 7, no. S2, pp. 33–39, 2011.
[17] D. Lowe, S. Murray, E. Lindsay, and D. Liu, "Evolving remote laboratory architectures to leverage emerging Internet technologies," *IEEE Trans. Learn. Technol.*, vol. 2, no. 4, pp. 289–294, Oct. 2009.
[18] P. Orduña, L. Rodriguez-Gil, D. López-de-Ipiña, and J. García-Zubia, "Sharing the remote laboratories among different institutions: A practical case," in *Proc. 9th Int. Conf. Remote Eng. Virtual Instrum. (REV)*, Jul. 2012, pp. 1–4.
[19] P. Zervas, A. Trichos, D. G. Sampson, and N. Li, "A responsive design approach for supporting mobile access to virtual and remote laboratories," in *Proc. IEEE 14th Int. Conf. Adv. Learn. Technol.*, Jul. 2014, pp. 11–13.
[20] G. Paravati, C. Celozzi, A. Sanna, and F. Lamberti, "A feedback-based control technique for interactive live streaming systems to mobile devices," *IEEE Trans. Consum. Electron.*, vol. 56, no. 1, pp. 190–197, Feb. 2010.
[21] H. Crompton, D. Burke, K. H. Gregory, and C. Gräbe, "The use of mobile learning in science: A systematic review," *J. Sci. Educ. Technol.*, vol. 25, no. 2, pp. 149–160, 2016.
[22] I. Angulo, L. Rodrìguez-Gil, and J. Garcìa-Zubìa, "Scaling up the lab: An adaptable and scalable architecture for embedded systems remote labs," *IEEE Access*, vol. 6, pp. 16887–16900, 2018.
[23] S. D. Bencomo, "Control learning: Present and future," *IFAC Proc. Volumes*, vol. 35, no. 1, pp. 71–93, 2002.
[24] J. G. Zubía and G. R. Alves, *Using Remote Labs in Education: Two Little Ducks in Remote Experimentation*. Bilbao, Spain: Univ. Deusto, 2012.
[25] R. Heradio, L. de la Torre, D. Galan, F. J. Cabrerizo, E. Herrera-Viedma, and S. Dormido, "Virtual and remote labs in education: A bibliometric analysis," *Comput. Educ.*, vol. 98, pp. 14–38, Jul. 2016.
[26] L. de la Torre, M. Guinaldo, R. Heradio, and S. Dormido, "The ball and beam system: A case study of virtual and remote lab enhancement with moodle," *IEEE Trans. Ind. Informat.*, vol. 11, no. 4, pp. 934–945, Aug. 2015.
[27] *UNED UNILabs Remote Laboratories Platform*. Accessed: Sep. 13, 2019. [Online]. Available: https://unilabs.dia.uned.es/?lang=es
[28] W. Hu, Z. Lei, H. Zhou, G. Liu, Q. Deng, D. Zhou, and Z. Liu, "Plug-in free Web-based 3-D interactive laboratory for control engineering education," *IEEE Trans. Ind. Electron.*, vol. 64, no. 5, pp. 3808–3818, May 2017.
[29] *RELLE RExLab Arduino Remote Laboratory*. Accessed: Jul. 23, 2019. [Online]. Available: http://relle.ufsc.br/labs/4
[30] *RELDES Arduino Remote Laboratory*. Accessed: Jan. 15, 2019. [Online]. Available: http://swed.zntu.edu.ua/index.php/iot/lightexp
[31] K. Henke, T. Vietzke, H.-D. Wuttke, and S. Ostendorff, "GOLDi—Grid of online lab devices Ilmenau," *Int. J. Online Eng.*, vol. 12, p. 11, Apr. 2016.
[32] I. Angulo, J. García-Zubía, U. Hernández-Jayo, I. Uriarte, L. Rodríguez-Gil, P. Orduña, and G. M. Pieper, "RoboBlock: A remote lab for robotics and visual programming," in *Proc. 4th Exp. Int. Conf. (exp.at)*, Jun. 2017, pp. 109–110.
[33] M. C. Rodriguez-Sánchez, A. Torrado-Carvajal, J. Vaquero, S. Borromeo, and J. A. Hernandez-Tamames, "An embedded systems course for engineering students using open-source platforms in wireless scenarios," *IEEE Trans. Educ.*, vol. 59, no. 4, pp. 248–254, Nov. 2016.
[34] M. El-Abd, "A review of embedded systems education in the Arduino age: Lessons learned and future directions," *Int. J. Eng. Pedagogy*, vol. 7, no. 2, pp. 79–93, 2017.
[35] S. Werner, A. Lauber, J. Becker, and E. Sax, "Cloud-based remote virtual prototyping platform for embedded control applications: Cloud-based infrastructure for large-scale embedded hardware-related programming laboratories," in *Proc. 13th Int. Conf. Remote Eng. Virtual Instrum. (REV)*, Feb. 2016, pp. 168–175.
[36] V. Fotopoulos, A. I. Spiliopoulos, and A. Fanariotis, "Preparing a remote conducted course for microcontrollers based on Arduino," *Διεθνές Συνέδριο για την Ανοικτή & εξ Αποστάσεως Εκπαίδευση*, vol. 7, no. 5B, pp. 1–8, 2016.
[37] J. P. C. de Lima, L. M. Carlos, J. P. S. Simão, J. Pereira, P. M. Mafra, and J. B. da Silva, "Design and implementation of a remote lab for teaching programming and robotics," *IFAC-PapersOnLine*, vol. 49, no. 30, pp. 86–91, 2016.
[38] P. Anzhelika, G. Olga, E. Ivanov, A. Sokolyanskii, and S. Kurson, "Development and application of remote laboratory for embedded systems design," in *Proc. 12th Int. Conf. Remote Eng. Virtual Instrum. (REV)*, Feb. 2015, pp. 69–73.
