UNIVERSITY OF DEUSTO

IMPROVING THE REMOTE LABORATORY EXPERIENCE THROUGH AUGMENTED CHARACTERISTICS BEYOND THE EXPERIMENT CORE

PhD thesis presented by Luis Rodríguez Gil

Directed by Dr. Javier García Zubía and Dr. Pablo Orduña Fernández

Bilbao, June 2017

UNIVERSITY OF DEUSTO

IMPROVING THE REMOTE LABORATORY EXPERIENCE THROUGH AUGMENTED CHARACTERISTICS BEYOND THE EXPERIMENT CORE

PhD thesis presented by Luis Rodríguez Gil

for the *Engineering for the Information Society and Sustainable Development* PhD program

Directed by Dr. Javier García Zubía and Dr. Pablo Orduña Fernández

The PhD candidate The advisors

Bilbao, June 2017

Author: Luis Rodríguez Gil
Advisors: Dr. Javier Garcia Zubia and Dr. Pablo Orduña Fernández

Text printed in Bilbao
First edition, June 2017

To my family

Abstract

Educational remote laboratory technologies allow students to access distant equipment, through the Internet. They can experiment and learn in a way that resembles the traditional hands-on laboratories of any educative institution. This not only provides convenience. Through remote laboratories, institutions can reduce their costs and expand their educational offer. They can share resources and reduce underusing of their laboratories. But not everything are advantages. The laboratory experience is different. A common topic in the literature is the comparison among hands-on, remote and virtual laboratories, and their effectiveness. Although there is no definite answer, results suggest that all of them, when properly designed and implemented, can provide satisfactory results. The value they give is, presumably, the sum of the different components that comprise their educative experience: the laboratory itself, the interaction between the students and the equipment, the guidance that teachers provide, and other characteristics. The goal of this dissertation is to improve the remote laboratory experience, adding value and leveraging advances in different fields to improve the aforementioned characteristics. For this, it adopts a three-pronged approach.

The first part of this thesis improves the quality of the interaction between users and remote laboratories, so that it can be closer to that of a real hands-on laboratory. A webcam is the window through which students interact with the equipment. Thus, this work analyzes the different interactive live-streaming approaches that are available, and designs and implements an architecture to satisfy the requirements effectively.

The second part of this thesis improves the guidance that students receive. In a traditional hands-on laboratory students benefit from the physical presence of the teacher. In a remote laboratory, only remote assistance can be provided. This work proposes and develops a new model of customizable tutoring agents that can be customized by teachers through a visual language. Through this, they can augment that guidance.

The third part of this thesis aims not to match, but rather to augment a remote laboratory beyond the limitations of a traditional hands-on one. One of the traditional advantages of virtual laboratories (simulations) is that they can simulate those realities that are not possible in a conventional hands-on laboratory, due to practical, financial or security reasons. This works explores the

hybrid laboratory model: remote laboratories that leverage not only real, but also, virtual components.

This three-pronged approach aims to advance the state of the art towards new generations and models of online laboratories, that can not only match conventional hands-on laboratories, but surpass them.

Resumen

Las tecnologías de laboratorios remotos educativos permiten acceder a equipamiento distante, a través de Internet. Los estudiantes experimentan y aprenden de forma similar a como lo harían en los laboratorios tradicionales de cualquier institución educativa. Esto no sólo aporta comodidad. Gracias a los laboratorios remotos, las instituciones pueden reducir sus costes y aumentar su oferta educativa, compartiendo recursos y mejorando la utilización de sus laboratorios. Pero no todo son ventajas. La experiencia de aprendizaje es diferente, y recurrentemente en la literatura surje la compración entre laboratorios reales, remotos, y simulados. No existe respuesta clara pero los resultados sugieren que todos ellos, correctamente implementados, pueden dar resultados satisfactorios. El valor que aportan es, presumiblemente, fruto de la suma de los componentes que forman su experiencia educativa: El propio laboratorio, la interacción del estudiante con el equipamiento, la guía que proporciona el profesor, y otras características. El objetivo de esta tesis es aumentar el valor de los laboratorios remotos, aprovechando y desarrollando avances en varios campos para mejorar dichas características. Para ello se adopta un triple enfoque.

En una primera parte, el objetivo es aumentar la calidad de la interacción con los laboratorios remotos, de tal forma que sea más equiparable a la de un laboratorio real. La ventana a través de la cual el estudiante interactúa con el equipamiento es la webcam. Por ello, en la tesis se analizan los diferentes enfoques disponibles y se diseña e implementa una arquitectura que satisfaga de forma más efectiva los requisitos.

En una segunda parte, el objetivo es mejorar la guía que el estudiante recibe. En un laboratorio tradicional el estudiante cuenta con la presencia física del profesor. En un laboratorio remoto debe contentarse con asistencia remota. En la tesis se propone y desarrolla un nuevo modelo de agentes conversaciones personalizables por profesores a través de un lenguaje visual. Con ello, pueden complementar dicha asistencia.

En la tercera parte, el objetivo no es igualar o equiparar una características, sino en aportarlas más allá de lo que posibilita la realidad de un laboratorio tradicional. Una de las ventajas tradicionales de los laboratorios virtuales (no remotos) es que pueden simular aquellas realidades que por limitaciones prácticas, económicas o de seguridad son inviables en uno tradicional. En la tesis se explora el modelo de los laboratorios híbridos: laboratorios remotos que incorporan, además, elementos virtuales.

Con este triple enfoque se espera avanzar el estado del arte hacia nuevas generaciones y modelos de laboratorios online, que puedan no solamente ser equiparables a laboratorios tradicionales sino incluso superiores.

Acknowledgements

There are many people without whom this work would not have been possible. First, I would like to thank my co-advisors, Javier García Zubia and Pablo Orduña Fernández. This work has only reached its completion through their unceasing support, guidance and hard-work.

Next, I would like to thank other members of the WebLab-Deusto research team and of the LabsLand company. Particularly, I would like to thank Ignacio Angulo, Unai Hernández and Esteban Azcuénaga. It is a great experience working with all of you.

I would also like to express my gratitude to the various anonymous reviewers who have provided valuable inputs for the articles that comprise this dissertation, and which have truly bolstered its quality.

I would like to thank Diego López de Ipiña, as leader of the DeustoTech Internet unit, for providing the resources, environment and support that were required for this work. Also, my co-workers and other members of the MORElab group (Aitor, Unai, and all others), who have always provided valuable comments and always been willing to attend presentations and rehearsals.

I am also very thankful to my family, who has accompanied me through this journey. Particularly, to Luis, my father, who has even proof-read parts of this work; and to my mother Cristina, my sister Virginia and my grandparents Claudio and Elisa, who have supported me throughout all of these years.

Many thanks, also, to my friends. To Victoria, David, Jon, Aitor, Lucia and Gabriela; and to those whose names, for brevity, I do not explicitly include here.

Finally, I would like to thank the Department of Education, Language policy and Culture of the Basque Government, who have funded this work through a Predoctoral Scholarship.

Thank you very much,

Luis Rodríguez Gil

June 2017

Contents

List of Figures xiii

List of Tables xv

Acronyms xvii

1 Introduction 1
- 1.1 Thesis Statement . . . 4
- 1.2 Main research questions . . . 5
- 1.3 Structure and general methodology . . . 5
- 1.4 Contributions . . . 5
  - 1.4.1 Main scientific contributions . . . 6
  - 1.4.2 Main technical contributions . . . 7
- 1.5 Publications . . . 7
  - 1.5.1 Paper I . . . 7
  - 1.5.2 Paper II . . . 8
  - 1.5.3 Paper III . . . 9
  - 1.5.4 Paper IV . . . 9
  - 1.5.5 Paper V . . . 9
  - 1.5.6 Previous publications . . . 10
- 1.6 Outline . . . 10

2 Interactive Live-Streaming for Remote Laboratories 11
- 2.1 Introduction . . . 12
- 2.2 Choosing interactive live-streaming approaches . . . 12
  - 2.2.1 Requirements and criteria . . . 13
  - 2.2.2 Approaches . . . 13
  - 2.2.3 Experiments and comparison . . . 14
  - 2.2.4 Results . . . 14
  - 2.2.5 Conclusions . . . 17
- 2.3 Goals and requirements . . . 17
- 2.4 Architecture . . . 18

2.4.1 Layers . . . 20
2.4.1.1 Input sources layer . . . 20
2.4.1.2 Redis cluster layer . . . 20
2.4.2 CamServers layer . . . 20
2.4.3 Browser layer . . . 21
2.5 Evaluation . . . 21
2.5.1 Methodology . . . 21
2.6 Results . . . 23

3 Teacher-defined tutoring agents 27
3.1 Introduction . . . 27
3.2 Purpose and research questions . . . 29
3.3 Background . . . 29
3.3.1 Agent authoring tools and non-programmers . . . 29
3.3.2 Visual programming languages . . . 30
3.4 Requirements . . . 31
3.5 Visual Domain-Specific Language . . . 31
3.6 Strengths and weaknesses . . . 32
3.7 Platform architecture . . . 32
3.7.1 Key technologies . . . 32
3.7.2 Perspectives . . . 33
3.8 Methodology . . . 34
3.9 Results . . . 35
3.9.1 How easy to learn is the proposed approach? . . . 35
3.9.2 How usable is the proposed approach? . . . 35
3.9.3 Is the proposed approach perceived as valuable and intuitive? . . . 36
3.10 Discussion . . . 36
3.11 Conclusions . . . 37

4 Hybrid laboratory models 39
4.1 Introduction . . . 39
4.2 Goals . . . 41
4.3 Architecture . . . 41
4.3.1 Client side . . . 42
4.3.2 Server side . . . 42
4.4 Architectures comparison . . . 42
4.5 Watertank FPGA laboratory . . . 44
4.6 Evaluation . . . 45
4.6.1 Technical evaluation . . . 45
4.6.2 Prospects and satisfaction survey . . . 45
4.7 Conclusions . . . 46

5 Conclusions and outlook **47**
5.1 Conclusions 47
5.2 Outlook 48

Bibliography **51**

A Paper I: ***Interactive live-streaming technologies and approaches for web-based applications*** **61**

B Paper II: ***An open and scalable web-based interactive live-streaming architecture: The WILSP platform*** **95**

C Paper III: ***Teacher-defined tutoring agents: Authoring through a visual domain-specific language*** **113**

D Paper IV: ***Hybrid laboratory for rapid prototyping in digital electronics*** **129**

E Paper V: ***Towards new multiplatform hybrid online laboratory models*** **149**

List of Figures

1.1 Characterization of some key components that form the educative experience that hands-on and remote laboratories provide. 3
1.2 Outline of the main parts of this dissertation and their related publications. . 3

2.1 Measuring the capture-render delay. Computer on the right is rendering the interactive live-stream from the IP camera (on the lower left.) 15
2.2 Client-side RAM usage comparison) 15
2.3 Client-side CPU usage comparison) 16
2.4 Client-side downstream bandwidth usage comparison 16
2.5 Architectural overview of the proposed platform and its components. 19
2.6 Interactive live-stream with an embedded QR timestamp to measure the true capture-render delay. 22
2.7 Results of Experiment 1. Feeder component using the image refreshing technique. 23
2.8 Results of Experiment 2. Feeder component using the H.264 format technique. 24
2.9 Results of Experiment 3. CamServer component using the image refreshing format technique. 24
2.10 Results of Experiment 4. CamServer component using the H.264 format technique. 25

3.1 Embedding a teacher-defined agent into a VISIR remote lab. 29
3.2 The Scratch visual programming language and environment. 30
3.3 Authoring Tool's main view. 33
3.4 Authoring system architecture overview. 34
3.5 Experiment stages that the participants follow. 34
3.6 Average time in seconds that participants spent in each stage (n=32). Error bars show 95% confidence intervals. 35

4.1 Hybrid laboratories within a characterization of labs (adapted from (Dormido Bencomo, 2004) and (Gomes and Bogosyan, 2009)) 40
4.2 Proposed hybrid laboratory architecture. 42
4.3 FPGA-Watertank components model 44

4.4 Results of Q1.2: perceived learning potential of RLs / ARLs ($n = 56$, $\overline{x} = 3.464$) 46

List of Tables

1.1 Summary of scientific and technical contributions 6

3.1 Results of the custom survey Q5-Q8 (n=32) 36

4.1 Comparison of the architecture characteristics 43
4.2 Results of the posterior survey Q2.5 — interest on the technologies (n=53; scale: 1-4) 46

Acronyms

AIML Artificial Intelligence Markup Language

AJAX Asynchronous JavaScript And XML

API Application Programming Interface

ARL Augmented Remote Laboratory

CA Conversational Agent

CTAT Cognitive Tutor Authoring Tools

DSL Domain-Specific Language

ECA Embodied Conversational Agent

H.264/AVC H.264 Advanced Video Coding

HTTP Hyper-Text Transfer Protocol

ID Identifier

IDE Integrated Development Environment

IEEE Institute of Electrical and Electronics Engineers

IP Internet Protocol

ITS Intelligent Tutoring System

JPEG Joint Photographic Experts Group

JSON JavaScript Object Notation

LMS Learning Management Systems

M-JPEG Motion-JPEG

MOOC Massively Open Online Course

MPEG Moving Picture Experts Group

MPEG-1 Moving Picture Experts Group 1

NLP Natural Language Processing

NTP Network Time Protocol

QR-Code Quick Response Code

RIA Rich Interactive Application

RL Remote Laboratory

RLMS Remote Laboratory Management System

SUS System Usability Scale

UMUX Usability Metric for User Experience

WebRTC Web Real Time Communication

WILSP Web-based Interactive Live-Streaming Platform

XML eXtensible Markup Language

The true method of knowledge is experiment.

William Blake

CHAPTER

1

Introduction

EDUCATIONAL technologies keep growing, empowered by the latest technical and scientific developments. Among those, a key technology are educational online laboratories (Ma and Nickerson, 2006; Corter et al., 2007; Nedic et al., 2003; de Jong et al., 2013; Heradio et al., 2016). They allow students to access equipment through the Internet. In the case of online virtual laboratories that equipment is simulated. In the case of online remote laboratories, however, the equipment is real. Students can access it remotely, view the equipment through one or more webcams, and even interact with it through virtualized controls (García-Zubia et al., 2009; Gomes and Bogosyan, 2009; Harward et al., 2008).

Remote laboratories offer significant advantages. Factors that support their use include cost-efficiency, security, reliability, flexibility and convenience (Lowe et al., 2009; Nedic et al., 2003). Students are no longer geographically constrained. They can access them from anywhere, anytime. Often, they will be able to access them using only their browsers, or even their mobile devices. Institutions can use them to reduce costs while increasing the value they offer to their students. They can share equipment with other institutions, thus increasing their educational offer at little cost (Tawfik et al., 2014; Orduña et al., 2014; Lowe et al., 2016; Orduña, 2013). Remotely available equipment can be easier to manage. It does require some maintenance, but constant supervision is no longer required. Students can use it safely on their own. Because use of the laboratory is no longer restricted to the standard working hours of the University, it can be used around the clock, by a higher number of people. This reduces underuse of the equipment.

Despite these advantages, a recurring question in the literature has been to what extent remote and virtual labs are effective, especially when compared to hands-on laboratories. There is no definite answer, but the results of various studies suggest that when properly designed and implemented, remote and virtual laboratories can be at least as effective as hands-on laboratories (Ma and Nickerson, 2006; de Jong et al., 2013; Tzafestas et al., 2006;

Corter et al., 2004; Nedic et al., 2003; Corter et al., 2011; Brinson, 2015). Although those types of laboratory can thus all meet similar practical and educational goals, the learning experience of each of them is different. They have different strengths and weaknesses, and different capabilities and limitations. For instance, an advantage of virtual laboratories is that they can adapt reality (de Jong et al., 2013). They can highlight the most relevant information, modify the time scale, or make unobservable phenomena visible (Trundle and Bell, 2010; Ford and McCormack, 2000). An advantage of hands-on laboratories is that students can learn to deal with unanticipated complexities, such as measurement errors (Toth et al., 2009). An advantage of remote laboratories is that they can provide a high realism and a fast and reliable setup (Corter et al., 2011; Nickerson et al., 2007).

The value that a laboratory provides is thus, presumably, the sum of the various components and affordances that form the particular educative experience that it offers. Figure 1.1 characterizes some of the components that form it, and how they differ in hands-on and remote laboratories. As the figure shows, the experiment core is essentially the same. Students, either directly or remotely, access a particular equipment and experiment with it. The interaction between users and the laboratory, however, is quite different. In a hands-on laboratory, students physically interact with the hardware. They can view it with their own eyes, and see the results of their own actions. They control it with their own hands. In a remote laboratory, however, they most often need to rely on a live-stream provided by a webcam to view the hardware, and they control it through virtualized controls (García-Zubia et al., 2009; Hashemian and Riddley, 2007; Jara et al., 2008). Teacher guidance is other significant aspect that determines the value of the laboratory experience. In hands-on laboratories, teachers and assistants are often available to provide direction, ensure that the students are conducting the experiment properly, and resolve problems and misunderstanding that arise. Comparing the different types of laboratory, Ma and Nickerson (2006) conclude that "it is clear that students learn not only from equipment, but from interactions with peers and teachers". In remote laboratories, teacher guidance can also be available, but generally it will need to be provided remotely. As previously described, research suggests that, overall, remote laboratories can be as effective as hands-on ones. However, it can be observed that at the previously described aspects, remote laboratories are at a disadvantage. Remote interaction is more difficult than quality hands-on interaction, and remote guidance is more difficult than in-person guidance. Therefore, the main goal of this dissertation is to increase the value of the remote laboratory experience by considering these aspects that go beyond the core experiment, and that are part of the educational experience that they offer.

In order to accomplish this goal, the thesis adopts a three-pronged approach. For each part, recent technical and scientific advances will be leveraged to forward the state of the art and improve the particular characteristic. An attempt is made to rely on the most appropriate methodology and evaluation for each of them, and an effort is made to ensure that, when possible, the advances are not only applicable to remote laboratories, but also to related fields that share the same needs. Figure 1.2 outlines the main parts of this dissertation and their related publications.

The first part of this thesis aims to increase the quality of the interaction between the

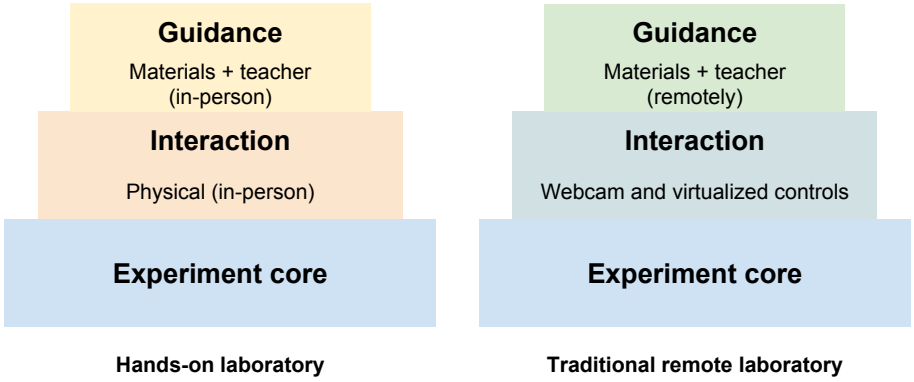

Figure 1.1: Characterization of some key components that form the educative experience that hands-on and remote laboratories provide.

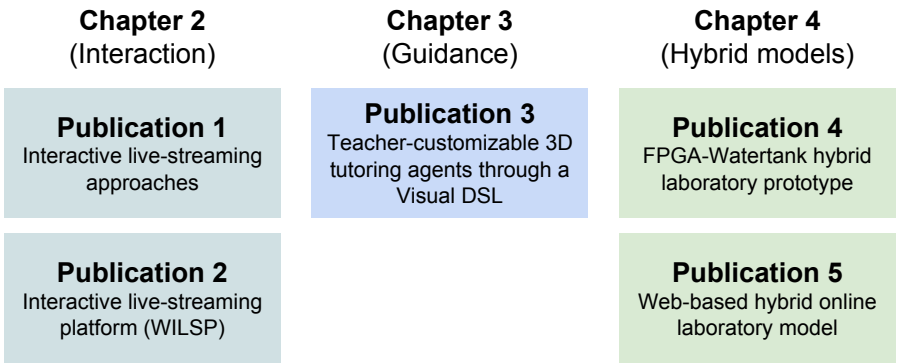

Figure 1.2: Outline of the main parts of this dissertation and their related publications.

students and the laboratory, so that it is closer to the one a hands-on laboratory provides. These improvements are centered in the interactive live-streaming (Zhang and Liu, 2015; Paravati et al., 2010) capabilities (through one or several webcams) that remote laboratory systems require (Yazidi et al., 2011; Jara et al., 2008). Traditionally, little attention has been dedicated to this particular aspect, but live-streams are the window through which students interact with the equipment. Thus, it is key to their user experience. This part of the thesis is centered around two publications. The first publication analyzes the interactive live-streaming needs of remote laboratories, and its requirements. From a mainly client-side perspective, it determines the most appropriate web-based interactive live-streaming schemes that are currently available and that meet these requirements. The second publication, from a mainly server-side perspective, designs, implements, and evaluates an open, web-based, interactive live-streaming architecture. The platform that results from that architecture is published as Open Source, and designed to be useful and satisfy the needs of remote laboratory researchers and developers.

The second part of this thesis is centered in the guidance aspect of a remote laboratory learning experience. In a traditional remote laboratory, students benefit from the physical presence of the teacher. In a remote laboratory, guidance can only be provided remotely. Research suggests that, all else being equal, face-to-face tutoring is generally more effective (Price et al., 2007). Nonetheless, with an increasing number of students (which decreases the attention that the teacher can dedicate to a single student), tools such as intelligent tutoring systems can become nearly as effective as human tutoring (VanLehn, 2011). This part is centered around one publication. It proposes a web-based platform and a visual Domain-Specific Language (DSL) that have been designed to allow non-programming users (such as teachers) to easily define their own 3D tutoring agents, and to integrate them into remote laboratories without requiring assistance from the laboratory developers.

The two previously described parts are oriented to compensate certain aspects in which remote laboratories are at a disadvantage compared to hands-on laboratories. The third part aims to research characteristics that go beyond those that a traditional hands-on laboratory can provide. As previously described, virtual (simulation-only) laboratories have certain strengths: they can adapt reality (de Jong et al., 2013) and they are typically less costly (Nedic et al., 2003; Valera et al., 2005; de Jong et al., 2013). This part explores and advances the concept of hybrid laboratories. Though there is no consensus on the precise definition of *hybrid*, this works considers as hybrid those laboratories that include both real and virtual (simulated) components (Gomes and Bogosyan, 2009). This part is centered around two publications. The first one presents an early example of hybrid remote laboratory for FPGA programming, which allows users to program a real FPGA that controls a virtual industrial water tank simulation. The second publication analyzes relevant hybrid laboratory models and their architectures. With the knowledge obtained from the early prototype and from that analysis, it proposes a hybrid online laboratory model that combines real and virtual components that interact among them. This model is evaluated against a new remote laboratory implementation, that relies on modern technologies such as Unity and which is fully web-based and cross-platform.

1.1 Thesis Statement

The value that a remote laboratory experience provides is not only determined by the core laboratory experience. It is also determined by the quality of the interaction between the students and the equipment, by the guidance that the students receive, and by additional functionalities offered by the laboratory.

Therefore, in order to increase the value offered by a remote laboratory, three areas can be improved. First, interaction can be improved through a web-based interactive live-streaming architecture that satisfies the particular requirements of remote laboratories. Second, student guidance can be improved by designing a platform and a visual domain-specific language that allows teachers to create and customize their own conversational agents. Third, additional value can be provided to students by developing new hybrid laboratory models that leverage virtual and real components.

1.2 Main research questions

This section enumerates the main research questions that this dissertation answers. Some of the questions lead to more concrete subquestions, which are answered in the publications that comprise this thesis, and which are summarized in the following chapters. The main research questions are the following:

1. Which are the most effective interactive live-streaming schemes for remote laboratories?

2. Can an interactive live-streaming architecture support the most effective schemes, while providing universality and scalability?

3. Can a model in which teachers rely on a visual domain-specific language to define their own tutoring agents, be used to improve the remote laboratory experience?

4. Can the remote laboratory experience be augmented through a hybrid architecture that mixes real and virtual components that interact with each other, while meeting the universality, security and power requirements of remote laboratories?

1.3 Structure and general methodology

This dissertation is comprised by five publications, which are listed in Section 1.5. Its main goal is to improve the remote laboratory experience. This is done in three different ways: improving the interaction between the students and the equipment, providing new means of teacher guidance, and providing a new hybrid laboratory model. A chapter is dedicated to each of these aspects. Each of those chapters summarizes the publication or publications that are dedicated to that same aspect. Each publication is self-contained. Therefore, each of them includes its own State of the Art section, proposes its research questions, and describes the particular methodology that it applies, which is tailored to these specific questions. The full outline of the thesis is described in Section 1.6.

Table 1.1: Summary of scientific and technical contributions

		Focus	Research question	Chapter	Publication (Appendix)
Scientific	SC 1	Interaction	1	Chapter 2	I (A)
	SC 2	Interaction	2	Chapter 2	II (B)
	SC 3	Guidance	3	Chapter 3	III (C)
	SC 4	Guidance	3	Chapter 3	III (C)
	SC 5	Hybrid models	4	Chapter 4	IV, V (D, E)
Technical	TC 1	Interaction	2	Chapter 2	II (B)
	TC 2	Guidance	3	Chapter 3	III (C)
	TC 3	Guidance	3	Chapter 3	III (C)
	TC 4	Hybrid models	4	Chapter 4	IV, V (D, E)
	TC 5	Hybrid models	4	Chapter 4	IV, V (D, E)

1.4 Contributions

This section summarizes the main contributions of this dissertation. Though the focus of a dissertation are its scientific contributions, several significant technical ones have also been made, and are included here. Table 1.1 outlines the contributions and indicates their focus and lists which research question they are related to, in which chapters they are described, and in which publications they are included.

1.4.1 Main scientific contributions

- **SC 1:** Study of the current interactive live-streaming schemes, analysis of the requirements for remote laboratories, and experimental comparison of the most relevant ones.
- **SC 2:** Architecture for a web-based, interactive live-streaming platform. It is designed to maximize scalability, has the Redis memory-based store engine at its core, and is designed for the needs of remote laboratories.
- **SC 3:** A visual domain-specific language oriented to non-programmers that can be used to define intelligent tutors. The language is based on Blockly and oriented towards non-programmers.
- **SC 4:** A method and web-based model for non-programmers to define their own intelligent tutors, relying on the aforementioned visual language; and to embed them into external resources. The tutors can act as guides for remote labs and other resources, and teachers do not need help from the laboratory developers to embed them.

- **SC 5:** A novel hybrid laboratory model that augments remote laboratories with virtual components, in which the real and simulated parts can interact with each other to save costs, make the laboratories more interesting, or for other purposes.

1.4.2 Main technical contributions

- **TC 1:** Implementation for WILSP (Web-based Interactive Live-Streaming Platform). Based on the designed and evaluated interactive live-streaming architecture. Released as Open Source[1]. Implemented with Python, Flask, Gevent, Redis and JavaScript-based technologies.

- **TC 2:** A web-based platform that allows teachers to define their own intelligent tutors and embed them into their remote laboratories or learning resources. Teachers use the previously described DSL to define the agents. Both the platform and the agents are designed to be fully web-based. They can act as guides for remote laboratories. Teachers can embed them on their own, without needing explicit help from the original developers of the learning resources. The client-side has been implemented in HTML5 and JavaScript based technologies. The server side has been implemented with Python Django and MySQL.

- **TC 3:** A 3D agents engine component. Developed in Unity3D and exportable to WebGL. Provides the actual 3D agent, chat and bubbles widgets, and the specific interpreter engine.

- **TC 4:** FPGA-Watertank remote laboratory. The server-side has been implemented in Python an integrated with the WebLab-Deusto RLMS. The client-side is fully web-based, relying on JavaScript and Angular JS. Additionally, it integration the watertank model simulation component.

- **TC 5:** Virtual simulation model component. Designed to be integrable into hybrid laboratories and to make the design of various simulated models easier. The watertank model simulation has been developed with Unity3D and is exported to WebGL. The 3D models have been created with Blender.

1.5 Publications

This thesis is a collection of five publications. Four of them are journal articles and one of them is a book chapter. Three of the journal articles have been published in JCR-indexed journals. The book chapter has been published by IFSA Publishing.The fourth journal article is currently under review. In this section, these publications are listed and briefly introduced. The publications are included in the appendices.

[1] https://github.com/zstars/wilsp

1.5.1 **Paper I**

Paper I (Appendix A) is titled '*Interactive live-streaming technologies and approaches for web-based applications*'. It has been published in Springer's '*Multimedia Tools and Applications*'. From a client-side perspective, it proposes several web-based interactive live-streaming approaches and determines which ones are the most effective for remote laboratories.

Often, remote laboratory researchers and developers do not dedicate much attention to the interactive live-streaming scheme of the laboratory. This can very negatively affect the users' experience, because that stream is often the main way through which users interact with the laboratory. The purpose of **Paper I** was to scientifically evaluate which of the currently used schemes are more effective, and whether certain novel schemes might provide better performance. To do this, the existing and the novel schemes are described. Then, a study was conducted, measuring and comparing their performance.

L. Rodriguez-Gil, P. Orduña, J. Garcia-Zubia, D. Lopez-de-Ipina. "Interactive live-streaming technologies and approaches for web-based applications." Multimedia Tools and Applications (2016). DOI: 10.1007/s11042-017-4556-6.

"Multimedia Tools and Applications" JCR IF (2016): 1.530.
Q2 in Computer Science, Software Engineering.
Q2 in Computer Science, Theory & Methods.

1.5.2 **Paper II**

Paper II (Appendix B) is titled '*An open and scalable web-based interactive live-streaming architecture: The WILSP platform*', and it extends **Paper I**. It has been published in '*IEEE Access*'. Focusing on the schemes that deliver the best results, it introduces a new architecture for effective and scalable web-based interactive live-streaming, implementing it as an open source platform.

The better known live-streaming platforms are commercial platforms and are unsuitable for the purposes of remote laboratory research and development. The capture-render delay that they have tends to be too high, which is only appropriate for streams with low interaction, such as live sports. Their architectures and platforms are closed-source, so researchers cannot easily learn from them, and they cannot be adapted. They are not designed to be integrated into external systems such as remote laboratories. The main goal of **Paper II** is to propose and architecture and platform that solves these shortcomings. It is specifically oriented towards interactive-level live-streaming. It is based on open technologies (such as Redis), so that it can be freely used and adapted. It is designed to be integrable into external systems, and to satisfy the particular requirements of remote laboratories.

L. Rodriguez-Gil, J. Garcia-Zubia, P. Orduña, D. Lopez-de-Ipina. "An Open and Scalable Web-Based Interactive Live-Streaming architecture: The WILSP Platform." IEEE Access (2017). DOI: 10.1109/ACCESS.2017.2710328.

"IEEE Access" JCR IF (2016): 3.244.
Q1 in Computer Science, Information Systems.
Q1 in Engineering, Electrical & Electronic.

1.5.3 **Paper III**

Paper III (Appendix C) is titled '*Teacher-defined tutoring agents: Authoring through a visual domain-specific language*'. It is currently under review in an IEEE journal. It introduces a novel method for non-programmers to define tutoring agents, that relies on a visual domain-specific language. It presents a web-based platform that provides that functionality.

In a hands-on laboratory teachers are often there to guide the students. In a remote laboratory they need to guide them remotely, if at all. The main goal of **Paper III** is to provide teachers with a complimentary way to provide guidance: defining their own tutoring agents through a visual blocks-based language that is very easy to understand. The paper describes that novel approach that is tailored to non-programmers. It implements it in a platform, and evaluates it through a user study.

L. Rodriguez-Gil, J. Garcia-Zubia, P. Orduña, D. Lopez-de-Ipina. "Teacher-defined tutoring agents: Authoring through a visual domain-specific language" Submitted for publication. (Under review on June 26, 2017; submitted March 3, 2017).

1.5.4 **Paper IV**

Paper IV (Appendix D) is titled '*Hybrid laboratory for rapid prototyping in digital electronics*'. It has been published as a book chapter in the book '*Online Experimentation: Emerging Technologies and IoT*'. It presents a prototype hybrid remote laboratory for rapid prototyping, which mixes virtual and real components.

Remote laboratories are a useful technology for embedded systems education. In them, users can upload their code into an already-setup remote board, and interact with it to verify that their code behaves as they intend. However, the exercises are not always engaging. Industrial hardware is expensive, so the input and output peripherals tend to be simplistic: LEDs, butotns and switches. The main goal of this paper was to present a laboratory for rapid prototyping that implements a new concept: combining both real and virtual components. Then, the experience can provide a high realism (the controller is real) but be more engaging (with a virtual industrial model to control).

L. Rodriguez-Gil, J. Garcia-Zubia, P. Orduña, I. Angulo, D. Lopez-de-Ipina. "Hybrid laboratory for rapid prototyping in digital electronics." Online Experimentation: Emerging Technologies and IoT, IFSA Publishing, 2015. ISBN: 978-84-608-5977-2.

1.5.5 Paper V

Paper V (Appendix E) is titled '*Towards new multiplatform hybrid online laboratory models*', and it extends **Paper IV**. It has been published in '*IEEE Transactions on Learning Technologies*'. It analyzes existing hybrid laboratories and proposes a novel architecture for hybrid laboratories in which real components interact with virtual ones in real time. It presents the FPGA-Watertank hybrid laboratory, that implements that architecture.

There are currently many hybrid laboratory models. These models are very different, aim for different goals, and implement different architectures. Some models are based on gamification. Other models are based on virtual worlds, or on augmented reality. The main goal of this paper was to classify these models and extract the main architectures, in order to propose a novel one. The proposed architecture is web-based, avoids some of the analyzed pitfalls, and is oriented towards a specific type of hybrid laboratories: those in which the virtual and hybrid components interact with each other in real-time. The proposed architecture is used to implement a particular remote lab: FPGA-Watertank. In it, users can program a real FPGA device, interact with it, and control a virtual industrial watertank. A user study is conducted to evaluate the laboratory and the proposed architecture.

L. Rodriguez-Gil, J. Garcia-Zubia, P. Orduña, D. Lopez-de-Ipina. "Towards new multiplatform hybrid online laboratory models" IEEE Transactions on Learning Technologies (2016). DOI: 10.1109/TLT.2016.2591953.

"IEEE Transactions on Learning Technologies" JCR IF (2016): 2.267.
Q2 in Computer Science, Interdisciplinary Applications.
Q1 (SSCI) in Education & Educational Research.

1.5.6 Previous publications

This dissertation is formed by the 5 aforementioned publications. However, it is noteworthy that, as of this date, as a member of the WebLab-Deusto research team, I have co-authored over 40 conference and journal contributions in the field of remote laboratories.

1.6 Outline

The thesis is divided into 5 chapters and 5 appendixes. After this introduction (Chapter 1), three chapters summarize the five publications that comprise this dissertation. Chapter 2 focuses on adding value to the laboratory experience by improving interactive live-streaming capabilities and presenting a novel interactive live-streaming architecture. Chapter 3 focuses on adding value to the laboratory experience by improving the guidance that students receive. It presents a novel model, based on a visual domain-specific language, that enables teachers and non-programmers to define their own tutoring agents. Chapter 4 focuses on adding value to the remote laboratory experience by leveraging advantages from virtual laboratories. A new hybrid model that combines both virtual and real components is presented. Chapter 5 summarizes the conclussions of this thesis.

The papers that comprise this thesis (see Section 1.5) are appended. Appendix A and B contain **Paper I** and **Paper II** respectively, both about interactive live-streaming. Appendix C contains **Paper III**, about the novel tutoring agent definition method. Appendix D and E contain **Paper IV** and **Paper V** respectively, both about the novel hybrid online laboratory model.

A camera is a tool for learning how to see without a camera

Dorothea Lang

CHAPTER

2

Interactive Live-Streaming for Remote Laboratories

THE interactive live stream that a webcam provides is the window through which users interact with most remote laboratories. In a conventional hands-on laboratory users are face to face with the equipment. But in a remote one they need to rely on that live-stream and on virtualized controls to see how the equipment is behaving, to interact with it, to see the results of their experiments, and to check whether further actions are required. The quality of the interactive live-stream that a laboratory provides is therefore particularly important for the experience as a whole, but, through the years, has often been neglected.

This chapter summarizes two of the published works that form this dissertation. It focuses on the interactive live-streaming requirements of remote laboratories. First, from a mainly client-side perspective, it analyzes potential web-based interactive live-streaming schemes, and determines which are most effective: a scheme based on image-refreshing, and a scheme based on H.264. Then, taking a server-side perspective, it designs and proposes an architecture that satisfies the requirements. The architecture is based on Redis, web-based, and highly scalable. It is evaluated through an implementation, the WILSP platform, which is released as Open Source. These contributions are expected to be useful for remote laboratory researchers and developers. They are expected to be able to integrate the platform into their laboratories, and to freely modify it to their needs.

2.1 Introduction

In the last few years many live-streaming platforms have emerged, such as YouTube Live[1], TwitchTV[2], Instagram Livestream[3], and Facebook Live[4]. These platforms are backed by large social media companies and are proprietary. They are effective for streaming non-interactive live-streaming content, such as live sports. However, they are often unsuitable for interactive live-streaming, because their capture-render delay is too high. The main reason is that, for their purposes, a low delay is less prioritary than scalability and high quality at low bit rates. Allowing for a relatively high delay lets them use techniques such as long buffers and codecs that provide heavy interframe compression. Zhang and Liu (2015) analyze the TwitchTV architecture, providing more insight into these aspects, and measuring the broadcast delay to vary between 12 and 21 seconds.

Besides that limitation, remote labs, both for research and practice, need an interactive live-streaming technology that can be integrated into the laboratories, and which can be modified to fit the particular needs of each one. The fact the aforementioned platforms are closed-source and proprietary is therefore aspect that makes them unsuitable for these purposes.

The main purpose of the works that are described in this chapter is to provide remote laboratory researchers and developers with an open interactive live-streaming architecture and an Open-Source interactive live-streaming platform that they can use, learn from, modify, and integrate into their own projects. For this purpose, the main goal of the first work is to determine which schemes, from a client-side perspective, are more effective for remote laboratories. This is important because, currently, there is no consensus in the remote laboratory community. Remote laboratories tend to rely on different simplistic interactive live-streaming schemes, and not all of them are as effective as they could be. Once the best schemes have been determined, the second work proposes an interactive live-streaming architecture that can support these schemes effectively, and that is Open Source, designed to be highly scalable and extensible, and designed to be integrable into external systems.

2.2 Choosing interactive live-streaming approaches

An interative live-streaming approach is, in the context of this work, the combination of techniques, formats and protocols that comprise a system that is able to provide a client with a low-delay stream to render. There is a very high number of potential schemes. Different encoding formats could be used (e.g., M-JPEG, MPEG-1, H.264), different communication protocols (Basic HTTP, AJAX, WebSockets, WebRTC), and different client-side rendering methods.

There is currently no consensus on which particular interactive live-streaming scheme (which particular combination) is more effective. Works can be found in the literature that

[1] https://www.youtube.com/live
[2] https://twitch.tv
[3] https://instagram.com/livestream
[4] https://live.fb.com

compare the performance of their components, such as the performance of H.264 against other codec types. Few works, however, compare the real-world performance of particular schemes. In this work, some of the web-based schemes that are most commonly used in remote laboratories, and some more modern schemes which are promising, will be experimentally compared. The goal is to determine which schemes are the most appropriate for the requirements or remote laboratories.

2.2.1 Requirements and criteria

The key difference between an interactive and a non-interactive live-streaming system is that an interactive one requires a lower capture-render delay (Zhang and Liu, 2015; Rodriguez-Gil et al., 2017; Paravati et al., 2010). In such a stream, viewer actions affect the stream itself. Therefore, if the delay is too high, their user experience will not be satisfying, because it will take too long for them to see the results of their actions and they will not be able to act on those results. A key goal is thus that the system be **near-real-time**. That is, that the system be able to provide a capture-render delay that is low enough for interactive live-streaming.

Other key goal is **universality** (García-Zubia et al., 2009). Remote laboratories, and therefore the interactive live-streaming scheme, should be compatible with a wide range of systems and platforms. As many people as possible should be able to access them. For this purpose, the schemes should be fully web-based and cross-platform, and they should be available across different devices, such as mobile phones and tablets. These are ever more important, and its use in education is growing (Crompton et al., 2016; Couse and Chen, 2010; Şad and Göktaş, 2014; de la Iglesia et al., 2015).

A last key goal is **security**. Remote laboratories are often hosted by educational institutions. In that context, it is particularly important that students not be exposed to security risks, for which the institution could be liable (García-Zubia et al., 2009). To promote security, the IT services of these institutions often restrict ports, protocols and sometimes non-standard browser. Schemes will thus preferibly rely on HTTP-based, standard technologies.

In order to initially compare the different schemes, the criteria will be client-side resource usage. Therefore, the variables that will be measured are the frame rate, the network bandwidth usage, the CPU usage, and the RAM usage. Server-side processing needs will be considered in depth in the second work that is presented in this chapter.

2.2.2 Approaches

The approaches that are considered are the following:

- **JavaScript-based Image Refreshing**: Requesting and repeteadly rendering discrete snapshots through JavaScript relying only on HTTP. Very simplistic, but it is a method used by many existing remote laboratories, is very cross-platform, and can adapt to different bandwidths automatically.

- **Native M-JPEG**: Rendering a M-JPEG stream using only native browser support for these streams. Though it is simple due to native support, that support tends to be lacking and little control is available programatically.

- **JavaScript-based M-JPEG**: Transmitting a M-JPEG stream through the SocketIO[1] library. It relies on either JavaScript or WebSockets. Then, the data is rendered with a JavaScript decoder and HTML5. Though it is a significantly more complex scheme than the previous one, it is more reliable and allows for greater control.

- **JavaScript-based MPEG-1**: Transmitting a MPEG-1 stream through SocketIO. MPEG-1 is a relatively old codec that supports interframe compression. It is then rendered with a JavaScript decoder.

- **JavaScript-based H.264/AVC**: Transmitting an H.264/AVC stream through SocketIO. H.264/AVC is a more modern format which supports high interframe compression and can provide good quality at lower bit rates than previous standards.

- **Approaches based on HLS and MPEG-DASH**: Those are promising standards which are nonetheless widely supported for now. Because a key goal is universality, these approaches are not considered for further testing.

- **Approaches based on WebRTC**: WebRTC is designed for multimedia applications, but it is still being standardized and, at least for now, is oriented towards peer to peer streaming. Therefore, approaches that rely on WebRTC are not considered either. When data communications are needed, SocketIO (and thus WebSockets) are used instead.

2.2.3 Experiments and comparison

In order to compare the most promising of the previously described methods, an experimental setup was designed. Five of the previous schemes have been implemented and are compared: Image-refreshing, native M-JPEG, JavaScript-based M-JPEG, MPEG-1, and H.264. The prototype interactive live-streaming platform can receive data from IP webcams as input, transcode it into various formats if needed, and serve it to the client's browser appropriately. The platform also serves various client-side widgets that are able to render the stream in the browser, depending on the particular scheme.

The experiments and performance analysis are designed to represent the real-world performance of these schemes, so no simulations are used. Instead, the system is deployed in real servers and the streams are transmitted and rendered by real browsers. CPU and RAM measurements are provided by the browser itself when rendering each scheme. Capture-render delay measurements are calculated by comparing pictures of the testing client computer against the source, as shown in Figure 2.1.