[39] P. J. Alexander and N. Radhakrishnan, "Remote lab implementation on an embedded Web server," in *Proc. Int. Conf. Circuits, Power Comput. Technol. (ICCPCT)*, Mar. 2015, pp. 1–5.
[40] H. Mostefaoui and A. Benachenhou, "Design of a remote electronic laboratory," in *Proc. Int. Conf. Interact. Mobile Commun. Technol. Learn. (IMCL)*, Nov. 2015, pp. 160–162.
[41] A. Fernández-Pacheco, S. Martin, and M. Castro, "Implementation of an Arduino remote laboratory with raspberry Pi," in *Proc. IEEE Global Eng. Educ. Conf. (EDUCON)*, Apr. 2019, pp. 1415–1418.
[42] J. Sarik and I. Kymissis, "Lab kits using the Arduino prototyping platform," in *Proc. IEEE Frontiers Educ. Conf. (FIE)*, Oct. 2010, pp. T3C-1–T3C-5.
[43] A. Albiol, A. Corbi, and D. Burgos, "Design of a remote signal processing student lab," *IEEE Access*, vol. 5, pp. 16068–16076, 2017.
[44] P. Liggesmeyer and M. Trapp, "Trends in embedded software engineering," *IEEE Softw.*, vol. 26, no. 3, pp. 19–25, May/Jun. 2009.
[45] M. Maksimović, V. Vujović, N. Davidović, V. Milošević, and B. Perišić, "Raspberry Pi as Internet of Things hardware: Performances and constraints," in *Proc. 1st Int. Conf. Elect., Electron. Comput. Eng.*, 2014, vol. 3, no. 8, pp. 1–6.
[46] P. Plaza, E. Sancristobal, G. Fernandez, M. Castro, and C. Pérez, "Collaborative robotic educational tool based on programmable logic and Arduino," in *Proc. Technol. Appl. Electron. Teach. (TAEE)*, Jun. 2016, pp. 1–8.
[47] P. Jamieson and J. Herdtner, "More missing the Boat—Arduino, Raspberry Pi, and small prototyping boards and engineering education needs them," in *Proc. IEEE Frontiers Educ. Conf. (FIE)*, Oct. 2015, pp. 1–6.
[48] A. Bashir, M. Alhammadi, M. Awawdeh, and T. Faisal, "Effectiveness of using Arduino platform for the hybrid engineering education learning model," in *Proc. Adv. Sci. Eng. Technol. Int. Conf. (ASET)*, Mar./Apr. 2019, pp. 1–6.
[49] H. Zieris, H. Gerstberger, and W. Müller, "Using Arduino-based experiments to integrate computer science education and natural science," in *Proc. KEYCIT*, in Key Competencies in Informatics and ICT, T. Brinda, N. Reynolds, R. Romeike, and A. Schwill, Eds., vol. 1, 2015, pp. 381–389.
[50] E. Keenan, M. Shoushtarian, C. K. Karmakar, and M. Palaniswami, "Developing stem skills using Arduino and heart rate variability analysis," in *Proc. STEM Educ. Conf.*, 2019, pp. 1–67.
[51] J. Carlson, *Redis in Action*. Greenwich, CT, USA: Manning Publications, 2013.
[52] V. A. Alexeev, P. V. Domashnev, T. V. Lavrukhina, and O. A. Nazarkin, "The design principles of intelligent load balancing for scalable WebSocket services used with grid computing," *Procedia Comput. Sci.*, vol. 150, pp. 61–68, Jan. 2019.
[53] S. Chen, X. Tang, H. Wang, H. Zhao, and M. Guo, "Towards scalable and reliable in-memory storage system: A case study with Redis," in *Proc. IEEE Trustcom/BigDataSE/ISPA*, Aug. 2016, pp. 1660–1667.
[54] A. D. J. Banks, G. D. Beardall, P. S. Dennis, A. D. Dick, and I. C. Vanstone, "Improving scalability and throughput of a publish/subscribe network," U.S. Patent 8 495 127, Jul. 23, 2013.
[55] D. Weintrop and U. Wilensky, "Comparing block-based and text-based programming in high school computer science classrooms," *ACM Trans. Comput. Educ.*, vol. 18, no. 1, 2017, Art. no. 3.
[56] P. Orduña, J. Garcia-Zubia, L. Rodriguez-Gil, I. Angulo, U. Hernandez-Jayo, O. Dziabenko, and D. López-de-Ipiña, "The WebLab-Deusto remote laboratory management system architecture: Achieving scalability, interoperability, and federation of remote experimentation," in *Cyber-Physical Laboratories in Engineering and Science Education*. Cham, Switzerland: Springer, 2018, pp. 17–42.
[57] P. Orduña, L. Rodriguez-Gil, I. Angulo, U. Hernandez, A. Villar, and J. Garcia-Zubia, "WebLabLib: New approach for creating remote laboratories," in *Cyber-Physical Systems and Digital Twins*, M. E. Auer and B. K. Ram, Eds. Cham, Switzerland: Springer, 2020, pp. 477–488.
[58] L. Rodríguez-Gil, J. García-Zubia, P. Orduña, and D. Lopez-de-Ipiña, "An open and scalable Web-based interactive live-streaming architecture: The WILSP platform," *IEEE Access*, vol. 5, pp. 9842–9856, 2017.
[59] C. A. Jara, F. A. Candelas, and F. Torres, "Virtual and remote laboratory for robotics e-learning," *Comput. Aided Chem. Eng.*, vol. 25, pp. 1193–1198, Jun. 2008.