Figure 2.1: Measuring the capture-render delay. Computer on the right is rendering the interactive live-stream from the IP camera (on the lower left.)

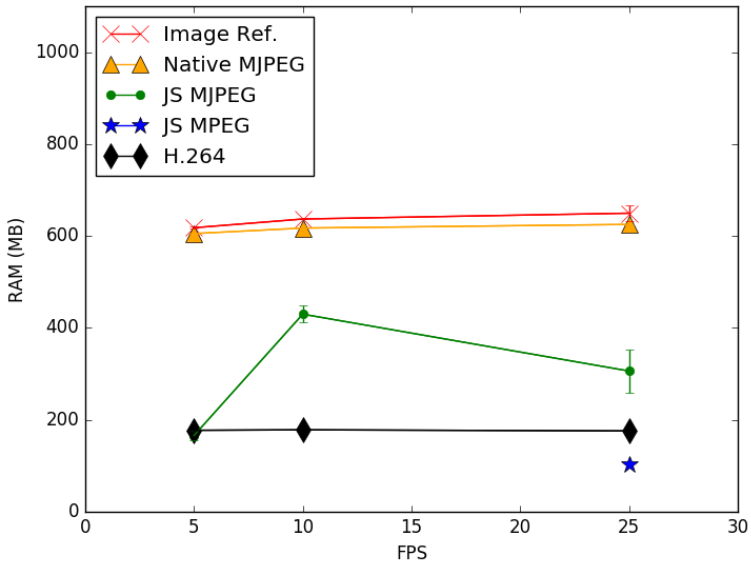

Figure 2.2: Client-side RAM usage comparison)

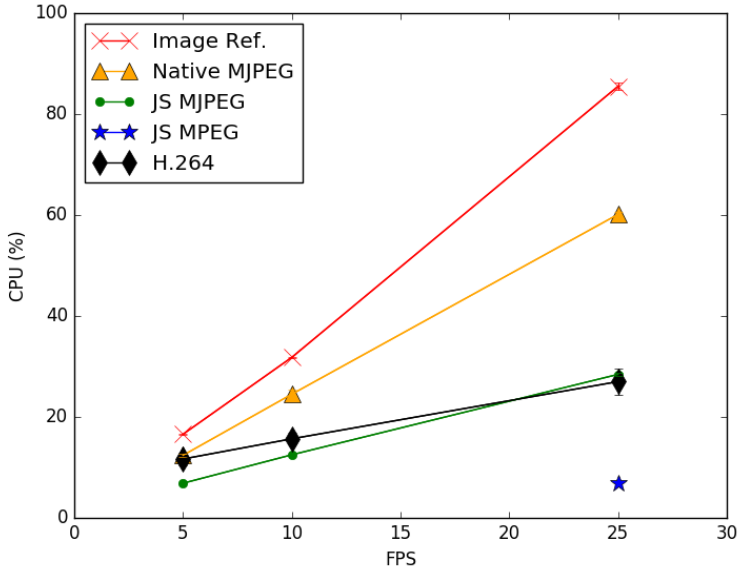

Figure 2.3: Client-side CPU usage comparison)

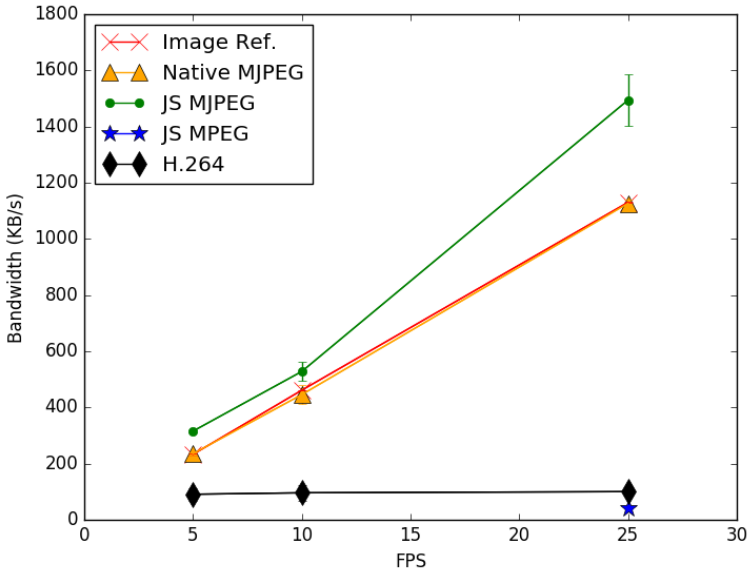

Figure 2.4: Client-side downstream bandwidth usage comparison

2.2.4 Results

The results of the experiment are summarized by Figure 2.2, Figure 2.3 and Figure 2.4. They are included in full in the related publication (see Appendix A).

They show that the performance of image-refreshing, native M-JPEG, and JavaScript-based M-JPEG is similar. They require a relatively high bandwidth due to the poor compression they provide, they require relatively high CPU and RAM usage, but they are able to provide a very low capture-render delay. All of them, including image-refreshing, can provide the maximum FPS for the experiments (which is 25) without issues. MPEG-1 and H.264 require much less bandwidth, and even require less CPU and RAM. However, their capture-render delay is significantly higher, probably due to the transcoding that is required to serve them.

2.2.5 Conclusions

The main conclusion is that there is no single best scheme for all cases. Native M-JPEG provides no significant advantage over the other schemes, so it should be avoided. This is an interesting conclusion because currently, several remote laboratories rely on it. Probably because, due to being supported natively, is simple to implement and is expected to be more efficient than a custom JavaScript-based renderer. But this does not seem to be the case, and most implementations are unreliable.

The experimental results and the experience with each implementation suggest that the schemes that best satisfy the requirements are **image-refreshing** and **H.264**. Image-refreshing has similar performance to JavaScript-based M-JPEG, but it is very easy to implement and deploy, relies only on standard HTML features, is able to adapt to varying bandwidths and to recover from error conditions automatically, and is also simple from a server-side perspective. Therefore, it provides a good compromise on performance and convenience. H.264 can provide good quality at a much lower bandwidth than image-refreshing. However, its capture-render delay is higher. This was expected, due to H.264 being a relatively powerful interframe compression scheme that requires transcoding and a buffer (even though it has in this case been configured to be very small).

Therefore, image-refreshing is an appropriate scheme to use for remote laboratories that require minimal capture-render delay. H.264 is appropriate for those that need to minimize resource usage but which can withstand a slightly higher capture-render delay. However, both schemes are likely to provide good results for a wide range of laboratories.

2.3 Goals and requirements

The interactive live-streaming platform to be designed should meet the goals of requirements of remote laboratories. The main goals are:

- **Universality:** The streams should be available for as many end-users as possible.

[1] https://socket.io

- **Efficiency and scalability:** The architecture should scale and support a large number of camera sources and users.
- **Openness:** The architecture should be open and rely on open technologies, so that it can be customized and used by researchers and developers.

The main requirements are:

- **Interactive live-streaming:** The platform should be capable of interactive live-streaming, so it should provide a low capture-render delay.
- **Supporting multiple input sources:** It should support multiple camera sources, and it should support several input formats.
- **Supporting multiple output schemes:** The architecture should be able to provide the streams through different output schemes, so that researchers and developers can choose the most appropriate for their use-cases, and extend them if needed. The image-refreshing scheme and the H.264 schemes that were selected through the work described in Section 2.2 will be the mainly supported output schemes, and the evaluation will be conducted with those.

2.4 Architecture

An overview of the proposed architecture is shown in Figure 2.5.

The architecture is designed in highly-decoupled layers. A Redis (Carlson, 2013) cluster is at the center to ensure that the main input and output layers are indeed decoupled and to maximize scalability. In line with this, the layers are designed to be horizontally scalable. An arbitrary number of input sources (IP cameras) can be supported by increasing the number of Feeders, which can be deployed in an arbitrary number of physical servers. Similarly, an arbitrary number of end-users can be served by increasing the number of CamServers.

The figure shows an example of deployment. Data flows from left to right. The IP cameras, to the left, feed their streams into the Feeder components. The Feeders can handle several formats. Most IP cameras provide JPG snapshots and M-JPEG video. The Feeders then forward the streams into the Redis clusters. Sometimes transcoding is required. For instance, transcoding is always required to serve an H.264 stream. In this case, the Feeder will rely on FFmpeg[1]. The Redis cluster is made of standard Redis instances (unlike the Feeders and CamServers, which have been designed and developed explicitly for the WILSP platform). It acts as a decoupling agent. It will not serve the streams until the CamServers ask. When a CamServer does request a stream from Redis, the data will be forwarded to it. Depending on the output format that the CamServer requests, it will rely on the key-value store of Redis, or on its message brokering capabilities. The CamServer will then forward the data to the end-user's browser. The end-user widgets, which can also be provided by the platform, are JavaScript-only and fully standard. They rely on either

[1] https://ffmpeg.org

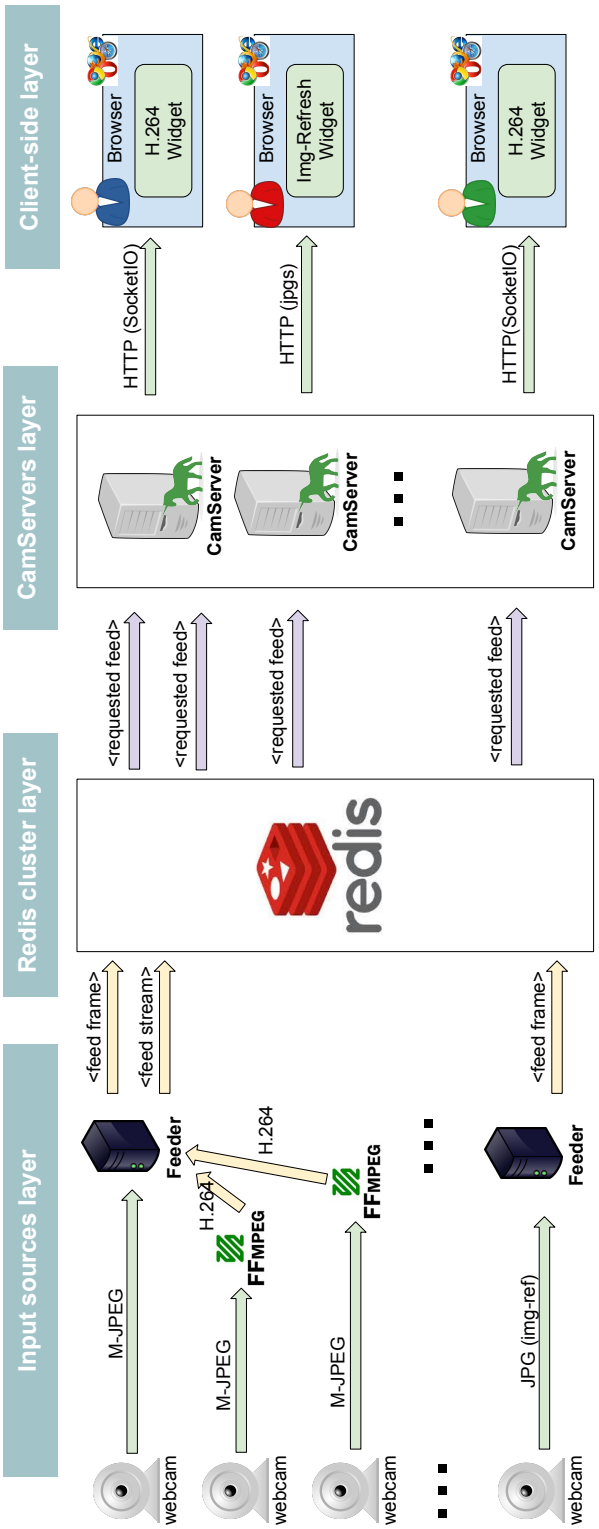

Figure 2.5: Architectural overview of the proposed platform and its components.

basic HTTP, Java, or SocketIO to accept the data from the CamServer. They then render the stream using one of the schemes that are summarized in Section 2.2.

2.4.1 Layers

2.4.1.1 Input sources layer

This layer encapsulates access to the webcams. This is important because IP webcams are not homogeneous, and often their hardware and software is limited. Especially over non-local networks and over relatively high-loads, their performance tends to be unreliable. Also, depending on the brand, different configuration options and formats are supported, and in some cases they have been known to present security vulnerabilities (McAfee, 2015). The Feeder components are designed to abstract out that complexity, making it transparent to the platform deployers and to end-users. The Feeder components provide well-known and reliable streams into the Redis clusters, transcoding them through FFmpeg if needed. The contributed implementation has been created in Python and relies on the Gevent[1] coroutine-based networking engine. An arbitrary number of Feeders and of IP cameras can be deployed.

2.4.1.2 Redis cluster layer

This is a key element of the architecture, providing decoupling and scalability between the Feeders and the CamServers. It receives the streams from the Feeders, and then forwards them in an on-demand basis to the CamServers. This scheme ensures that the performance of the Feeders is not affected by the number of end-users (but only by the number of cameras) and that the performance of the CamServers is not affected by the number of cameras (but only by the number of end-users). Redis is not specifically oriented towards video streaming. However, both its short-lived key-value storage features and its short-lived message brokering capabilities are particularly appropriate for this architecture. Furthermore, it is well-tested, supports clustering and sharding (Redis, 2017), has been used successfully to achieve scalability, and research has shown that it provides better performance than other alternatives (Abubakar et al., 2014).

2.4.2 CamServers layer

This layer contains the CamServers, which request the streams that users request and serve them to each end-user. When clients request image-refreshing they rely on Redis key-value store features to independently retrieve each frame. When clients request a true stream such as H.264 they rely on Redis short-lived message brokering features to receive the appropriate stream. Then, they serve the stream to the end-users either through standard HTTP or through SocketIO, depending on the requested format. In the case of SocketIO, it is noteworthy that internally it relies either on WebSockets or in AJAX. The CamServes have been implemented in Python and use Gevent. Additionally, they rely on the WSGI

[1] http://www.gevent.org

and on Gunicorn[2] to be horizontally scalable. An arbitrary number of CamServers can be started, to be able to serve an arbitrary number of end-users.

2.4.3 Browser layer

The platform includes several client-side widgets, libraries and video players to render each stream in the user's browser. The platform supports several schemes, some of which were described in Rodriguez-Gil et al. (2017). The ones that seem to be most effective and that are considered in this work, however, are image-refreshing and H.264. In the case of image-refreshing, the widget is simplistic: rendering a snapshot stream only requires standard HTML and basic JavaScript. It is not particularly efficient, but higher than 30 FPS can be supported without issues in modern browsers, even in mobile devices. H.264 is significantly more complex. The contributed implementation relies on a customized, purpose-specific version of the Broadway[1] H.264 decoder. The core is programmed in C but compiled into JavaScript through EmScripten[2]. It is therefore both cross-platform and highly efficient. In WebGL-compatible devices, it is designed to leverage low-level hardware access and to rely on graphical hardware acceleration to achieve better performance. In this case, the stream data is received by the client through SocketIO, which uses either WebSockets or AJAX depending on browser-level support.

2.5 Evaluation

2.5.1 Methodology

The architecture has been implemented as Open Source. The resulting platform is named WILSP (Web-based Interactive Live Streming Platform). The architecture is evaluated experimentally through that implementation. It is highly-decoupled, supports an arbitrary number of cameras and end-users, and the input and output layers are independent from each other. Therefore, the evaluation is designed to consider those two layers independently. First, the input (Feeders) layer is evaluated, to measure whether its performance is as expected. Second, the output (CamServers) layer is evaluated, to measure, again, whether its performance is as expected. The study also needs to consider the different supported output schemes, which are also remarkably different from each other. Therefore, four different experiments are to be conducted:

- **Experiment 1**: Feeder component performance in snapshots mode (image-refreshing).
- **Experiment 2**: Feeder component performance in stream mode (H.264).
- **Experiment 3**: CamServer component performance in snapshots mode (image-refreshing).

[2]http://gunicorn.org/

[1]https://github.com/mbebenita/Broadway

[2]https://github.com/kripken/emscripten

- **Experiment 4**: CamServer component performance in stream mode (H.264).

The experiments use three different physical servers: An *experiment server* to hold the components being tested, a *support server* to hold the components that are not being tested, and a *GUI server* that is necessary for latency measurement. To be able to obtain meaningful results and to be able to simulate loads, two additional benchmarking components have been developed: a *FakeWebcam* component that is able to simulate an IP webcam, serving a never-ending video clip, and a *Requester* component, that is able to simulate end-user requests.

Most measurements are taken at the *experiment server*, though the capture-render measurement technique is more complex and requires the *GUI server* to really render the stream. The FakeWebcam component that provides the testing stream can embed a QR with a timestamp into the source image. That stream, with its embedded QR containing the original timestamp, flows through the system as delay builds up. The benchmarking script has been developed to, when appropriate, open a real browser (using Selenium[1]) for each experiment step. The script then takes a screenshot of the rendered stream, which contains the QR. The QR is then parsed, and the real capture-render delay can be calculated. The servers are synchronized through the NTP protocol. Figure 2.6 shows the stream with the embedded QR.

Figure 2.6: Interactive live-stream with an embedded QR timestamp to measure the true capture-render delay.

[1] http://www.seleniumhq.org/

Each experiment is actually run 50 times, and for each, 50 measurement sets are taken. In the case of the Feeder experiments, each time it is conducted with a different number of cameras (from 1 to 50). In the case of the CamServer experiments, each time it is conducted with a different number of Requesters (taking the place of end-users) (from 1 to 50). The ranges for the experiments are specifically chosen to represent how the platform scales on its intended working conditions, without reaching any bottleneck. If a bottleneck were to be reached, the behavior of the platform would depend on the particular bottleneck and format combination, and would most likely result in poor or unacceptable performance.

2.6 Results

The result of the experiments that evaluate the Feeder layer are summarized by Figure 2.7 and Figure 2.8. The results of the experiments that evaluate the CamServer layer are summarized by Figure 2.9 and Figure 2.10. The results are included in full in the related publication (see Appendix B).

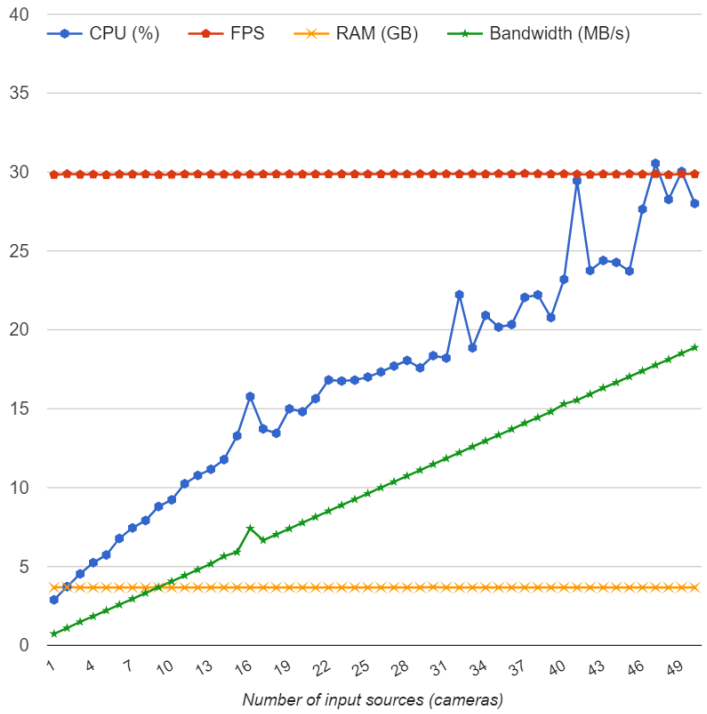

Figure 2.7: Results of Experiment 1. Feeder component using the image refreshing technique.

These results show that the platform scales as designed. All components for all schemes can maintain the target 30 FPS without issues independently from the load. RAM is mostly constant, as the frames for the interactive live-streaming platform are short-lived by design. CPU and bandwidth scale linearly with both the number of input sources (in the case of

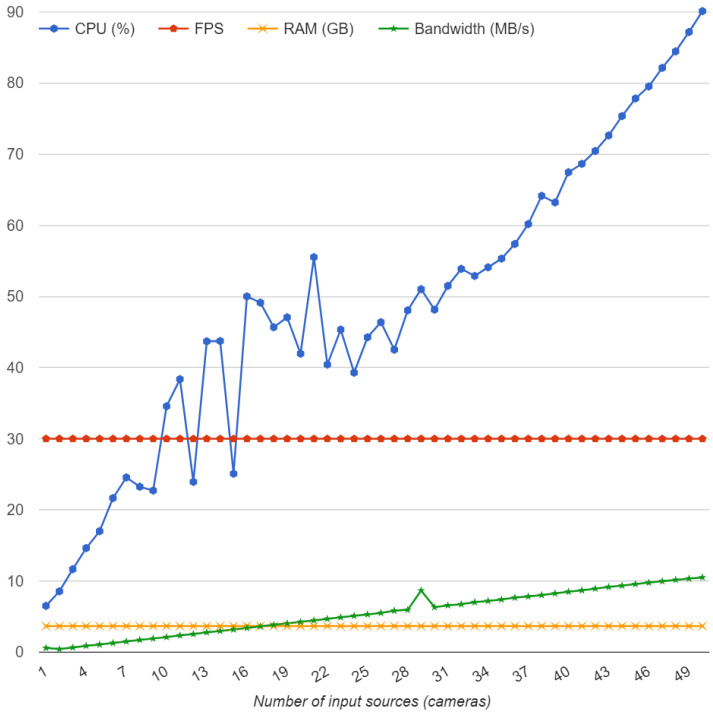

Figure 2.8: Results of Experiment 2. Feeder component using the H.264 format technique.

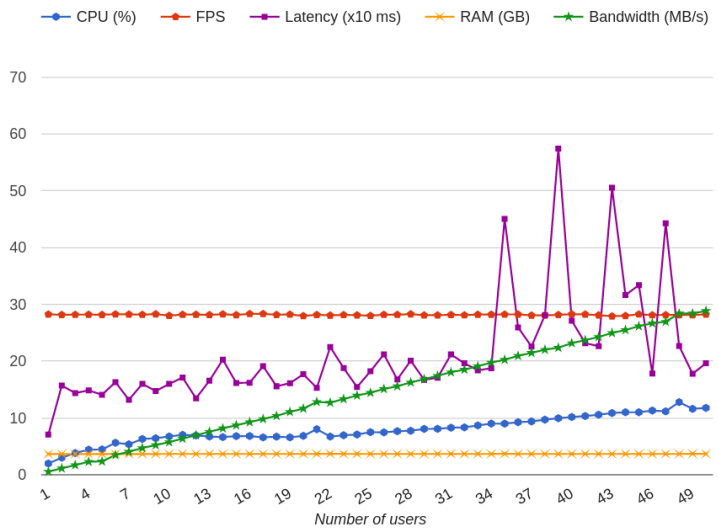

Figure 2.9: Results of Experiment 3. CamServer component using the image refreshing format technique.

the Feeders) and with the number of end-users (in the case of the CamServers). Therefore, when the platform is deployed, these can be expected to be the main potential bottlenecks.

During experiment 2, the Feeders serving H.264 reach around 85% CPU usage at 50 input cameras, as opposed to around 30% when serving image-refreshing snapshots. This is expected because in the case of H.264, transcoding is required, and H.264 provides good compression and is relatively expensive computationally. Nonetheless, it would be a factor to take into account for CPU-constrained servers. However (also as expected), H.264 is significantly more efficient from a bandwidth consumption perspective. With 50 cameras, image-refreshing consumes roughly 28.8 MB/s (576 KB/s per camera), while H.264 consumes only around 10.8 MB/s (216 KB/s per camera).

Ensuring a low capture-render delay is in fact one of the main requirements of an interactive live-streaming server. In this case, both schemes are able to deliver particularly small capture render delays. Lower than 200 ms in average and with maximum peaks of 574 ms. This is a satisfying result for an interactive live-streaming system, especially using the H.264 format, which often delivers considerably worse delays. In this case, the FFmpeg was carefully configured for interactive speeds, ensuring that the proper flags are set and that the buffer lengths are minimal and that it favours speed over compression. A natural consequence of that is, naturally, that bandwidth requirements are higher than they would be with a conventional high-delay H.264 stream.

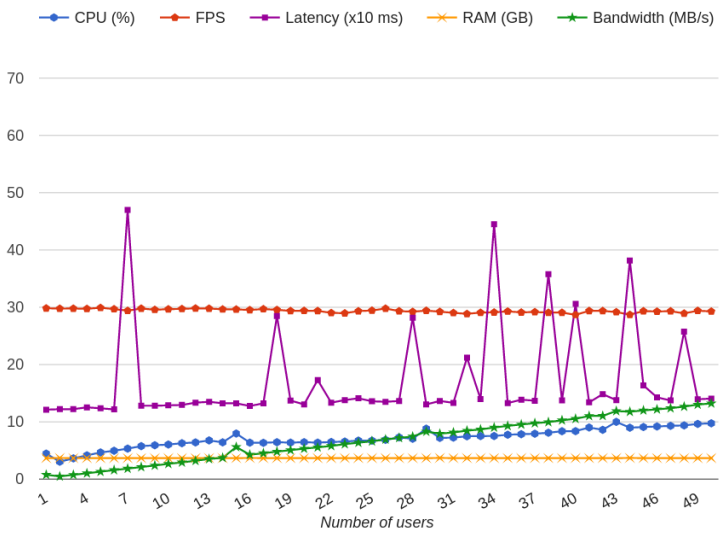

Figure 2.10: Results of Experiment 4. CamServer component using the H.264 format technique.

Judge a man by his questions rather than by his answers

Voltaire

CHAPTER

3

Teacher-defined tutoring agents

TEACHERS play a very significant role in a laboratory experience. Students are normally not alone when experimenting. Instead, a teacher is present to provide guidance, advice and supervision. This can be considered an advantage that hands-on laboratories have over remote laboratories. Research shows that in-person tutoring tends to be the most effective, and guidance in a remote laboratory is, often, harder to provide.

This chapter summarizes one of the published works that form this dissertation. Its goal is to propose a novel model for teachers to provide additional guidance to students. The model is based on intelligent tutors, which have been shown to be effective. A Domain-Specific Language (DSL) has been designed. Based on Blockly, it allows non-programming experts to visually define the content and behavior of tutoring agents. A platform allows teachers to use that DSL to define their own agents, and integrate them into third-party educational resources. Thus, it is possible to embed a teacher-customized tutoring agent into educational resources such as remote laboratories, without assistance from the original laboratory developers.

3.1 Introduction

In the last decades, several types of systems have been researched that are able to provide tutoring and guidance. Two of those types are Intelligent Tutoring Systems (ITSs) and Conversational Agents (Nye et al., 2014; Cassell et al., 1999; Weizenbaum, 1966). Though they have similarities, their focus is different. The former are focused on providing human-like conversation, for different purposes. Often they are even Embodied Conversational Agents (ECAs), which not only can have conversations but also feature a virtual animated

body (Kopp et al., 2005; Bickmore et al., 2016). The latter are focused on teaching, and they tend to not be limited to a conversational engine. Instead, they tend to be task based and try to resemble human tutoring (Vanlehn, 2006).

These systems have many potential applications in education and in other fields. They can be used for tutoring or question answering (Kerry et al., 2009), provide customer service or information, or act as a virtual companion or website tour guide (Rubin et al., 2010). However, creating them is difficult. Technical and domain expertise is often required, and domain experts do not always have programming experience. Authoring tools can reduce those issues (Aleven et al., 2009, 2015). Authoring tools are, indeed, the focus of this work.

Remote laboratories and other educational contents often require explanations and guidance. This is either due to the laboratory complexity itself, or to the theoretical content that should be taught alongside. Teachers are particularly effective at providing such tutoring, and research has in fact shown that face-to-face tutoring tends to be the most effective (Price et al., 2007). However, in practise, they often can only dedicate a limited amount of time to each student. In this scenario, as the number of students grows, the relative effectiveness of artificial intelligent systems increases, to a point where some ITSs can even be considered as equivalent (VanLehn, 2011).

Due to this, teachers and students can benefit from adding intelligent tutors to remote laboratories or other educational contents. The goal is not to replace teachers. It is, instead, to provide teachers with the means to create and integrate their own agents, to embed into those contents. This concept is illustrated with the example in Figure 3.1. In this case, a teacher-customized virtual agent is embedded in a VISIR remote laboratory: a remote laboratory that allows students to learn electronics by building their own circuits. Once the teachers have created and embedded a tutoring agent, they can then rely on it to provide basic guidance to any number of students. This is ever more important, with the increasing relevance of online courses, MOOCs (Pappano, 2012), and online laboratories (Ma and Nickerson, 2006; Dormido Bencomo, 2004; Froyd et al., 2012). Customizability is also a key feature. Research works and even large initiatives, such as the European project Go-Lab, suggest that for the wide and successful adoption of these tools teachers should be able to provide customized experiences for their students (de Jong et al., 2014; Rodríguez-Triana et al., 2014; Gillet et al., 2013; Govaerts et al., 2013; Rodriguez-Gil et al., 2014).

To accomplish the goal, this work first establishes the requirements that teachers may have, focusing on the previously described use-case. Then, it proposes a visual Domain-Specific Language (DSL). This language is based in Google Blockly[1], a visual blocks-based language that has been designed to be simple and intuitive and has been successfully used to teach programming to young children (Doe, 2017; Trower and Gray, 2015). Then, it describes an architecture that relies on that DSL and that aims to satisfy the requirements. A platform that implements that model and architecture is created. A methodology is designed to evaluate the architecture experimentally. A testing platform is designed and developed to lead experiment participants through various stages. Throughout these, they learn to use and create agents. The gathered data, an standardized UMUX (Finstad, 2010; Berkman and Karahoca, 2016) survey and another non-standardized survey are used to

[1] https://developers.google.com/blockly/

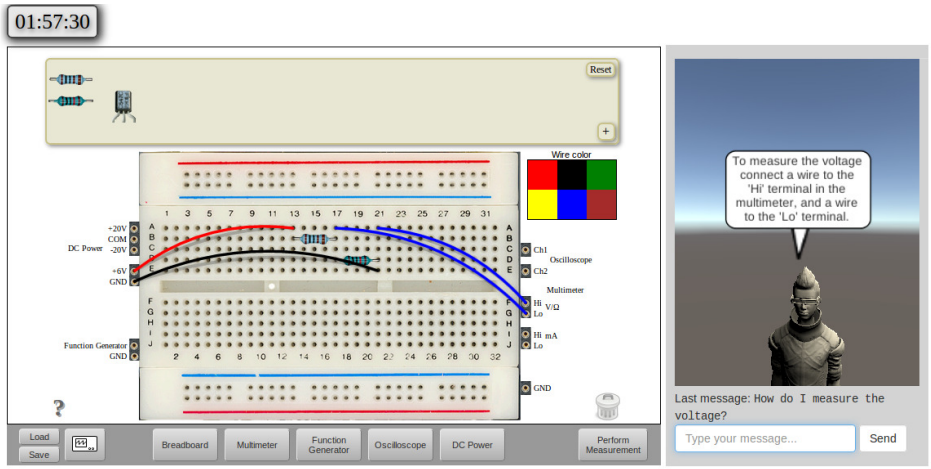

Figure 3.1: Embedding a teacher-defined agent into a VISIR remote lab.

draw conclusions.

3.2 Purpose and research questions

The purpose is to propose a novel approach to allow teachers (non-programming users) to define conversational agents using a Blockly-based DSL. A web-based authoring platform is to be created, that satisfies the requirements and can be used to answer the following research questions:

1. How easy to learn is the proposed approach? Can non-programmers learn it in a reasonable time?
2. How usable is the proposed approach?
3. Is the proposed approach perceived as valuable and intuitive?

3.3 Background

3.3.1 Agent authoring tools and non-programmers

Creating CAs and intelligent tutors can be costly in time, effort, and expertise required. Authoring tools can be used to create agents more efficiently, with a lower amount of resources (Olsen et al., 2014; Lester et al., 2015). Significant research efforts are dedicated to different types of authoring tools, such as Olsen et al. (2014); Kumar and Rose (2011); Olsen et al. (2013); Ray and Gilbert (2013); Virvou and Alepis (2005).

The main aim of some of those research works is to provide non-programmers with tools that can enable them to create the agents they need (Ray and Gilbert, 2013; Aleven

et al., 2016a; Lane et al., 2015; Koedinger et al., 2004; Bickmore and Ring, 2010). Different models exist to define agents, with different strengths and weaknesses. Tools such as CTAT (Aleven et al., 2006) cognitive tutors rely on XML and Jess rules[1]. These languages are not programming-languages but they are nonetheless relatively complex, and using the tools still requires significant technical knowledge. Especially, since they rely on complex IDEs such as Flash IDE or Eclipse. Simpler methods exist. Some tools rely on conversational trees (Lane et al., 2015). Other tools such as CTAT's *example-tracing* tutors avoid requiring formally-specified rules (Aleven et al., 2016b). Some tools can even generate the agents automatically by parsing large amounts of text to automatically extract information using NLP techniques (Shawar and Atwell, 2004; Feng et al., 2003).

In the case of this work, it proposes a novel model for specifying the agents. Using a visual DSL that can be used by non-programmers, it is possible to define agents in a way that is simple for non-programmers, powerful (because advanced blocks are possible, and Blockly can be a fully-fledged programming language), and extensible (because new blocks can be added to the DSL).

3.3.2 Visual programming languages

Visual programming languages are based on visual components. Instead of writing their program, users place visual blocks, drag-and-drop them, and conduct other spatial actions. A field where those languages have proven their usefulness is education (Sáez-López et al., 2016). Languages such as Scratch (Maloney et al., 2010) or Blockly (Doe, 2017) are intuitive and easy to learn, and have been used to teach programming to young learners (Kalelioğlu, 2015; Kumar, 2014). For this purposes, they have significant advantages, because students do not need to learn a syntax, they can start getting results in a very short time, and they can be sure that if the blocks fit, the program will at least run (Kuan et al., 2016). Figure 3.2

Google Blockly is Open Source, and very similar to Scratch. It has the aforementioned advantages, it has been designed to be extensible and completely customizable, and many success stories are available in the literature on its use as a base for a DSL (Iturrate et al., 2013; Serna et al., 2015; Wiriyakul and Senivongse, 2015). As such, it is an appropriate base for the DSL that is proposed in this work.

3.4 Requirements

The approach and platform that are proposed in this work have certain key requirements, some of which were described in previous sections. The main design goal is that teachers should be able to create and customize agents with it. Therefore, the skill requirements should be low. Also, the agents should be integrable into remote laboratories and other educational resources. This leads to certain technical requirements. In order to be properly integrable, the agents should be cross-platform and web-based. They should be integrable without requiring help from the original laboratory or educational resource developer.

[1] http://ctat.pact.cs.cmu.edu

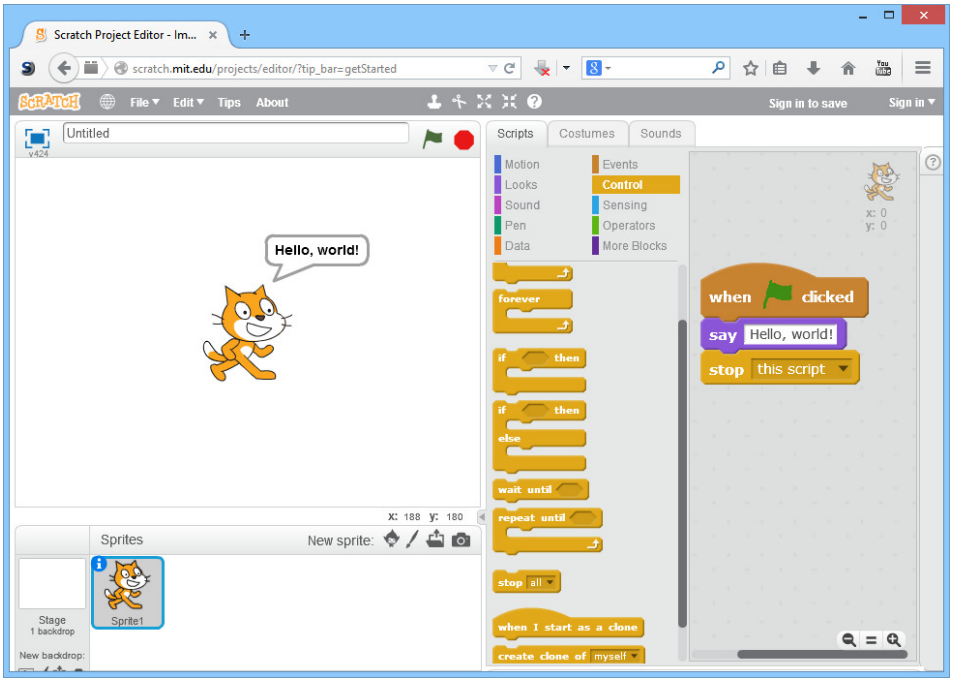

Figure 3.2: The Scratch visual programming language and environment.

Teachers should have precise control over the behavior of the agent. Content creators should be able to predict the output. Though such a requirement can lead to less natural conversations, the model that this work proposes is deliberately willing to sacrifice believability so that the behavior is easier to define and so that teachers can be confident that they how the bot will behave, and so that they don't require datasets.

3.5 Visual Domain-Specific Language

As described in previous sections, non-programmers should be able to create their own agents using the Visual DSL. Google Blockly is used as a base. Its Open Source and extensible nature makes it easy to define custom blocks and rules. Code generation can also be customized easily, so that these blocks get translated into a text-based language. Normally, the text-based language will be a mainstream one, such as JavaScript or Python, but custom generators can be defined.

In this case, the Visual DSL that is proposed here is made of various custom blocks, and generates a domain-specific JSON. That JSON can then be interpreted by the custom virtual agents engine to define its behavior. The DSL and platform have been designeded to be easily extended, but the first version, described (and implemented) in this work, is meant to be very simple. Users will be able to place *Conversation Node* blocks. Each block has an attached number of *Condition* blocks and an attached number of *Action* blocks. The most common action is simply for the bot to *say* something. It is noteworthy that such an

approach is useful for interacting with external systems or even for ECAs (Cassell, 2000), because there could be actions such as 'open a website' or 'move the virtual agent'.

The main condition blocks are summarized in the following list:

- **Contains-words condition block:** Triggers the conversation node when the user's input contains certain words.
- **Logic condition block:** Lets users use conventional logic operators such as AND or OR to combine conditions.
- **Previous-node block:** Triggers a node only if the previous triggered node was a specific one.
- **Default condition block:** The conversation node will be triggered by default when an input is received but no other node is triggered.

The main action blocks are summarized in the following list:

- **Say block:** To have the virtual agent say something when the node is triggered.
- **Raise-event block:** Raises a custom event that can be captured by the website the agent is integrated into. Useful for advanced interaction with external systems.

3.6 Strengths and weaknesses

The proposed approach has both strengths and weaknesses. It does not aim to be the best choice for all cases, but it aims to be particularly effective for some of them. The main advantages are its simplicity, the fine-grained control the non-programming authors have, the power that an extensible language provides, the integration capability it offers, the predictability of the created virtual agents, the fact that no datasets are requierd, and the fact that the agents are language-agnostic (bots can be created for any language). Nonetheless, as is often the case, some of the advantages have drawbacks. The main ones are that the scope of the agents is limited (they cannot learn or adapt, and defining huge bodies of knowledge would be tedious). The conversations with them are not particularly natural.

Due to these circumstances, the conclusion is that the agents are suitable for the use-cases for which they are designed: those in which the author wants a narrow-scope agent that can be integrated into an external resource (such as a remote laboratory) to provide in-context guidance.

3.7 Platform architecture

In order to satisfy the requirements that were described in Section 3.4 the architecture is fully web-based and end-users do not require any proprietary or non-standard technology to use or customize agents. The architecture contemplates two perspectives: the **authoring tool**, intended for teacher and domain-expert, and the **conversational agent component**, which will be deployed into the LMS, RLMS, or specific remote laboratory.

3.7.1 Key technologies

Key technologies for the architecture are Google Blockly (see Section 3.5), Unity3D[1] and WebGL. Google Blockly has already been described. Unity3D is a 3D applications engine, commonly used as a game engine. It is cross-platform and supports building and deploying for many different platforms. One of such platforms is WebGL (Marrin, 2011). WebGL is a web standard by the Chronos group which is not part of the HTML5 standard but which is open and widely supported by browsers. It provides low-level web-based access to OpenGL graphics, including 3D acceleration. Through WebGL it is possible to support complex and efficient virtual environments, which can leverage the full graphical power of the host machine in a cross-platform way. That includes advanced shaders and animations. WebGL is also currently supported in most mobile device browsers.

3.7.2 Perspectives

The platform is used through two perspectives, the **authoring** one and the **end-user** one. The authoring tool is also web-based. It prominently features the Blockly JavaScript-based engine, modified to support the implemented visual DSL, support its custom blocks, and be able to generate the appropriate JSON code. Figure 3.3 shows the main view of the Authoring Tool. The editor for the Blockly-based Visual DSL is to the left. Users can pick the blocks they want and place them appropriately in the canvas. To the right, an instance of the virtual agent engine can be used to test the logic that is being configured in real time. Figure 3.4 provides an overview of this architecture. Authors create agents through the web-based platform, which seamlessly integrates the tools they require.

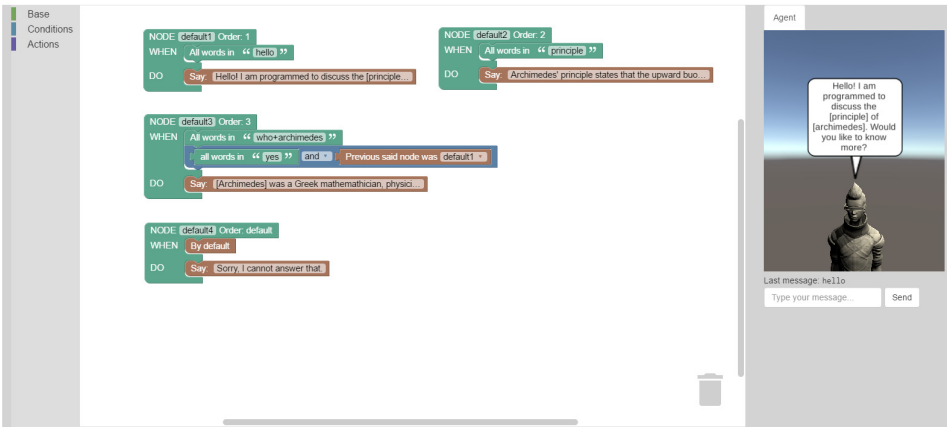

Figure 3.3: Authoring Tool's main view.

The end-user's perspective is different. End-users view only the virtual agent (similarly to the test agent in the authoring tool), but it will normally be integrated into a remote

[1] https://unity3d.com

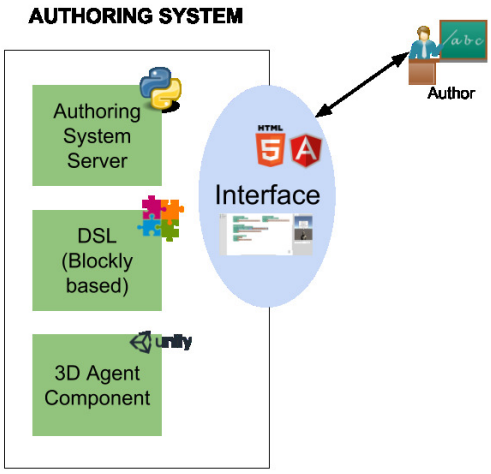

Figure 3.4: Authoring system architecture overview.

laboratory or educational system. This perspective is illustrated by Figure 3.1. The integration should be as seamless as possible, and teachers should not require assistance from the original content developers to include them. This is achieved, in part, by relying on the `window.postMessage` API.

3.8 Methodology

In order to evaluate the proposed model, experiments are conducted against the implemented platform. Evaluating the platform is challenging because the end-user results depend not only on the capability of the authoring platform, but also on the specific resulting virtual agents. And those agents will vary greatly depending on the domain-expert (the teacher) who creates them. A purpose-specific testing platform has been designed and developed. It leads participants through several stages, in order. These experimental stages are shown in Figure 3.5.

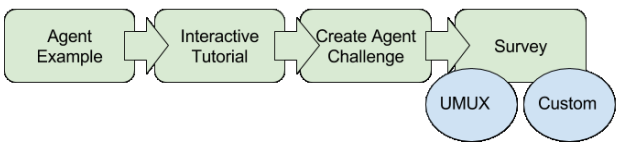

Figure 3.5: Experiment stages that the participants follow.

First, participants are shown how the resulting agents work. This is an introductory stage, for participants to understand what they can expect. Second, participants are taught how to create their own agents through an interactive tutorial. The tutorial has several

steps. They are not allowed to continue until they finish each step. Third, they are asked to use the knowledge they have to create a specific virtual agent. All participants are asked to create the same one, and are given some specifications. Again, they are not allowed to continue until the agent they have created meets certain basic specifications. Fourth, they fill an standardized UMUX (Finstad, 2010; Berkman and Karahoca, 2016) questionnaire to measure usability. UMUX is a four-items Likert scale designed to correlate with SUS (Brooke et al., 1996; Bangor et al., 2008). They are also asked to fill another non-standardized purpose-specific questionnaire. Additionally, throughout all the stages the participants' behavior is recorded by the system, and the time they spend in each step is measured.

32 first-year students from the University of Deusto took part in the study. They were split in two different sessions. Participation was voluntary. The students played the role of content creators, and they can be considered to represent the non-programming audience of the tool closely enough. The procedure was the following: two sessions were conducted. Each student had a computer, and an hour of time was allocated. The students were introduced to the platform, and then were asked to go through each stage at their own pace. The interactive tutorial of the platform is designed to be self-sufficient, but the participants were nonetheless allowed to ask questions and receive directions when they got stuck in particular steps or came across potential technical issues.

3.9 Results

The results are organized around the research questions that were raised in Section 3.2. Note that the results are summarized here, but are included in full in the related publication (see Appendix C).

3.9.1 How easy to learn is the proposed approach?

The platform took measurements of the time participants spend in each stage. Those are shown in Figure 3.6. The error bars are displayed for the 95% ($\alpha = 0.05$) confidence intervals. The sum of average times for every stage is 2 732 seconds (45.5 minutes).

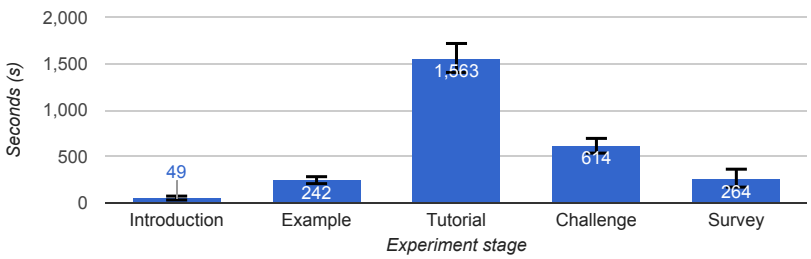

Figure 3.6: Average time in seconds that participants spent in each stage (n=32). Error bars show 95% confidence intervals.

Table 3.1: Results of the custom survey Q5-Q8 (n=32)

	Question (1-7 points Likert-type scale)	Mean	S.D.
Q5	*Non-programmers would be capable of creating bots with a Visual Language such as the one we used*	5.78	1.008
Q6	*It may be useful to integrate bots into some educational systems [...]*	6.00	0.916
Q7	*The visual block-based language to program the bots is intuitive*	5.94	1.076
Q8	*I have come across technical issues during the experiment*	3.62	2.196

3.9.2 How usable is the proposed approach?

Usability relies on a conventional UMUX questionnaire, which has proven its effectiveness. As Finstad (2010) explains, UMUX is scored through its 4 Likert-style scale questions to yield a SUS-like score in the 0-100 range. The UMUX mean score for the 32 participants of the study is 73.31 ($\mu = 15.62$), with a confidence interval half-width of 5.41 ($[67.89 - 78.72]$). This score is in the *acceptable* range and can be considered 'good' though not 'excellent' (Bangor et al., 2008). Considering that the platform is a research prototype and that the participants had a short time to learn to use the relatively powerful visual language, this result can be considered satisfactory.

3.9.3 Is the proposed approach perceived as valuable and intuitive?

The second part of the questionnaire is not standardized. Nonetheless, just like the UMUX part, it uses a 7-point Likert-scale. This is only for consistency. The results of that part of the questionnaire are summarized in Table 3.1. They show that users overwhelmingly believe that everyone would be capable of creating bots with such a visual language ($\bar{x} = 5.78$), that it would be useful to integrate bots into educational content ($\bar{x} = 6.00$), and that the proposed visual DSL is intuitive ($\bar{x} = 5.94$). The few participants who responded to the free-text question of the questionnaire left positive feedback.

3.10 Discussion

The results suggest that the proposed approach can be a useful way for teachers and other non-programmers to define tutoring agents. These agents cannot be expected to be suitable for all cases: they are, by design, meant to be domain-specific and simplistic. They can nonetheless be very appropriate for the proposed use-case: integrating them as aids into remote laboratories or other educational contents. Other approaches such as text-based formal languages (e.g., AIML or XML) may be more powerful, but can be expected to be

harder to learn. At the same time, the approach is powerful enough to be extensible and support advanced blocks, so it is more powerful than certain approaches such as providing pre-defined inputs and responses. This is in fact one of the strengths of the system. The blocks can be tailored to support advanced functionalities. For instance, it would be relatively easy to add a *text-to-speech* block, a *speech-to-text* block, blocks to control the behavior of the embodied agent, or blocks to help the teacher automatically evaluate the student. Those blocks can easily leverage advances in the state of the art of related fields as these fields evolve.

From a technical perspective, the implemented platform is satisfactory. The technology choices were appropriate, and the experiments show (through the UMUX standardized questionnaire and through the specific questionnaire) that users find it satisfying, especially considering that it is a prototype. The use of Unity3D and WebGL has also been appropriate, and it has worked as expected throughout the experiments. Every device the platform was tested on was able to render the WebGL-powered virtual agents with no issues.

The interactive tutorial itself served its purpose (to help evaluate the proposed approach and the platform) but it would have to be improved before teachers or students use it on their own. Though generally participants in the study were able to advance through it without help, at times they got stuck for a length of time during certain steps. In these cases the educational goals were achieved, but participants found these events frustrating. Ways to improve this issue could be explored, though they are beyond the scope of this work.

3.11 Conclusions

This work has proposed a novel approach for creating effective educational agents that is both simple and expressive. Teachers can use the platform and a Visual DSL to easily define the behavior of an agent. They can then integrate that agent into external remote laboratories or educational resources, without requiring assistance from the original developers. The results show that both the approach itself, and the prototype platform, are promising. No previous works in the literature (to the extent I know) propose a system to define and integrate agents in such a way. The results of the conducted experiments suggest that indeed, users can learn to create their agents fast, and find the experience and the platform satisfying. It can therefore be concluded that the proposed approach can indeed be used to add value to the learning experience of a remote laboratory, by easily integrating the teacher-defined agents into it to provide guidance or teaching.

At its very core, virtual reality is about being freed from the limitations of actual reality

John Carmack

CHAPTER

4

Hybrid laboratory models

DIFFERENT types of laboratories have many different strengths and many different weaknesses. Hands-on laboratories are often expensive, and time-consuming for students. Remote laboratories are often hard to maintain and provide limited interaction. Virtual laboratories are often inexpensive but unrealistic. But as technology and science advances, new models of laboratory become possible: Hybrid laboratories that are not limited to the conventional classification, but which instead combine features and leverage advantages from online and remote labs.

This chapter summarizes two of the published works that form this dissertation. It analyzes the literature and the deployed implementations of some of the most significant hybrid laboratories, most of which are in a prototype stage. Their architectures are extracted and compared. A new prototype hybrid laboratory is designed and implemented. Then, a novel hybrid model and architecture are proposed. Through them, it is possible to create hybrid web-based laboratories on which a real hardware board controls a simulated model. Both types of components interact with each other. Thus, the user experience can be fully realistic (the component they control is real), but at the same time it is more engaging and instructive. They can program the device against an arbitrarily challenging simulation. This hybrid model and architecture are evaluated against an implementation: a remote laboratory in which students can program a real FPGA device, and use it to control a virtual industrial water tank. This scheme can also save costs: real industrial models can be hard to maintain, prohibitively expensive, or even dangerous. By allowing students to program real boards that control simulated models, they benefit from both worlds.

4.1 Introduction

Traditionally, online laboratories have been classified into two distinct groups: remote laboratories and virtual laboratories (Dormido Bencomo, 2004; Auer et al., 2003; Tzafes-

tas et al., 2006). Though both groups share similar objectives, their approaches and even philosophies have been very different. On one side, remote laboratories provide access to real physical equipment through the Internet. On the other side, virtual laboratories provide access to a simulation, with no real components behind them. Research suggests that both types of laboratory have different sets of strengths and weaknesses (Nedic et al., 2003; Nickerson et al., 2007). Both can be effective for educational purposes, and sometimes one type is more adequate than the other (de Jong et al., 2013; Ma and Nickerson, 2006).

One of the main avantages of remote labs is that they are real. When students program a FPGA remotely, they can be sure that a real FPGA would indeed behave that way, because it *is* a real FPGA. One of the main advantages of virtual labs is that they are generally less expensive, and that they can adapt reality (de Jong et al., 2013). If a teacher wants students to experiment with a hydroelectric energy plant, a simulation will generally be more affordable and scalable than a real plant, or even an industrial model. Also, its virtual nature provides flexibility. Virtual labs can add visualizations, change the time dimension, or let the students conduct experiments that would be unsafe with real equipment.

The main goal of the work that is summarized in this chapter is to propose a novel hybrid architecture of remote laboratories that can leverage advantages from both areas. The concept is that a hybrid laboratory can add additional value to students by featuring both real and virtual components. An example would be a laboratory whose main goal were to teach students how to program FPGA for industrial control. If every component is virtual, the laboratory is unlikely to be effective. FPGA devices are hard to simulate, and those simulators that exist are often expensive and/or not particularly realistic. If every component is real, the laboratory would require some kind of industrial component to be controlled through the FPGA. Such a component is likely to be expensive and hard to maintain. With a hybrid architecture that supports laboratories with interactions between real and virtual components, however, it would be possible to provide a laboratory in which students can program real FPGA devices that control a virtual industrial model of arbitrary complexity.

Figure 4.1 shows a classical characterization of online laboratories, adapted to include hybrid ones. The exact definition of hybrid laboratories is not clearly established. Some authors refer to them as either virtual or remote, depending on which characteristics they emphasize. In the context of this work, a hybrid laboratory is simply an online laboratory which mixes virtual and real components (Gomes and Bogosyan, 2009). In the literature, several examples of laboratories that meet that criteria can be found, and are often very different in nature. For example, some are based on augmented reality (AR) (Vargas et al., 2009, 2011), others on virtual worlds (García-Zubia et al.; Scheucher et al., 2009; Callaghan et al., 2013), others on gamification (Callaghan et al.).

This work analyzes relevant hybrid architectures, extracting them from existing implementations if needed. Then, it proposes certain criteria that, according to the literature, would be adequate for an architecture for hybrid online labs based on real and virtual components that can interact with each other. An architecture that satisfies this novel laboratory model is then proposed. In order to evaluate the architecture, a hybrid laboratory that implements it is designed and developed. The *Watertank FPGA* laboratory is a hybrid online

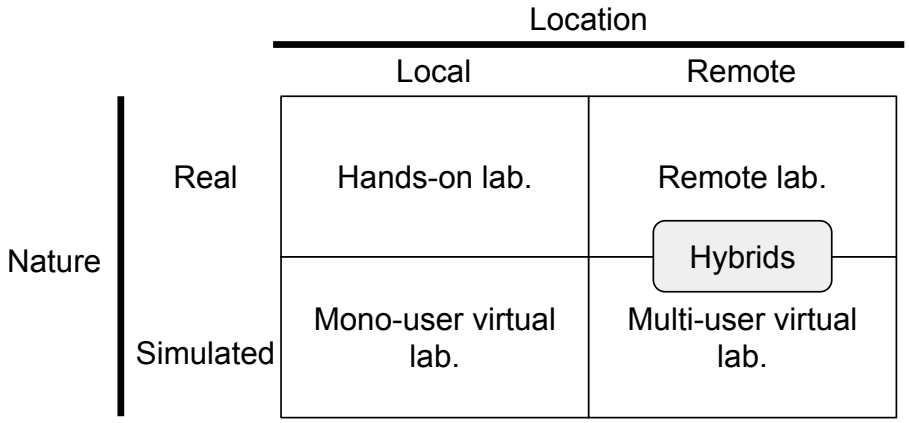

Figure 4.1: Hybrid laboratories within a characterization of labs (adapted from (Dormido Bencomo, 2004) and (Gomes and Bogosyan, 2009))

lab in which students are invited to program a real FPGA that can be used to control a virtual industrial watertank. This implementation is described and compared against the requirements. It is evaluated from a technical perspective, and then it is evaluated through a prospects and satisfaction survey.

4.2 Goals

The architecture is oriented towards laboratories with virtual and real components that can interact with each other in real time. Its main design guidelines are:

- **Universality**: Accessible to as many as possible. This involves relying only on standard technologies and avoiding proprietary and desktop-only technologies.
- **Security**: Minimize the security risk that its users are exposed to.
- **Power**: Support relatively advanced technical features such as rich media (graphics, sound and video) or low-latency bidirectional communications.

4.3 Architecture

Figure 4.2 shows an overview of the proposed architecture. The client-side is fully contained within the web-browser. This is critical, because, as previously mentioned, one of the core goals is universality. Therefore, the client-side needs to be fully web-based. In the server-side, the RLMS acts as a gateway. It communicates with the client through HTTP (speficially, through AJAX). The Experiment Server handles access to the hardware, receiving commands and providing responses to the RLMS. More details are provided in the following subsections.

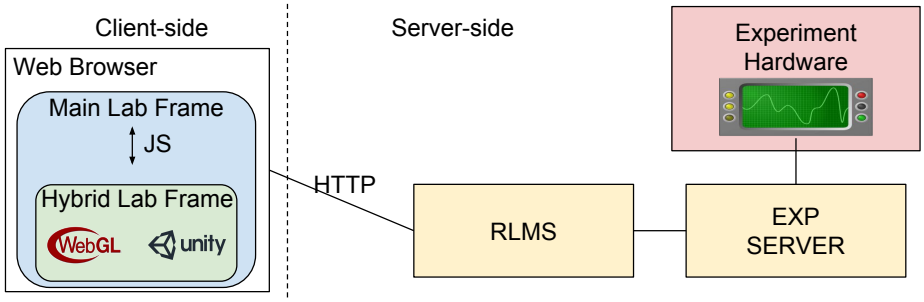

Figure 4.2: Proposed hybrid laboratory architecture.

4.3.1 Client side

The client-side needs to be fully web-based in order to be *universal* enough. For the same reason, it also needs to avoid non-standard ports, and plugins such as Adobe Flash and Java Applets (García-Zubia et al., 2009). Time ago, those technologies were frequently used to provide RIA features. Using them it was possible to rely on raw sockets for TCP and UDP, similarly to a desktop application. Similarly, using them it was possible to rely on advanced 2D or 3D accelerated graphics. Nowadays, with HTML5 and its related standards, this is no longer necessary. The proposed architecture relies on AJAX for communications. It has been chosen over Web Sockets because its performance is sufficient, and it provides slightly better compatibility with older firewalls and proxy servers. It relies on WebGL (Khronos, 2014) for accelerated 3D graphics. Authoring the WebGL components is done through the Unity 3D[1] technology.

4.3.2 Server side

The core of the server-side is the RLMS server. The proposed architecture, particularly, relies on the WebLab-Deusto[2] RLMS for that purpose. The RLMS provides common features such as authentication, user management and laboratory federation. It handles communication with the client, which is done through AJAX. It also handles communication with the Experiment Servers, so that the client does not need to connect to them directly. The Experiment Servers encapsulate access to the hardware. They can both conduct interaction requests from the client and report results. All of this is done in almost real-time, so that user interaction with the hardware can be fluid enough.

4.4 Architectures comparison

Table 4.1 summarizes the comparison among the proposed architecture and other architectures that were extracted from the literature and from existing laboratories that mix real

[1] https://unity3d.com
[2] http://weblab.deusto.es

and virtual components. This comparison is included in full in the related publication (see Appendix E).

4.5 Watertank FPGA laboratory

The purpose of many existing remote laboratories is to let users interact with hardware development boards. Those include FPGAs (El Medany, 2008; Lobo, 2011), PLDs (Garcia-Zubia et al., 2011), PLCs (Besada-Portas et al., 2013) and microcontrollers (Gilibert et al., 2006). The University of Deusto, for example, has long offered a FPGA laboratory (Orduña et al., 2011), providing access to a Xilinx FPGA development board. Students create the VHDL code and remotely program it into the board. The board provides several input and output peripherals that the student can program and interact with. Those peripherals include LEDs and switches.

These laboratories are definitely useful. Students can program the device remotely as if they had the device in front of them. However, the exercises they are asked to complete are not particularly engaging. The output peripherals are limited, so if the exercise is to design the control system for an industrial watertank, they need to imagine the system, and deduct from the LED (or other peripheral) outputs whether it is working correctly or not. An alternative would be to connect the outputs to a real model of a watertank, but such models are expensive to purchase and maintain.

A hybrid FPGA-Watertank laboratory that implements the proposed architecture (see Section 4.3) is described in this section. The interaction model and the relationship between its different components is summarized in Figure 4.3.

As the figure shows, the laboratory includes both a real FPGA board (that the students can program with their own VHDL logic) and a virtual model of an indistrial watertank. Therefore, students program a fully real control device, but that control device controls a virtual model. Bidirectional interaction is supported. The virtual model has multiple sensors and outputs: it has 3 level sensors, 2 overheating sensors and 2 water pumps. Students can read the virtual sensors from the real FPGA device, and turn on or off the virtual water pumps.

This has some significant advantages. The realism of the exercise is high: the VHDL students program is run in a completely real FPGA device. The exercises, however, are now more challenging and engaging: they no longer have to imagine whether their system is indeed controlling a watertank. They can now see it, and check whether the logic they have programmed, over time, is working as expected. This approach is also highly scalable. Because the industrial model is virtual, additional models could be provided very easily. These models could be served by the same FPGA device. Therefore, it would be possible to have a certain number of homogeneous FPGA device instances, to serve an arbitrary number of students an arbitrary number of virtual models.

Table 4.1: Comparison of the architecture characteristics

		UPM3DLabs	**iLab-TEALsim**	**CW1**	**CW3**	**ARL**	**Proposed**
Approach	Application type	Virtual world	Virtual world	Virtual world	Serious game	Augmented RL	Augmented RL
	Remote lab type	RLMS	RLMS	Domain-specific	Domain-specific	Domain-specific	RLMS
	Interaction type	Batch	Interactive	Batch	Batch	Interactive	Interactive
Universality	Desktop browsers	✗	✗	✗	✓	✗	✓
	Mobile browsers	✗	✗	✗	✓	✗	✓
	HTTP/HTTPS	✗	✗	✗	✓	✓	✓
	Built-in tech.	✗	✗	✗	✓	✗	✓
Security	HTTP/HTTPS	✗	✗	✗	✓	✓	✓
	Non-intrusive	✗	✗	✗	✓	✗	✓
	Built-in tech.	✗	✗	✗	✓	✗	✓
Power	Hardware acceleration	✓	✓	✓	✓	✓	✓
	Audio & video	✓	✓	✓	✓	✓	✓
	Network protocol	Opensim	Wonderland	Opensim	HTTP	Custom	HTTP

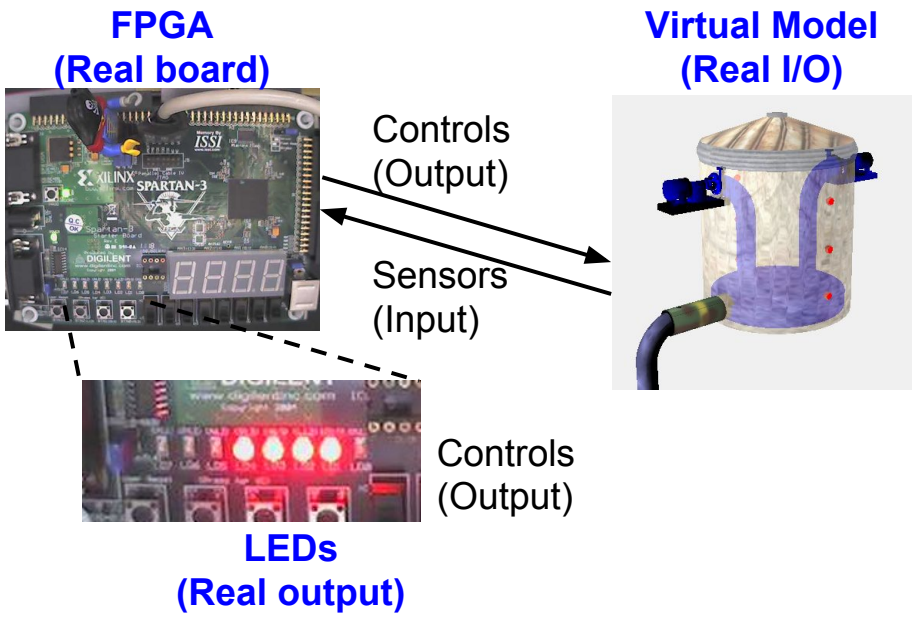

Figure 4.3: FPGA-Watertank components model

4.6 Evaluation

4.6.1 Technical evaluation

A technical analysis has been performed to verify the technical requirements. Tests were run against various desktop and mobile browsers to verify that the laboratory can run under them. It does indeed run on all modern browsers without requiring plugins. That includes, among others, Google Chrome, Mozilla Firefox, Internet Explorer, Microsoft Edge, Safari, Mobile Firefox and Mobile Chrome. The laboratory requires no non-standard ports, multiple virtual models can be deployed with relative ease, and the system does indeed rely on free technologies.

4.6.2 Prospects and satisfaction survey

In order to evaluate the potential of the model for education, and the user satisfaction, a study is conducted with the described implementation. The participants in the study complete two questionnaires. The first one is presented before they have used the laboratory. It obtains information about their expectations. Especifically, it measures their interest in different types of online laboratory technologies. The second questionnaire is presented after they have used the laboratory. It measures their satisfaction with that particular laboratory model and whether their expectations were met. 56 students filled the first questionnaire and 57 students filled the second.

Table 4.2: Results of the posterior survey Q2.5 — interest on the technologies (n=53; scale: 1-4)

Technology	Mean	S.D.
Pen and paper	2.604	0.809
Simulators	3.189	0.646
Remote Labs	3.189	0.585
Hybrid (Augmented) Remote labs	3.359	0.676

Figure 4.4 shows the results of one of the questions, about the perceived learning potential of RLs (Remote Laboratories) or ARLs (Augmented Remote Laboratories) (Andujar et al., 2011). Table 4.2 shows the results of other question, about user interest on each technology type. The full results can be found in the related publication (see Appendix E).

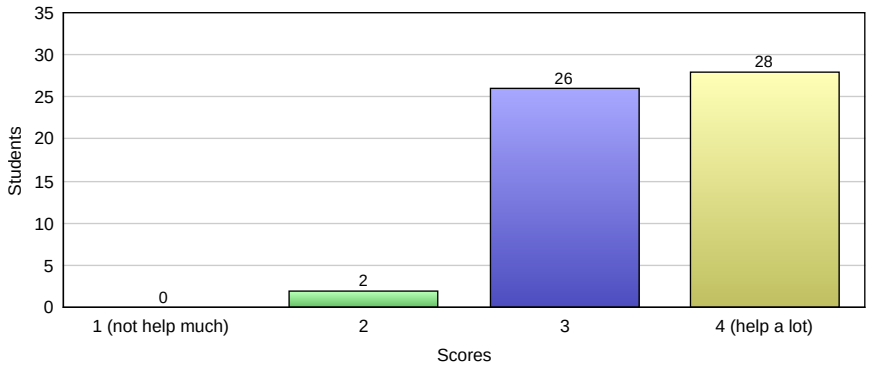

Figure 4.4: Results of Q1.2: perceived learning potential of RLs / ARLs ($n = 56$, $\overline{x} = 3.464$)

4.7 Conclusions

The proposed architecture meets the requirements. It is highly cross-platform and able to provide highly-interactive hybrid laboratories which combine real and virtual components.

The results of the user study show that students are indeed interested in laboratory technologies. Of those (hands on labs, virtual labs, remote labs, augmented remote labs), the augmented remote lab model, proposed in this work, ranks the highest. These findings suggest that, indeed, the architecture that is proposed here not only meets the described requirements, but it can lead to more satisfying and engaging laboratories. The results also show that after using the Watertank-FPGA laboratory, their opinion did not vary significantly. That is, they still rated the hybrid laboratory model higher than the other laboratory technologies. Most students found the laboratory satisfactory, and most thought that it had added value to their learning. Determining whether that perception is true would require

further study, and falls beyond the scope of this work. Students also agreed that augmenting a remote laboratory with a virtual environment is indeed valuable.

The mystery of life isn't a problem to solve, but a reality to experience.

Frank Herbert, Dune

CHAPTER 5

Conclusions and outlook

THROUGHOUT this dissertation one main goal has been pursued: to add value to the remote laboratory experience by improving the key aspects that comprise it. This consists on improving its interactive live-streaming capabilities, its teacher guidance capabilities, and by enabling augmented hybrid laboratories featuring virtual and real components. This chapter summarizes the main conclusions that have been drawn. Then, it presents an outlook for potential applications and future research.

5.1 Conclusions

As described throughout this dissertation, the remote laboratory experience is the result of the sum of several components. Three of them have been the focus of this work, and thus a three-pronged approach has been followed.

The first part of this thesis added value to the remote laboratory experience by improving user-laboratory interaction. For this, a novel interactive live-streaming architecture was proposed, implemented and evaluated. The results suggest that, indeed, the architecture and its platform accomplish that goal. The interactive live-streaming requirements are met. The architecture can provide high quality at a low latency and it is fully web-based. Furthermore, it is open-source, extensible and integrable into external systems (such as the remote labs). Therefore, it should be useful for remote laboratory researchers and developers, and the user-laboratory interaction of remote laboratories that use it should be improved.

The second part of this thesis added value to the remote laboratory experience by providing further means for teachers to guide their students on their use. A novel method has been proposed that allows teachers to define their own 3D tutoring agents using a visual Domain-Specific Language. Unlike most alternatives, this novel approach can be effectively used by non-programmers, thus enabling them to define the agents without assistance. Those agents can be integrated into external systems, such as remote laboratories,

to provide context-specific guidance. A prototype platform that supports that method has been implemented. After evaluating the method and platform through user studies, the results suggest that this new approach for tutoring agent creation is indeed effective. It is therefore a promising new way for non-programmers to define their own agents, which provides an interesting compromise between simplicity and power.

The third part of this thesis added value to the remote laboratory experience by providing a new online laboratory model: a hybrid laboratory which mixes both virtual and real components to leverage advantages from both types of laboratory. The model and architecture was successfully implemented. It led to the FPGA-Watertank laboratory, which was successfully tested with students. Results suggest that hybrid laboratories that follow that model can indeed be developed and may provide significant value to the learning experience in the cases on which they are appropriate.

Therefore, through this approach, this dissertation provides evidence showing that the value of the remote laboratory experience is increased by improving its interactive live-streaming interaction, by providing additional teacher guidance based on tutoring agents, and by augmenting the laboratory with the virtual components of a hybrid architecture.

5.2 Outlook

Technology keeps advancing at an impressive rate. Many fields, including education, are being reshaped. Today, learners of all ages can benefit from MOOCs, from interactive whiteboards and tablets, from new *maker* resources to conduct their own active learning and innovation. Among those potentially disrupting technologies are remote laboratories. Nonetheless, and though many institutions successfully use remote labortories today, their impact is still far from reaching their potential. The goal of this dissertation is to build towards that end.

Improving the interactive live-streaming capabilities of the laboratories makes the experience closer to a conventional laboratory. It should reduce the distance and lack of immersion from which remote laboratories sometimes suffer. Today, many remote laboratories, which are otherwise very useful and remarkable, provide sub-optimal streaming. The architecture and platform that are described should be useful to those research teams, or even to commercial initiatives that are arising. Its open nature makes it possible to customize the platform easily to different needs, and to build on it as required. Though oriented towards remote laboratories, this work presents potential applications in other areas that rely on interactive live-streaming as well.

Providing the means for teachers to create and integrate their own tutoring agents into the laboratories is also a step towards an improved experience. The method itself, that relies on a visual Domain-Specific Language for such a task, could in the future be explored for other purposes. Visual languages, so far, have proven their effectiveness for learners, and in some other specific areas such as hardware control. It is very likely, nonetheless, that they could be very successfully applied many more fields. Especially, to those where non-programmers need to specify or configure complex behaviors.

The last part of this dissertation explored the concept of hybrid laboratories: an archi-

tecture that provides laboratories in which virtual and real components interact with each other. Though, as discussed, some prototypes have been tried already, this sub-field of remote labs is still at its beginning. And it shows great potential. Today, industries and policy makers demand ever more laboratory and practical experiences. Distance education courses are offered to ever more people, and the institutions that offer them need to be ever more competitive. In this context, remote laboratories need to be both varied, useful and cost-effective. Hybrid laboratories can help provide more variety in a cost-effective way, while providing still enough realism. They can therefore provide capabilities that a simulation hardly will (e.g., allowing students to control a real, physical FPGA). And even, capabilities that a conventional hands-on laboratory could not provide (e.g., allowing students to make a virtual industrial watertank explode when they fail to control it correctly).

Through all of these contributions, it is my hope that this dissertation becomes a solid step towards better remote laboratory experiences, so that the disrupting potential of remote laboratories can eventually be trully unleashed, leading to an education that is of higher quality, more resource-efficient, and available to more people.

Bibliography

Abubakar, Y., Adeyi, T., and Auta, I. G. (2014). Performance evaluation of NoSQL systems using YCSB in a resource austere environment. *Performance Evaluation*, 7(8).

Aleven, V., Baker, R., Wang, Y., Sewall, J., and Popescu, O. (2016a). Bringing non-programmer authoring of intelligent tutors to MOOCs. In *Proceedings of the Third (2016) ACM Conference on Learning@ Scale*, pages 313–316. ACM.

Aleven, V., McLaren, B. M., Sewall, J., and Koedinger, K. R. (2006). The cognitive tutor authoring tools (CTAT): preliminary evaluation of efficiency gains. In *Intelligent Tutoring Systems: Proceedings of the 8th International Conference, ITS 2006, Jhongli, Taiwan, June 26-30, 2006.*, pages 61–70. Springer Berlin Heidelberg.

Aleven, V., Mclaren, B. M., Sewall, J., and Koedinger, K. R. (2009). A new paradigm for intelligent tutoring systems: Example-tracing tutors. *International Journal of Artificial Intelligence in Education*, 19(2):105–154.

Aleven, V., McLaren, B. M., Sewall, J., van Velsen, M., Popescu, O., Demi, S., Ringenberg, M., and Koedinger, K. R. (2016b). Example-tracing tutors: intelligent tutor development for non-programmers. *International Journal of Artificial Intelligence in Education*, 26(1):224–269.

Aleven, V., Sewall, J., Popescu, O., van Velsen, M., Demi, S., and Leber, B. (2015). Reflecting on twelve years of ITS authoring tools research with CTAT. In *Design recommendations for adaptive intelligent tutoring systems*, pages 263–283, Orlando. US Army Research Laboratory.

Andujar, J. M., Mejías, A., and Marquez, M. A. (2011). Augmented reality for the improvement of remote laboratories: an augmented remote laboratory. *IEEE Trans. Educ.*, 54(3):492–500.

Auer, M., Pester, A., Ursutiu, D., and Samoila, C. (2003). Distributed virtual and remote labs in engineering. In *2003 IEEE Int. Conf. Ind. Technol.*, volume 2, pages 1208–1213. IEEE.

Bangor, A., Kortum, P. T., and Miller, J. T. (2008). An empirical evaluation of the system usability scale. *Intl. Journal of Human–Computer Interaction*, 24(6):574–594.

Berkman, M. I. and Karahoca, D. (2016). Re-assessing the usability metric for user experience (UMUX) scale. *Journal of Usability Studies*, 11(3):89–109.

Besada-Portas, E., Lopez-Orozco, J. A., De La Torre, L., and de la Cruz, J. M. (2013). Remote control laboratory using ejs applets and twincat programmable logic controllers. *IEEE Trans. Educ.*, 56(2):156–164.

Bickmore, T. and Ring, L. (2010). Making it personal: end-user authoring of health narratives delivered by virtual agents. In *International Conference on Intelligent Virtual Agents*, pages 399–405. Springer.

Bickmore, T. W., Utami, D., Matsuyama, R., and Paasche-Orlow, M. K. (2016). Improving access to online health information with conversational agents: a randomized controlled experiment. *Journal of medical Internet research*, 18(1).

Brinson, J. R. (2015). Learning outcome achievement in non-traditional (virtual and remote) versus traditional (hands-on) laboratories: A review of the empirical research. *Computers & Education*, 87:218–237.

Brooke, J. et al. (1996). SUS-A quick and dirty usability scale. *Usability evaluation in industry*, 189(194):4–7.

Callaghan, M., McCusker, K., Losada, J. L., Harkin, J., and Wilson, S. (2013). Using game-based learning in virtual worlds to teach electronic and electrical engineering. *IEEE Trans. Ind. Informat.*, 9(1):575–584.

Callaghan, M., McShane, N., and Eguiluz, A. G. Using game analytics to measure student engagement/retention for engineering education. In *Proc. REV 2014 Conf., Porto*, pages 297–302. IEEE.

Carlson, J. L. (2013). *Redis in Action*. Manning Publications Co.

Cassell, J. (2000). Embodied conversational interface agents. *Communications of the ACM*, 43(4):70–78.

Cassell, J., Bickmore, T., Billinghurst, M., Campbell, L., Chang, K., Vilhjálmsson, H., and Yan, H. (1999). Embodiment in conversational interfaces: Rea. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 520–527. ACM.

Corter, J. E., Esche, S. K., Chassapis, C., Ma, J., and Nickerson, J. V. (2011). Process and learning outcomes from remotely-operated, simulated, and hands-on student laboratories. *Computers & Education*, 57(3):2054–2067.

Corter, J. E., Nickerson, J. V., Esche, S. K., and Chassapis, C. (2004). Remote versus hands-on labs: A comparative study. In *Frontiers in Education, 2004. FIE 2004. 34th Annual*, pages F1G–17. IEEE.

Corter, J. E., Nickerson, J. V., Esche, S. K., Chassapis, C., Im, S., and Ma, J. (2007). Constructing reality: A study of remote, hands-on, and simulated laboratories. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 14(2):7.

Couse, L. J. and Chen, D. W. (2010). A tablet computer for young children? Exploring its viability for early childhood education. *Journal of Research on Technology in Education*, 43:75–96.

Crompton, H., Burke, D., Gregory, K. H., and Gräbe, C. (2016). The use of mobile learning in science: a systematic review. *Journal of Science Education and Technology*, 25(2):149–160.

de Jong, T., Linn, M. C., and Zacharia, Z. C. (2013). Physical and virtual laboratories in science and engineering education. *Science*, 340(6130):305–308.

de Jong, T., Sotiriou, S., and Gillet, D. (2014). Innovations in STEM education: the Go-Lab federation of online labs. *Smart Learning Environments*, 1(1):3.

de la Iglesia, D. G., Calderón, J. F., Weyns, D., Milrad, M., and Nussbaum, M. (2015). A self-adaptive multi-agent system approach for collaborative mobile learning. *IEEE Transactions on Learning Technologies*, 8(2):158–172.

Doe, R. (2017). Google Blockly - A visual programming editor. http://code.google.com/p/blockly. Accessed Jan 2017.

Dormido Bencomo, S. (2004). Control learning: Present and future. *Annual Reviews in control*, 28(1):115–136.

El Medany, W. M. (2008). Fpga remote laboratory for hardware e-learning courses. In *Computational Technologies in Electrical and Electronics Engineering, 2008. SIBIRCON 2008. IEEE Region 8 Int. Conf. on*, pages 106–109. IEEE.

Feng, J., Bangalore, S., and Rahim, M. (2003). Webtalk: Mining websites for automatically building dialog systems. In *Automatic Speech Recognition and Understanding, 2003. ASRU'03. 2003 IEEE Workshop on*, pages 168–173. IEEE.

Finstad, K. (2010). The usability metric for user experience. *Interacting with Computers*, 22(5):323–327.

Ford, D. N. and McCormack, D. E. (2000). Effects of time scale focus on system understanding in decision support systems. *Simulation & Gaming*, 31(3):309–330.

Froyd, J. E., Wankat, P. C., and Smith, K. A. (2012). Five major shifts in 100 years of engineering education. *Proceedings of the IEEE*, 100(Special Centennial Issue):1344–1360.

García-Zubia, J., Irurzun, J., Angulo, I., Orduña, P., Ruiz-de Garibay, J., Hernández, U., and Castro, M. Developing a second-life-based remote lab over the weblab-deusto architecture. In *Proc. REV 2010 Conf., Stockholm, Sweden*, pages 171–176.

Garcia-Zubia, J., Orduna, P., Angulo, I., Hernandez, U., Dziabenko, O., Lopez-Ipina, D., and Rodriguez-Gil, L. (2011). Application and user perceptions of using the weblab-deusto-pld in technical education. In *Frontiers in Education Conf. (FIE), 2011*, pages GOLC1–1. IEEE.

García-Zubia, J., Orduna, P., Lopez-de Ipina, D., and Alves, G. R. (2009). Addressing software impact in the design of remote laboratories. *IEEE Transactions on Industrial Electronics*, 56(12):4757–4767.

Gilibert, M., Picazo, J., Auer, M., Pester, A., Cusidó, J., Ortega, J. A., et al. (2006). 80c537 microcontroller remote lab for e-learning teaching. *Int. Journal of Online Engineering (iJOE)*, 2(4).

Gillet, D., De Jong, T., Sotirou, S., and Salzmann, C. (2013). Personalised learning spaces and federated online labs for STEM education at school. In *Global Engineering Education Conference (EDUCON), 2013 IEEE*, pages 769–773. IEEE.

Gomes, L. and Bogosyan, S. (2009). Current trends in remote laboratories. *IEEE Transactions on industrial electronics*, 56(12):4744–4756.

Govaerts, S., Cao, Y., Vozniuk, A., Holzer, A., Zutin, D. G., Ruiz, E. S. C., Bollen, L., Manske, S., Faltin, N., Salzmann, C., et al. (2013). Towards an online lab portal for inquiry-based STEM learning at school. In *International Conference on Web-Based Learning*, pages 244–253. Springer.

Harward, V. J., Del Alamo, J. A., Lerman, S. R., Bailey, P. H., Carpenter, J., DeLong, K., Felknor, C., Hardison, J., Harrison, B., Jabbour, I., et al. (2008). The ilab shared architecture: A web services infrastructure to build communities of internet accessible laboratories. *Proceedings of the IEEE*, 96(6):931–950.

Hashemian, R. and Riddley, J. (2007). Fpga e-lab, a technique to remote access a laboratory to design and test. In *Microelectronic Systems Education, 2007. MSE'07. IEEE International Conference on*, pages 139–140. IEEE.

Heradio, R., de la Torre, L., Galan, D., Cabrerizo, F. J., Herrera-Viedma, E., and Dormido, S. (2016). Virtual and remote labs in education: A bibliometric analysis. *Computers & Education*, 98:14–38.

Iturrate, I., Martín, G., García-Zubia, J., Angulo, I., Dziabenko, O., Orduña, P., Alves, G., and Fidalgo, A. (2013). A mobile robot platform for open learning based on serious games and remote laboratories. In *Engineering Education (CISPEE), 2013 1st International Conference of the Portuguese Society for*, pages 1–7. IEEE.

Jara, C. A., Candelas, F. A., and Torres, F. (2008). Virtual and remote laboratory for robotics e-learning. *Computer Aided Chemical Engineering*, 25:1193–1198.

Kalelioğlu, F. (2015). A new way of teaching programming skills to K-12 students: Code.org. *Computers in Human Behavior*, 52:200–210.

Kerry, A., Ellis, R., and Bull, S. (2009). Conversational agents in e-learning. In *Applications and Innovations in Intelligent Systems XVI*, pages 169–182. Springer.

Khronos (2014). WebGL specification. Technical report, Khronos WebGL Working Group. Available: https://www.khronos.org/registry/webgl/specs/1.0/.

Koedinger, K. R., Aleven, V., Heffernan, N., McLaren, B., and Hockenberry, M. (2004). Opening the door to non-programmers: Authoring intelligent tutor behavior by demonstration. In *International Conference on Intelligent Tutoring Systems*, pages 162–174. Springer.

Kopp, S., Gesellensetter, L., Krämer, N. C., and Wachsmuth, I. (2005). A conversational agent as museum guide–design and evaluation of a real-world application. In *International Workshop on Intelligent Virtual Agents*, pages 329–343. Springer.

Kuan, W.-H., Tseng, C.-H., Chen, S., and Wong, C.-C. (2016). Development of a computer-assisted instrumentation curriculum for physics students: Using LabVIEW and Arduino platform. *Journal of Science Education and Technology*, 25(3):427–438.

Kumar, D. (2014). Digital playgrounds for early computing education. *ACM Inroads*, 5(1):20–21.

Kumar, R. and Rose, C. P. (2011). Architecture for building conversational agents that support collaborative learning. *IEEE Transactions on Learning Technologies*, 4(1):21–34.

Lane, H. C., Core, M. G., Hays, M. J., Auerbach, D., and Rosenberg, M. (2015). Situated pedagogical authoring: Authoring intelligent tutors from a student's perspective. In *International Conference on Artificial Intelligence in Education*, pages 195–204. Springer.

Lester, J., Mott, B., Rowe, J., and Taylor, R. (2015). Principles for pedagogical agent authoring tools. In *Design recommendations for intelligent tutoring systems: Authoring tools and expert modeling techniques*, page 151. Robert Sottilare.

Lobo, J. (2011). A remote reconfigurable logic laboratory for basic digital design. In *1st Experiment@ Int. Conf.–Remote & Virtual Labs–exp. at*, volume 11.

Lowe, D., Murray, S., Lindsay, E., and Liu, D. (2009). Evolving remote laboratory architectures to leverage emerging internet technologies. *IEEE Transactions on learning technologies*, 2(4):289–294.