[60] H. Vargas, G. Farias, J. Sanchez, S. Dormido, and F. Esquembre, "Using augmented reality in remote laboratories," *Int. J. Comput. Commun. Control*, vol. 8, no. 4, pp. 622–634, 2013.
[61] L. Rodriguez-Gil, P. Orduña, J. García-Zubia, and D. López-de-Ipiña, "Interactive live-streaming technologies and approaches for Web-based applications," *Multimedia Tools Appl.*, vol. 77, no. 6, pp. 6471–6502, 2018.
[62] K. A. Al-Busaidi and H. Al-Shihi, "Instructors' acceptance of learning management systems: A theoretical framework," *Commun. IBIMA*, vol. 2010, no. 2010, pp. 1–10, 2010.
[63] A. Corbi and D. Burgos, "OERaaS: Open educational resources as a service with the help of virtual containers," *IEEE Latin Amer. Trans.*, vol. 14, no. 6, pp. 2927–2933, Jun. 2016.

AITOR VILLAR-MARTÍNEZ received the B.Sc. and M.Sc. degrees in telecommunication engineering from the University of Deusto, in 2016 and 2018, respectively. He is currently pursuing the Ph.D. degree with the University of Deusto, which involve working and researching about remote laboratories and the technologies used in them. Since 2014, he has collaborated with some projects at the University of Deusto, such as Smart Moto Challenge, in 2015 and 2016 editions. Since 2017, he has been a Hardware/Software Developer with the LabsLand (spin-off of the WebLab-Deusto project). Since 2018, he has been collaborating with the MORElab Research Group, DeustoTech—Deusto Institute of Technology.