Lowe, D., Yeung, H., Tawfik, M., Sancristobal, E., Castro, M., Orduña, P., and Richter, T. (2016). Interoperating remote laboratory management systems (rlmss) for more efficient sharing of laboratory resources. *Computer Standards & Interfaces*, 43:21–29.

Ma, J. and Nickerson, J. V. (2006). Hands-on, simulated, and remote laboratories: A comparative literature review. *ACM Computing Surveys (CSUR)*, 38(3):7.

Maloney, J., Resnick, M., Rusk, N., Silverman, B., and Eastmond, E. (2010). The Scratch programming language and environment. *ACM Transactions on Computing Education (TOCE)*, 10(4):16.

Marrin, C. (2011). Webgl specification. *Khronos WebGL Working Group*.

McAfee (2015). McAfee Labs Threats Report, May 2015. Technical report, Intel Security.

Nedic, Z., Machotka, J., and Nafalski, A. (2003). *Remote laboratories versus virtual and real laboratories*, volume 1. IEEE.

Nickerson, J. V., Corter, J. E., Esche, S. K., and Chassapis, C. (2007). A model for evaluating the effectiveness of remote engineering laboratories and simulations in education. *Computers & Education*, 49(3):708–725.

Nye, B. D., Graesser, A. C., and Hu, X. (2014). Autotutor and family: A review of 17 years of natural language tutoring. *International Journal of Artificial Intelligence in Education*, 24(4):427–469.

Olsen, J. K., Belenky, D. M., Aleven, V., and Rummel, N. (2013). Intelligent tutoring systems for collaborative learning: Enhancements to authoring tools. In *International Conference on Artificial Intelligence in Education*, pages 900–903. Springer.

Olsen, J. K., Belenky, D. M., Aleven, V., Rummel, N., Sewall, J., and Ringenberg, M. (2014). Authoring tools for collaborative intelligent tutoring system environments. In *International Conference on Intelligent Tutoring Systems*, pages 523–528. Springer.

Orduña, P. (2013). Transitive and scalable federation model for remote laboratories. *Q Bilbao*.

Orduña, P., Bailey, P. H., DeLong, K., López-de Ipiña, D., and García-Zubia, J. (2014). Towards federated interoperable bridges for sharing educational remote laboratories. *Computers in Human Behavior*, 30:389–395.

Orduña, P., Irurzun, J., Rodriguez-Gil, L., Zubía, J. G., Gazzola, F., and López-de Ipiña, D. (2011). Adding new features to new and existing remote experiments through their integration in weblab-deusto. *iJOE*, 7(S2):33–39.

Pappano, L. (2012). The year of the MOOC. *The New York Times*, (12).

Paravati, G., Celozzi, C., Sanna, A., and Lamberti, F. (2010). A feedback-based control technique for interactive live streaming systems to mobile devices. *IEEE Transactions on Consumer Electronics*, 56(1).

Price, L., Richardson, J. T., and Jelfs, A. (2007). Face-to-face versus online tutoring support in distance education. *Studies in Higher Education*, 32(1):1–20.

Ray, C. and Gilbert, S. (2013). Bringing authoring tools for intelligent tutoring systems and serious games closer together: Integrating GIFT with the Unity game engine. In *AIED 2013 Workshops Proceedings*, volume 7, page 37.

Redis (2017). Redis clustering tutorial. https://redis.io/topics/cluster-tutorial (Accessed: 2017-04-28).

Rodriguez-Gil, L., Latorre, M., Orduña, P., Robles-Gómez, A., Sancristobal, E., Govaerts, S., Gillet, D., Lequerica, I., Caminero, A. C., Hernandez, R., et al. (2014). OpenSocial application builder and customizer for school teachers. In *Advanced Learning Technologies (ICALT), 2014 IEEE 14th International Conference on*, pages 31–33. IEEE.

Rodriguez-Gil, L., Orduna, P., Garcia-Zubia, J., and Lopez-de Ipina, D. (2017). Interactive live-streaming technologies and approaches for web-based applications. *Multimedia Tools and Applications*.

Rodríguez-Triana, M. J., Govaerts, S., Halimi, W., Holzer, A., Salzmann, C., Vozniuk, A., de Jong, T., Sotirou, S., and Gillet, D. (2014). Rich open educational resources for personal and inquiry learning: Agile creation, sharing and reuse in educational social media platforms. In *Web and Open Access to Learning (ICWOAL), 2014 International Conference on*, pages 1–6. IEEE.

Rubin, V. L., Chen, Y., and Thorimbert, L. M. (2010). Artificially intelligent conversational agents in libraries. *Library Hi Tech*, 28(4):496–522.

Şad, S. N. and Göktaş, Ö. (2014). Preservice teachers' perceptions about using mobile phones and laptops in education as mobile learning tools. *British Journal of Educational Technology*, 45(4):606–618.

Sáez-López, J.-M., Román-González, M., and Vázquez-Cano, E. (2016). Visual programming languages integrated across the curriculum in elementary school: A two year case study using "Scratch" in five schools. *Computers & Education*, 97:129–141.

Scheucher, B., Bailey, P. H., Gütl, C., and Harward, J. V. (2009). Collaborative virtual 3d environment for internet-accessible physics experiments. *iJOE*, 5(S1):65–71.

Serna, M. Á., Sreenan, C. J., and Fedor, S. (2015). A visual programming framework for wireless sensor networks in smart home applications. In *Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2015 IEEE Tenth International Conference on*, pages 1–6. IEEE.

Shawar, B. A. and Atwell, E. (2004). A chatbot as a novel corpus visualization tool. In *LREC*. Citeseer.

Tawfik, M., Salzmann, C., Gillet, D., Lowe, D., Saliah-Hassane, H., Sancristobal, E., and Castro, M. (2014). Laboratory as a service (laas): A novel paradigm for developing and implementing modular remote laboratories. *International Journal of Online Engineering*, 10(4).

Toth, E. E., Morrow, B. L., and Ludvico, L. R. (2009). Designing blended inquiry learning in a laboratory context: A study of incorporating hands-on and virtual laboratories. *Innovative Higher Education*, 33(5):333–344.

Trower, J. and Gray, J. (2015). Blockly language creation and applications: Visual programming for media computation and bluetooth robotics control. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, pages 5–5. ACM.

Trundle, K. C. and Bell, R. L. (2010). The use of a computer simulation to promote conceptual change: A quasi-experimental study. *Computers & Education*, 54(4):1078–1088.

Tzafestas, C. S., Palaiologou, N., and Alifragis, M. (2006). Virtual and remote robotic laboratory: Comparative experimental evaluation. *IEEE Transactions on education*, 49(3):360–369.

Valera, A., Díez, J. L., Vallés, M., and Albertos, P. (2005). Virtual and remote control laboratory development. *IEEE Control Systems*, 25(1):35–39.

Vanlehn, K. (2006). The behavior of tutoring systems. *International journal of artificial intelligence in education*, 16(3):227–265.

VanLehn, K. (2011). The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems. *Educational Psychologist*, 46(4):197–221.

Vargas, H., Sánchez-Moreno, J., Dormido, S., Salzmann, C., Gillet, D., and Esquembre, F. (2009). Web-enabled remote scientific environments. *Comp. in Sci. & Eng.*, 11(3):36–46.

Vargas, H., Sanchez Moreno, J., Jara, C. A., Candelas, F. A., Torres, F., and Dormido, S. (2011). A network of automatic control web-based laboratories. *IEEE Trans. Learning Technol.*, 4(3):197–208.

Virvou, M. and Alepis, E. (2005). Mobile educational features in authoring tools for personalised tutoring. *Computers & Education*, 44(1):53–68.

Weizenbaum, J. (1966). ELIZA - a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1):36–45.

Wiriyakul, J. and Senivongse, T. (2015). A visual editor for language-independent scripting for bpmn modeling. In *Computer Science and Software Engineering (JCSSE), 2015 12th International Joint Conference on*, pages 156–161. IEEE.

Yazidi, A., Henao, H., Capolino, G.-A., Betin, F., and Filippetti, F. (2011). A web-based remote laboratory for monitoring and diagnosis of ac electrical machines. *IEEE Transactions on Industrial Electronics*, 58(10):4950–4959.

Zhang, C. and Liu, J. (2015). On crowdsourced interactive live streaming: a twitch. tv-based measurement study. In *Proceedings of the 25th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 55–60. ACM.

APPENDIX

Paper I

Interactive live-streaming technologies and approaches for web-based applications

This paper was published in the journal '*Multimedia Tools and Applications*' by Springer. It was first online on 11 March 2017. Its DOI is: 10.1007/s11042-017-4556-6. It is published as Open Access and can be accessed online from: https://link.springer.com/article/10.1007/s11042-017-4556-6. The published version is appended here.

Multimed Tools Appl
DOI 10.1007/s11042-017-4556-6

Interactive live-streaming technologies and approaches for web-based applications

Luis Rodriguez-Gil[1,2] · Pablo Orduña[1,2] · Javier García-Zubia[1,2] · Diego López-de-Ipiña[1,2]

Received: 25 August 2016 / Revised: 9 January 2017 / Accepted: 27 February 2017

Abstract Interactive live streaming is a key feature of applications and platforms in which the actions of the viewers affect the content of the stream. In those, a minimal capture-display delay is critical. Though recent technological advances have certainly made it possible to provide web-based interactive live-streaming, little research is available that compares the real-world performance of the different web-based schemes. In this paper we use educational remote laboratories as a case study. We analyze the restrictions that web-based interactive live-streaming applications have, such as a low delay. We also consider additional characteristics that are often sought in production systems, such as universality and deployability behind institutional firewalls. The paper describes and experimentally compares the most relevant approaches for the study. With the provided descriptions and real-world experimental results, researchers, designers and developers can: a) select among the interactive live-streaming approaches which are available for their real-world systems, b) decide which one is most appropriate for their purpose, and c) know what performance and results they can expect.

Keywords Webcam · Live streaming · Remote laboratories · Online learning tools · Rich interactive applications

✉ Luis Rodriguez-Gil
luis.rodriguezgil@deusto.es

Pablo Orduña
pablo.orduna@deusto.es

Javier García-Zubia
zubia@deusto.es

Diego López-de-Ipiña
dipina@deusto.es

[1] Faculty of Engineering, University of Deusto, Avda. Universidades, 24, 48007, Bilbao, Spain

[2] DeustoTech - Deusto Foundation, Avda. Universidades, 24, 48007, Bilbao, Spain

Published online: 11 March 2017

1 Introduction

The latest social trends and technological advances have led to the emergence of various popular web-based live streaming platforms, such as YouTube Live,[1] TwitchTV,[2] Instagram Livestream[3] and Facebook Live.[4] These platforms are designed to maximize scalability and, though they are indeed live, they still allow a relatively high delay (several seconds or more). This enables them to use a larger buffer, heavier compression and more effective transcoding techniques than they otherwise could. The work in [48] provides further detail on these issues and outlines the TwitchTV architecture, which is a good example. Specifically, the measured broadcast delay of that platform varies between 12 to 21 seconds. The negative impact on user experience is not too high, because for non-interactive live-streaming applications —such as live sports—, such a delay is acceptable.

However, there are also many applications of live streaming which need to be interactive. In interactive live streaming systems, the viewers affect the content of the stream. A common example is a videoconference application, in which viewers interact with each other. Other example are remote laboratories, which will be used in this work as the main case study. These labs allow remote students to view specific hardware through a webcam and interact with it remotely in close to real time. Figure 1 characterizes the different types of streaming and some of its applications.

Interactive live-streaming systems share some challenges with standard live-streaming platforms. One of those is the importance of being web-based. Throughout the last years there has been a powerful trend towards shifting applications to the Web. However, certain features, such as multimedia, have traditionally had more limited support [31, 41]. Applications that depended on them had to find workarounds: many chose to rely on non-standard plugins [9], such as Java Applets[5] or Adobe Flash.[6] Others accepted a significant decrease in their quality or performance, or could not be migrated at all. Today, with HTML5 [16] and with other related Web standards such as WebGL [44], this is starting to change. One of the features for which applications have traditionally had to rely on external plug-ins was video streaming. Now, as an example, large websites such as Youtube or Netflix[7] rely by default on HTML5 [47].

Applications that require interactive live-streaming, however, have additional requirements, expectations, and limitations. VOD (Video-On-Demand) streaming applications, such as Youtube or Netflix, are the most common platforms. Because videos exist far in advance before the users view them, they can be preprocessed at will. They can use heavy compression and prepare the video for different qualities and transmission rates. Also, they can be streamed through adaptive streaming with relative ease. Also, they rely on buffering to provide a greater quality despite network issues, and to be able to use a larger compression frame. Live applications, however, have limitations at those respects. As previously mentioned, those that are not interactive (e.g., broadcasting a live sports event), can withstand

[1] https://www.youtube.com/live

[2] https://twitch.tv

[3] https://instagram.com/livestream

[4] https://live.fb.com

[5] http://java.com

[6] http://www.adobe.com/es/products/flashplayer.html

[7] https://www.netflix.com

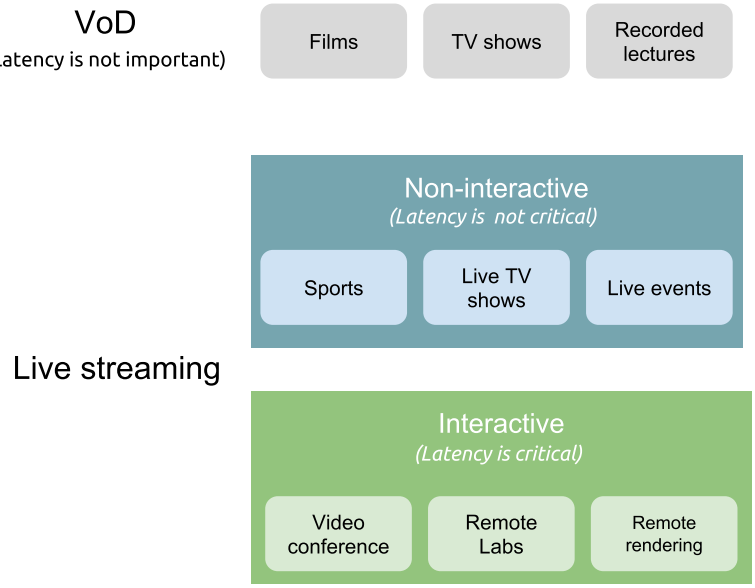

Fig. 1 Characterization of the different types of streaming and some of its applications

several seconds delay without issues. For those that are interactive (e.g., remote laboratories, collaborative tools, video conferencing applications) more than a second delay is already high: according to some HCI analysis, beyond a 0.1 seconds delay the user can notice a system is not reacting instantaneously, and beyond 1 second the user's flow of thought is interrupted [27].

In this context, researchers and system designers and developers that want to implement interactive live-streaming systems face certain difficulties. Major live-streaming platforms are closed and proprietary. It is difficult to use them for learning and research purposes [48], and they are not suitable for interactive live-streaming or as middleware for other applications. Moreover, the schemes that are available for implementing interactive live-streaming are complex. For a real-world usage, the adequacy of a scheme may depend on the video format, on the communication scheme, on the compatibility of different browsers, on the resources and bandwidth available, etc. Most of those aspects, individually, are examined in the literature. However, the real-world performance and limitations of the different real-world schemes cannot be readily predicted from it. There is little real-world experimental data that researchers and developers may use to take a truly informed decision on the approaches they choose. The main goal of this work is to provide them with that information.

In this paper, in Section 2, we describe the goals and contributions of this work and the particular requirements of web-based interactive applications that rely on live-streaming that we will consider. To illustrate the case practically, we put special focus on remote laboratories and educational applications. We propose some criteria through which the effectiveness of each approach can be compared. Then, in Section 3, we examine and describe several approaches that Web-based interactive applications may use for providing live-streaming capabilities. The five of them that seem potentially more relevant according to the previously defined criteria are described in more detail, and selected for further experimental comparison. In Section 4 we describe the experiments that have been conducted to measure the effectiveness and real-world performance of those five approaches. In Section 5, we compare the results of the different experiments. In Section 6 we examine the results and comparison and we offer an interpretation and some guidelines for potential application.

Finally, in Section 7 we draw a number of conclusions and we outline some possible future lines of work.

2 Motivation

2.1 Challenge and purpose

In a live streaming system the content is typically produced while it is being broadcast. That is, essentially, what differentiates it from non-live systems. However, there is still a very significant delay between the moment a frame is captured and the moment it is displayed in the target device [1, 15]. This delay is not only the result of network or hardware latency. It is built into the design to achieve a higher scalability [43]. In non-interactive live streams, a delay of seconds does not typically harm the QoE (Quality of Experience), and it makes it possible to leverage techniques such as buffering, video segmentation and high-compression motion codecs. An example of this is the case of the Twitch[8] platform. It relies on different techniques depending on the target device, but it tends to have a higher than 10 seconds delay [48]. Other example is the YouTube[9] live streaming platform, which lets users choose between better quality or lower latency. Even in the lower latency mode a capture-display delay higher than 20-30 seconds is, reportedly, not unexpected.[10] This is appropriate for several types of applications, such as broadcasting a sports event. However, for certain interactive applications, delays higher than a second, as previously established, can already be considered high.

An *interactive* live streaming application differs from a non-interactive one. Users are not simply passive spectators to the content. Instead, they are able to interact with it or through it, affecting the stream [48]. This imposes a strong constraint on the maximum acceptable capture-display delay that is not present in other types of live streaming. It could thus also be considered as *near-real-time* streaming. Some of the potential applications for interactive live streaming (see also Fig. 1) are the following:

- **Videoconferencing software**, such as Skype,[11] Google Hangouts,[12] or Apple Facetime,[13] in which users view, listen and react to each other in *near-real-time*.
- **Surveillance systems**, in which the viewer should be able to see what is happening in almost real-time.
- **Remote rendering systems**, in which the server handles the rendering and sends the video to the client in real time. An example is cloud-based gaming [36]: rendering a videogame in the server-side and forwarding the input from the client. Other example is free-viewpoint rendering [40]: in such a system, with many video inputs, the server has a huge amount of video data. To reduce bandwidth requirements, only the relevant portions are served to the client in real time.

[8] http://www.twitch.tv

[9] http://www.youtube.com

[10] Though no official figures are provided by YouTube, several observations and informal tests are available, such as those found at http://blog.ptzoptics.com/youtube-live/low-latency-streaming/ or at the Google product forum (https://productforums.google.com/forum/).

[11] https://www.skype.com

[12] https://hangouts.google.com

[13] https://apple.com/facetime

– **Interactive remote laboratories**, in which users interact with real physical equipment located somewhere else with a webcam stream as their main input.

The contributions of this work are intended to be useful for any web-based interactive live streaming application. However, due to the experience and background of the authors, the examples of this work will mainly relate to this fourth type of application: remote laboratories. Nowadays, remote laboratories often rely on relatively old technologies and approaches to provide interactive live streaming. Examples of such an approach is refreshing an image from JavaScript, or relying on the M-JPEG codification scheme. It is currently not clear, however, which of these relatively old approaches are more effective. Also, it is not clear whether newer approaches are not being used due to:

– Inertia and developer preference.
– More advanced technologies (such as adaptive streaming, video segmentation, or high-compression codecs) not being effective for near-real-time streaming.
– Newer approaches having significant real-world issues, such as portability issues, low reliability or difficulties to deploy behind institutional proxies.
– No literature available on the approaches available, their effectiveness, and the expected real-world outcome.

This work thus aims to shed more light in that area. The goal is that the remote laboratory community in particular and other interactive live streaming applications in general have the information to make better decisions on which streaming approaches to implement. And, moreover, so that they can know what effectiveness and performance they can expect by doing so. It also aims, specifically, to describe the currently used approaches and their architecture, and to propose some novel ones.

2.2 Contributions

The contributions of this work are thus the following:

– A brief analysis of which characteristics are important for interactive live streaming applications.
– Description and architecture of the most common interactive live streaming approaches that are currently used by remote laboratories (JavaScript-based image refreshing, and native M-JPEG).
– Description and architecture of some more advanced approaches, which, to our knowledge, have not been used in real-world remote laboratories but which could be superior. (JavaScript-based M-JPEG, JavaScript-based MPEG-1 and JavaScript-based H.264/AVC, all three relying on Web Sockets as a transport).
– Experimental analysis of the support for these approaches across all major desktop and mobile browsers.
– Experimental performance comparison of those described approaches that are most relevant.
– Scientific knowledge for existing developers of systems that rely on interactive live-streaming, that enables them to make educated decisions on the feasibility and convenience of incorporating alternative technical approaches into their implementations.
– Conclusions, based on the results of the experiments, on which approaches would be more appropriate depending on the type of remote laboratory required.

Of all of those contributions, the main one is the experimental performance comparison among the most relevant web-based approaches.

2.3 Remote laboratories

A remote laboratory is a software and hardware tool. It allows students to remotely access real equipment located somewhere else [9, 13, 24]. They can thus learn to use that equipment and experiment with it without having it physically available. Research suggests that learning through a remote laboratory, if it is properly designed and implemented, can be as effective as learning through a hands-on session [5]. Additionally, they can offer advantages such as reducing costs [26] and promoting sharing of equipment among different organizations [29]. Many remote laboratories feature one or several webcams. Through them, users can view the remote equipment. Simultaneously, they can interact with the physical devices using means such as virtual controls that are physically mapped to them. (e.g., [14, 20, 42, 46]). Some remote laboratories are even designed to allow access from mobile devices [8]. An example of remote laboratory is depicted in Fig. 2. In this particular case,[14] the students experiment with the Archimedes' principle. They can interact with 6 different instances of equipment, for which 6 simultaneous webcam streams are needed.

2.4 Technical goals and criteria

We propose a set of technical goals and criteria to compare and evaluate the different interactive live streaming approaches that will be examined.

The key technical goals that will be considered are the following:

- **Near-real-time**: The delay between the actual real-life event and the time the user perceives it —the latency— should be minimum for the interaction to be smooth.
- **Universality**: The applications should be deployable under as many platforms, systems and networks and as easily as possible.
- **Security**: The applications should be secure.

Though less critical, the following traits significantly affect the Quality of Experience and will be taken into account when evaluating the different possible approaches:

- **Frame rate**: The higher the better.
- **Quality**: The higher the better.
- **Network bandwidth usage**: The lower the better.
- **Client-side resources**: CPU and RAM usage. The lower the better.

Server-side processing is also an important consideration, especially for production systems. Though it will be considered and discussed evaluating it quantitatively is beyond the scope of this work — which focuses mainly on the client-side. Therefore, the experiments themselves include no server-side measurements.

A last consideration is the **implementation complexity** of each approach. Beyond the previously mentioned criteria, in practise, the knowledge, cost and effort required for implementing a specific interactive live-streaming approach is also, in many cases, a determining factor. Evaluating this complexity quantitatively is beyond the scope of this work.

In the following subsections we briefly describe a simplified streaming platform model. Additionally, we provide further detail and rationale about the aforementioned technical goals and criteria.

[14]The Archimedes' principle remote laboratory is usually publicly available at: https://weblab.deusto.es/weblab/labs/Aquatic%20experiments/archimedes/

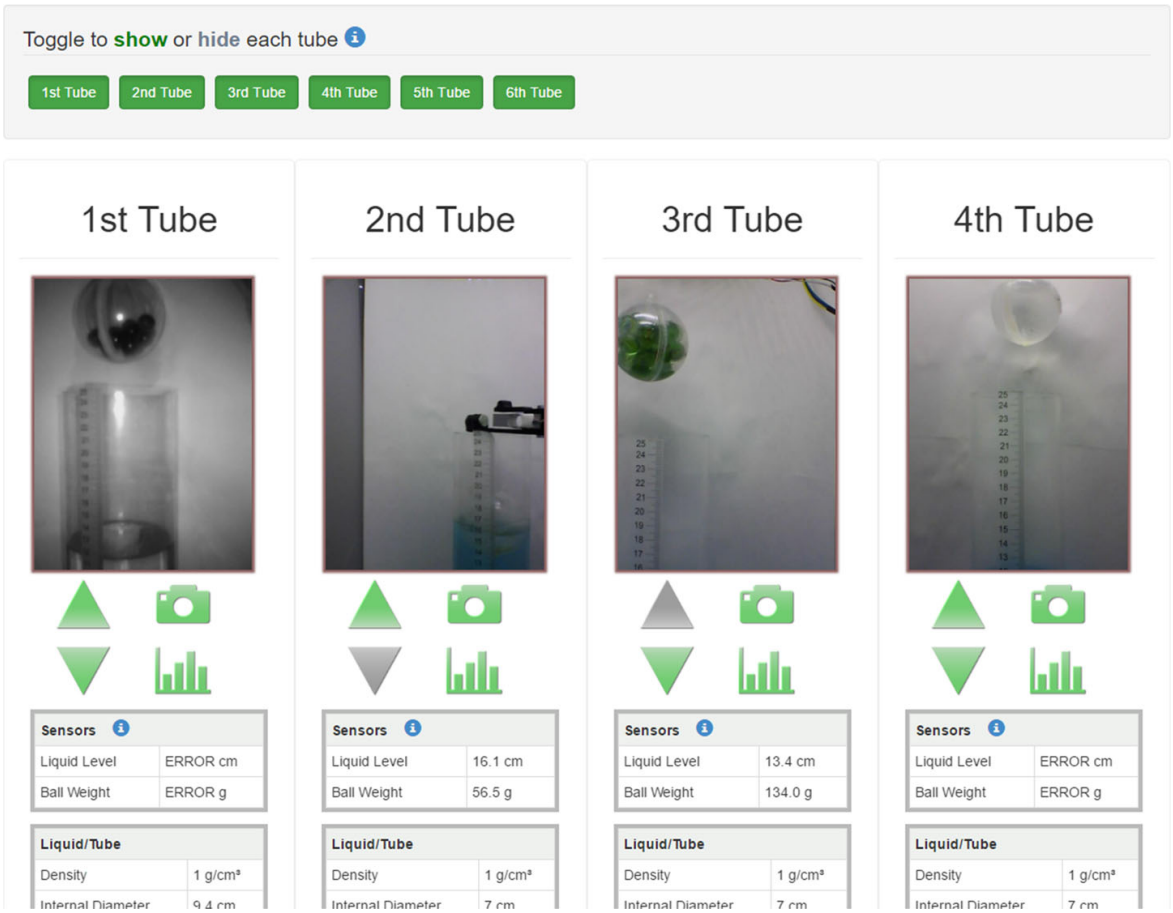

Fig. 2 Archimedes principle remote laboratory at the University of Deusto

2.4.1 Simplified live-streaming platform model

Different live-streaming platform models may exist. A simplified one is shown in Fig. 3. It is also the general model that is considered in this work. A set of IP cameras provide their input to the streaming platform through a camera output format. The particular format will vary, because different camera models support different formats. Common ones are, for instance, JPG snapshots, M-JPEG streams, and, in newer models, the H.264 format. The streaming platform receives the input and transcodes it into the target format. Often, the platform will also briefly act as a cache server for the input, so that it can scale for

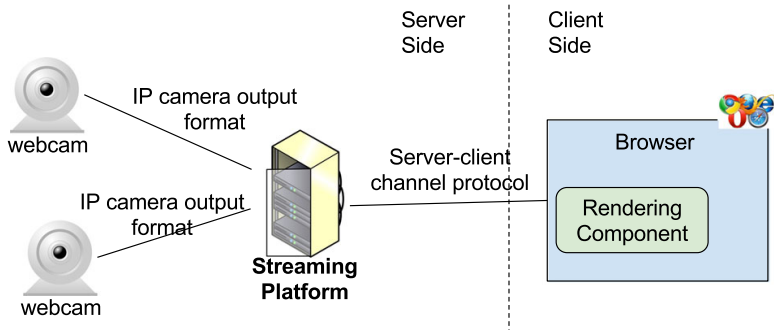

Fig. 3 Simplified live-streaming platform model

an arbitrary number of users without increasing the load on the webcams. The transcoded output is served through the server-client channel protocol (e.g., standard HTTP, AJAX, Web Sockets) to the client's browser. Depending on the approach, the browser will render it natively or through other means.

2.4.2 Near-real-time

In a live-streaming context end-to-end latency (sometimes also known just as *latency*), is generally considered to be the time that elapses between the instant a frame is captured by the source device and the time it is displayed on the target device. For most types of live streaming applications a relatively high (some seconds) latency can be tolerated without significantly harming the user's experience [6, 32]. Latency is introduced in each stage of the process. Noteworthy delays are the latency introduced by the camera, the latency introduced by the server-side encoding, the latency introduced by the network and the latency introduced by the client (decoding and displaying). These sources of latency are analyzed and discussed in detail in the white paper by Axis Communications [22]. Tolerating a relatively high delay is a significant advantage. Especially in a bandwidth-constrained network, codecs that provide large compression but which require heavy pre-processing can be used. Issues such as *jitter* can be solved with a longer buffer. Most HTTP streaming methods rely on buffering to provide adaptation for bandwidth fluctuation, and often separate the stream into multiple media segments. This adds an unavoidable capture-display delay [23].

Interactive live-streaming applications are much less tolerant to latency. The actions of the users depend directly on what they are currently seeing on the stream. A few seconds delay is enough to severely harm their Quality of Experience. Exactly how much latency can be tolerated and how much it affects user experience varies depending on the application. For example, some works report that in conversational applications (e.g., IP telephony, videoconferencing) 150 ms is a good latency, while 300-400 ms is not acceptable [32]. For cloud-based games some studies suggest that approximately for each additional 100 ms latency there is a 25% decrease in player performance [3]. For many other types of common interactive live streaming applications, such as remote laboratories, there is, to our knowledge, little specific research available on how much increased latency affects user experience. However, the interaction style and pace of many of them, such as remote laboratories themselves, is generally similar to that of a standard application or interactive website. Thus, it is reasonable to assume that generalist interaction conclusions are appropriate. In this line, according to works such as [27], beyond a 0.1 seconds delay the user can notice that a system is not reacting instantaneously, and beyond 1 second the user's flow of thought is interrupted.

Due to all this, supporting near-real-time (which for the purpose of this work, we will consider as being able to provide a relatively low end-to-end latency) is a particularly important requirement for an effective interactive live streaming approach, and the set of techniques that can be applied are significantly different than those that are applied for standard live-streaming or for VoD (Video on Demand). Modern techniques which are very popular and effective for standard streaming are sometimes not an option anymore, or are severely limited:

- **Buffering**: Would add a delay of at least the buffer length, so it can't be used or needs to have a minimal length.
- **Segmented streams**: Would add a delay of at least the segment length.
- **Pre-transcoding**: Not really an option if a small delay is required.

2.4.3 Universality

The meaning and usage of *universality* varies between contexts, but in this paper we will use it to refer to the degree to which an application is technically available to those who may want to use it. Aspects which increase universality are, among others, the following:

- Being cross-platform
- Being web-based
- Being available across many types of devices (PCs, mobile phones, tablets)
- Having less technical requirements to run properly
- Requiring less user privileges to run
- Being deployable behind more strict institutional firewalls and proxies

Universality is generally positive, but it is important to note that, in practise, it often implies important trade-offs. Depending on the particular context, needs and requirements of an application, the actual importance of universality will vary. In the case of remote laboratories, research suggests that it is one of the most important characteristics [9], but in other cases this might differ. It is noteworthy that this work aims to contribute to web-based interactive applications, which, for being web-based, already tend to provide relatively high universality.

2.4.4 Security

Being secure can be considered a goal of any application. However, the importance will vary depending on the context. Some technologies tend to provide greater security than others. For example, remote laboratories and other educational applications are often hosted by universities. Their IT teams are often hesitant to offer intrusive technologies to students to avoid exposing them to security risks, for which the university could be liable [9]. All things equal, *non-intrusive* technologies are thus preferred.

2.4.5 Frame rate

The frame rate is measured through the frames-per-second (FPS) metric. In some contexts, 50-60 FPS is considered to be a satisfactory visual quality at which increases can hardly be noticed. However, in practise, in many cases, significantly lower frame rates are used [32]. This is often in fact the case for many interactive live streaming applications.

2.4.6 Quality

Quality is hard to measure because it is actually a qualitative perception that is affected by many (qualitative and quantitative) variables. Sometimes (e.g., in Youtube) it is used as a synonym for *resolution* or *pixel density*. For simplification, in the comparison of the different approaches, we will rely the most on the *resolution*. The particular video codec that is used also has great influence in the final quality of the stream.

2.4.7 Network bandwidth usage

Live-streaming applications consume significant amounts of network bandwidth. This is because video content tends to consume significant bandwidth itself, and because often it has to be provided to many users [38]. Bandwidth usage can thus be a significant cost

and limitation, and all things equal approaches that preserve network bandwidth are preferred. Unfortunately, there tends to be an inverse correlation between network bandwidth usage and required server-side and client-side processing. That is, the codecs that require the less bandwidth tend to also be the ones that require the more processing power to code (server-side) and to decode (client-side). Sometimes specialized hardware is relied upon to provide more efficient decoding. Adding to the difficulty, some network setups, particularly mobile ones, are inherently unstable and their bandwidth capacity cannot be predicted reliably [23, 39].

2.4.8 Client-side resources

Different approaches and implementations require different amounts of CPU power and RAM. The codecs used, particularly, have a very significant influence at that respect. Client-side processing effort tends to be higher for the codecs that require the less bandwidth. To compensate for this, however, many devices also provide hardware-level support for particular codecs. Relying on hardware-level support is most of the time significantly more efficient, in terms of processing and energy usage. At the same time, because support tends to vary between different devices, it can sometimes make portability harder. In this work, the client-side processing effort will be measured in terms of CPU and RAM usage, though additional variables could be taken into account, such as energy cost, I/O usage or discrete graphic card usage. It is noteworthy that some applications have different client-side processing restrictions than others. All things equal, lower resources usage is better: A Video on Demand (VoD) application, for instance, could admit a relatively high usage in exchange of low bandwidth and high quality. There is a single active stream and the user is not expected to be multi-tasking. However, a remote laboratory or an IP surveillance application which requires being able to observe many cameras at once would often have stricter limits. See for instance the remote laboratory in Fig. 2. The students have access to 6 different simultaneous streams. Through them, they must be able to interact with the equipment in real-time. Thus, the resource usage of an individual stream must be significantly conservative.

2.4.9 Server-side processing

Server-side processing can be very high due to the pre-processing, compression and encoding that is sometimes used. Large media servers and systems, and especially those that aim to scale to many concurrent users per stream, such as Wowza,[15] YouTube and Netflix, encode a given source video into many separate formats and qualities. Thus, they can dynamically adapt the stream to the bandwidth and technical restrictions of each user. A higher or lower quality stream can be served depending on the bandwidth that the user has available. Also, one format or another can be served depending on whether the user's device or browser supports that format or not, and depending even on whether the user's device supports hardware acceleration for that format. For interactive applications the possible choice of codecs and formats is more limited, because the latency cannot exceed certain values. Also, it is noteworthy that for applications which do not aim to scale to many concurrent users per stream, but which instead aim to serve a relatively high number of different streams (such as many remote laboratories) it is sometimes convenient to accept a higher bandwidth usage in exchange of a lower processing effort.

[15] http://www.wowza.com

Though server-side processing could thus be an important consideration, this paper focuses on the client-side and therefore, though server-side considerations will be briefly described, experiments will focus on the client-side.

2.4.10 Implementation complexity

In practise, in production systems, the main factor for choosing an interactive live-streaming approach will often not be the technical characteristics or performance, but its implementation complexity. Technically superior approaches may be overlooked in favour of approaches that require less knowledge and effort and have a lower cost to implement. The quantitative evaluation of the implementation complexities of each of the different approaches is beyond the scope of this work. It would be hard to do and very difficult to reach meaningful results, due to its often developer-specific and subjective nature. Nonetheless, the architecture used for each experiment will be described and thus the implementation complexity may be partially inferred from it.

3 Interactive live-streaming approaches

In this section we will describe and analyze some of the different approaches that are available for web-based interactive live streaming. The first of them (image-refreshing and native M-JPEG based approaches) are often used by the architectures, use-cases and implementations of live streaming applications that can be found in the literature. Additionally, we include some novel approaches which are more rarely used, some of which have only recently become available due to their reliance on new or under-development Web standards.

The diagram in Fig 4 shows the generalized (and simplified) process that the interactive live streaming platforms follow from the time a frame is captured to the time it is displayed. The flow starts when the IP webcam captures a frame. Many different IP webcam models exist, and different models support different output formats. Some of the most common are JPG (discrete images), M-JPEG and H.264 streams. Through those formats, the output

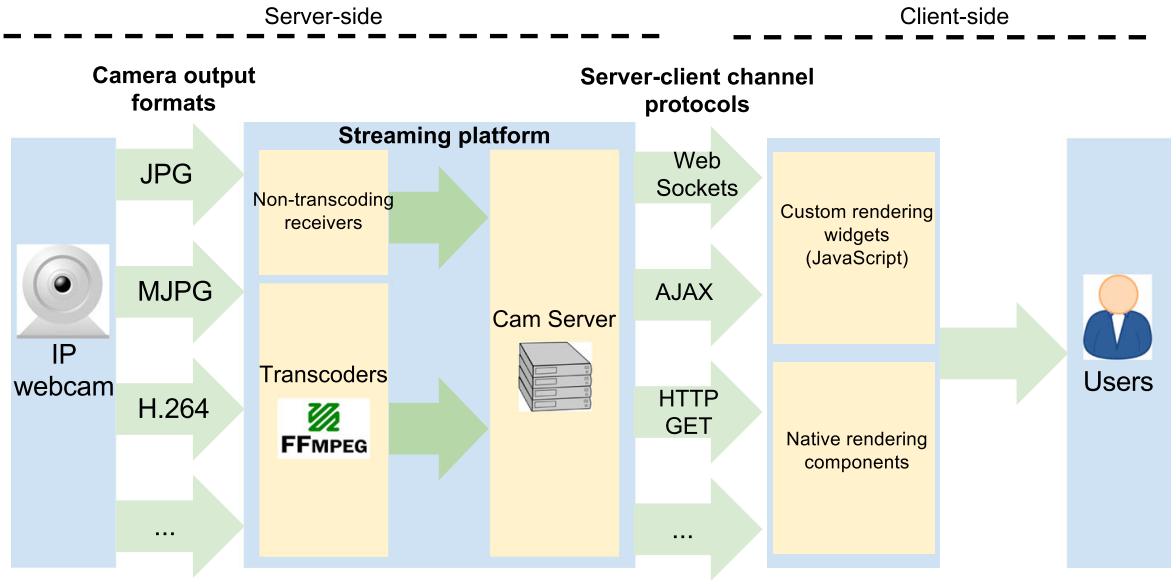

Fig. 4 Interactive live-streaming platform process

from the cameras is forwarded to the streaming platform. Depending on the source and target formats, it might or might not be necessary to transcode it into a different format. Transcoding can take a significant amount of processing power and adds some latency. The cameras server will briefly store these images, and will be responsible of serving them to the browser. Different server-client channel protocols are available. Once delivered, the browser will render the provided data to the user. Depending on the particular scheme, it will rely on a native component (such as for native M-JPEG videos or for image refreshing) or it will process, decode and render the data through JavaScript.

In the following subsections we will describe in more detail the different schemes. We will describe more thoroughly those schemes which best seem to meet the criteria described in Section 2, and which we will select for the experimental comparison. For most schemes, we will describe separately the server-side and client-side.

3.1 JavaScript-based image refreshing

This is a mature technique which is technically very simplistic, both in the client and the server-side. The server simply provides individual access to the current frame and the client repeatedly asks for a new frame using JavaScript. Despite its technical shortcomings it is used by many applications in the literature, sometimes as a fallback. Particularly, it is used by many remote laboratories for which a high FPS isn't absolutely necessary, including most of the remote laboratories of WebLab-Deusto,[16] LabsLand[17] and RemLabNet[18] [34].

3.1.1 Client-side

In the client-side only a browser with JavaScript support is required. The HTML contains an `<img>` tag which points at the webcam image. Then, from JavaScript, the image tag's `src` attribute is constantly modified. Under all modern browsers —and most legacy ones— when the `src` tag is changed the new image is loaded. To prevent some technical issues, generally some additional low-level considerations are taken:

- To modify the webcam image's URL in some way (such as by adding a random number to the *GET query* parameters) so that no cache issues occur.
- To query for a new image only after the `load` event has fired, sometimes after a small delay, so that requests don't accumulate on slow networks.

3.1.2 Server-side

Server-side this particular approach is also relatively simple. Most IP webcams provide an image URL that serves one frame, so that URL just needs to be made available either directly or through a standard web server. Some applications require several concurrent users and the hardware of webcams is sometimes not particularly powerful, so a cache server can be used. In that case, a cache server in the same network would continuously request frames to the camera. Then, whenever the web server asks for a frame, the last cached frame is served.

[16] https://weblab.deusto.es

[17] http://labsland.com

[18] http://remlabnet.eu

3.2 Motion JPEG

Motion JPEG, often known as just M-JPEG, is a video format with intraframe-only compression: each frame is essentially a separately compressed JPEG image. Because of this, its compression rate tends to be significantly lower than that of most modern interframe formats [4] (e.g., H.264/MPEG-4 AVC [19], H.265 [18], VP8 [2], VP9 [11]). However, it does have some significant advantages [21]: it is simple to implement, it requires little memory and processing power, it responds better than other formats to packet loss, fast image changes [4], and network jitter [28]. Many IP webcams provide native support for M-JPEG streaming, and M-JPEG is supported by most modern browsers, including the desktop and mobile versions of Google Chrome, Mozilla Firefox, Apple Safari and Microsoft Edge. It is not natively supported, however, by Microsoft Internet Explorer, and there have been significant issues with the implementations of most of these browsers. Examples of interactive live streaming applications that currently rely on M-JPEG are the RexLab.[19] Other example are laboratories based on the iSES[20] SDK [33], which support both browser-native M-JPEG and a JavaScript-based M-JPEG decoder which receives the frame from the server through WebSockets.

Natively, browsers and HTTP servers implement M-JPEG by sending the videos through special mimetypes such as `multipart/x-mixed-replace` and using the Chunked Based Coding feature of HTTP 1.1 (RFC2616 [7]), which is a data transfer mechanism that allows the transmission of dynamically generated content. When the request is started the total length does not need to be known. Instead, an undetermined number of *chunks* can be sent, and the request can be completed any time by a final zero-length chunk. In the case of a M-JPEG stream, servers keep a long-running request, sending the separate JPEG images that compose the video as separate chunks.

3.2.1 Limitations of browser-native M-JPEG implementations

Although the M-JPEG video format itself is relatively effective for live streaming [10, 21] and it is indeed a popular format for applications such as webcams, digital cameras or remote laboratories, there are in practise some technical and reliability issues with the native M-JPEG implementations of current browers.

Though most major browsers nominally support M-JPEG natively, it fails to work under certain versions of Google Chrome, Mobile Safari and Firefox. Also, when the stream is interrupted or fails, browsers do not recover. Under some circumstances, such as receiving at a higher FPS than the bandwidth allows, Chrome and Firefox often stop displaying the image but keep internally receiving them.

Motion-JPEG has no particular bandwidth-control or timing provisions. Browsers tend to render the frames just as they are received. This works as expected when the bandwidth is high enough for the FPS. The latency is even particularly low when compared to modern formats such as H.264 because there is no interframe compression or buffering delay. However, when the server is sending frames at a higher rate than the client can receive and display, the frames tend to accumulate and thus a significant and growing latency is

[19] Brazillian consortium headed by the Federal University Santa Catarina (UFSC) remote laboratories. On 21th April 2016 at least 8 different remote laboratories are available at http://relle.ufsc.br, all of which rely on browser-native M-JPEG.