LUIS RODRÍGUEZ-GIL received the dual degree in computer engineering and industrial organization engineering, the M.Sc. degree in information security, and the Ph.D. degree in computer science from the University of Deusto, in 2013, 2014, and 2017, respectively. During the Ph.D., he co-founded the LabsLand remote labs company, where he is currently a full-time CTO. Since 2009, he has been a part of the WebLab-Deusto Research Group, collaborating in the development of the WebLab-Deusto RLMS. He has authored various peer-reviewed publications and contributed to several open source projects.

IGNACIO ANGULO presented the Ph.D. thesis on Open Architecture for the Deployment of Remote Laboratories on Embedded Systems, in 2015. Since 2002, he has been with the University of Deusto, where he has been with the Department of Information Technology, Electronics and Communication. He has participated over 25 research projects. He has collaborated in the writing of 32 scientific articles published in magazines of international impact.

PABLO ORDUÑA (M'05) received the degree in computer engineering and the Ph.D. degree from the University of Deusto, in 2007 and 2013, respectively. During the Ph.D., he was a Visiting Researcher with the MIT CECI, in 2011, and the UNED DIEEC, in 2012, for six weeks each. He was a Researcher and the Project Manager of the MORElab (DeustoTech Internet), until 2017. He has also attended two programs for entrepreneurship training at Singularity University: Global Solutions Program and Launchpad. Since 2004, he has also been with the WebLab-Deusto Research Group, leading the design and development of WebLab-Deusto. He is currently the CEO with LabsLand (spin-off of the WebLab-Deusto project), and an External Collaborator with the DeustoTech—Deusto Institute of Technology.

JAVIER GARCÍA-ZUBÍA (M'08–SM'11) received the Ph.D. degree in computer science from the University of Deusto, Spain. He is currently a Full Professor with the Faculty of Engineering, University of Deusto. He is the Leader of the WebLab-Deusto Research Group. His research interests include remote laboratory design, implementation, and evaluation.

DIEGO LÓPEZ-DE-IPIÑA received the Ph.D. degree from the University of Cambridge, in 2002. Responsible for several modules in the B.Sc. and M.Sc. degrees in computer engineering, he is interested in pervasive computing, the IoT, semantic service middleware, open linked data, and social data mining. He is currently an Associate Professor and a Principal Researcher with the MORElab Group, Faculty of Engineering, University of Deusto. He is taking and has taken part in several big consortium-based research European (EDI, GreenSoul, WeLive, SIMPATICO, IES CITIES, or GOLAB) and Spanish projects, and has more than 140 publications in relevant international conference and journals, including more than 60 JCR-indexed articles.