[20] http://www.ises.info

introduced. Avoiding this is non-trivial and is not always possible, and the implementations observed in this work do not make any particular provision at this respect.

3.2.2 Client-side

The client-side is, at least in principle, straightforward for those browsers that natively support M-JPEG, because including an `<img>` tag is all that is needed. In practise, some implementations will provide a fallback mechanism (e.g., in [25]) for browsers that do not support it or that encounter issues, which is not uncommon.

As described in the previous section, reliability and bandwidth control is often an issue. Browsers provide no particular Motion-JPEG-related functionality or semantics through JavaScript, so detecting and recovering from failures is non-trivial. Under the tested browsers (Google Chrome, Mozzilla Firefox) no *load* or *error* events are raised on the `<img>` element that hosts the M-JPEG image when the stream is interrupted. In principle, the binary data of the image could be repeatedly accessed in JavaScript through the HTML5 APIs and explicitly compared to verify whether it has changed or not. In practise, however, this is very costly in terms of performance and is by default disallowed for externally hosted images due to CORS restrictions. Furthermore, browsers seem to have issues handling high FPS, with Chrome, for instance, soon using 100% CPU on that tab and showing a blank image.

Most observed implementations simply choose to stream at a fixed and relatively low FPS to partially avoid these issues, or let the user or administrator configure the FPS.

An alternative is to avoid relying on the native M-JPEG capability of the browsers (which tends to be flawed) and to receive and render the stream through JavaScript instead.

3.2.3 Server-side

Many IP webcam models support M-JPEG natively, so some implementations simply redirect that stream to the end-users. However, if the system should support several concurrent users, a caching server is required to achieve an acceptable Quality of Service. The server can try to adapt to available bandwidth by avoiding to queue frames and instead sending always the latest captured frame. Thus, theoretically, if the client received and displayed the latest frame just when it is received, the capture-display delay would be minimized and the server and client would be mostly synchronized. In practise, however, intermediate systems (including the browsers themselves) often buffer the TCP stream, so this scheme does not always work as well as expected.

To avoid relying on the (often flawed) native M-JPEG browser support, and to rely on JavaScript instead, the server will have to send the stream through an alternative means, such as Web Sockets, WebRTC or AJAX. This way, the previously mentioned glitches can be bypassed, and it becomes feasible to adapt to the available bandwidth. This comes at the cost of a higher client-side processing.

3.3 High-compression formats

Image-refreshing and M-JPEG, as discussed, provide relatively poor compression, but they are nonetheless often used for certain types of real-time interactive applications. That seems to be, mostly, because:

– They require little client-side and server-side processing power.

– They are simple to implement and maintain.
– Encoding and decoding takes very little time so very little capture-display latency is added.
– They can be used in almost any platform and browser.

In other contexts, however, such as non-interactive live streaming, or such as VoD, those formats are generally not used and would often be regarded as suboptimal formats. Instead, those contexts often favour formats and approaches that take higher processing time, require a more complex architecture, and add some latency; but that require lower bandwidth and provide higher quality. Example formats are, for instance: MPEG-1 [17], H.264/MPEG-4 AVC [19], H.265 [18], VP8 [2], or VP9 [11]. Today, some of those are natively supported by some browsers and systems —though not necessarily for live-streaming—, and some formats can be decoded, at a potentially high processing cost, through JavaScript. Therefore, in the current state of things, approaches that rely on this kind of codecs could today be an effective alternative for near-real-time live video streaming. Especially, since they offer a particularly high compression-rate and a relatively high quality [12].

3.3.1 Client-side

Client-side there are, again, two possibilities. First, in the most ideal case, the browser and hardware will support the format natively through the HTML `<video>` tag, and the support will also provide enough facilities for near-real-time live streaming of that format. Unfortunately, this is not always a valid approach due to the following:

– The HTML5 standard provides the `<video>` tag but it does not include live-streaming support. Though that is likely to change in the future through the MediaExtensions API.
– Each browser supports a different set of codecs, so the server needs to generate different streams to be truly cross-platform.

As an alternative, or as a fallback, it is possible to use JavaScript-based decoders for some of the formats. Such a system receives the stream data through Web Sockets, AJAX, or similar; decodes it through JavaScript, which depending on the format can require significant resources; and then renders it into an HTML5 canvas. Though more costly in terms of processing power, it is truly cross-platform and compatible with any modern browser.

3.3.2 Server-side

Server-side this approach tends to be significantly more costly in terms of resources, especially if several streams need to be provided. If a single stream is provided, it is also more costly than in the case of image-refreshing or M-JPEG due to the compression and intra-frame nature of these formats. A significant difference is also that for an user to be able to join an ongoing stream, initialization steps are required. That sometimes involves sending initialization packets through a secondary data channel, and/or waiting for specific periodic frames before being able to join.

3.4 Non-standard plugins

Traditionally, multimedia features in the browser have been limited. This has led many media-dependant applications to rely on non-standard plugins such as Adobe Flash or Java

Applets. The remote laboratory described in [37], for instance, displays a webcam through the YawCam[21] Java Applet.

Adobe Flash, Java Applets and similar plug-ins have access to native TCP and UDP sockets, which makes it possible to use non-web streaming protocols such as RTSP [35]. Although this makes this approach particularly powerful, it also implies that it is less universal. The plugins themselves need to be previously installed, which requires administrator privileges that are not always available. Also, Java Applets, for instance, are not supported in mobile devices, and Flash support is very limited. Support in desktop browsers is better, but still, Chrome has dropped support for Java Applets, and Firefox is expected to drop it soon. Other browsers are likely to follow similar paths. These issues are also discussed in [30]. Furthermore, the plugins themselves and the usage of non-HTTP protocols have security implications. As a result, applications that rely on those would be unable to be deployed under many institutional firewalls and proxies without significant configuration and policy changes [9].

3.5 WebRTC

WebRTC (Web Real-Time Communication) is an API standard that is currently under development by the W3C. It is oriented for peer-to-peer multimedia communication. This technology can be very useful for certain applications, such as videoconferencing ones, which can benefit from being decentralized and peer-to-peer. However, it has certain constraints:

- It still requires a server to handle the connection process and signalling; and to route all data for those cases where the NATs or firewalls of the clients prevent them from directly connecting to each other.
- It is oriented towards peer-to-peer connections, so its usefulness is limited when the source of the content is a traditional centralized server.
- It is not yet an accepted standard, and it is not yet supported by every major browser.

For these reasons, WebRTC-based approaches will not be considered for the purposes of this work, though it may be useful to also compare them in the future.

3.6 HLS and MPEG-DASH

HLS (HTTP Live Streaming) is a protocol created by Apple which is intended to answer some of the challenges described in this work. However, it is, at least for now, not natively supported by all browsers, and it is not standard, so *universality* is limited.

MPEG-DASH (Dynamic Adaptive Streaming over HTTP) is an adaptive bitrate streaming standard that is currently in draft form. Support is growing and in the future it is likely to be a very effective alternative.

3.7 Limitations

Though several approaches were described in the previous section, there are many more potential ones which could be feasible, and more are likely to appear in the future. Thus, it is noteworthy that the previous list or this comparison is not meant to be exhaustive.

[21] http://www.yawcam.com/

4 Experimental work

In this section, we experimentally evaluate the performance of the five most relevant schemes (described in the previous section). First, we detail the experimental setup and methodology. Next, for each scheme, we:

- Implement the scheme.
- Conduct qualitative experiments to verify whether the scheme does indeed run under different systems and devices.
- Conduct quantitative experiments to evaluate its performance.

4.1 Experimental setup

At this stage we have selected 5 approaches to be compared quantitatively (which we will refer to as image-refreshing, native M-JPEG, JS M-JPEG, MPEG-1, H.264/AVC). We are interested in measuring the performance under real-world conditions. Thus, beyond the specific scheme being analyzed, there are several other variables which may affect the measurements. Some of these are the following:

- **FPS:** Target Frames Per Second.
- **Client device:** Computer or mobile device to render and take measurements in.
- **Network:** Latency, available bandwidth, etc.
- **Browser:** Browser and specific version.

Conducting experiments for every combination would be impractical, and not particularly meaningful. Certain restrictions have been applied for each of these variables, and will be described next.

4.1.1 FPS

Though some systems rely on a variable FPS, in this case we will set a target FPS. This makes it possible to obtain comparable results for RAM, CPU and bandwidth, and is consistent with real-world usage. When applicable,[22] we will conduct the experiments against three different FPS values: 5, 10 and 25. For some schemes, we will also measure the maximum average FPS they can achieve.

4.1.2 Client device

All the quantitative experiments have been conducted under *Device A*. In addition to the quantitative experiments, several qualitative ones were conducted to verify whether the specific schemes are indeed cross-platform. For those, two additional devices were used. The specification of the three devices are the following:

- **Device A** Mac Book Pro 13' Mid 2014: 2.6 GHz Intel Core i5, 8 GB RAM, 256 GB SSD, Intel Iris 1536 MB Graphics Card. Running OS X 10.11.5).
- **Device B** Desktop PC. Intel Core i7, 8 GB RAM. Running Windows 10.
- **Device C** Samsung Galaxy S7 Edge (SM-G935F).

[22] As described in more detail in later subsections, MPEG-1 will only be measured with 25 FPS, because its standard does not allow 5 or 10 FPS.

4.1.3 Network

There are mainly two network parameters which could be considered: bandwidth and latency. The experiments were conducted in a local network, with around 100 Mbps of bandwidth and around 20 ms of latency. Though both parameters are important, preliminary experiments suggest that they do not significantly affect the comparison.

In those preliminary experiments, the bandwidth did not affect the measurements, except when the measured bandwidth usage started getting close to the maximum network bandwidth. In these cases, either the target FPS could not be met (image-refreshing scheme) or the capture-render delay started growing beyond what could be reasonable for an interactive live-streaming system (the other schemes). Differences in latency seemingly have no effect in RAM, CPU or bandwidth. As expected, however, an increased network latency results in an increased capture-render delay. The relationship seems to be, as expected, mostly linear. So, though the measurements would vary on a slower network, the comparison and the conclusions should not.

4.2 Methodology and measurements

The implementations have been deployed in a server in the local network. The video feed is obtained from a local IP webcam. The metrics that will be measured are the RAM usage, CPU usage, downstream bandwidth and capture-render latency. They all will be measured in relation to a target FPS. We have kept them as separate metrics because, depending on the specific application, the most significant ones might vary. For instance, in certain mobile networks minimizing the bandwidth usage could be the most important criteria [39]. In other cases, such as in networks with more bandwidth, minimizing RAM or CPU usage could be more appropriate.

Bandwidth measurements are incoming-only, and have been obtained either through the Chrome Task Manager or the Mac OS activity monitor (because web socket bandwidth usage is not shown in Chrome). RAM and CPU were measured through the Chrome Task Manager. 5 measurements were taken in each case. The highest and lowest measurement were discarded. The 3 remaining measurements were used to compute the average and the standard deviation, which are listed in tables for each scheme.

For measuring the latency, an IP webcam was pointed towards a desktop computer displaying a clock on the screen. Then, the test laptop (Device A) was placed next to it, rendering the webcam image through the experimental streaming system, using the particular settings of each experiment. The latency is thus equal to the difference between the live clock (in the desktop) and its image rendered in the test laptop. For each experimental combination (FPS and streaming approach) 5 pictures were taken of both screens, and the difference measured. For these 5 measurements, the average and standard deviation was computed. Figure 5 shows a picture of this setup. The screen (in the middle) shows a clock. An IP webcam (in the lower left) is pointing to it and forwarding the stream to our interactive live-streaming platform. Then, the stream is being rendered in the laptop (to the right), using one of the schemes and configurations. When a picture is taken of both screens, the capture-render delay, at that moment, will thus be the difference between the clock and its render.

4.3 JavaScript-based image refreshing

The setup for this set of experiments can be observed in Fig. 6. The live images are retrieved from the IP camera and stored into a Redis cache server. A Python-Flask-based server serves

Fig. 5 Picture measuring the capture-render delay

those images to the browser, which simply applies the image refreshing technique from JavaScript to obtain the images through standard HTTP and then displays them at a given frame rate.

The implementation was run on device A (with Chrome, Firefox, Safari), device B (with Chrome, Firefox, Internet Explorer and Edge) and device C (with mobile Chrome and mobile Firefox). Works as expected and without noticeable issues in all of them.

The performance of the experimental implementation (conducted under device A and Chrome) is summarized in Table 1. It is noteworthy that RAM usage is particularly high. It increases steadily since the first image is loaded, and, after a while, it stabilises at around 600 MB. It is hypothesised that this is because of the Chrome caching and memory optimization schemes, which retains copies of the previously loaded images up until a certain point. The latency (capture-display delay) is relatively low. For all the tested frame rates, it is within the 223ms to 316ms range. The latency is lowest for 25 FPS. Although not covered in the table, it is noteworthy that at 10 FPS but with a constrained, simulated *Good 2G* connection, the delay is stable at around 1726 ms.

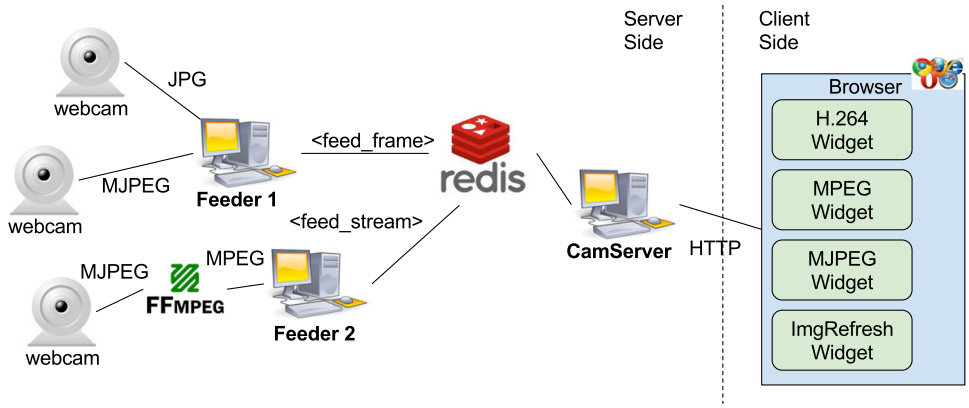

Fig. 6 Architecture and deployment setup for the different experiments

Table 1 JavaScript-based image refreshing performance

		Mean	S.D.
5 FPS	RAM	618.0 MB	2.646
	CPU	16.57%	0.115
	Bandwidth	233.0 KB/s	4.359
	Latency	316 ms	104.083
10 FPS	RAM	634.7 MB	0.577
	CPU	31.8%	0.100
	Bandwidth	461.3 KB/s	0.577
	Latency	246 ms	104.083
25 FPS	RAM	649.67 MB	17.098
	CPU	85.5%	0.709
	Bandwidth	1130.3 KB/s	1.528
	Latency	223 ms	107.480
Max	Achieved FPS	41 FPS	
	RAM	671 MB	
	CPU	116.9%	
	Bandwidth	2100 KB/s	

4.4 Native M-JPEG

The setup for these experiments is similar to the previous ones, and also depicted in Fig. 6. The live images are retrieved from the IP camera and stored into a Redis cache server. A Python-Flask-based server provides a M-JPEG stream from them to the client at a fixed target frame rate. The client displays the M-JPEG stream natively by simply using an ¡img¿ element to the stream. No particular JavaScript is needed. Thus, no particular 'Native M-JPEG' widget is present in the figure.

The implementation was tried on device A (with Chrome, Firefox, Safari), device B (with Chrome, Firefox, Internet Explorer and Edge) and device C (with mobile Chrome and mobile Firefox). Runs in all browsers except on Internet Explorer.

The performance of the experiment (conducted under device A and Chrome) is summarized in Table 2. Similarly to the previous experiment, the RAM raises steadily since the first image is rendered, and increases steadily until it reaches around 600 MBs. When not bandwidth-constrained, the capture-display delay is in the 218-515 ms range, being lowest for 25 FPS. When bandwidth-constrained, the delay steadily increases and can be higher than 60 seconds. No data for a maximum FPS is provided, because at higher FPS the stream fails sooner. This is likely due to the native M-JPEG limitations described in Subsection 3.2.1. Particularly, it seems that once a single image fails, the browser stops updating the image. And because there is no JavaScript API and no JavaScript error event raised, recovery is non-trivial.

4.5 JavaScript-based M-JPEG

The setup for this set of experiments is also depicted in Fig. 6. The live images are retrieved from the IP camera and stored into a Redis cache server. A Python-Flask-based server provides a M-JPEG stream from them to the client at a fixed target frame rate through

Table 2 Native M-JPEG

		Mean	S.D.
5 FPS	RAM	605.3 MB	1.155
	CPU	12.40%	0.265
	Bandwidth	235.3 KB/s	4.726
	Latency	515 ms	149.921
10 FPS	RAM	617.3 MB	6.658
	CPU	24.5%	0.866
	Bandwidth	445.7 KB/s	32.624
	Latency	356 ms	51.394
25 FPS	RAM	625.3 MB	1.155
	CPU	60.13%	0.503
	Bandwidth	1126.7 KB/s	12.014
	Latency	218 ms	55.426

Web Sockets and the *socket.io* library. The client renders each frame to an HTML5 Canvas through JavaScript.

The implementation was tried on device A (with Chrome, Firefox, Safari), device B (with Chrome, Firefox, Internet Explorer and Edge) and device C (with mobile Chrome and mobile Firefox). Runs in all browsers. No particular issues were observed in any of them.

The performance of the experiment (conducted under device A and Chrome) is summarized in Table 3. The RAM usage seems to increase slowly but it lowers periodically, and does not increase proportionally to the FPS. When not bandwidth-constrained the latency is

Table 3 JavaScript-based M-JPEG

		Mean	S.D.
5 FPS	RAM	166.0 MB	11.136
	CPU	6.867%	0.058
	Bandwidth	315.7 KB/s	3.215
	Latency	289 ms	71.924
10 FPS	RAM	430.0 MB	18.330
	CPU	12.5%	0.100
	Bandwidth	528.0 KB/s	34.511
	Latency	216 ms	57.735
25 FPS	RAM	306.3 MB	47.480
	CPU	28.433%	0.231
	Bandwidth	1494.3 KB/s	92.376
	Latency	284 ms	54.262
Max	Achieved FPS	115-127 FPS	
	RAM	263 MB	
	CPU	20.8%	
	Bandwidth	8434.3 KB/s	

relatively low and quite stable. It is within the 284-289 range, and it is lowest for 10 FPS, though given the small difference, it might be due to random fluctuations.

Higher than 115 FPS could be reached, and, strangely, in that case the CPU usage was actually lower than at 25 FPS.

4.6 JavaScript-based MPEG-1

MPEG-1 [17] is a very mature format and in many traditional streaming contexts it could be considered *legacy*. More modern formats such as H.264 —which will also be tested in later sections,— provide better quality and compression [45]. However, potentially, its simplicity also implies a lower processing cost and a lower latency. Considering this and that currently many applications (such as most remote laboratories) still rely on apparently less efficient approaches such as image-refreshing or M-JPEG; MPEG-1 could still be an effective option in some current contexts.

The setup for this set of experiments is also depicted in Fig. 6. It is slightly more complex than the previous ones because a transcoding component (from M-JPEG to MPEG) is necessary. Thus, in the server, a *ffmpeg* instance retrieves the live stream from the IP webcam (through M-JPEG), transcodes it into MPEG-1 and sends it to a *feeder* server. The *feeder* forwards the stream to the web server through Redis channels. The client receives the stream from that server through Web Sockets and *socket.io*. The stream is decoded using a JavaScript decoder and rendered into an HTML5 canvas. The MPEG-1 standard supports a limited number of FPS options, so in this case, the experiment was conducted only with 25 FPS (5 FPS and 10 FPS are not really allowed by the standard). The codec was configured to use an 800 Kbps constant bitrate.

Specifically, the system relies on the following ffmpeg command to transcode the stream:

```
ffmpeg -r 30 -f mjpeg -i <webcam_mjpeg_url> -f mpeg1video
-b 800k -r 25 pipe:1
```

Client-side, a pure JavaScript decoder has been used. The decoder is Open Source and was originally created by Phoboslab.[23] It has been modified to add support for the *socket.io* library, which is a wrapper around Web Sockets but falls back to a long-polling system if the specific deployment does not support the former. The modified decoder is available at Github.[24]

The setup was tried on device A (with Chrome, Firefox, Safari), device B (with Chrome, Firefox, Internet Explorer and Edge) and device C (with mobile Chrome and mobile Firefox). Runs in all browsers. No particular issues were observed in any of them. It is noteworthy, though, that MPEG-1 quality was lower than the previous approches (which was to be expected due to the higher compression).

The performance of the experiment (conducted under device A and Chrome) is summarized in Table 4. The RAM usage is very steady. The latency is on average 610 ms when not bandwidth-constrained. Though not included in the table summary, it is noteworthy that when simulating a *Good 2G* connection under Chrome, the latency is similar, probably because due to its low bandwidth requirements, the restricted bandwidth is not really constrained either.

[23] http://phoboslab.org

[24] https://github.com/zstars/jsmpeg

Table 4 JavaScript-based MPEG-1

		Mean	S.D.
25 FPS	RAM	104.0 MB	0.000
	CPU	6.9%	0.351
	Bandwidth	39.57 KB/s	3.066
	Latency	610 ms	325.087

4.7 JavaScript-based H.264/MPEG-4 AVC

H.264/MPEG-4 AVC [19] is currently a popular high-compression format, which is particularly appropriate for streaming, and is commonly supported as an output format for many modern IP webcams. The performance of a system relying on H.264 will vary significantly depending on the specific codec implementation, on the specific parameters for the codec, and on other factors. Thus, these experiments are not intended to evaluate the performance of the H.264 format itself, but, instead, the potential performance of a real-life interactive live streaming system that relies on it.

The setup for these experiments is depicted in Fig. 6. Its architecture is similar to the one that was used for the MPEG-1 experiment, and it also relies on a transcoding component (from M-JPEG to H.264). Thus, in the server, a *ffmpeg* instance retrieves the live stream from the IP webcam (through M-JPEG), transcodes it into H.264 and sends it to a *feeder* server. The *feeder* forwards the stream to the web server through Redis channels. The client receives the stream from that server through Web Sockets and *socket.io*. The stream is decoded using a JavaScript decoder and rendered into an HTML5 canvas.

The *ffmpeg* instance was configured to use the *libx264* codec, with the baseline profile, constant bitrate mode (set to 1500Kbps) and with several low latency flags enabled. Specifically, the *ffmpeg* commandline that was used is:

```
ffmpeg -r 30 -f mjpeg -i <webcam_mjpeg_url> -flags +low_delay
-probesize 32 -c:v libx264 -tune zerolatency -preset:v ultrafast
-r <target_fps> -f h264 1500k pipe:1
```

The client renders the stream through a modified *Broadway.js* decoder. *Broadway.js* is a heavily optimized, Open Source, H.264 JavaScript decoder, which has been compiled through Emscripten and is futher optimized to use WebGL. To it, we have added *socket.io* support, so that it can gracefully fall back to AJAX under systems and deployments where Web Sockets are not functional. The modified decoder is Open Source and is hosted at GitHub.[25]

The setup was tried on device A (with Chrome, Firefox, Safari), device B (with Chrome, Firefox, Internet Explorer and Edge) and device C (with mobile Chrome and mobile Firefox). Runs in all browsers. No particular issues were observed in any of them.

The performance of the experiment (conducted under device A and Chrome) is summarized in Table 5. RAM usage is steady despite the FPS. CPU usage increases with the FPS. Latency is highest at 5 FPS (under which it averages 1083 ms) and lowest at 25 FPS (under which it averages 417 ms). This difference is quite significant, and, in part, is probably due to internal buffers in the codec being filled faster at higher framerates.

[25] https://github.com/zstars/h264-live-player

Table 5 JavaScript-based H.264/AVC

		Mean	S.D.
5 FPS	RAM	177.3 MB	1.528
	CPU	11.67%	1.155
	Bandwidth	90.9 KB/s	22.848
	Latency	1083 ms	29.445
10 FPS	RAM	178.3 MB	5.859
	CPU	15.67%	1.155
	Bandwidth	96.2 KB/s	1.258
	Latency	734 ms	27.731
25 FPS	RAM	176.0 MB	1.000
	CPU	27.0%	2.646
	Bandwidth	100.3 KB/s	8.314
	Latency	417 ms	160.728

5 Comparison

Table 6 summarizes the browser support for each chosen approach and implementation. For the most part, it is very wide. This is to be expected because they were chosen, precisely, for providing relatively high *universality*.

Figure 7 compares the client-side RAM usage observed during the experiments. In the case of image refreshing and native M-JPEG it is, apparently, very high. However, it is noteworthy that when the page loads it starts small, and progressively grows until it stabilizes at the figures that are displayed. That is likely due to Chrome's caching and RAM management scheme. In the case of JavaScript-rendered M-JPEG RAM usage is significantly smaller, and it is rather volatile —which probably explains why for 25 FPS the measured RAM usage is smaller. The MPEG-1 and H.264 implementations consume, by far, the least RAM.

Figure 8 compares CPU usage. As one would normally expect, it seems to increase almost linearly with the FPS. Interestingly, however, native M-JPEG and image-refreshing consume the most CPU, while JavaScript-based M-JPEG and the high-compression-based methods —MPEG-1 and H.264— consume the least. This could be considered counter-intuitive, because MPEG-1 and H.264 have a significantly more powerful and complex, interframe compression. Although explaining these results would require further analysis

Table 6 Browser support for each approach and implementation

	Chrome	Safari	IE	Edge	Firefox	Chrome Mobile	Samsung Mobile	Firefox Mobile
Image Ref.	✓	✓	✓	✓	✓	✓	✓	✓
Native M-JPEG	✓	✓	✗	✓	✓	✓	✓	✓
JS M-JPEG	✓	✓	✓	✓	✓	✓	✓	✓
JS MPEG-2	✓	✓	✓	✓	✓	✓	✓	✓
JS H.264	✓	✓	✓	✓	✓	✓	✓	✓

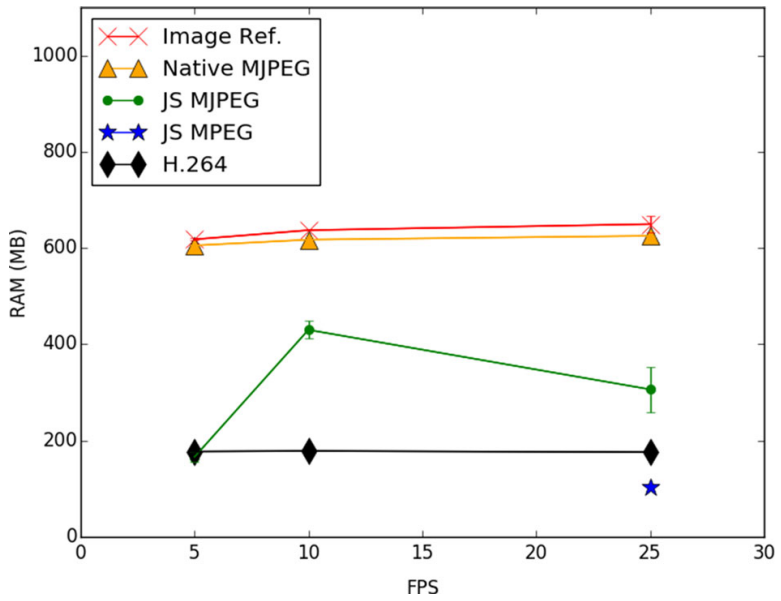

Fig. 7 Client-side RAM usage comparison

and is not within the scope or target of this work, some of the factors involved could potentially be:

- Higher bandwidth-requirements of the lower-compression formats increase the CPU usage as compared to the higher-compression but lower-bandwidth ones, especially in a browser environment where probably the buffers involved are copied several times.
- Image-refreshing is not what the browsers' image component and system was designed for, and modifying an image repeatedly is not necessarily meant to be efficient: it involves cache and DOM operations which are appropriate for single images but not so much for videos.

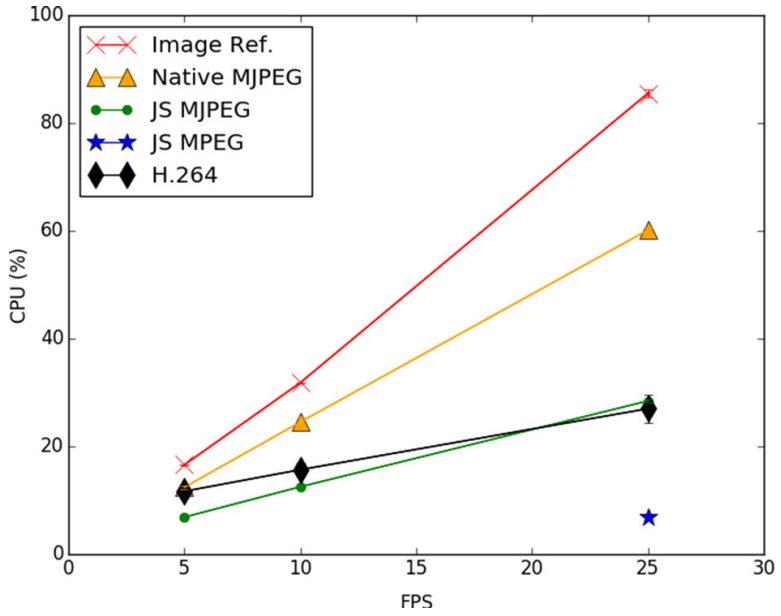

Fig. 8 Client-side CPU usage comparison

– The native M-JPEG implementation of Chrome might be significantly suboptimal, while the JavaScript engine is very heavily optimized, which would explain the difference between the JavaScript-based M-JPEG and the native M-JPEG experiments.
– The H.264 decoder has the advantage of being heavily optimized (compiled into JavaScript through Emscripten, and even using WebGL for some specific tasks such as color conversion) while the MPEG-1 decoder has the advantage of being relatively simple.

Figure 9 compares downstream bandwidth usage. Comparatively, MPEG-1 and H.264 consume a very small amount of bandwidth. Image-refreshing, native M-JPEG and JavaScript-based M-JPEG, which do not rely on interframe compression, consume several times more bandwidth and it increases proportionally to the FPS. JavaScript-based M-JPEG seems to consume slightly more than native, specially at higher framerates, possibly due to *socket.io* or WebSocket overheads.

Figure 10 compares the latency of the different approaches. For the low framerates image-refreshing and JavaScript-based M-JPEG are the fastest. H.264 is the slowest. However, for the higher framerates, the difference among the difference approaches decreases. The change is most apparent for H.264, which goes from an around 1100 ms delay at 5 FPS to a quite low around 500 ms delay at 25 FPS. This non-linear change is probably due to the codec's implementation details. Particularly, internal buffers are probably being filled faster at the higher framerates.

6 Discussion

We have examined and compared some techniques and implementations for web-based near-real-time interactive live streaming. Of cross-platform web-based techniques, image refreshing and native M-JPEG are currently among the most commonly used ones for applications such as remote laboratories. We have also examined and compared some approaches (Java-Script based M-JPEG rendering, JavaScript-based MPEG-1 rendering,

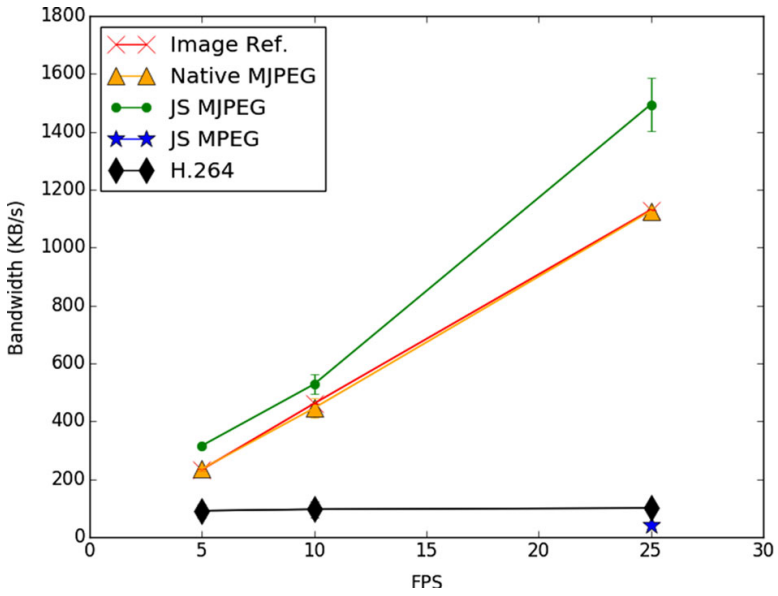

Fig. 9 Client-side downstream bandwidth usage comparison

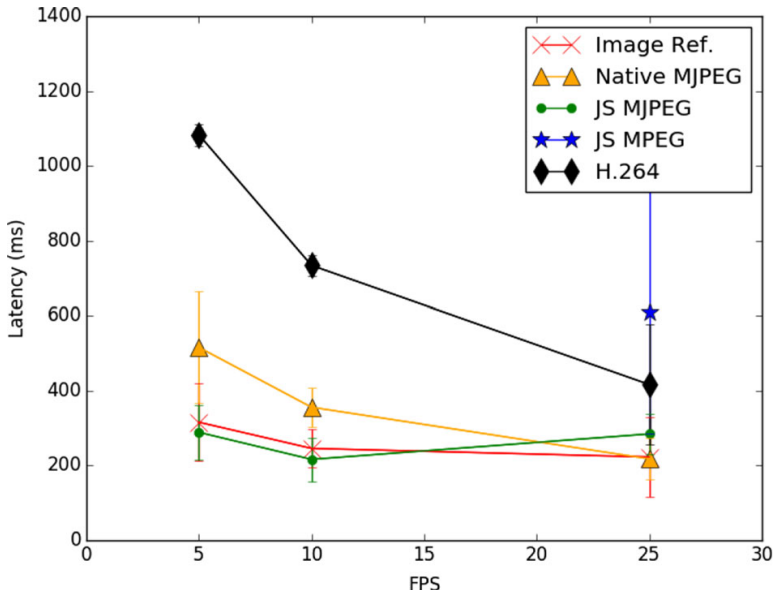

Fig. 10 Latency comparison

and JavaScript-based H.264/AVC rendering) which as far as we know, have not been used for remote laboratories (and possibly not for similar purposes such as IP camera servers).

The formats that some of those approaches rely on (particularly, M-JPEG and MPEG-1) are far from new. However, M-JPEG is still very popular for interactive live streaming because of its simplicity and because of the low capture-display latency that it provides. MPEG-1, similarly, is very mature. Before the appearance of newer formats, it used to be popular for web streaming. Some of those newer formats are H.264 AVC [19], H.265 [18], VP8 [2] and VP9 [11]. They provide a significantly better quality and compression rate, but they are more complex. They generally have a higher processing cost. Of those newer formats, we have considered an approach relying on JavaScript and H.264.

The results of the experiments show that no single approach is necessarily the best for all cases. They show, however, that some approaches can probably be significantly more advantageous than others for certain purposes.

One of the first conclusions that we learn from the experiments is that relying on the native M-JPEG scheme does not seem to be convenient in any case. This is remarkable because, currently, many remote laboratories and other applications, such as IP camera servers, rely on it. However, the only advantage seems to be a slightly lower bandwidth usage than JavaScript-based M-JPEG. In exchange:

- A major browser lacks support (Internet Explorer)
- Other browsers have several reported bugs and a relatively poor track-record at supporting it
- It consumes significantly more RAM
- It consumes significantly more CPU power

Additionally, other significant observation is that the performance of the most simplistic approach —image refreshing— is as good as that of native M-JPEG in RAM and bandwidth usage, and is not far in CPU power. Considering the aforementioned issues that come with native M-JPEG, and the significant advantages of image refreshing such as simplicity and trivial error-recovery and bandwidth-adaptation capabilities, we conclude again that in most cases, there would be no advantage in relying on native M-JPEG.

Other significant observation is that JavaScript-based MPEG-1 and JavaScript-based H.264 seem to be significantly advantageous in many of the evaluated variables. This is remarkable because, as far as we know, they have not been used in the context of remote laboratories. They require much less RAM, extremely lower amounts of bandwidth than the other approaches, and also less CPU power —especially in the case of MPEG-1—. The lower bandwidth requirement is expected due to their interframe compression, and the fact that a fixed bitrate was used. However, it is still noteworthy that it can be achieved while maintaining a (subjectively) good image quality. The lower CPU requirement is less obvious (some possible explanations are suggested in Section 5). These results suggest that, for some remote laboratories, particularly those that require a high FPS and which are particularly bandwidth-constrained (for example, because they are intended to be used on mobile devices, or because they feature many simultaneous webcams) relying on MPEG-1 and H.264 could be a very effective choice.

These formats, however, do have some disadvantages. First, the image quality —with the constant-bitrate, low-latency configuration that was chosen— is not bad, but is worse than for the other approaches. Second, they do indeed raise the latency. Nonetheless, by choosing the appropriate low-latency configuration, most measurements remain lower than 1 second. This is still acceptable for many interactive live streaming applications. However, it is certainly higher than with other more *traditional* approaches such as image refreshing or M-JPEG. Thus, despite the mostly superior results, they are not necessarily the best choice for all applications. It is also noteworthy that they have some additional disadvantages:

- The implementation and deployment is more complex. It requires a transcoding server, and a specific JavaScript-based player in the client.
- Deployment is more complex due to the transcoding server that is required. Server-side, more processing resources are required.
- Error recovery and bandwidth adaptation is harder than with other approaches such as image-refreshing (for which it is trivial).

7 Conclusions and future work

In this work we have described two of the most common approaches that are nowadays used for web-based near-real-time interactive live streaming (image refreshing and native-M-JPEG). Additionally, we have proposed and compared three additional approaches (JavaScript-based M-JPEG, JavaScript-based MPEG-1 and JavaScript-based H.264). Those last, to our knowledge, are not currently used in this context.

The results suggest that, in many cases, avoiding the native M-JPEG scheme in favour of one of the other four would be advisable. It seems to provide very little benefit over the alternatives. They also suggest that the three JavaScript-based alternative approaches (JavaScript-based M-JPEG, JavaScript-based MPEG-1 and JavaScript-based H.264/AVC) provide comparatively good performance.

The schemes based on MPEG-1 and H.264/AVC could be the most appropriate for certain types of interactive applications. Particularly, for those, such as certain remote laboratories, which can withstand a relatively high latency (around 1 second) but which require a low bandwidth usage at a high FPS. Nonetheless, the image-refreshing scheme, despite its simplicity, could still be the most appropriate scheme for those interactive applications which require a very low latency, especially at lower FPS rates.

In the future, it would be interesting to compare new approaches. MPEG-DASH is a promising HTML5-related standard which will likely provide near-real-time live-streaming. Once the support for it is wider, it would be useful to evaluate whether a low capture-display delay can be achieved, and how its performance compares against the alternatives.

Additionally, it would be interesting to evaluate more modern interframe compression formats, and with more configurations. Although the most modern formats (such as VP-9) are expected to require a much higher amount of resources, especially if decoding through JavaScript, a format such as VP-8 could still give interesting results. Especially, if the implementation was heavily optimized and relied on *asm.js* or a similar scheme.

Acknowledgments This work has received financial support by the Department of Education, Language policy and Culture of the Basque Government through a Predoctoral Scholarship granted to Luis Rodriguez-Gil.

References

1. Akhshabi S, Begen AC, Dovrolis C (2011) An experimental evaluation of rate-adaptation algorithms in adaptive streaming over HTTP. In: Proceedings of the second annual ACM conference on multimedia systems. ACM, pp 157–168
2. Bankoski J, Wilkins P, Xu Y (2011) Vp8 data format and decoding guide. Tech. rep., available: http://tools.ietf.org/html/rfc6386 (accessed: 2016-07-07)
3. Claypool M, Finkel D (2014) The effects of latency on player performance in cloud-based games. In: 2014 13th annual workshop on Network and systems support for games (netgames). IEEE, pp 1–6
4. Cozzolino A, Flammini F, Galli V, Lamberti M, Poggi G, Pragliola C (2012) Evaluating the effects of mjpeg compression on motion tracking in metro railway surveillance. In: Advanced concepts for intelligent vision systems. Springer, pp 142–154
5. De Jong T, Linn MC, Zacharia ZC (2013) Physical and virtual laboratories in science and engineering education. Science 340(6130):305–308
6. Deshpande H, Bawa M, Garcia-Molina H (2001) Streaming live media over a peer-to-peer network. Tech. rep
7. Fielding R, Gettys J, Mogul J, Frystyk H, Masinter L, Leach P, Berners-Lee T (2006) Hypertext transfer protocol–HTTP/1.1, 1999. RFC2616
8. Garcia-Zubia J, López-de-Ipiña D, Orduña P (2008) Mobile devices and remote labs in engineering education. In: 2008 eighth IEEE international conference on advanced learning technologies. IEEE, pp 620–622
9. García-Zubia J, Orduña P, López-de-Ipiña D, Alves GR (2009) Addressing software impact in the design of remote laboratories. IEEE Trans Ind Electron 56(12):4757–4767
10. Golparvar-Fard M, Bohn J, Teizer J, Savarese S, Peña-Mora F (2011) Evaluation of image-based modeling and laser scanning accuracy for emerging automated performance monitoring techniques. Autom Constr 20(8):1143–1155
11. Grange A, Rivaz P, Hunt J (2016) Draft vp9 bitstream and decoding process specification. Tech. rep., available: http://www.webmproject.org/vp9/ (accessed: 2016-07-07)
12. Grois D, Marpe D, Mulayoff A, Itzhaky B, Hadar O (2013) Performance comparison of H.265/mpeg-hevc, vp9, and H.265/MPEG-AVC encoders. In: Picture coding symposium (PCS), 2013. IEEE, pp 394–397
13. Harward VJ, Del Alamo JA, Lerman SR, Bailey PH, Carpenter J, DeLong K, Felknor C, Hardison J, Harrison B, Jabbour I et al (2008) The ilab shared architecture: a web services infrastructure to build communities of internet accessible laboratories. Proc IEEE 96(6):931–950
14. Hashemian R, Riddley J (2007) Fpga E-lab, a technique to remote access a laboratory to design and test. In: Microelectronic systems education, 2007. IEEE International Conference on MSE'07. IEEE, pp 139–140

15. Hei X, Liang C, Liang J, Liu Y, Ross KW (2007) A measurement study of a large-scale P2P IPTV system. IEEE Trans Multimed 9(8):1672–1687
16. Html5 specification (2016) Tech. rep., W3, available: https://www.w3.org/TR/html5
17. ISO/IEC (2015) Iso/iec 11172:1993. coding of moving pictures and associated audio for digital storage media at up to about 1.5 mbit/s. Tech. rep., available: http://www.iso.org/iso/catalogue_detail?csnumber=19180 (Accessed: 2016-07-07)
18. ITU-T (2015) High efficiency video coding. Tech. rep., available: https://www.itu.int/rec/t-REC-h.265-201504-i/en (accessed: 2016-07-07)
19. ITU-T (2016) Advanced video coding for generic audiovisual services. Tech. rep., available: http://www.itu.int/rec/T-REC-H.264-201602-I/en (Accessed: 2016-07-07)
20. Jara CA, Candelas FA, Torres F (2008) Virtual and remote laboratory for robotics e-learning. Comput Aided Chem Eng 25:1193–1198
21. Kaspar M, Parsad NM, Silverstein JC (2010) Cowebviz: interactive collaborative sharing of 3D stereoscopic visualization among browsers with no added software. In: Proceedings of the 1st ACM international health informatics symposium. ACM, pp 809–816
22. Latency in live network video surveillance (2015) Tech. rep., axis communications, available: http://bit.ly/2izYVOb (accessed: 2016-07-07)
23. Kim K, Cho BY, Ro WW (2016) Server side, play buffer based quality control for adaptive media streaming. Multimed Tools Appl 1–19
24. Ma J, Nickerson JV (2006) Hands-on, simulated, and remote laboratories: a comparative literature review. ACM Comput Surv (CSUR) 38(3):7
25. Martinez G, Angulo I, Garcia-Zubia J (2016) Weblabmicroscope: A remote laboratory for experimenting with digital microscope. In: 2016 13th international conference on remote engineering and virtual instrumentation (REV). IEEE, pp 159–162
26. Nedic Z, Machotka J, Nafalski A (2003) Remote laboratories versus virtual and real laboratories. In: FIE 2003 33rd annual, vol 1. IEEE
27. Nielsen J (1994) Usability engineering. Elsevier
28. Nishantha D, Hayashida Y, Hayashi T (2004) Application level rate adaptive Motion-JPEG transmission for medical collaboration systems. In: Proceedings of 24th international conference on distributed computing systems workshops, 2004. IEEE, pp 64–69
29. Orduña P, Bailey PH, DeLong K, López-de-Ipiña D, García-zubia J (2014) Towards federated interoperable bridges for sharing educational remote laboratories. Comput Hum Behav 30:389–395
30. Quax P, Liesenborgs J, Barzan A, Croonen M, Lamotte W, Vankeirsbilck B, Dhoedt B, Kimpe T, Pattyn K, McLin M (2016) Remote rendering solutions using web technologies. Multimed Tools Appl 75(8):4383–4410
31. Rodriguez-Gil L, Orduña P, García-Zubia J, Angulo I, López-de-Ipiña D (2014) Graphic technologies for virtual, remote and hybrid laboratories: Weblab-fpga hybrid lab. In: 2014 11th international conference on remote engineering and virtual instrumentation (REV). IEEE, pp 163–166
32. Salkintzis A, Passas N (2005) Emerging wireless multimedia: services and technologies. Wiley
33. Schauer F, Lustig F, Ožvoldová M (2009) Ises-internet school experimental system for computer-based laboratories in physics. Innovations 109–118
34. Schauer F, Krbecek M, Beno P, Gerza M, Palka L, Spilaková P (2014) Remlabnet-open remote laboratory management system for e-experiments. In: 2014 11th international conference on remote engineering and virtual instrumentation (REV). IEEE, pp 268–273
35. Schulzrinne H, Rao A, Lanphier R (1998) Rtsp: real time streaming protocol. IETF RFC2326, april
36. Shea R, Liu J, Ngai ECH, Cui Y (2013) Cloud gaming: architecture and performance. IEEE Netw 27(4):16–21
37. Soares J, Lobo J (2011) A remote fpga laboratory for digital design students. In: 7th portuguese meeting on reconfigurable systems (REC 2011). pp 95–98
38. Sripanidkulchai K, Ganjam A, Maggs B, Zhang H (2004) The feasibility of supporting large-scale live streaming applications with dynamic application end-points. In: ACM SIGCOMM computer communication review, vol. 34. ACM, pp 107–120
39. Tan WL, Lam F, Lau WC (2008) An empirical study on the capacity and performance of 3g networks. IEEE Trans Mob Comput 7(6):737–750
40. Ueberheide M, Klose F, Varisetty T, Fidler M, Magnor M (2015) Web-based interactive free-viewpoint streaming: a framework for high quality interactive free viewpoint navigation. In: Proceedings of the 23rd ACM international conference on multimedia. ACM, pp 1031–1034
41. Van Lancker W, Van Deursen D, Mannens E, Van de Walle R (2012) Implementation strategies for efficient media fragment retrieval. Multimed Tools and Appl 57(2):243–267

42. Vargas H, Farias G, Sanchez J, Dormido S, Esquembre F (2013) Using augmented reality in remote laboratories. Int J Comput Commun Control 8(4):622–634
43. Wang B, Zhang X, Wang G, Zheng H, Zhao BY (2016) Anatomy of a personalized livestreaming system. In: Proceedings of the 2016 ACM on internet measurement conference. ACM, pp 485–498
44. Webgl specification (2014) Tech. rep., khronos webGL working group, available: https://www.khronos.org/registry/webgl/specs/1.0/
45. Wiegand T, Sullivan GJ, Bjontegaard G, Luthra A (2003) Overview of the h. 264/avc video coding standard. IEEE Trans Circuits Syst Video Technol 13(7):560–576
46. Yazidi A, Henao H, Capolino GA, Betin F, Filippetti F (2011) A web-based remote laboratory for monitoring and diagnosis of ac electrical machines. IEEE Trans Ind Electron 58(10):4950–4959
47. Youtube now defaults to HTML5 video (2016) https://youtubeeng.googleblog.com/2015/01/youtube-now-defaults-to-html5_27.html (accessed: 2016-07-07)
48. Zhang C, Liu J (2015) On crowdsourced interactive live streaming: a Twitch.TV-based measurement study. In: Proceedings of the 25th ACM workshop on network and operating systems support for digital audio and video. ACM, pp 55–60

Luis Rodriguez-Gil is a PhD student at DeustoTech Internet group. He finished his studies of a double degree in Computer Eng. and Industrial Org. Eng. in 2013, and he completed a MSc in Information Security in 2014. Since 2009, he has been involved in the WebLab-Deusto Research Group, collaborating in the development of the WebLab-Deusto RLMS. He has published several peer-reviewed publications and contributed to some Open Source projects.

Pablo Orduña is a full time researcher and project manager at the MORElab Research Group at DeustoTech Internet. He finished Computer Engineering in 2007 and his PhD in 2013 in the University of Deusto. During his PhD he was a visiting researcher twice for 6 weeks each, in the MIT CECI in 2011 and UNED DIEEC in 2012. Since 2004, he has also been involved in the WebLabDeusto Research Group, leading the design and development of WebLab-Deusto.

Javier García-Zubia holds a PhD in Computer Sciences by the University of Deusto. He is a full professor in the Faculty of Engineering of the University of Deusto, Spain. His research interest is focused on remote laboratory design, implementation and evaluation. He is the leader of the WebLab-Deusto research group.

Diego López-De-Ipiña is an associate prof. and P.R. of MORElab group and director of DeustoTech Internet unit, and of the PhD program within the Faculty of Eng. of the University of Deusto. He received his PhD from the University of Cambridge in 2002. Responsible for several modules in the BSc and MSc in Comp. Eng. degrees, he is interested in pervasive computing, IoT, semantic service middleware, open linked data and social data mining. He is taking and has taken part in several big consortium-based research european (IES CITIES, MUGGES, SONOPA, CBDP, GO-LAB, LifeWear) and Spanish projects, and has more than 70 publications in relevant int. conf. and journals, including more than 25 JCR-indexed articles.

APPENDIX

B

Paper II

An open and scalable web-based interactive live-streaming architecture: The WILSP platform

This paper was published in the journal '*IEEE Access*'. Its publication date is 1 June 2017. Its DOI is: 10.1109/ACCESS.2017.2710328. It is published as Open Access and can be accessed online through the IEEE Xplore Digital Library at: http://ieeexplore.ieee.org/document/7937791/. The published version is appended here.

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 4, 2017, accepted May 25, 2017, date of publication June 1, 2017, date of current version June 28, 2017.

Digital Object Identifier 10.1109/ACCESS.2017.2710328

An Open and Scalable Web-Based Interactive Live-Streaming architecture: The WILSP Platform

LUIS RODRÍGUEZ-GIL[1,2], JAVIER GARCÍA-ZUBIA[2], (Senior Member, IEEE), PABLO ORDUÑA[1,2], (Member, IEEE), AND DIEGO LÓPEZ-DE-IPIÑA[1,2]

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

Corresponding author: Luis Rodriguez-Gil (luis.rodriguezgil@deusto.es)

This work was supported by the Department of Education, Language Policy and Culture of the Basque Government through a Predoctoral Scholarship to Luis Rodriguez-Gil.

ABSTRACT Interactive live-streaming applications and platforms face particular challenges: the actions of the viewer's affect the content of the stream. A minimal capture-render delay is critical. This is the case of applications, such as remote laboratories, which allow students to view specific hardware through a webcam, and interact with it remotely in close to real time. It is also the case of other applications, such as videoconferencing or remote rendering. In the latest years, several commercial live-streaming platforms have appeared. However, the most of them have two significant limitations. First, because they are oriented toward standard live-streaming, their capture-render delay tends to be too high for interactive live-streaming. Second, their architectures and sources are closed. That makes them unsuitable for many research and practical purposes, especially when customization is required. This paper presents the requirements for an interactive live-streaming platform, focusing on remote lab needs as a case study. Then, it proposes an architecture to satisfy those requirements that relies on Redis to achieve high scalability. The architecture is based on open technologies, and has been implemented and published as open source. From a client-side perspective, it is web-based and mobile-friendly. It is intended to be useful for both research and practical purposes. Finally, this paper experimentally evaluates the proposed architecture through its contributed implementation, analyzing its performance and scalability.

INDEX TERMS Webcam, live streaming, live streaming platform, remote laboratories, online learning tools, open.

I. INTRODUCTION

Throughout the last few years many live-streaming platforms have emerged, such as YouTube Live,[1] TwitchTV,[2] Instagram Livestream,[3] and Facebook Live.[4] These platforms tend to be backed by large social media companies and be proprietary. They are designed for scalability and are able to provide live-streaming to a large number of users. However, for some purposes, they have significant limitations. Although they are effective for live-streaming non-interactive content, such as live sports, they tend to be unsuitable for interactive live-streaming. In interactive live-streaming, users react to the stream and affect it in close to real-time. Thus, a minimal capture-render delay is critical. Standard live-streaming platforms, such as the aforementioned ones, tend to have a relatively high delay of at least several seconds. This is by design. It is, among other reasons, because they rely on transcoding, buffering, and heavy interframe compression techniques to maximize scalability and minimize networking issues. The work in [1], for example, outlines the TwitchTV architecture, providing further detail in these aspects. In that case, the broadcast delay of the platform is measured to vary from 12 to 21 seconds. Other limitation is that these major platforms are proprietary, and their architectures and sources are closed. This makes it difficult to rely on them for learning and research purposes. They are also impractical for applications that would need to customize them, or integrate them as middleware.

Interactive live-streaming has particular requirements, limitations, and expectations. Figure 1 characterizes interactive live-streaming among other types of streaming: standard live-streaming and Video on Demand (VoD). There are

[1]https://www.youtube.com/live
[2]https://twitch.tv
[3]https://instagram.com/livestream
[4]https://live.fb.com

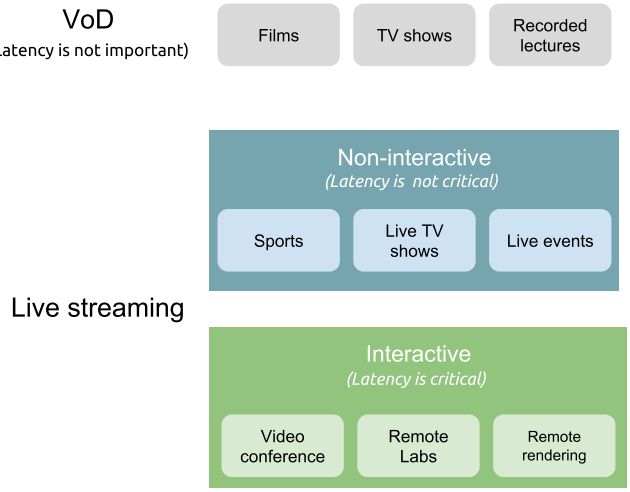

FIGURE 1. Characterization of the different types of streaming and some of its applications. Reproduced from [4].

several aspects that differ, but the most remarkable one is the capture-render delay. The videos in a VoD application are created far in advance. VoD platforms, such as Youtube or Netflix, can use the heaviest compression techniques, and convert the video in advance for different configurations and network conditions. Thus, a high-bandwidth client in a desktop will receive a high-quality and high-bandwidth stream; while a low-bandwidth client in a mobile device will receive a lower-quality but lower-bandwidth stream. They can rely on adaptive streaming to adapt to varying conditions, and they can use buffering to provide high quality in a relatively unstable network, avoiding network jitter [2]. Standard live-streaming platforms have more limitations at this respect. They no longer have nearly-unlimited time to convert the input streams to different formats, nor can they use as large a buffer. Most commercial platforms still allow a significant capture-render delay. This allows them to partially leverage the previously mentioned techniques. However, for interactive live-streaming applications, such as remote labs, collaborative tools, video conferencing applications or remote rendering applications, a low capture-render delay is critical. According to Human Computer Interaction (HCI) research, a second in delay is high. Beyond 0.1 seconds the user can notice a system does not react instantaneously. Beyond a second the user's flow of thought is interrupted [3].

In this context, this work proposes a novel architecture for interactive live-streaming. A platform, named WILSP,[5] has been designed and implemented according to that architecture. It is designed to overcome the aforementioned limitations in existing platforms. Firstly, it is designed for interactive live-streaming, ensuring a low capture-render delay. Secondly, it is open source and relies on open technologies. It can be used for research and practical purposes, customized freely and integrated as middleware. From a technical perspective, it is designed to be a distributed, scalable platform by relying on Redis.[6] It is extensible and supports different video formats and techniques, such as H.264 [5] and image-refreshing, which previous research shows as effective for this purpose [4]. Also, from a client-side perspective, it is fully web-based. This is remarkable because, traditionally, certain features such as multimedia have had more limited support on the Web [6], [7]. This trend has changed and applications no longer need to depend on non-standard plugins [8], such as Adobe Flash[7] or Java Applets.[8] Now they can rely on HTML5 [9] and other related Web standards such as WebGL [10]. Today, major platforms such as Youtube or Netflix[9] rely on HTML5 [11].

The proposed architecture is designed to be generalistic and integrable as middleware into different applications, but its initial and main use-case are remote labs. Remote labs are software and hardware tools that enable remote users to access real, remote equipment through a website [12]–[14]. They can view and interact with that equipment through a webcam. To evaluate the proposed architecture, this work analyzes whether certain requirements are met, focusing particularly on this use-case. Furthermore, a study to measure the performance and scalability of the platform is conducted.

The paper is organized as follows: Section II describes in more detail the purpose and contributions of this research, and the relevant state of the art on interactive live-streaming and remote labs. Section III analyzes the requirements for the proposed platform. Section IV presents an overview of the proposed architecture. Section V describes each layer in more detail. Section VI describes the methodology of the performance and scalability study. Section VII enumerates and explains the conducted experiments. Section VIII describes the results. Section IX discusses those results and potential applications. Section X summarizes the conclusions and proposes future lines of work.

II. MOTIVATION

A. INTERACTIVE LIVE-STREAMING

In an interactive live-streaming system viewers are expected to interact with the stream: they are not simply passive spectators [1], [4], [15]. This is not necessarily the case in a standard live-streaming system. In any live-streaming system there is a capture-render delay: an unavoidable delay between the moment a frame is captured by the source camera, and the moment it is rendered on the viewer's screen. However, the maximum delay that interactive live-streaming systems can allow while still providing an acceptable Quality of Experience for the user is much smaller than for standard live-streaming systems. Although it is not always noticeable for users, most popular standard live-streaming systems nowadays have a significant capture-render delay [16], [17].

[5]The WILSP platform has been released as Open Source and is available at: https://github.com/zstars/wilsp

[6]https://redis.io/
[7]http://www.adobe.com/products/flashplayer.html
[8]http://java.com
[9]https://www.netflix.com

For example, the TwitchTV[10] platform tends to have a higher than 10 seconds delay [1], and the YouTube[11] live-streaming platform seems to commonly have a 20-30 seconds delay in its low-latency configuration.[12]

This delay is purposefully built into the design of their architectures [18]. It allows them to leverage techniques such as buffering, heavy-compression codecs, and video segmentation to maximize scalability and performance. Through these, they can withstand higher network jitter and provide better quality at a lower bandwidth. The high capture-render delay that results is generally not a problem, due to the non-interactive nature of their live-streaming applications. Live sports streaming or live-shows are some common uses of these platforms. The content is indeed being produced while it is broadcast, but users interact very little with the stream, so they can withstand a high capture-render delay. For example, if users are viewing a football match with a 30 seconds delay, their user-experience is unlikely to be affected significantly.

As described above, however, interactive live-streaming applications allow only for a much smaller delay. Some applications of this kind are videoconferencing ones (e.g., Skype,[13] Google Hangouts),[14] surveillance systems (those which require real-time monitoring), remote rendering systems (e.g., [19], [20]), or remote laboratories. Remote labs are in fact the main use-case of the platform that is proposed in this work, and a use-case for this study. They will be described in further detail in later sections.

Currently, the better-known live-streaming platforms are thus not suitable for interactive live-streaming. Most of them are purpose-specific, proprietary, and closed-source, which hinders their use for learning or research purposes. Nonetheless, some proprietary engines which are designed to be used as middleware do exist, such as Wowza.[15] Some live-streaming open source projects can also be found. A remarkable one is the *nginx-rtmp-module*,[16] a module for the Nginx web server. It can live-stream through protocols such as RTMP, HLS or MPEG-DASH.

B. REMOTE LABORATORIES

Remote laboratories allow users to access remote equipment through the Internet [8], [12], [21], [22]. Nowadays, remote laboratories are often educational. Students can access, from anywhere, remote equipment that is located in institutions across the globe. Research has shown that when they are properly designed and implemented they can be as educationally effective as a standard hands-on lab [14], [23], [24].

[10] http://www.twitch.tv

[11] http://www.youtube.com

[12] YouTube provides no official figures or guarantees, but observations and informal tests can be found, such as those at http://blog.ptzoptics.com/youtube-live/low-latency-streaming/ or at the Google product forum (https://productforums.google.com/forum/).

[13] http://www.skype.com

[14] https://hangouts.google.com

[15] https://www.wowza.com

[16] https://github.com/arut/nginx-rtmp-module

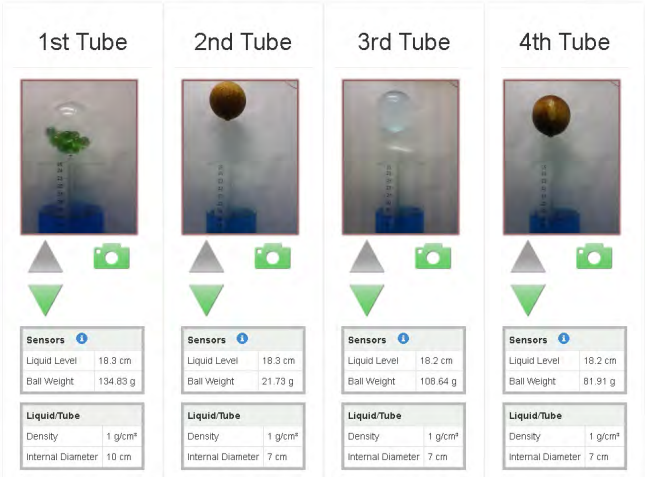

FIGURE 2. Archimedes remote lab. Users experiment with the principle of Archimedes by raising and lowering objects into different liquids and obtaining sensor readings. From http://weblab.deusto.es.

They have many advantages. By relying on remote laboratory technology, institutions that are geographically separated can share expensive equipment. This makes more equipment available for their students, reduces their costs and reduces underusing of equipment [25], [26].

Remote laboratories are formed by multiple hardware and software components [8]. Often they are developed by universities and other institutions as part of research projects. Often they are built on top of Remote Laboratory Management Systems, such as WebLab-Deusto [8] or MIT iLabs [21]. Those systems provide common features, such as authentication, user management, learning analytics, or laboratory federation.

Most remote laboratories feature one or more live-streams of the equipment. In a hands-on laboratory, students see the equipment through their own eyes and interact with it as needed. In a remote laboratory, the interaction needs to be different. Students view the equipment through one or more webcams and interact with it through virtualized controls [27]–[30]. For an interactive remote laboratory an interactive live-stream is required. When lab users conduct an action, such as pressing a button, they expect to see the result immediately.

Figure 2 shows the GUI of the Archimedes remote lab. In it, students can raise and lower objects into different liquids. Sensors provide them with the object weight and liquid height. They can thus verify how the Principle of Archimedes works in reality. As the figure shows, this remote lab includes different simultaneous live-streams. Figure 3 shows a different remote lab. In it, students can create a program using a visual programming language and run it on a real, remote, Arduino-based robot. They have access to several input peripherals (e.g., sensors, buttons) and output ones (LEDs, a serial terminal). In this case, a webcam provides a live-stream as well, and it is also key to the user experience. Users monitor the robot through it and check

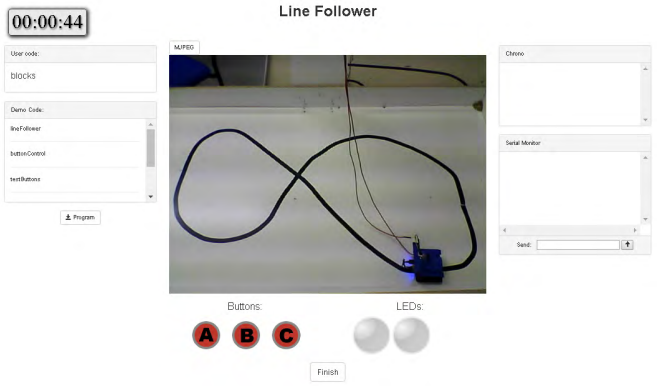

FIGURE 3. Arduino robotics remote lab. Users program their own Arduino robot remotely and are able to interact with it. From http://labsland.com.

whether their program is making the robot behave as they expect. They can interact with the robot in near real-time through the buttons and serial terminal, so the capture-render delay must be low. In this case, because the robot moves relatively fast, the video, ideally, needs to provide a relatively high FPS.

C. CHALLENGE AND CONTRIBUTIONS

As described above, an interactive live-stream is a key feature of most remote laboratories. The stream is the window through which remote students see and interact with the equipment, and is thus particularly important for their user experience. However, traditionally, in remote laboratory research and practice, little attention has been paid to this aspect [4]. The purpose of this work is to analyze the requirements for an interactive live-streaming architecture that is general-purpose but particularly suited for remote laboratories, to design and implement it, and to evaluate it. The platform architecture is designed to be general-purpose but optimal for interactive remote laboratories. It is web-based, based on open technologies, and open source. This is key because the goal is for the platform to be useful for remote laboratory researchers and developers. It is also designed to be distributed and scalable, so that it can manage a large number of input cameras, and so that these streams can be served to a high number of users.

The contributions of this work are the following:

- An analysis of the requirements for an interactive live-streaming platform that is optimized for remote laboratories.
- Architecture for an interactive live-streaming platform that is distributed, based on open technologies, and highly scalable.
- Implementation for the proposed architecture, which is made available as Open Source.
- Design and implementation of two supporting tools for performance evaluation (an IP webcam simulation and a *requester* load-testing script).
- Novel use of the Redis in-memory store engine as a middleware for interactive live-streaming.
- Experimental performance analysis and evaluation of the proposed architecture.
- Conclusions, based on the conducted experiments, on the performance and suitability of such an architecture for remote laboratory research and development.

III. PLATFORM GOALS AND REQUIREMENTS

The main target of the interactive live-streaming platform that is proposed in this work is to satisfy the requirements of remote laboratories, for both research and production contexts. Even though, it is also designed to be general-purpose, and should thus be suitable for additional applications that have similar interactive live-streaming needs.

The main general goals of the platform, which are in line with the needs described in the previous sections, are the following:

- **Universality:** The streams, from a technical perspective, should be available to as many end-user configurations as possible, independently of their platform or device type.
- **Efficiency and scalability:** The architecture should scale horizontally for a large number of camera sources and viewers.
- **Openness:** The architecture should rely on open technologies and be open itself, so that researchers and developers can learn from it, build on it, or use it for research purposes.

The main requirements are the following:

- **Interactive live-streaming:** The platform should, as previously described, be capable of interactive live-streaming. The live-streams it provides should have a small capture-render delay.
- **Supporting multiple input sources:** Being able to handle many video sources (IP cameras) with different stream formats.
- **Supporting multiple output schemes:** Being able to provide different types of stream, from a client-side perspective, depending on the particular needs.

In the following subsections, these goals and requirements will be described in further detail.

A. UNIVERSALITY

The meaning of universality varies across different contexts. In this work, we refer to the goal that, from the client-side perspective, the streams that the platform provides should be as broadly compatible as possible. Universality is particularly important for remote laboratories and similar applications [12], [31].

To promote universality the platform is intended to be fully web-based. Researchers and developers will thus be able to integrate the streams into any web-based application, and users will be able to view them through any web browser. Being web-based is the most significant step towards universality, but there are additional concerns that will be considered. The first of them is that non-standard browser plug-ins should be avoided. Traditionally, some web applications have

relied on technologies such as Java Applets [32] or Adobe Flash [33] to provide certain advanced features such as graphics or non-HTTP networking. For example, [34] describes a remote laboratory which relies on the YawCam[17] Java Applet for streaming its webcam. Using them was once necessary, and the networking access they provide makes protocols such as RTSP [35] available. However, they hinder universality because they are rarely supported in mobile phones and nowadays some popular browsers are even dropping support and discouraging their use on desktops. They can also pose a security risk [36]. For modern applications, generally, standard non-intrusive technologies are preferred. Remote laboratories are often deployed behind the institutional networks of universities. Their IT teams and policies often avoid offering intrusive technologies to students, because their institution could, in fact, be liable were them to be exposed to security issues [8].

Currently, certain technologies that are potentially useful for live streaming (e.g., MPEG-DASH, Media Source Extensions, WebRTC) are being standardized. To promote universality, the interactive live-streaming schemes described in this work rely only on approved standards, and technologies that are widely supported by all browsers. Nonetheless, the proposed architecture is designed to be extensible, and these might be considered in the future.

Finally, it is important to remark that a wide range of different user devices should be supported. This is critical for education: nowadays students from institutions across the world often rely on mobile and tablet devices [37]–[40].

B. EFFICIENCY AND SCALABILITY

In order for the platform to be useful for researchers and developers, it needs to be reasonably efficient and scalable. Institutions that host remote laboratories often host several different ones, each with potentially several live-streams to manage. For example, Figure 2 shows a single laboratory that has had up to 7 simultaneous ones. Other laboratories only require a single stream, such as the robot shown in Figure 3. However, even in that case, the institution often hosts several instances of the same laboratory. For example, in the case of the robot, users upload their own program into it, watch it execute, and interact with it. Only one user can thus have control over it at a given time. To support several simultaneous users, several instances of the laboratory exist. It is also noteworthy that the number of supported viewers per stream should also be scalable, especially for these cases where the laboratory is publicly viewable or collaborative. Currently, Remote Laboratory Management Systems are used to host a large number of different remote laboratories and instances. It is therefore convenient for a single platform to be able to handle all interactive streams.

[17]http://www.yawcam.com

C. OPENNESS

Most popular live-streaming middlewares and platforms are closed-source and proprietary. Nonetheless, they attract significant research attention. Those researchers deduce their architectures from practical experiments, reverse-engineering, and other sources. For example, works can be found on TwitchTV [1], YouTube Live [41], YouNow [42], Meerkat [43] or Periscope [43], [44].

Unfortunately, the closed-source nature of these platforms makes them unsuitable for certain research and practical purposes. As far as we know, for example, no remote laboratory has tried to use any of the above platforms, in a research or production context. Most of the proprietary platforms are not intended to be integrated into other systems, and, as explained in previous sections, do not generally satisfy the low-latency constraints that interactive live-streaming entails.

The goal of the proposed platform is thus to be open, customizable and flexible. This is in line with works such as [45], where it is also stated that the customizability and flexibility that being open-source ensures is particularly critical for academia. Researchers and developers are meant to be able to freely use and modify the platform, adapt it to their needs, and integrate it into their own systems (of which remote labs are a good example).

In line with this, the platform will rely on open technologies, such as Redis, Flask[18] and FFmpeg.[19]

D. INTERACTIVE LIVE-STREAMING

Although it has also been described in earlier sections, it is important to remark that the main requirement of the platform is being able to serve *interactive* live-streams. That is, they should have a low-enough latency, which guarantees that even for interactive applications, the user's Quality of Experience is satisfactory. The actual value for that satisfactory threshold will inevitably vary among users and applications. In cloud-based games, for example, the maximum delay is very low: for each 100 ms of latency, there is a 25% decrease in performance [46]. The threshold for videoconferencing applications is closer to the 300-400ms range [47]. For other contexts where specific research is not available, such as remote labs themselves, general guidelines can be considered. According to Nielsen [3], if a system's response has a delay higher than a second, the user's flow of thought is interrupted, and a 0.1 seconds delay is enough for users to notice.

The proposed architecture will take this into account, and avoid those techniques which tend to increase latency, such as heavy compression and long buffering times.

E. SUPPORTING MULTIPLE INPUT SOURCES

Multiple input sources should be supported, as mentioned in Subsection III-B. The main input sources will be IP webcams.

[18]http://flask.pocoo.org
[19]https://ffmpeg.org

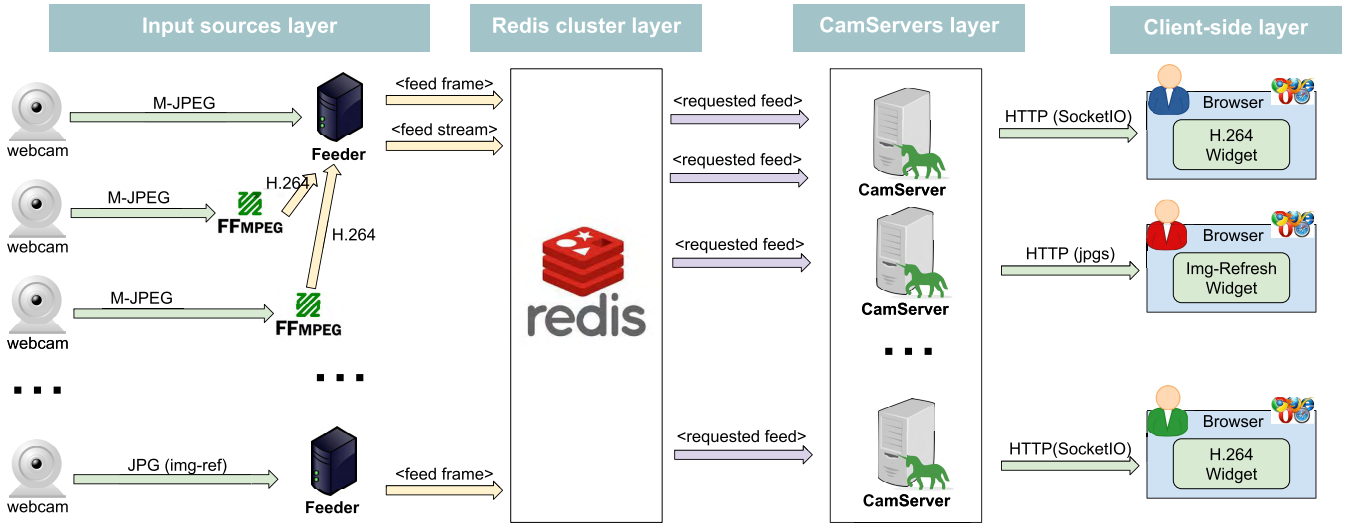

FIGURE 4. Architectural overview of the proposed platform and its components.

Those webcams support different output formats. Common ones are snapshots (which can be turned into a video by simply requesting them periodically fast enough) and M-JPEG, though some modern cameras also support formats such as H.264, and some models even protocols such as RTSP. The platform should be extensible, so that several formats can be supported and new ones added in the future.

F. SUPPORTING MULTIPLE OUTPUT SCHEMES

The output schemes are those that will be used to send the stream to the end user's browser, and actually render it. Traditionally, some applications, including remote laboratories, have dedicated little attention to this aspect [4]. However, it plays a very significant part in promoting universality (being accessible across platforms and devices), a high enough reliability, and, in general, a good enough Quality of Experience.

Previous research suggests that there is currently no single best scheme for all purposes [4]. Several components intervene in a scheme, including the communications protocol, the container format, the video codec and the rendering technology. Many valid combinations may exist. Due to this, the architecture is designed to support various schemes and to be easily extensible.

IV. ARCHITECTURE OVERVIEW

As described in Section III, the proposed architecture is designed to be highly scalable, both in respect to the number of input sources and to the number of end-users. In order to achieve close-to-horizontal scalability, the architecture has several layers. A standard Redis server is at the core of the architecture. Redis is a popular Open Source in-memory data store and message broker, which has successfully been used in many well-known projects to provide scalability.

Figure 4 shows an overview of the proposed architecture and its main components. The architecture is divided into several layers which are highly decoupled, in order to promote scalability and to better separate the concerns of each layer. Particular deployments may have any number of webcams, Feeders, CamServers and clients. Only a fixed number are represented in the picture.

It can be observed that the data flows from left to right. Firstly, the IP webcams, to the left, capture the initial stream. The architecture is designed to scale to an arbitrary number of source webcams and to support different input formats. M-JPEG and continuous snapshots are the most common ones, and the ones the contributed implementation provides. Note that the ones in the figure have been chosen arbitrarily: all Feeders support all formats. The one to be used will depend on the particular IP camera. Secondly, as the figure shows, the streams are directed to Feeder servers. In some cases, the streams are transcoded through FFmpeg, depending on whether the particular input and output format combinations require transcoding or not. The Feeder servers are also designed to scale horizontally, and they forward the streams into the Redis cluster layer.

The Redis layer contains a standard Redis cluster (or a single Redis instance, depending on the scalability needs), and acts as a central decoupling element. Redis is a well-known Open Source middleware designed for in-memory storage, message-brokering, and scalability. Its goal here is to receive the streams, store them in-memory very briefly and efficiently, and distribute them as requested by the CamServers.

The CamServers, which are also designed to be horizontally scalable, retrieve, from Redis, the streams that users are currently requesting. They support different output schemes. When a client requests a stream, they serve it in the expected one. The contributed implementation and the experiments conducted in this work focus on two: image-refreshing

and H.264. Previous research suggests that they are effective and have different advantages [4].

The client-side layer is fully web-based and is formed by different client-side technologies (mostly JavaScript-based) that are served by the CamServers themselves. It provides libraries and widgets that are able to request the streams in different formats and render them.

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. See again Figure 4 for a general overview of these layers.

A. INPUT SOURCES LAYER

The purpose of this layer is to encapsulate access to the input sources (webcams), forwarding the stream into the Redis cluster, after having transcoded it into an appropriate format if required.

This scheme has several advantages. Some applications and remote laboratories, for simplicity, access the stream provided by IP cameras directly. Experience shows, however, that this can lead to different significant issues which are not necessarily simple to foresee. Some of them are mentioned in the previous work [4]. From a technical perspective, the software and hardware of these cameras are often limited. While normally they will be able to provide a reasonable Quality of Experience for a single user (or a few), under higher loads their performance tends to be unpredictable. Also, they are heterogeneous. They support different stream formats and specifications, which are not always documented, and which may need to be handled in different ways.

Most IP cameras, for instance, provide an M-JPEG stream. This is a common format because it is simple to implement, requires little encoding, and adds little delay. However, the FPS rate tends to be fixed, either specified in the configuration of the IP webcam, or fixed in their firmware. When users are unable to process the images fast enough, due for example to network jittering or bandwidth constraints, a capture render delay builds up that can add up to many seconds. As a result, applications that rely on directly rendering the stream often become unusable under certain conditions, with no trivial way to notice programmatically or even warn the user. As mentioned, there are also differences between webcams. For example, some provide different qualities and video resolutions, others provide different formats such as an H.264 stream, others use specific non-standardized HTTP headers to transmit timestamps.

The Feeder servers abstract out these idiosyncrasies, providing well-known and reliable streams into the Redis cluster. The contributed implementation, for example, relies mostly on the different M-JPEG streams that the IP webcams provide. No matter how many simultaneous users the interactive live-streaming platform has, a single Feeder accesses a single IP webcam at any time. The IP webcam can thus safely stream reliably a high-quality stream into the Feeder, at a high FPS (generally at 30 FPS). Also each webcam is intended to be located in the same LAN as its Feeder, so there is very little risk of a delay building up, and the chance of webcam-side issues is minimized.

```
ffmpeg -r 30 -f mjpeg -i <src> -flags +low_delay \
  -probesize 32 -c:v libx264 -tune zerolatency \
  -preset:v ultrafast -r 30 -f h264 -s 480x640 \
  -b:v 1500k -g 100 -pix_fmt yuv420p pipe:1
```

FIGURE 5. Example of FFmpeg configuration for transcoding M-JPEG into H.264 with minimal latency.

When transcoding is necessary, the scheme works similarly. A single FFmpeg instance accesses the webcam at any one time. For example, one of the main output schemes that the contributed platform provides is based on H.264 [5]. This is a relatively strong, interframe compression codec. It is not possible to rely on webcams supporting it, and even those that do often do not support it at an interactive live-streaming level. To provide a very low latency it is necessary to configure the H.264 codec appropriately.[20] The FFmpeg instance takes a stream as input from the webcam, in a low-latency format that the webcam supports, such as M-JPEG. Then it transcodes it into the target format (e.g., H.264), sends it to the Feeder, and the Feeder forwards it to the Redis server through its publish-subscribe scheme. See Figure 5 for an example of the FFmpeg configuration that can be used to transcode M-JPEG into H.264 with minimal latency.

The Feeder server, in the contributed implementation, has been created in Python and relies on the Gevent[21] coroutine-based networking engine. It is designed to scale horizontally, supporting any number of instances. Each instance can handle many cameras. The source code is available in the repository.

B. REDIS CLUSTER LAYER

The Redis[22] [48] cluster is a central element of the architecture. It decouples the Feeders (which abstract out access to the input sources and transcoding) from the CamServers (which serve the appropriate stream to the end-users, in whatever format and framerate is required).

Redis is an Open Source in-memory data store. It is used as an in-memory cache and message broker. The proposed architecture makes use of a standard, unmodified Redis cluster (which can, in fact, be a single Redis instance up until a significantly high number of users and cameras, as the experiments in Section VII will show). Redis databases can be replicated through a master-slave model and current versions support a form of sharding [49].

Redis is not specifically targeted towards video streaming. However, it has certain characteristics that make it appropriate for this purpose. Redis is well-known to provide high

[20] Depending on the codec, the processing power, the buffering size, the H.264 mode, and many other configuration details, the capture-render delay, quality and bandwidth usage will vary widely.

[21] http://www.gevent.org/

[22] https://redis.io/

performance and to have been used to provide scalability for many applications. It is well-tested, well-maintained and there is a strong community of developers behind it.[23] Research works suggest that its performance is excellent. For example, [50] compares several NoSQL systems including MongoDB, ElasticSearch and OrientDB, and Redis shows the best performance.

The architecture relies on two different Redis subsystems. In case no transcoding is necessary, which is typically the case when the output format is based on image-refreshing or M-JPEG, it relies on Redis' standard key-value storage features to store individual image frames. The Feeder servers place frames into Redis in a particular Redis key for each active stream. Meanwhile, the CamServers read them as needed. Through this, the Feeders and the CamServers are fully decoupled, and can work at different framerates without issues. Normally the Feeder will work at the maximum framerate for the camera (e.g., 30 FPS). The CamServer can retrieve snapshots from Redis at a slower FPS (e.g., 25 FPS), or even at an adaptive FPS. Some frames will be skipped but no delay is built up. Simplistic as it might seem, it is a very suitable choice for many applications [4], especially since many remote laboratories today rely on those rendering schemes.

When transcoding is necessary, such as when a CamServer requests an H.264 stream, the architecture relies on Redis' message brokering features (Redis channels). FFmpeg transcodes into the Feeder servers, and the Feeder servers publish the output directly into a specific Redis channel for the stream configuration.

As previously described, Redis is not generally geared towards multimedia. However, it fits the proposed platform needs particularly well. In the case of the key-value storage system (for the image-refreshing scheme), the proposed architecture relies on short-lived, non-permanent, readily-replaced frames. These are key features of the Redis store system. Similarly, in the case of the channel-based system (for the H.264 scheme), Redis provides memory-only, short-lived, efficient messages. This is also what the platform requires. Therefore, an interesting contribution of this work is also this novel use of the Redis engine.

C. CamServers LAYER

CamServers obtain the stream data from Redis and serve them to each individual user. They are based on Python's Flask microframework and the Gevent coroutine engine. In order to provide horizontal scalability, an arbitrary number of CamServer instances can be deployed, relying on the WSGI and Gunicorn[24] technologies. Each CamServer instance can serve a number of clients. The architecture supports several types of schemes for this last streaming layer. The main ones are a simple image-refreshing scheme and an H.264 streaming scheme. Previous research suggests that those schemes are effective [4].

When a client is requesting image-refreshing the CamServers simply need to serve discrete frames. This scheme adapts automatically to the client's FPS and bandwidth requirements. The CamServer retrieves each frame directly from Redis, from the appropriate Redis key for the stream. That key's value, at any given moment, will be the latest frame pushed into Redis by the Feeder.When a client is requesting H.264 streaming the CamServer registers itself to listen to the appropriate Redis channel. Then, it forwards that channel's stream data to the client through SocketIO.[25] The architecture can be extended easily to support additional formats based on these schemes, and some additional ones are in fact supported by the contributed platform implementation.

The CamServer component, in the contributed implementation, has also been implemented in Python using Gevent. Each individual CamServer component instance can thus handle several clients, but, additionally, it relies on Gunicorn and WSGI to provide horizontal scalability. Through them, an arbitrary number of CamServer processes on an arbitrary number of hardware servers can be started. The source code is available in the repository.

D. CLIENT-SIDE LAYER

Evaluating potential interactive live-streaming schemes from a client-side perspective, and determining the ones with the most potential for remote laboratories, was the main focus of previous research [4]. As a result, the schemes that are mainly considered in this work are a simple image-refreshing scheme and a more complex H.264 based scheme.

Both the image-refreshing and the H.264 schemes rely only on standard HTML5 and JavaScript. Dependence on non-standard technologies such as Adobe Flash and Java Applets is purposefully avoided. The implementation includes all schemes as widgets, so that they can be easily integrated into other systems. The image-refreshing scheme is the simplest. The widget simply retrieves frames at a high-rate from the CamServer using standard HTTP requests. Though this is not particularly efficient, in relatively modern devices 30 FPS or even higher framerates can be achieved without issues.

The H.264 scheme relies on a customized version of the Broadway[26] H.264 decoder. The core of the decoder is programmed in C but compiled into highly-optimized JavaScript through the EmScripten[27] technology, and can render even more efficiently in WebGL-compatible devices. More details about this scheme and its client-side performance can be found in [4]. The CamServer forwards the stream data to the browser and the decoder through the SocketIO library. SocketIO relies on WebSockets in compatible environments, and falls back to less efficient transports if needed.

[23] The official repository can be found at: https://github.com/antirez/redis. At the moment of writing this, over 8,725 developers have forked the code, and over 23,145 have starred (are following) it.

[24] http://gunicorn.org/

[25] https://socket.io/

[26] https://github.com/mbebenita/Broadway

[27] https://github.com/kripken/emscripten

It is noteworthy that the source code of these widgets is also provided in the repository, in both the JavaScript and the TypeScript languages.

VI. METHODOLOGY

The architecture has been implemented as an Open Source platform. In order to evaluate the architecture, various experiments are conducted on the contributed implementation. The performance is experimentally analyzed through several benchmarks, and the results compared against the previously proposed goals and requirements.

The architecture is formed by several components, which are designed to scale horizontally. A Redis cluster is at the center. The number of Feeders can be increased as the number of camera sources increases, the number of CamServers can be increased as the number of end-users increases. In order to experimentally analyze the performance of the system, the CamServer and the Feeder components are analyzed through separate experiments. That way it is possible to gain insight on what kind of performance we can expect as the number of cameras increases, and what kind of performance we can expect as the number of end-users increases. Feeders and CamServers are decoupled through the Redis cluster. Thus, their performance can mostly be expected to be independent.

It is noteworthy that analyzing the behavior of the platform when a resource limit (e.g., CPU, memory, bandwidth) is reached is out of the scope of this study. Therefore, the ranges (from 1 to 50 cameras, and from 1 to 50 users) have been chosen to avoid this eventuality. It is expected that whenever such a limit is reached the performance of the streams can no longer be guaranteed to satisfy the requirements appropriately. In that case, deployers would be advised to add a new computer to the cluster hosting additional instances of the constrained component. The actual effect of an unsolved constraint would depend on the resource whose limit was reached, on the component that suffers it, and on the stream format. Additional detail on this, especially from a client-side perspective, is described in [4]. The most likely results would be an always-increasing delay, or, in the best-case, a reduced FPS.

The architecture supports several different video schemes. The ones which are most convenient according to previous research, and the ones which are the focus of this work, are image-refreshing and H.264. Because significantly different performance for each of those can be expected, all the experiments are also conducted for those two schemes. Therefore, four different experiments are conducted:

- **Experiment 1**: Feeder component performance in snapshots mode (image-refreshing).
- **Experiment 2**: Feeder component performance in stream mode (H.264).
- **Experiment 3**: CamServer component performance in snapshots mode (image-refreshing).
- **Experiment 4**: CamServer component performance in stream mode (H.264).

A. MATERIALS

All the experiments are conducted on a Gigabit LAN. Thus, the latency of this network is minimal and the bandwidth is much higher than the maximum amount required for the highest load in the experiments. This is critical so that the network does not add a significant amount of latency and so that the bandwidth does not become a bottleneck, affecting the measurements. Three different physical servers are used:

- **Experiment Server**: Intel i7-6700 CPU (3.40GHz, 4 cores, 8 threads), 16 GB RAM.
- **Support Server**: Intel Xeon E5-2630 v3 CPU (2.40GHz, 8 cores, 16 threads), 48 GB RAM.
- **GUI Server**: Intel Core 2 Duo CPU E8400 (3.00GHz, 2 cores), 4 GB RAM.

The Experiment Server, for each experiment, holds only the component being tested. Measurements are taken on it. The Support Server holds the components that are not being tested and also hosts the load simulation components. It is significantly more powerful than the Experiment Server so that, again, it does not act as a bottleneck or affect the results. The GUI server is specifically to support a real graphical browser and measure the capture-render delay.

B. SUPPORTING COMPONENTS

Supporting components have been developed to help benchmark. Firstly, A *FakeWebcam* component simulates an IP camera and provides a looping M-JPEG stream. This way it is possible to reliably simulate an arbitrary number of equal high-performing input sources. It also can embed a QR code with a timestamp into the image, so that the receiver can calculate the capture-render delay. This component has been implemented in Python and supports WSGI through Flask and WSGI. Therefore, an arbitrary number of worker processes can be started, and it can simulate an arbitrary number of cameras. Secondly, a *Requester* component can simulate client loads by requesting and reading a stream through SocketIO, as a real client's browser would do. In this case, it has been implemented in NodeJS. Similarly, several instances can be started, and each instance can simulate several clients.

The clip that the FakeWebcam loops, and that is used for the experiments, shows a remote laboratory. It has a 480x640 resolution and there is a total of 928 frames which loop continuously. Each frame is originally JPEG-compressed and is roughly 13 KB in size.

The source code for the supporting components and for the benchmarking scripts, and the video clip itself, are also available as Open Source.

C. MEASUREMENTS

For the Feeder component experiments the measured variables are CPU usage, the RAM usage, outgoing bandwidth usage and FPS. For the CamServer component experiments the measured variables are CPU usage, RAM usage, incoming bandwidth usage, FPS, and latency.

Measurements are taken on the Experiment Server, which holds the component that is under testing for each

FIGURE 6. Interactive live-stream with an embedded QR timestamp to measure the true capture-render delay.

experiment. Exceptions are the FPS and latency measurements for the CamServer experiments, which are taken in the GUI Server. Every measurement in the Experiment Server is taken 50 times and the average is computed. Every measurement in the GUI Server is taken 30 times and the average is computed.

Due to the nature of an interactive live-streaming system, measuring the capture-render delay is important. This is a challenge because not all streaming formats allow inserting a text-based timestamp into the frames. To work around this limitation, the servers used during the experiments are synchronized through the NTP protocol. A timestamp is calculated, and a QR code containing it is generated and embedded into each frame.[28] Figure 6 shows the live-stream with the embedded QR timestamp. When the frame is rendered, as in the image, the QR code can be read on the target computer. It can be compared to the current time to obtain an accurate capture-render delay. These measurements are taken on the GUI server, because a server with a window system (X Server in this case) is required to open a browser, use the platform's

[28] QR generation and embedding has been measured to take around 23 milliseconds in the Support Server. A small part of the latency measurement will thus be due to this.

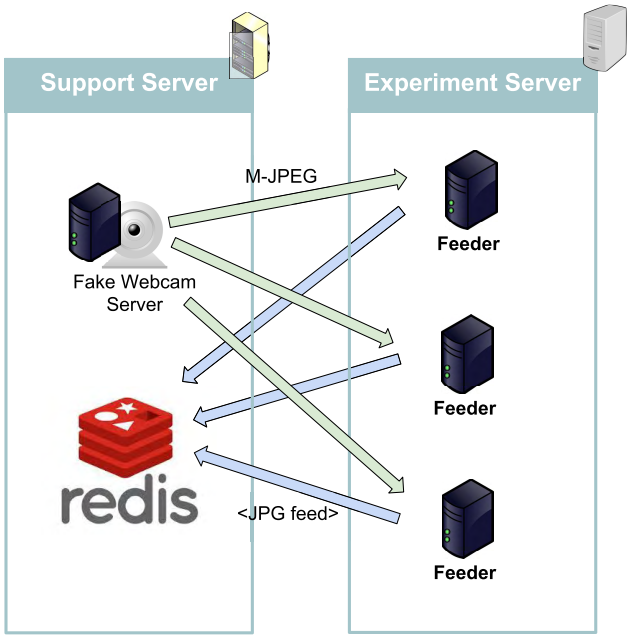

FIGURE 7. Setup and components for Experiment 1: Feeder components and image-refreshing.

JavaScript-based widgets to render the stream, and saving the image with the QR code timestamp.

VII. EXPERIMENTS

A. EXPERIMENT 1

Experiment 1 measures the performance of the Feeder component with the simple image-refreshing scheme. Figure 7 shows this setup. The Support Server hosts a Redis instance and FakeWebcam instances. The Experiment Server hosts the Feeder component. The experiment is run 50 times, with an increasing number of cameras, from 1 to 50. The Feeders read the stream from them and forward the frames into the Redis servers in the image-refreshing format. The number of Feeder instances is increased proportionally to the number of cameras (an instance is added for every 10 cameras). All the measurements are taken in the Experiment Server, which contains the Feeders. As described in Section VI, each measurement is taken 50 times, and the average is computed.

B. EXPERIMENT 2

Experiment 2 measures the performance of the Feeder component with the H.264 format scheme. Figure 8 shows this setup, which is similar to Experiment 1's. In this case, the H.264 format scheme is evaluated instead. The experiment is also run 50 times, with an increasing number of cameras (from 1 to 50). This time, however, transcoding is required, so FFmpeg instances to transcode into H.264 are used. These instances forward the stream into the Feeders and the Feeders into Redis, using the channel-based scheme. All the measurements are taken in the Experiment Server, which contains the Feeders and the FFmpeg instances. As described in Section VI, each measurement is taken 50 times, and the average is computed.

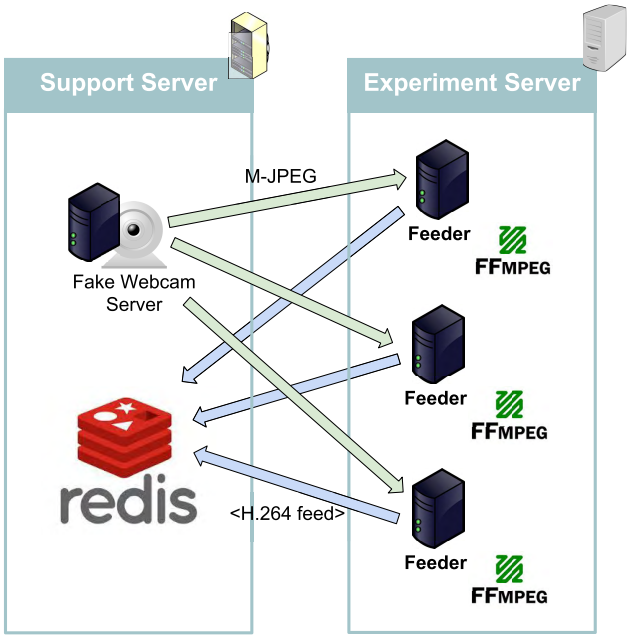

FIGURE 8. Setup and components for Experiment 2: Feeder components and H.264.

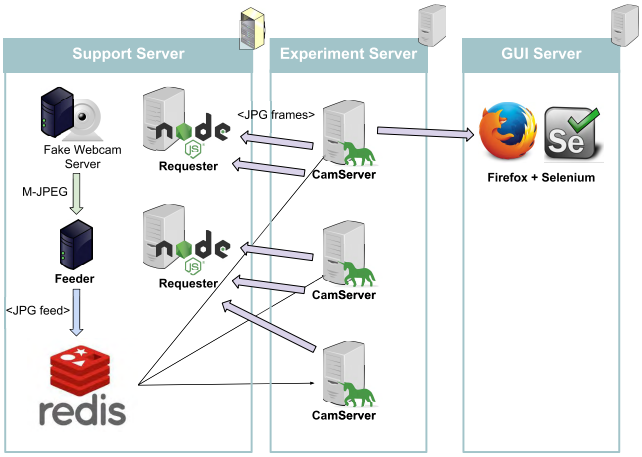

FIGURE 9. Setup and components for Experiment 3: CamServer components and image-refreshing.

C. EXPERIMENT 3

Experiment 3 measures the performance of the CamServer component with the simple image-refreshing scheme. Figure 9 shows this setup. It is significantly more complex than the setup for the previous experiments. The Support Server hosts the Redis instance, a single Feeder component, and a varying number of *Requester* components to simulate end-user load. The Experiment Server hosts several CamServer components (6 Gunicorn-initiated worker processes of Gevent type). The GUI Server hosts a Selenium script[29] that also simulates load for a single client. A script in the GUI servers opens a real browser, renders the stream, and automatically extracts the timestamp from the frame's QR code,

[29]http://www.seleniumhq.org/

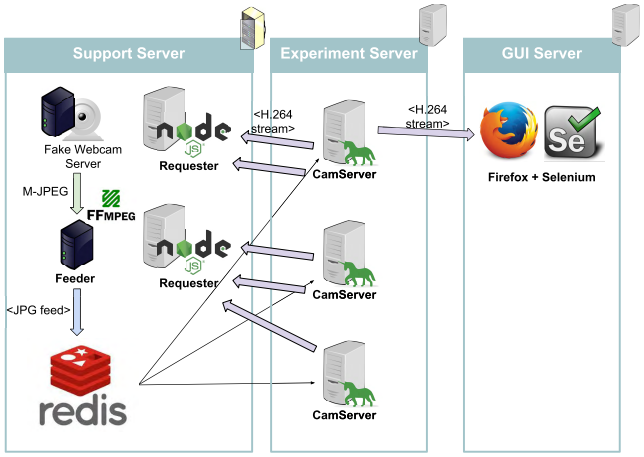

FIGURE 10. Setup and components for Experiment 4: CamServer components and H.264.

measuring the FPS and capture-render delay. The other measurements are taken from the Experiment Server. Note that in this case FPS and latency measurements are particularly realistic, because they are measured in a real browser while rendering the frames. As described in Section VI, each measurement in the Experiment Server is taken 50 times, and the average is computed. Measurements in the GUI Server (FPS and latency) are taken 30 times, and the average is computed as well.

D. EXPERIMENT 4

Experiment 4 measures the performance of the CamServer component with the H.264 format scheme. Figure 10 shows this setup. It is very similar to Experiment 3's, except that in this case the H.264 format scheme is evaluated instead. Transcoding is needed again, so the single Feeder instance is aided by an FFmpeg. Redis channels are used this time to transit the H.264 stream. In this case, the stream is transmitted using SocketIO. It is noteworthy that properly rendering the H.264 stream using the provided widgets requires WebGL, so for this experiment, relying on the GUI Server to open a real browser and measure the capture-render delay is particularly important. It can't easily be calculated without truly rendering the stream. As described in Section VI, each measurement in the Experiment Server is taken 50 times, and the average is computed. Measurements in the GUI Server (FPS and latency) are taken 30 times, and the average is computed as well.

VIII. RESULTS

The results of the experiments are presented in this section. Charts are provided for all of them. The charts combine different variables. Units have been chosen so that they are reasonably proportioned on the vertical axis. CPU is presented in usage percent. RAM is presented in GB. Bandwidth is presented in megabytes per second. Latency (capture-render delay) is presented in centiseconds (the unit is thus 10ms), so that the scales match.

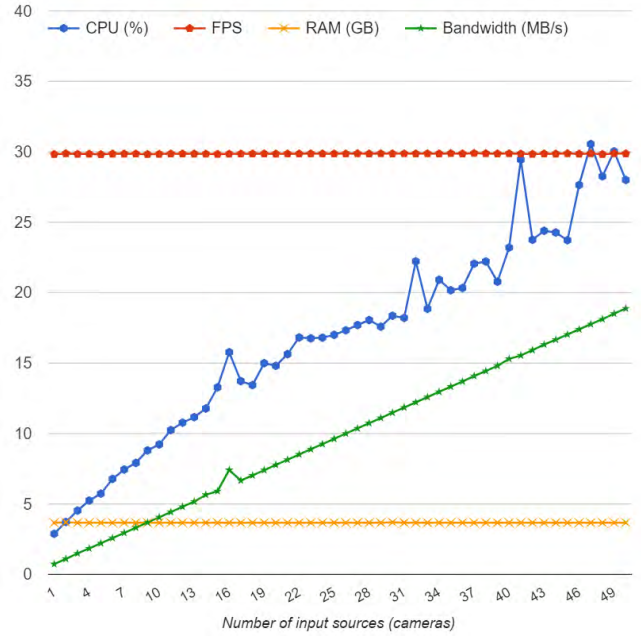

FIGURE 11. Results of Experiment 1. Feeder component using the image-refreshing technique.

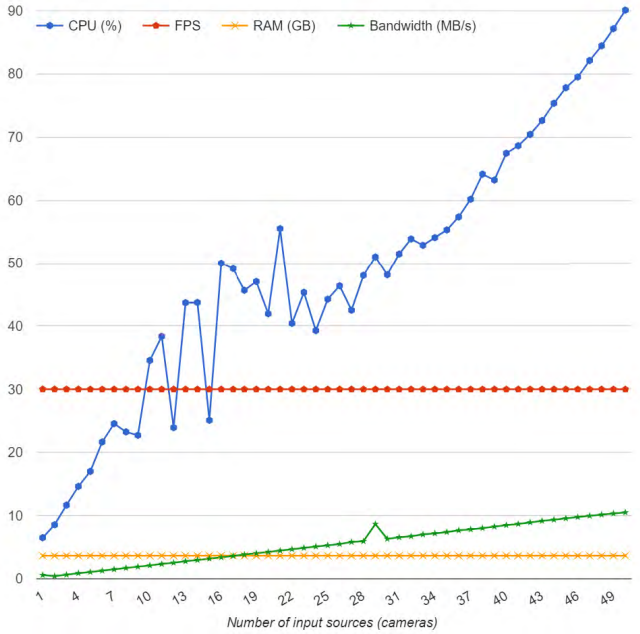

FIGURE 12. Results of Experiment 2. Feeder component using the H.264 format technique.

A. EXPERIMENT 1

The results of Experiment 1 are summarized in Figure 11. The system's RAM usage remains nearly constant. This is expected, because individual frames are small and the architecture is designed to replace old frames with updated ones: no frames are stored. The value itself (around 3.6 GB) is not particularly relevant (most of it is allocated by the system itself and not the platform). The target 30 FPS is maintained without issues. Bandwidth and CPU usage increase linearly with the number of cameras. CPU usage raises up to around 30% at 50 cameras. Bandwidth raises up to 28.8 MB/s at 50 cameras, which is roughly 576 KB/s per camera.

B. EXPERIMENT 2

The results of Experiment 2 are summarized in Figure 12. RAM usage, again, remains nearly constant. The target 30 FPS is maintained throughout. Bandwidth and CPU usage increase linearly with the number of cameras, using up to around 85% CPU at 50 cameras. This is significantly higher than the CPU usage in the previous experiment, which is understandable because in this case all the streams are being transcoded from M-JPEG into H.264. At the same time, nonetheless, the bandwidth requirements are significantly lower. This is expected, as the compression of H.264 is significantly more effective than that of image-refreshing, which is similar to M-JPEG. At 50 cameras it consumes around 10.8 MB/s, which is roughly 216 KB/s per camera.

C. EXPERIMENT 3

The results of Experiment 3 are summarized in Figure 13. RAM usage is mostly constant. The FPS is also mostly constant at 28 FPS. Bandwidth and CPU increase linearly with the number of users, with CPU usage at

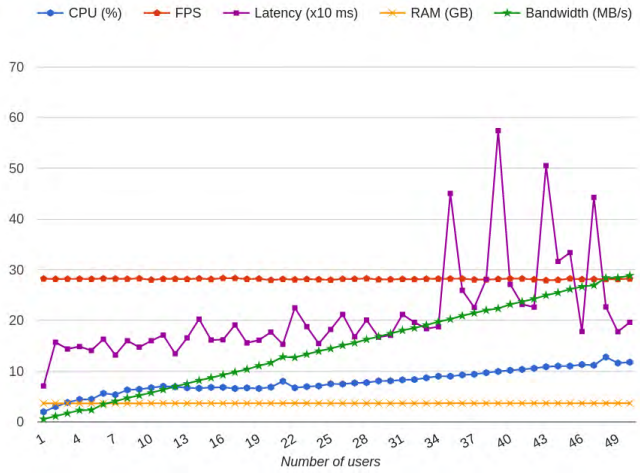

FIGURE 13. Results of Experiment 3. CamServer component using the image-refreshing technique.

around 12% for 50 users, and bandwidth usage at around 28.8 MB/s (576 KB/s per stream). In this experiment, the latency (capture-render delay) is measured as well. To do this, as previously described, the stream is rendered in a real Firefox browser and the QR timestamping technique is used to determine the delay. In this case, the delay remains mostly constant. The average is 213 ms ($\mu = 95$), though there are some peaks of up to 574 ms. It is noteworthy that though there might seem to be a significant variation in latency in the chart due to the chosen scale, the absolute value is quite low, always below 0.6 seconds.

D. EXPERIMENT 4

The results of Experiment 4 are summarized in Figure 14. RAM usage is again mostly constant. The target 30 FPS is

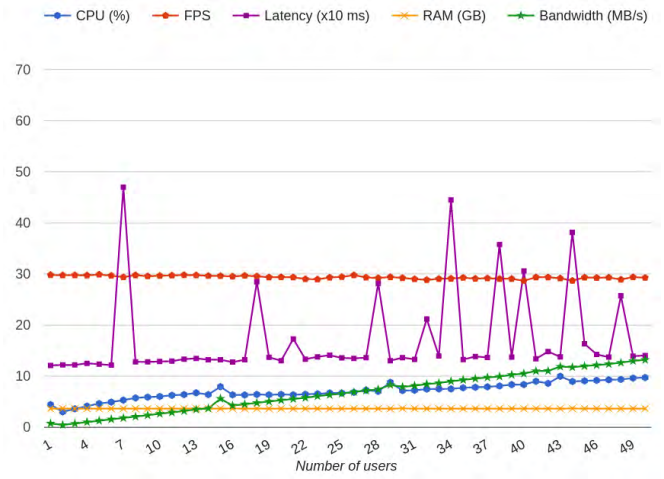

FIGURE 14. Results of Experiment 4. CamServer component using the H.264 format technique.

mostly maintained, sometimes decreasing very slightly to around 29 FPS. Bandwidth and CPU increase linearly with the number of users, with CPU usage at around 10% for 50 users, and bandwidth usage at around 19.9 MB/s (around 398 KB/s per stream). The capture-render delay remains mostly constant as well. The average is 171 ms ($\mu = 85$), though there are some peaks of up to 383 ms. It might again be noteworthy that the absolute value is nonetheless quite low (always below 0.4 seconds in this case).

IX. DISCUSSION

In Section III we described the goals and requirements of the proposed platform. The evaluation, through the contributed implementation and the described experiments, suggests that those goals and requirements are indeed met.

A. GOALS

The goals were the following:

- Universality
- Efficiency and scalability
- Openness

The platform, from a client-side perspective, can indeed be considered to provide high universality. The two main streaming approaches (which are image-refreshing and H.264), described in more detail in [4], are fully web-based. They rely only on JavaScript and HTML5 and are compatible with any modern device, including mobile phones and tablets. The communication protocols that are required are HTTP and optionally WebSockets.

The *efficiency and scalability* of the platform are satisfactory as well. The results suggest that both the Feeders and CamServers are efficient for both image-refreshing and H.264. As expected, there are some performance differences between them. The most significant one is that image-refreshing takes less CPU but takes significantly more bandwidth. This is understandable, because image-refreshing is essentially an M-JPEG variation: there is no interframe compression. No transcoding is needed, but, in exchange, the compression rate is worse than for H.264.

The capture-render delay that the platform provides, for both schemes, is similarly low. Some minor peaks are present but they are within reasonable bounds. It is noteworthy that this H.264 delay is particularly small, considering that it is an interframe-compression format. The codec has been configured to very aggressively minimize latency, reducing buffer sizes to a minimum and sacrificing compression. In exchange, as observed, the bandwidth usage, while not particularly large, is higher than that of a typical H.264 stream.

A single server can handle more than 50 cameras or users (in some cases many more, depending on the format). Moreover, both the Feeders and CamServers scale horizontally. This has not been explicitly tested during the experiments, but they are based on technologies such as WSGI which rely on multiple independent processes. And, indeed, multiple independent processes were used during the experiments.

The *openness* goal is also satisfied. The libraries and technologies that the platform relies on are Open Source. The implementation of the platform itself has also been released as Open Source.

B. REQUIREMENTS

The requirements that were stated were the following:

- Interactive live-streaming
- Supporting multiple sources
- Supporting multiple output schemes

These particular technical requirements are indeed met by the platform. The experiments show that the capture-render delay that the platform can provide is indeed low enough to be considered interactive-level. It supports an arbitrary number of input cameras. They also support multiple output schemes. These include the image-refreshing scheme and the H.264-based schemes that have been discussed throughout this work and evaluated through the experiments. They also include some additional schemes, such as rendering M-JPEG through JavaScript in the client-side. These have not been discussed but are present in the contributed source.

With all these goals and requirements met, we expect that this work will be useful for both research and practical purposes. It is our intention, particularly, that the remote laboratory community may benefit from an easy to deploy, integrable, interactive live-streaming system that improves the user experience of remote laboratories.

X. CONCLUSIONS AND FUTURE WORK

This work has proposed goals and requirements for an open, web-based, interactive live-streaming platform. These goals and requirements are particularly well-suited for the field of remote laboratories, though the platform is intended to be useful for any other applications which have similar requirements. Then, an architecture to meet those goals has been designed. An implementation of that architecture has been

created and released as Open Source.[30] That contributed implementation has been used to experimentally evaluate the architecture. Results suggest that the goals and requirements are indeed met and that the proposed architecture will indeed be useful for practical and research purposes. Most live-streaming platforms, and especially interactive live-streaming ones, are not open. It is expected that the proposed platform and its implementation, due to its open nature, can be integrated, used and customized as needed by other researchers or developers.

These results are promising. Nonetheless, some lines of work remain open and could be pursued in the future. To our knowledge, there are no other interactive live-streaming architectures that are specifically oriented towards remote laboratories. However, some alternative open source platforms, with different goals, do exist. An example is *nginx-rtmp-module*.[31] It would therefore be interesting to compare these architectures in terms of performance and in terms of remote laboratory requirements. It would also be interesting to analyze the architecture's performance under varying network conditions. Furthermore, in the future, new interactive live-streaming schemes may be added to the platform and evaluated. In the latest times, new promising standards such as MPEG-DASH have appeared. Although they are not necessarily suited for interactive live-streaming, as they are developed and become more mainstream they may be explored as additional live-streaming schemes. Also, live-streaming additions to the HTML5 *video* tag are expected to appear in the future. Once such additions are standardized and well-supported by modern browsers, they may be explored as well.

REFERENCES

[1] C. Zhang and J. Liu, "On crowdsourced interactive live streaming: A twitch. TV-based measurement study," in *Proc. 25th ACM Workshop Netw. Oper. Syst. Support Digit. Audio Video*, 2015, pp. 55–60.

[2] D. Nishantha, Y. Hayashida, and T. Hayashi, "Application level rate adaptive motion-JPEG transmission for medical collaboration systems," in *Proc. IEEE 24th Int. Conf. Distrib. Comput. Syst. Workshops*, 2004, pp. 64–69.

[3] J. Nielsen, *Usability Engineering*. Amsterdam, The Netherlands: Elsevier, 1994.

[4] L. Rodriguez-Gil *et al.*, "Interactive live-streaming technologies and approaches for Web-based applications," *Multimedia Tools Appl.*, pp. 1–32, 2017, doi: 10.1007/s11042-017-4556-6.

[5] *Advanced Video Coding for Generic Audiovisual Services*, document ITU-T Rec. H.264 & ISO/IEC 14496-10 AVC, v3:2005, Amendment 3: Scalable Video Coding.

[6] W. Van Lancker, D. Van Deursen, E. Mannens, and R. Van de Walle, "Implementation strategies for efficient media fragment retrieval," *Multimedia Tools Appl.*, vol. 57, no. 2, pp. 243–267, 2012.

[7] L. Rodriguez-Gil, P. Orduña, J. García-Zubia, I. Angulo, and D. López-de-Ipiña, "Graphic technologies for virtual, remote and hybrid laboratories: WebLab-FPGA hybrid lab," in *Proc. IEEE 11th Int. Conf. Remote Eng. Virtual Instrum. (REV)*, Feb. 2014, pp. 163–166.

[8] 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.

[9] (Oct. 2014). "HTML5 specification," W3, Tech. Rep. [Online]. Available: https://www.w3.org/TR/html5

[30] Available at: https://github.com/zstars/wilsp

[31] https://github.com/arut/nginx-rtmp-module

[10] (Oct. 2014). "WebGL specification," Khronos WebGL Working Group, Tech. Rep. [Online]. Available: https://www.khronos.org/registry/webgl/specs/1.0/

[11] (2017). *YouTube Now Defaults to HTML5 Video*. [Online]. Available: http://youtubeeng.blogspot.com.es/2015/01/youtubenowdefaultstohtml5_27.html

[12] L. Gomes and S. Bogosyan, "Current trends in remote laboratories," *IEEE Trans. Ind. Electron.*, vol. 56, no. 12, pp. 4744–4756, Dec. 2009.

[13] J. G. Zubía and G. R. Alves, Eds., *Using Remote Labs in Education: Two Little Ducks in Remote Experimentation*, vol. 8. Bilbao, Spain: Universidad de Deusto, 2012.

[14] 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.

[15] 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.

[16] S. Akhshabi, A. C. Begen, and C. Dovrolis, "An experimental evaluation of rate-adaptation algorithms in adaptive streaming over HTTP," in *Proc. 2nd Annu. ACM Conf. Multimedia Syst.*, 2011, pp. 157–168

[17] X. Hei, C. Liang, J. Liang, Y. Liu, and K. W. Ross, "A measurement study of a large-scale P2P IPTV system," *IEEE Trans. Multimedia*, vol. 9, no. 8, pp. 1672–1687, Dec. 2007.

[18] B. Wang, X. Zhang, G. Wang, H. Zheng, and B. Y. Zhao, "Anatomy of a personalized livestreaming system," in *Proc. ACM Internet Meas. Conf.*, 2016, pp. 485–498.

[19] R. Shea, J. Liu, E. C.-H. Ngai, and Y. Cui, "Cloud gaming: Architecture and performance," *IEEE Netw.*, vol. 27, no. 4, pp. 16–21, Jul./Aug. 2013.

[20] M. Ueberheide, F. Klose, T. Varisetty, M. Fidler, and M. Magnor, "Web-based interactive free-viewpoint streaming: A framework for high quality interactive free viewpoint navigation," in *Proc. 23rd ACM Int. Conf. Multimedia*, 2015, pp. 1031–1034.

[21] V. J. Harward *et al.*, "The iLab shared architecture: A Web services infrastructure to build communities of Internet accessible laboratories," *Proc. IEEE*, vol. 96, no. 6, pp. 931–950, Jun. 2008.

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

[23] J. J. Rodriguez-Andina, L. Gomes, and S. Bogosyan, "Current trends in industrial electronics education," *IEEE Trans. Ind. Electron.*, vol. 57, no. 10, pp. 3245–3252, Oct. 2010.

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

[25] Z. Nedic, J. Machotka, and A. Nafalski, "Remote laboratories versus virtual and real laboratories," in *Proc. IEEE 33rd Annu. FIE*, vol. 1. Nov. 2003, pp. T3E-1–T3E-6.

[26] P. Orduña, P. H. Bailey, K. DeLong, D. López-de-Ipiña, and J. García-Zubia, "Towards federated interoperable bridges for sharing educational remote laboratories," *Comput. Human Behavior*, vol. 30, pp. 389–395, Jan. 2014.

[27] R. Hashemian and J. Riddley, "FPGA e-Lab, a technique to remote access a laboratory to design and test," in *Proc. IEEE Int. Conf. Microelectron. Syst. Edu. (MSE)*, Jun. 2007, pp. 139–140.

[28] 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, 2008.

[29] 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.

[30] 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.

[31] L. Rodriguez-Gil, J. García-Zubia, P. Orduña, and D. López-de-Ipiña, "Towards new multiplatform hybrid online laboratory models," *IEEE Trans. Learn. Technol.*, 2016, doi: 10.1109/TLT.2016.2591953.

[32] (2017). *Java Website*, accessed on Apr. 30, 2017. [Online]. Available: https://www.java.com

[33] (2017). *Adobe Flash Website*, accessed on Apr. 30, 2017. [Online]. Available: http://www.adobe.com/es/products/flashplayer.html

[34] J. Soares and J. Lobo, "A remote FPGA laboratory for digital design students," in *Proc. 7th Portuguese Meeting Reconfigurable Syst. (REC)*, 2011, pp. 95–98.

[35] H. Schulzrinne, A. Rao, and R. Lanphier, *RTSP: Real Time Streaming Protocol*, document IETF RFC2326, Apr. 1998.
[36] McAfee, "McAfee Labs threats report, May 2015," Intel Secur., Santa Clara, CA, USA, Tech. Rep., May 2015.
[37] H. Crompton, D. Burke, K. H. Gregory, and C. Gräbe, "The use of mobile learning in science: A systematic review," *J. Sci. Edu. Technol.*, vol. 25, no. 2, pp. 149–160, 2016.
[38] L. J. Couse and D. W. Chen, "A tablet computer for young children? Exploring its viability for early childhood education," *J. Res. Technol. Edu.*, vol. 43, no. 1, pp. 75–96, 2010.
[39] S. N. Şad and Ö. Göktaş, "Preservice teachers' perceptions about using mobile phones and laptops in education as mobile learning tools," *Brit. J. Edu. Technol.*, vol. 45, no. 4, pp. 606–618, 2014.
[40] D. G. de la Iglesia, J. F. Calderón, D. Weyns, M. Milrad, and M. Nussbaum, "A self-adaptive multi-agent system approach for collaborative mobile learning," *IEEE Trans. Learn. Technol.*, vol. 8, no. 2, pp. 158–172, Apr./Jun. 2015.
[41] K. Pires and G. Simon, "YouTube live and Twitch: A tour of user-generated live streaming systems," in *Proc. ACM 6th ACM Multimedia Syst. Conf.*, 2015, pp. 225–230.
[42] D. Stohr, T. Li, S. Wilk, S. Santini, and W. Effelsberg, "An analysis of the YouNow live streaming platform," in *Proc. IEEE 40th Local Comput. Netw. Conf. Workshops (LCN Workshops)*, Oct. 2015, pp. 673–679.
[43] J. C. Tang, G. Venolia, and K. M. Inkpen, "Meerkat and periscope: I stream, you stream, apps stream for live streams," in *Proc. CHI Conf. Human Factors Comput. Syst.*, 2016, pp. 4770–4780.
[44] M. Siekkinen, E. Masala, and T. Kämäräinen, "Anatomy of a mobile live streaming service: The case of Periscope," in *Proc. IMC*, 2016, pp. 1–8.
[45] C. Ceglie, G. Piro, D. Striccoli, and P. Camarda, "*3DStreaming*: An open-source flexible framework for real-time 3D streaming services," *Multimedia Tools Appl.*, vol. 75, no. 8, pp. 4411–4440, 2016.
[46] M. Claypool and D. Finkel, "The effects of latency on player performance in cloud-based games," in *Proc. IEEE 13th Annu. Workshop Netw. Syst. Support Games (NetGames)*, Dec. 2014, pp. 1–6.
[47] A. Salkintzis and N. Passas, Eds., *Emerging Wireless Multimedia: Services and Technologies*. Hoboken, NJ, USA: Wiley, 2005.
[48] J. L. Carlson, *Redis in Action*. Greenwich, CT, USA: Manning Publications Co., 2013.
[49] (2017). *Redis Clustering Tutorial*, accessed on Apr. 28, 2017. [Online]. Available: https://redis.io/topics/cluster-tutorial
[50] Y. Abubakar, T. S. Adeyi, and I. G. Auta, "Performance evaluation of NoSQL systems using YCSB in a resource austere environment," *Int. J. Appl. Inf. Syst.*, vol. 7, no. 8, pp. 23–27, 2014.

LUIS RODRÍGUEZ-GIL received the double degree in computer engineering and industrial organization engineering in 2013, and the M.Sc. degree in information security in 2014. He is currently pursuing the Ph.D. degree with the DeustoTech Internet Group, University of Deusto. Since 2009, he has been with the WebLab-Deusto Research Group, collaborating in the development of the WebLab-Deusto RLMS. He has authored a number of peer-reviewed publications and contributed to several Open Source projects.

JAVIER GARCÍA-ZUBIA (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, and embedded systems design. He is the President of the Spanish Chapter of the IEEE Education Society.

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

DIEGO LÓPEZ-DE-IPIÑA received the Ph.D. degree from the University of Cambridge in 2002. He is currently an Associate Professor and Principal Researcher of the MORElab Group, and the Director of the DeustoTech Internet Unit, and the Ph.D. program within the Faculty of Engineering, University of Deusto. Responsible for several modules in the B.Sc. and M.Sc. degrees in computer engineering. He has over 70 publications in relevant International conferences and journals, including over 25 JCR-indexed articles. He is interested in pervasive computing, Internet of Things, semantic service middleware, open linked data, and social data mining. He is taking and has taken part in several big consortium-based research European, including IES CITIES, MUGGES, SONOPA, CBDP, GO-LAB, and LifeWear, and Spanish projects.

APPENDIX

C

Paper III

Teacher-defined tutoring agents: Authoring through a visual domain-specific language

This paper is currently being considered for publication in the journal '*IEEE Transactions on Learning Technologies*'. It was submitted on 3 Mar 2017. The author-submitted version is appended here.

Teacher-defined tutoring agents: Authoring through a visual domain-specific language

Luis Rodriguez-Gil, Javier García-Zubia, *Senior Member, IEEE,* Pablo Orduña, *Member, IEEE* and Diego López-de-Ipiña

Abstract—Intelligent tutors and conversational agents have proven to be useful learning tools. They have potential not only as stand-alone devices but also as integrable components to enrich and complement other educational resources. For this, new authoring approaches and platforms are required. They should be accessible to non-programmers (such as most teachers) and they should be integrable into current web-based educational platforms. This work proposes a new approach to define such agents through a visual domain-specific language based on Google Blockly (a Scratch-like language). It also develops a web-based integrable authoring platform to serve as a prototype, describing the requirements and architecture. To evaluate whether this novel approach is effective a multi-stage experiment was conducted. Firstly, participants learned to use the prototype authoring platform through an interactive tutorial. Secondly, they created a conversational agent with a specific domain model. Times and performance were measured. Finally, they answered a standardized usability questionnaire (UMUX) and a purpose-specific survey. Results show that participants were able to learn to use the domain-specific language in a short time. Moreover, the purpose-specific survey indicates that their perception of the approach (and its potential) is very positive. The standardized questionnaire indicates that even in its prototype stage, its usability is satisfactory.

Keywords—visual programming languages, customizable systems, conversational agents, intelligent tutoring systems, online learning, online labs

1 Introduction

Conversational agents (CAs) and intelligent tutoring systems (ITSs) have existed for decades [1], [2]. Those two types of systems have different characteristics. The former focus on representing a human, often featuring a virtual body. The latter provide a learning environment, are often task-based, and try to resemble human tutoring to leverage its advantages [3]. They have aspects in common, and some ITSs include or are based on CAs [1].

Throughout the years, significant research efforts have been dedicated to these systems. Traditionally, one of the aspects that have drawn more attention is their natural language processing (NLP) capabilities. For example, this was the focus of Eliza [4], one of the first and most influential CAs. It relied on a simple pattern-matching algorithm to create the illusion of intelligence. Other researchers have experimented with Embodied CAs [5], [6], which aim to increase believability by also having a virtual animated body. Attention has also been directed towards approaches for building systems with comprehensive domain models, and powerful authoring tools [7], [8].

CAs have many potential applications in education (e.g., tutoring or question answering [9]). In other fields, they can also provide customer service, information, or act as a virtual companion or website tour guide [10]. Interest in them keeps growing. However, they can be expensive to create; and domain experts do not always have programming experience. Effective authoring tools are important to reduce those obstacles [11], [7].

In this work we describe a novel approach for the creation of web-based CAs for tutoring, and we present the architecture of a web-based authoring platform that uses it. The approach and the platform aim to be used by non-programmers. The approach relies on a simple Domain-Specific Language (DSL) based on Google Blockly [12] —a visual programming language that has been successfully used by non-programmers, including children [13]— and are designed to create domain-independent agents that may be integrated into external platforms.

Though this work could be applied for the authoring of many types of CA, it is oriented towards the creation of embeddable educational agents. Throughout the last years, the use of educational technologies such as online courses, MOOCs [14], and online laboratories [15], [16], [17] has increased, and the trend is expected to continue. Some research works and large initiatives, such as the European project Go-Lab, suggest that for the wide and

- *L. Rodriguez-Gil, P. Orduña and D. López-de-Ipiña are with the Faculty of Engineering, University of Deusto, Avda. Universidades, 24, 48007, Bilbao, Spain and DeustoTech - Deusto Foundation, Avda. Universidades, 24, 48007, Bilbao, Spain. E-mail: luis.rodriguezgil@deusto.es.*
- *J. García-Zubia is with the Faculty of Engineering, University of Deusto, Avda. Universidades, 24, 48007, Bilbao, Spain.*
- *This paper has supplementary downloadable material available at http://ieeexplore.ieee.org, provided by the authors. This includes two screencasts, which briefly show the authoring platform and the experiment stages.*

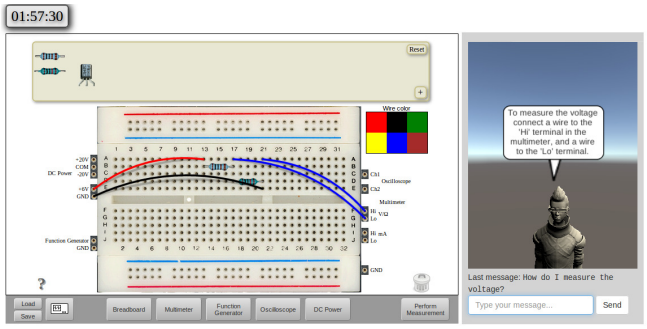

Fig. 1. Embedding a teacher-defined agent into a VISIR remote lab

successful adoption of these tools it is important that teachers are able to provide customized experiences for their students [18], [19], [20], [21], [22]. In this context, educational agents that are customizable by teachers and easily embedded into other educational content (e.g., online exercises or online labs) to provide guidance may add significant value. This is the intended main practical application of this scheme. Figure 1 illustrates this with an example: a teacher-defined CA accompanies an instance of the VISIR [23] remote lab, in order to provide specific guidance for a practice.

This contribution focuses on providing the means for the general non-programming public (such as most teachers) to create and customize their own agents. It does not aim to obtain agents with improved NLP capabilities, more natural conversations or higher believability. Non-programming users define the knowledge domain of the agent through the Google Blockly based DSL. It is intended to be most useful for use cases where the number of rules is not large, the user input is predictable and a well-defined conversational output is preferred. The authoring platform that is proposed in this work is designed to create relatively simplistic agents. However, Google Blockly has been used as a programming language, and it is designed to be extensible. As such, future systems that use this approach could provide arbitrarily advanced functionality through additional custom *blocks*, which in Google Blockly are akin to functions in traditional programming languages.

This paper is organized as follows: Section 2 describes in more detail the relevant state of the art on customizable CAs, authoring tools, and languages oriented for non-programmers. Section 3 analyzes the requirements for the proposed approach and the authoring platform. Section 4 presents the proposed approach. Section 5 details the platform architecture. Section 6 describes the methodology of the study that has been conducted to evaluate how easily such an authoring approach can be learned, how usable it is, and what the user perception is. Section 7 describes the results of the study, organized around the research questions. Section 8 discusses these results. Section 9 summarizes the conclusions. Finally, Section 10 proposes future lines of work and potential applications.

1.1 Purpose and research questions

The purpose of this work is to propose a novel approach to enable non-programmers to define custom CAs through the use of a Visual DSL; to create a prototype web-based authoring platform that implements that approach and that satisfies real-world technical requirements; and to answer the following research questions:

1) How easy to learn is the proposed approach? Can non-programmers learn it in a reasonable time?
2) How usable is the proposed approach?
3) Is the proposed approach perceived as valuable and intuitive?

2 Background

2.1 Agent authoring tools and non-programmers

Creating agents can take significant time and effort. Authoring tools can be leveraged to create agents faster and more efficiently, reducing the amount of time and resources required [8], [24]. Research efforts are dedicated towards their development and improvement. For example, research is being conducted on authoring tools for collaborative ITS and CA environments [25], [8], [26], on integrating ITSs into serious games [27], on developing authoring models such as Authoring by Tutoring [28], on making mobile authoring tools [29], and on making authoring tools available to non-programmers [27], [30], [31], [32], [33]. This last goal is, indeed, the main focus of this work.

Past research efforts have led to various authoring tools and frameworks which aim to reduce the skills and amount of time required to create tutors. Tools such as CTAT [34], which is a well-known set of tools that relies on the Jess rule-language; frameworks such as GIFT [35] or other authoring tools such as TDK [36]. Different models are available to specify the ITSs and CAs, with varying levels of power and complexity, and different advantages and disadvantages. For example, CTAT's cognitive tutor creation relies on XML and Jess rules[1]. It is used by systems such as [30]. It is powerful, but, though it does indeed require no programming experience, it tends to require considerable IT expertise. Specifically, it requires knowledge of computer languages such as the aforementioned ones, and, generally, it involves using relatively complex environments such as the Flash IDE or Eclipse. Other tools rely on specific XML-based languages [33]. CTAT's *example-tracing* tutor creation requires no JESS knowledge and is simpler [37]. Works such as [31] rely on conversation trees. In this case, the non-programming authors build a tree structure, specifying for each branch a selection of responses among which the end-user can choose. Other systems

1. http://ctat.pact.cs.cmu.edu

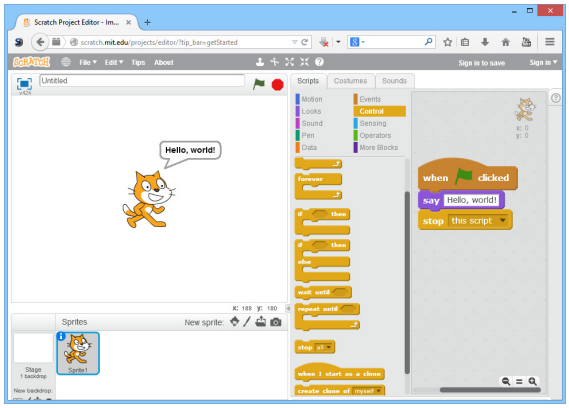

Fig. 2. The Scratch visual programming language and environment.

rely on NLP and other techniques to extract information from different sources and automatically generate an ITS or a CA [38], [39].

In this work we propose a novel model to define such an agent. A domain specific language (DSL) that relies on a Visual Programming Language specifically designed for non-programmers has been created. Through it, even those users who are not experienced in IT can specify rules and customize the behavior and domain-model of the agent.

2.2 Visual programming languages

Visual programming languages are those that are based on graphical elements. Examples of those elements are puzzle blocks, in the case of Google Blockly [12] and Scratch [40]; or block diagrams, in the case of LabView [41]. They rely on drag-and-drop and other spatial actions instead of being based on text. As an example, Figure 2 shows the Scratch environment, oriented for children.

Visual programming languages have proven their usefulness in certain domains such as education [42]: Google Blockly or Scratch have been designed to be intuitive and easily understood [43], even by very young students. Google Blockly has been used to teach them programming [44], [45]. It is particularly fit for these tasks [43], because, among other reasons:

- It is intuitive: students can see the blocks that are available and check whether they connect.
- Students do not need to remember a syntax and adhere to it before they start getting results.
- No syntax errors: if the blocks fit, then the program is valid[2].

Google Blockly is Open Source. It is based on blocks that the user can drag-and-drop to connect to each other and nest them. Once the blocks are arranged, Blockly can generate executable code for text-based languages (e.g., JavaScript, Python, and other languages), so that it can be run. JavaScript is the one most commonly used, because then it can run straightaway in the browser. Although it is originally designed to teach young students programming, it is also designed to be completely customizable, including the blocks, the connections, the generated code and the looks. For that reason, it is particularly effective for creating visual DSLs. It has been used for purposes as varied as educational robots scripting [46], smart home applications [47] and business processes modeling [48].

2. This does not necessarily imply that the program does what the user expects. It only implies that it can be translated into a text-based language with no syntax errors.

3 Requirements

This section describes, first, the general requirements for the proposed approach. Second, it describes additional, more specific requirements for the prototype authoring platform. The main design goals for the proposed approach are the following:

- **Low skill requirements**: Non-programmers should be able to use it.
- **Simple and predictable output**: The CA creators should be able to easily predict the output, which should be well-defined.
- **Integrable into other resources**: The produced agents are meant to be integrated into other educational resources.

Additionally, the authoring platform that implements the proposed approach should meet the following practical and technical requirements:

- **Web-based and mobile-friendly**: The tools and the produced intelligent tutors should be fully web-based, thus available anywhere.
- **Integrable agents into external systems**: As previously described, the generated agents are not meant to be stand-alone intelligent tutors. Instead, they are meant to be integrated into other systems such as Learning Management Systems or Remote Labs, as an aide. It should be technically possible to integrate them with minimal intervention from the external systems.

The proposed approach is designed to be extensible. Advances in related fields could be leveraged to provide advanced features for the produced agents (e.g., speech recognition, realistic conversations). However, those capabilities are beyond the scope of this work. Therefore, the authoring platform aims for simplicity and is explicitly willing to compromise on certain aspects. It does not aim for the agents to be able to converse realistically or to make use of advanced Natural Language Processing (NLP) techniques. It does not aim for the agents to be able to learn by themselves through Machine Learning techniques. Instead, it focuses on being able to produce predictable, known and controlled output, without requiring data sets. It does not aim for the agents to be particularly powerful from an Artificial Intelligence point of view, but aims for simplicity instead.

In the next subsections, the aforementioned goals and requirements are described in more detail.

3.1 Low skill requirements

Teachers and domain experts do not necessarily have programming or advanced IT skills. So that they can create and customize the agents, it is important that the authoring tool be easy to use and intuitive. It should avoid requiring knowledge of programming or of computer languages (such as XML, JSON or AIML) that are not necessarily easy to understand for the general public.

3.2 Simple and predictable output

The design goal of some CAs is to be as natural as possible. Such an agent should be able to provide varied and often unexpected output, like a human would. Some agents, such as Tay by Microsoft, even rely on previous conversations to learn new answers. This is, however, not the goal of the model proposed in this work. Natural agents are not without drawbacks. For instance, the agent by Microsoft, which has since been shut down, is known to have learned to be racist, utter profanities, and, in short, behave in a way that was particularly unexpected and undesirable for its creators[3]. The model proposed in this work explicitly sacrifices believability so that it can be easier to define, the output can be predictable and more easily understood, and no data sets are required.

3.3 Integrable into other resources

The CAs produced by the authoring platform are not meant to be used stand-alone. They are meant to be integrated into other educational contents, such as online labs. Those can be relatively complex virtual labs, as in the case of the PhET simulations [49]. Or even online interfaces to real equipment, as in the case of remote labs [50], [15]. Online labs are often designed to satisfy many use-cases, which makes it hard for students to use them without guidance. A teacher-defined CA would be able to provide context-specific information to the students, provide non-intrusive help when needed, and enhance the lab experience.

3.4 Web-based and mobile friendly

The goal of the described architecture is to be accessible to as many people as possible. For that purpose, it is very important that it is based on standard web technologies and that it supports mobile devices. This is critical for education: nowadays many schools and students rely on mobile and tablet devices [51], [52], [53], [54]. This implies that the architecture must only rely on web standards, and avoid certain proprietary technologies, such as Adobe Flash or Java Applets. Though common in the past, they no longer have wide support and have significant security implications [55].

This is also important for the platform to be deployable easily in different networks and environments. School and institutional networks are often restricted. Relying on non-HTTP ports, or on plugins or technologies which require administrator privileges to deploy, would make the system unacceptable to many potential users.

3.5 Integrable agents into external systems

As previously described, the produced CAs are meant to be integrable into other educational resources. From a technical point of view, those external resources are typically web-based platforms (such as Moodle instances or online labs) hosted by external providers. To be able to be used effectively, a key feature is that the produced agents should be able to be integrated into those resources without requiring intervention from the administrators of the external system (As an example, see Figure 1).

4 Proposed approach

One of the main focuses for the proposed approach is to allow non-programmers to create and customize CAs. Therefore, as described, it relies on a visual DSL. Its base (Google Blockly) was initially oriented towards children. This language is described in more detail in later sections. It is remarkable that it is oriented towards extensibility. The functionality of Google Blockly, and thus, the functionality of the DSL, depends on the *blocks* they provide. A *block* is, in that way, akin to a function in a standard programming language.

The non-programming authors, who are the domain experts, define the behavior of the CAs through that visual DSL. The base block is a *Conversation Node* block. The non-programming authors attach conditions and actions to them, which also take the form of blocks. A very simple condition block, for instance, would be a *words detected in input* block. A very simple action block, for instance, would be a *say* block. The variety, power, and complexity of action and condition blocks would vary depending on the purpose, needs, audience and platform, and can be tailored and extended easily.

With no extensions, the language and the prototype authoring platform are designed to be simplistic. This makes the agents most useful for creating narrow-scope agents, in which the inputs of the end-user are predictable and in which the author wants fine-grained control over the output of the agent. However, it makes them unsuitable for realistic human-like conversations.

One of the main motivations of this work is to enable teachers to create and integrate bots in educational content such as MOOCs, online courses, and remote [56], [57], virtual [49] or hybrid labs [58]. For this purpose, the approach is also designed for straightforward interaction with these systems. The bots themselves, as the prototype platform shows, can be integrated along with that

3. http://money.cnn.com/2016/03/24/technology/tay-racist-microsoft/

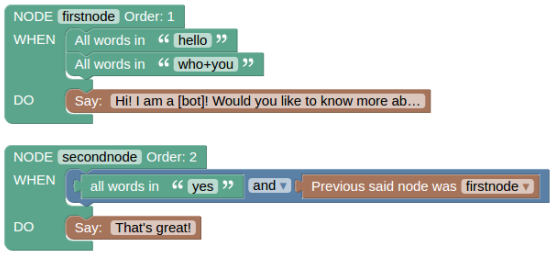

Fig. 3. Example of two base nodes with condition and action blocks within.

content. Also, the language is built to support straightforward interaction. An example is the *raise event* block. It can be used, for instance, for proposing and evaluating teacher-defined practical exercises. Thus, teachers could create an agent that proposes an exercise, integrate it into a generic virtual lab, and have their students solve the particular exercise using that generic lab. The agent would describe the exercise, provide guidance, evaluate the response, and raise a particular event when the exercise is solved.

Although not explored in detail in this work, this scheme is suitable for Embodied CAs (ECAs) [59], because the described DSL, with the proper blocks, would allow the author to control accurately the behavior and actions of the virtual character.

4.1 Visual Domain-Specific Language

In this section we detail the visual DSL and its basic blocks. However, as previously described, the system is designed to be extensible. Depending on the platforms, target, and advances in the state of the art, specific platforms that implement it may easily extend it.

4.1.1 Base conversation node

The basic block is the *Conversation Node*. This block establishes the conditions under which the node will be triggered, and the actions that will be executed when the conditions are met. Figure 3 shows an example with two base conversation nodes, each of which contains condition and action blocks. Users can attach or remove actions and conditions by simply dragging-and-dropping them.

4.1.2 Condition blocks

- **Contains-words condition block:** *Contains-words* is a basic condition block. It triggers the conversation node when the user's input contains all of the specified words. More than one word can be specified by separating them with plus signs or with spaces, as shown in Figure 3.
- **Logic condition block:** The logic condition blocks let users combine conditions with logical operators. An example is the *AND* block. They can be nested to build complex conditions.
- **Previous-node block:** The previous-node block can be used to specify that the node should only be triggered when the last triggered node was a specific one. That is a simple way to handle, for instance, responses to yes/no questions without requiring the use of contexts, or more complicated schemes.
- **Default condition block:** There is a special block that can be used to specify nodes that will be triggered by default when no other node is triggered. This block can be used to specify default responses. Another way is to simply rely on the ordering of the blocks.

4.1.3 Action blocks

- **Say block:** The most basic action block is the *say* block. In the case of the prototype authoring platform, it will simply display the text in a speech bubble above the virtual agent's head.
- **Raise-event block:** Raises a specific custom event. It is intended to be captured by the external system the CA is integrated into. The actual purpose of this will depend upon the external system, which can provide specific events. Those can, for example, enable teachers to propose custom exercises through the CA system.

4.1.4 Node ordering

Each node is ordered vertically, and the order is explicitly shown as a number. This defines the priority of a specific node. Users can rely on ordering to trigger different nodes depending on whether the higher-priority nodes conditions are met or not. Thus, when two different nodes have conditions that partially overlap (for example, one node is to be triggered when the user says "who are you" and the other when the user says something with "you"), the less restricting ones will normally be lower-priority and act as context-specific defaults.

4.2 Strengths and weaknesses

The main strengths of the proposed approach are the following:

- **Simplicity**: Non-programmers can learn and use it easily.
- **Fine-grained control**: Agent authors can control exactly what the bot says and when.
- **Power**: Despite its simplicity, Blockly can support a fully-fledged programming language. With extended blocks, it can provide as much power as one. Alternative approaches that are oriented for non-programmers such as example-tracing tend to be less powerful.
- **Integration**: The approach can be integrated easily into other systems. In more automated systems, or in systems on which the author does not have a fine-grained control over the output, it can be less obvious how to interact with other systems reliably.

Likewise, it makes it suitable for controlling a virtual agent, and thus the behavior of an ECA.
- **Predictability**: Authors can easily predict the exact output of the bot under given conditions. Thus, for a given question, they can know exactly if the bot will be able to answer, and how; avoiding surprises and indeterminacy.
- **No data or training required**: The system does not rely on corpora, datasets, or training.
- **Language-agnostic**: While systems that rely on NLP techniques and data corpora tend to be tailored towards a specific language (such as English), the described approach, without specialized blocks, is language-agnostic.

And the main weaknesses would be the following:
- **Limited scope**: This approach is not suitable for the creation of agents with a large body of knowledge. Too many nodes would be time-consuming to create manually.
- **Realistic language**: Without advanced NLP blocks, it is not suitable for generating realistic conversations. For this purpose, the intended predictability is also a disadvantage: being unpredictable would add realism.
- **No automation**: No automatic learning or information extraction functionalities are present.

As a consequence of this, these agents are suitable for these cases in which the authors want to obtain a narrow-scope agent, in a context in which they can predict the input, and want fine-grained control over the results. This is the case of several educational contexts, such as a remote lab, in which the range of possible questions and interactions is limited and can be predicted by the domain expert (the teacher) and in which the said expert does not need (or even want) the students to have broad-scope conversations with the agent. This is also an advantage for these contexts in which authors may want to use different languages, because the approach is language-agnostic. It is also important to remark that the approach is intended to lead to embeddable agents, not stand-alone ones (for which realistic conversations would probably be more important).

5 Platform Architecture

The architecture is fully web-based and it relies on certain key technologies to provide the required features while maintaining platform independency. This section will describe the proposed architecture, which has been designed, developed and evaluated. To better understand how the authoring platform looks and works in practice, a short screencast is supplied as an online Multimedia Material ('authoring.mp4').

To describe the architecture it is important to remark the two different perspectives:
- The **authoring tool**: For the authors, who normally will be teachers and domain experts.
- The **conversational agent component**: Which will be integrated into an external system, such as an LMS or a remote lab, and implement the agent defined by the author.

5.1 Key technologies

5.1.1 Google Blockly

Google Blockly[4] [12], which was briefly described earlier, is an Open Source visual programming language created by Google. It is originally intended to teach basic programming to young students. They can first learn basic programming concepts, such as logic, variables and loops in Blockly, and once they understand them they can move to the normally less-intuitive text-based languages.

Google Blockly is designed to be customized and modified easily. Completely new blocks and code generators can be developed, so it is well-suited to create visual domain-specific programming languages. In this work, we have designed and developed such a language so that non-programming users can very easily be able to define the logic of their CAs, which will be described in the next sections.

5.1.2 Unity and WebGL

Unity3D[5] is a very popular 3D application creation tool. The 3D applications created in Unity can be exported to different platforms. One of those are browsers, through WebGL [60]. WebGL is a standard by the Chronos group: a web-based graphical API that makes accelerated 3D graphics available on the Web. Though it is not part of the HTML5 standard itself, it has strong ties to it and is supported by every major browser, including mobile ones. Although SVG or Canvas can also be used for web-based graphics, WebGL is more powerful; it supports full 3D acceleration, shaders and relatively low-level access to the graphics card.

Thus, using Unity and WebGL for the CAs we can get agents that:
- Are fully 3D and can rely on advanced shaders and animation and be in a virtual environment.
- Are fully web-based and cross-platform, being deployable even in mobile and tablet devices.

5.2 Authoring Tool

As described in earlier sections, the goal of the authoring tool is to enable users, including those without programming or IT knowledge, to create and customize their own CA relying on a visual domain-specific language that has been designed and implemented for that purpose. A screen capture of the main view of the Authoring Tool can be observed in Figure 4. An overview of the architecture of the authoring system is depicted in Figure 5.

4. https://developers.google.com/blockly/
5. https://unity3d.com

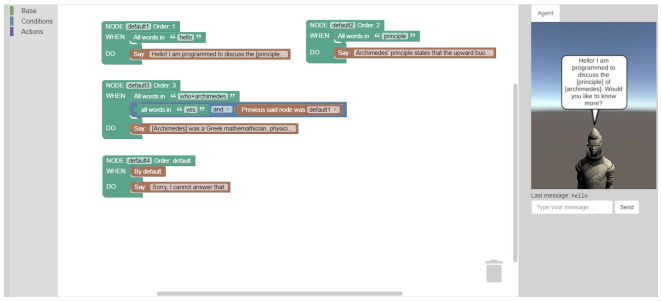

Fig. 4. Authoring Tool's main view

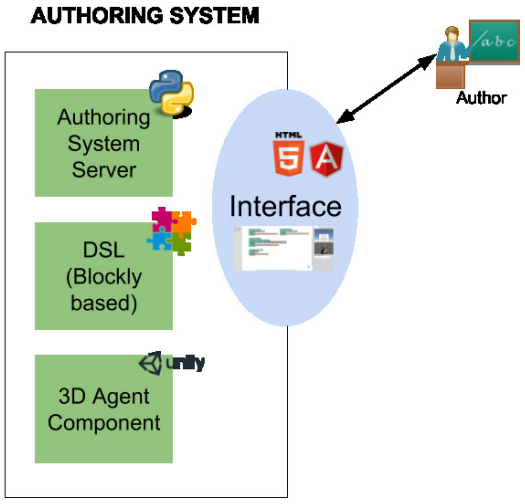

Fig. 5. Authoring system architecture overview.

Several components form the authoring tool. The authors —who will generally be teachers and/or domain experts— interact with it through the *Interface*, which is a web-application and which is thus supported in all the main browsers and in tablets. The authoring tool web-app as a whole is managed by the *Authoring System Server*, which is, essentially, the server-side. The teachers use the *Google Blockly based visual DSL* to create and customize the agents (thus defining the domain model). This component is also responsible for generating the JSON specification (see Figure 6) from the visual code, and forwarding it to either the server (for storage) or to the 3D agent component. The *3D agent component* within the Authoring System is there so that the authors can test their agents in real-time and check whether they function as they expect.

5.3 End-user perspective

From the perspective of the platform described in this work, there are two different kind of users.

```
{
    "formatVersion": 2,
    "conversationNodes": [

        {
            "node": "default1",
            "order": 1,
            "on": [
              "hello",
              "hi"
            ],
            "do": [
              { "say": "Hello, I can explain to you the basic [electrical c
            ]
        },

        {
            "node": "default2",
            "order": 2,
            "on": [
              "electrical components",
              { "$and": ["yes", { "last_node": "default1" }] }
            ],
            "do": [
              { "say": "You can use [resistors], [capacitors] or [diodes]."
            ]
        },

        {
            "node": "default4",
            "order": 3,
            "on": [
              "bye",
              "goodbye"
            ],
            "do": [
              { "say": "You can use [resistors], [capacitors] or [diodes]."
            ]
        },

```

Fig. 6. Example of the advanced JSON view for an agent. This is transparent to the user and normally not visible.

- **Authors**: They will use the authoring tool to create and customize conversational agents. They are not expected to have programming or IT knowledge, but can be considered domain-experts.
- **End-users**: They will use the CAs created and customized by the author-users. They will not necessarily have programming or IT knowledge either.

The CAs created through this platform are purposefully simplistic, and are not meant to be used as standalone intelligent tutors. Instead, they are integrable and add value to external educational systems, such as LMSs or remote labs. The end-user perspective is summarized in Figure 7 and an example can be observed in Figure 1. The end-users interact with an external system, which could be for instance a web-based remote lab. The 3D agent component would then be displayed as an *iframe* within that remote lab. The remote lab would forward interactions to the component, which implements the agent previously defined by an author-user. The interactions are forwarded through a bidirectional JavaScript API based on *window.postMessage*, which will be described in more detail in later sections. The purpose of this API is to guarantee that the agent component can be integrated into other systems, including those that are hosted in a different domain than the actual tool.

5.4 Components

This section describes in a lower-level detail the key components of the platform.

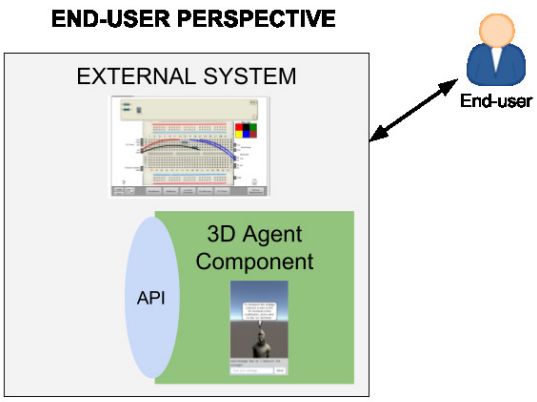

Fig. 7. The end-user perspective.

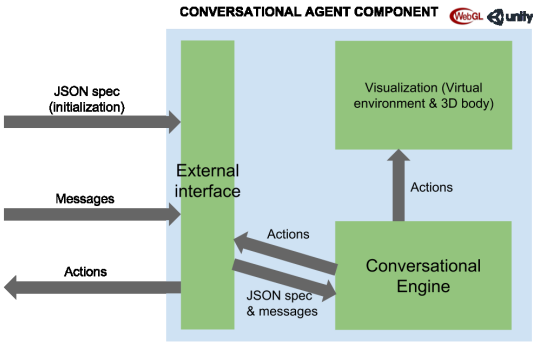

Fig. 8. The Conversational Agent component architecture.

5.4.1 Google Blockly based DSL

Throughout this work we have developed a Blockly-based DSL based on rules to enable the users (non-programmers) to easily define the behavior of the agent. The language relies on a set of custom blocks that internally generate JSON code (transparently to the users, unless they choose otherwise). Then, the CA engine interprets that JSON code to behave in the way that the non-programmer user defined.

The language for this implementation, which is meant to act as an example implementation for the model, is purposefully designed to be very simplistic. It is based on trigger-nodes. Users specify the *conditions* under which the node will be triggered, and the *actions* that will take place once the node is triggered. The most common and relevant action is to *say* a specific sentence, though the system itself supports other actions.

5.4.2 Integrable 3D tutor component

The 3D tutor component itself, as previously described, is created in Unity3D and exported to WebGL (JavaScript). Its architecture is depicted in Figure 8. This component contains the conversation engine that receives the JSON DSL —which is essentially the knowledge model— and the input from the user, and chooses the appropriate actions. It also provides a virtual 3D environment with a 3D body for the tutor, which can speak according to the engine and carry out other actions.

5.4.3 Integration

The architecture described in this work is meant to be integrable into other systems such as Learning Management Systems, virtual laboratories and remote laboratories. For that purpose, as described in Section 2, the 3D tutor component has been created in Unity3D to be exported into WebGL and integrated anywhere as an HTML5 *iframe*. It contains the CA engine that interprets the provided JSON, receives input from the user, and runs the specified actions when appropriate. It communicates through the JavaScript *window.postMessage* API instead of communicating through JavaScript directly so that it can avoid cross-domain issues that would arise when hosting the 3D tutor component into a different domain's *iframe*.

6 Methodology

To evaluate the proposed approach and to explore the research questions that were specified in Section 1.1, user tests were conducted. For this, the following software platforms were created:

- The prototype web-based *authoring platform* whose architecture was described in Section 5.
- A web-based *testing platform* to lead the study participants through several predefined stages, during which they use the authoring platform, and data is collected.

In the final stage, participants were asked to fill a standardized UMUX questionnaire (that measures usability), and a custom survey to evaluate their perception of the authoring platform and of the proposed approach.

6.1 The testing platform

A web platform was created to lead participants through the stages depicted in Figure 9. First, in the *example* stage, they see and use a resulting CA for the first time. Second, in the *tutorial* stage, they learn how to use the authoring tools through an interactive tutorial. Third, in the *challenge* stage, they are given a topic and several constraints and are asked to create a CA. Fourth, in the *survey* stage, they provide feedback about their experience.

A short screencast is provided as an online Multimedia Material ('experiment.mp4'), briefly demonstrating the testing platform and how users go through its various stages. It may be noteworthy that in the video the stages are completed very fast: real first-time users will need to read through the text, make more errors, and take much more time (as the time measurements in Section 7 show).

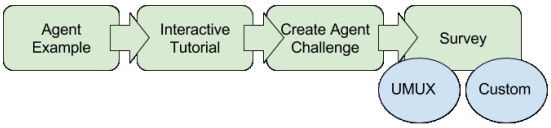

Fig. 9. Experiment stages that the participants follow

6.2 Stages

6.2.1 Example stage

The goal of this stage is to familiarize the participant with the CAs. This specific CA speaks about the laws of thermodynamics. However, the content itself is not important, because in practice it will depend on each agent's author. To ensure that the participant uses the agent sufficiently, we measure the different agent responses that are triggered. Once 5 different responses are triggered, the participants may continue to the next step.

6.2.2 Tutorial stage

The goal of this stage is to teach the test subject to create CAs using the prototype authoring tools and the visual DSL. This is challenging, because even though the visual DSL is designed to be intuitive, time is constrained and concepts such as CAs, computer languages, or logical conditions are novel to most participants. To be effective, we designed the tutorial to be interactive. Users are given examples at all times, and they practice what they learn. The system automatically evaluates whether what they are doing is right or wrong, provides tips, and suggests the next step to take. Users cannot skip to the next step until their current step works as intended.

6.2.3 Challenge stage

In the challenge stage, participants apply what they have learned to create a CA. We provide a topic —in this case, a museum guide— and several constraints. This way, we ensure that the resulting data is comparable and that they use several types of visual blocks, including complex ones. In this stage they can still see a 'cheatsheet' with examples, but they receive no guiding or tips. They only receive feedback about whether the constraints are currently met or not. A screenshot of this stage can be seen in Figure 10.

6.2.4 Survey stage

In this last stage the participants fill a 9-question survey, which is actually split in two different parts. The first part is a standardized UMUX (Usability Metric for User Experience) survey [61], [62]. UMUX is a four-items Likert scale to assess an application's perceived usability. It is designed to measure the three dimensions of usability defined by the ISO 9241-11 standard [63] (effectiveness, efficiency and satisfaction) and to provide similar results to the longer ten-items System Usability Scale (SUS) [64],

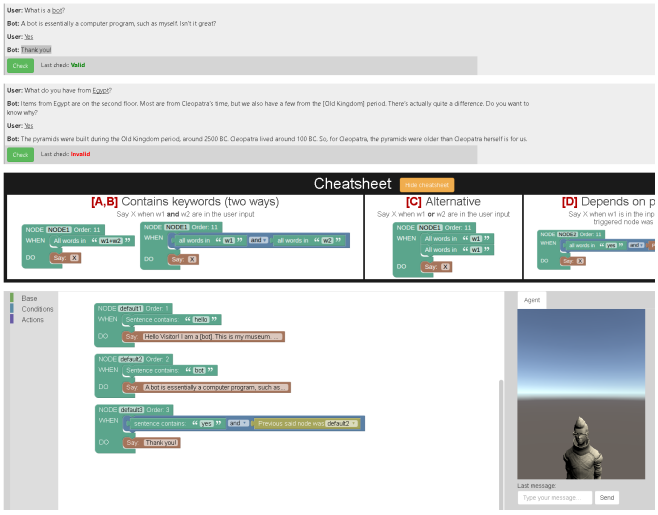

Fig. 10. Screenshot of the Challenge stage of the experiment

[65], but with less questions. Through this questionnaire the usability of the authoring platform prototype and the visual DSL that it relies on is measured. The second part has six questions, which are more specific, and are designed to evaluate the perception of the participants and their satisfaction with the approach and the authoring tools. All participants were Spanish-speaking, so the original questionnaires were in Spanish. The translated survey questions are detailed in Table 1.

6.3 Participants

32 first-year students from the University of [removed for blind review], in Spain, took part in the study. 14 of them were from the campus in [removed for blind review] and were enrolled in a Business Administration degree. 18 of them were from the campus in [removed for blind review] and were enrolled in an Electronics Engineering degree. Participation was voluntary. Although currently the main intended audience of the authoring platform are teachers, the students played the role of content creators. This way it was possible to obtain a larger number of participants, and the conclusions may be extended to different audiences. The students were non-programmers but all were familiar with technology such as graphical interfaces and web browsing. All were native Spanish speakers.

6.4 Procedure

Two group sessions were held in total, one for each campus. The first one had 14 of the participants, while the second had 18. Each student had access to a computer. An hour of time was allocated for each session. During the session, students were briefly introduced to the topic of CAs and the purpose of the authoring platform. Then,

TABLE 1
UMUX and custom questionnaires

Standard UMUX questionnaire	
Q1	*The Authoring Tool's capabilities meet my requirements* 7 point Likert-type scale (1 to 7), from *Strongly Disagree* to *Strongly Agree*
Q2	*Using the Authoring Tool is a frustrating experience* 7 point Likert-type scale (1 to 7), from *Strongly Disagree* to *Strongly Agree*
Q3	*The Authoring Tool is easy to use* 7 point Likert-type scale (1 to 7), from *Strongly Disagree* to *Strongly Agree*
Q4	*I have to spend too much time correcting things with the Authoring Tool* 7 point Likert-type scale (1 to 7), from *Strongly Disagree* to *Strongly Agree*
Custom questionnaire	
Q5	*Non-programmers would be capable of creating bots with a Visual Language such as the one we used* 7 point Likert-type scale (1 to 7) from *No teacher could do it* to *Yes, everyone could do it*
Q6	*It would be useful to integrate bots into other educational content (such as online courses; to support the student)* 7 point Likert-type scale (1 to 7), from *Strongly Disagree* to *Strongly Agree*
Q7	*It can be useful to integrate bots into some educational systems (as support for online courses; or virtual or remote labs)* 7 point Likert-type scale (1 to 7), from *Strongly Disagree* to *Strongly Agree*
Q8	*I found technical problems during the experiment* 7 point Likert-type scale (1 to 7), from *Strongly Disagree* to *Strongly Agree*
Q9	*Please, make any observation, comment or complaint* Free text response

they were directed to the tutorial platform and provided some general advice (such as making sure to follow the instructions at each stage very precisely, because the system is automated and they would be unable to advance otherwise). The participants then went through each of the stages, directed by the testing platform, and at their own pace. The interactive tutorial of the platform is designed to be self-sufficient, but the participants were allowed to ask questions and they received directions when they got stuck in a particular step or came across a potential technical issue.

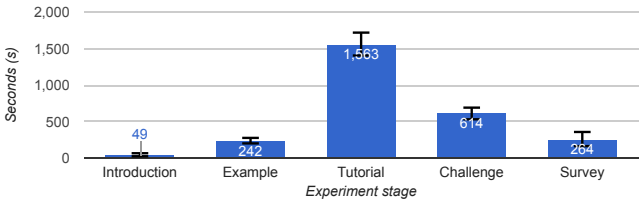

Fig. 11. Average time in seconds that participants spent in each stage (n=32). Error bars show 95% confidence intervals.

7 Results

The results of this research are organized around the Research Questions enumerated in Section 1.1.

7.1 How easy to learn is the proposed approach?

The testing platform measured the time that the participants spent in each stage. The average times in seconds are shown in Figure 11. The error bars show the 95% ($\alpha = 0.05$) confidence intervals. The sum of the average times for each stage is 2732 seconds for all stages (45.5 minutes).

The tutorial stage, as expected, took the longest: 1563 seconds (26.05 minutes) on average. The fastest participant took 1112 seconds (18.5 minutes) while the slowest took 2920 seconds (48.67 minutes). The challenge stage took 614 seconds (10.23 minutes) on average. The fastest participant took 293 seconds (4.88 minutes) while the slowest took 1183 seconds (19.71 minutes).

7.2 How usable is the proposed approach?

As previously described, the evaluation of the authoring platform usability has relied on a conventional UMUX questionnaire. As [61] explain, odd items are scored as $[score - 1]$ and even items are scored as $[7 - score]$. The preliminary UMUX score is thus in the 0-24 range. The sum for each participant is then divided by 24 and multiplied by 100 to convert it to the 0-100 SUS-like scale. The mean score is then calculated.

The UMUX mean score for the 32 participants is 73.31. The standard deviation is 15.62, with a 95% ($\alpha = 0.05$) confidence interval half-width of 5.41 ($[67.89 - 78.72]$). According to [65], this score is in the *acceptable* range and can be considered "good", though not "excellent". However, considering that the platform is a research prototype, and that the participants had a short time to learn to use the relatively powerful visual language, the result can be considered satisfactory.

TABLE 2
Results of the custom survey Q5-Q8 (n=32)

	Question (1-7 points Likert-type scale)	Mean	S.D.
Q5	*Non-programmers would be capable of creating bots with a Visual Language such as the one we used*	5.78	1.008
Q6	*It may be useful to integrate bots into some educational systems [...]*	6.00	0.916
Q7	*The visual block-based language to program the bots is intuitive*	5.94	1.076
Q8	*I have come across technical issues during the experiment*	3.62	2.196

7.3 Is the proposed approach perceived as valuable and intuitive?

The second part of the survey consisted of 5 questions. The first four, for consistency, use a Likert-scale with 7 points. However, unlike the UMUX ones, they are specific and are considered independently rather than as a metric. The last question is an invitation to freely comment anything. The results of this questionnaire are summarized in Table 2. They show that users overwhelmingly believe that everyone would be capable of creating bots with such a visual language ($\bar{x} = 5.78$), that it would be useful to integrate bots into educational content ($\bar{x} = 6.00$), and that the proposed visual DSL is intuitive ($\bar{x} = 5.94$).

Few participants chose to respond to the free-comments question (Q9), but all those that did were remarkably positive, with comments (translated) such as:

- "It is extremely easy to use. Very intuitive and simple."
- "It's a very good idea and I hope it moves forward!"
- "I would remark how intuitive it is"
- "Very good"

8 Discussion

This work has proposed an approach to create and customize CAs using a visual DSL that is friendly for non-programmers. An authoring platform has been created to showcase the approach and to study its effectiveness, and a user study has been conducted.

8.1 The proposed approach

The experience and the results suggest that, indeed, the approach can be a useful way to define CAs, especially for non-programmers. This does not necessarily mean that it is the best approach for all cases, but it may be the most appropriate for some of them. Alternative approaches such as using a text-based formal language (such as AIML or XML) may yield more flexibility, but in exchange can generally be expected to be harder to learn. On the contrary, approaches such as example-tracing may be easier to learn than the proposed visual DSL, but are (at least if not combined with other approaches) significantly less powerful. Thus, using a visual DSL can be considered a compromise between simplicity and power that may be very appropriate for some domains.

An additional advantage of this approach is that its capabilities and complexity can be tailored freely. In such a visual language, those depend on the *blocks* that are provided by the authoring platform. Thus, the platform can offer blocks that are more or less complex depending on the needs and audience, and advances in related areas could be leveraged easily. For instance, some advanced functionalities that may be integrated easily into the blocks system would be:

- Text-to-speech: Integrated into the *say* block, using an API such as Google's.
- Speech-to-text: Integrated into the *keywords-recognition* block, using an API such as Google's[6], and automatically providing the API with clues about the expected words to increase accuracy.
- Blocks to control the behavior of the 3D agent to leverage advances in Embodied Conversational Agent (ECA) research.
- Blocks to raise evaluation events so that the teacher can create, propose and evaluate customized exercises for online laboratories without requiring collaboration from the author of the lab.

8.2 Authoring platform prototype

In general the results have been satisfactory. Usability, according to the UMUX questionnaire and the previously mentioned thresholds, can be considered good enough, though there is still some room for improvement. The perception that users had of the platform was very positive (and generally, higher than their usability perception).

From a technical perspective, the technology choices were appropriate. Google Blockly was chosen as a base for the visual DSL, and it has met the requirements. It has been stable, easy to extend and intuitive enough for the participants to learn to use it in a very short time and with very little help. Similarly, the choice of Unity and WebGL as a rendering engine to implement the CA's interpreter engine and ECA has been satisfactory. It allows it to be web-based and can be run, as was the goal, in almost any browser, and thus meets the universality, security and deployability goals that we had set in section 2.

Though exploring it in detail was not the focus of this work, it is also noteworthy that the interactive tutorial could be improved. Although it was effective (participants were able to learn how to create agents in a short time) some participants found it frustrating. Not allowing users to skip to the next step until they have done the current one properly is an effective means to guarantee that the learning goals are achieved, but at the same time can be found frustrating if they are unable to

6. https://cloud.google.com/speech/

find the mistake in a short time, or if they don't receive appropriate feedback. Ways to improve this issue could be explored.

9 Conclusions

To create effective educational CAs, domain knowledge is required. This work has proposed a novel method to define such agents that is both simple —accessible to non-programmers— and expressive —capable of supporting complex rules and being extended with additional *blocks*—. It has also described an authoring platform that leverages that method to allow non-programmers to define their own agents. These agents are embeddable into external educational content, without requiring explicit collaboration from the content's creator.

The results of the study suggest that a visual DSL based on Blockly is indeed a promising way to define CAs for non-programmers. Even though the platform is still a prototype, the participants of the study were able to learn to use them fast, and created their own CA. Additionally, the standard UMUX usability test suggests that the authoring tools prototype and the visual DSL that it relies on have satisfactory usability already. With additional work it can be expected to produce better results. The specific survey, as well, shows that participants are very satisfied with their use of the tool, and believe that it has significant potential. The approach itself (using a visual DSL to allow non-programmers to create or customize CAs) seems promising. Although the visual DSL is expressive, non-programmers can learn and understand it in a short time. An additional advantage of this approach is that the visual language could be extended with custom blocks that provide additional capabilities and that leverage advances in related areas.

10 Future work

In the future certain improvements and modifications to the CA model could be investigated:

- The language processing system is currently very simplistic and based on keywords. It might be worthwhile to find ways to apply more advanced NLP techniques to the CA engine, and to check whether the new capabilities are useful enough, taking into account the potential cost in simplicity.
- The set of potential actions of the engine is currently limited. It might be worthwhile to add new synchronized animations or 3D interactions, relying on the existing literature about embodied CAs.

Apart from those lines of research that mainly involve improvements to the CA model and engine, certain potential applications that relate to online laboratories and that could be explored are worth mentioning:

- As a teacher-customizable intelligent tutor for an online lab, which may guide the student and provide specific advice on its usage, expectations and results.
- Additionally, as a teacher-customizable intelligent tutor which also offers automatic evaluation capabilities. Thus, the teachers could integrate a tutor into the online lab of their choice, design the problem description and the expected output, and have the intelligent agent automatically evaluate the students. This would be particularly useful because virtual and remote labs often just offer an open environment but not a customized practice session or experience, and because in distance education the teacher often has to evaluate a large number of students, and tools which make this easier can have a significant impact.

Acknowledgments

This work has received financial support by the Department of Education, Language policy and Culture of the Basque Government through a Predoctoral Scholarship granted to Luis Rodriguez-Gil.

References

[1] B. D. Nye, A. C. Graesser, and X. Hu, "Autotutor and family: A review of 17 years of natural language tutoring," *International Journal of Artificial Intelligence in Education*, vol. 24, no. 4, pp. 427–469, 2014.

[2] J. Cassell, T. Bickmore, M. Billinghurst, L. Campbell, K. Chang, H. Vilhjálmsson, and H. Yan, "Embodiment in conversational interfaces: Rea," in *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*. ACM, 1999, pp. 520–527.

[3] K. Vanlehn, "The behavior of tutoring systems," *International journal of artificial intelligence in education*, vol. 16, no. 3, pp. 227–265, 2006.

[4] J. Weizenbaum, "ELIZA - a computer program for the study of natural language communication between man and machine," *Communications of the ACM*, vol. 9, no. 1, pp. 36–45, 1966.

[5] S. Kopp, L. Gesellensetter, N. C. Krämer, and I. Wachsmuth, "A conversational agent as museum guide–design and evaluation of a real-world application," in *International Workshop on Intelligent Virtual Agents*. Springer, 2005, pp. 329–343.

[6] T. W. Bickmore, D. Utami, R. Matsuyama, and M. K. Paasche-Orlow, "Improving access to online health information with conversational agents: a randomized controlled experiment," *Journal of medical Internet research*, vol. 18, no. 1, 2016.

[7] V. Aleven, J. Sewall, O. Popescu, M. van Velsen, S. Demi, and B. Leber, "Reflecting on twelve years of ITS authoring tools research with CTAT," in *Design recommendations for adaptive intelligent tutoring systems*. Orlando: US Army Research Laboratory, 2015, pp. 263–283.

[8] J. K. Olsen, D. M. Belenky, V. Aleven, N. Rummel, J. Sewall, and M. Ringenberg, "Authoring tools for collaborative intelligent tutoring system environments," in *International Conference on Intelligent Tutoring Systems*. Springer, 2014, pp. 523–528.

[9] A. Kerry, R. Ellis, and S. Bull, "Conversational agents in e-learning," in *Applications and Innovations in Intelligent Systems XVI*. Springer, 2009, pp. 169–182.

[10] V. L. Rubin, Y. Chen, and L. M. Thorimbert, "Artificially intelligent conversational agents in libraries," *Library Hi Tech*, vol. 28, no. 4, pp. 496–522, 2010.

[11] V. Aleven, B. M. Mclaren, J. Sewall, and K. R. Koedinger, "A new paradigm for intelligent tutoring systems: Example-tracing tutors," *International Journal of Artificial Intelligence in Education*, vol. 19, no. 2, pp. 105–154, 2009.

[12] R. Doe. (2017) Google Blockly - A visual programming editor. http://code.google.com/p/blockly. Accessed Jan 2017.

[13] J. Trower and J. Gray, "Blockly language creation and applications: Visual programming for media computation and bluetooth robotics control," in *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*. ACM, 2015, pp. 5–5.

[14] L. Pappano, "The year of the MOOC," *The New York Times*, no. 12, 2012.

[15] J. Ma and J. V. Nickerson, "Hands-on, simulated, and remote laboratories: A comparative literature review," *ACM Computing Surveys (CSUR)*, vol. 38, no. 3, p. 7, 2006.

[16] S. D. Bencomo, "Control learning: Present and future," *Annual Reviews in control*, vol. 28, no. 1, pp. 115–136, 2004.

[17] J. E. Froyd, P. C. Wankat, and K. A. Smith, "Five major shifts in 100 years of engineering education," *Proceedings of the IEEE*, vol. 100, no. Special Centennial Issue, pp. 1344–1360, 2012.

[18] T. de Jong, S. Sotiriou, and D. Gillet, "Innovations in STEM education: the Go-Lab federation of online labs," *Smart Learning Environments*, vol. 1, no. 1, p. 3, 2014.

[19] M. J. Rodríguez-Triana, S. Govaerts, W. Halimi, A. Holzer, C. Salzmann, A. Vozniuk, T. de Jong, S. Sotirou, and D. Gillet, "Rich open educational resources for personal and inquiry learning: Agile creation, sharing and reuse in educational social media platforms," in *Web and Open Access to Learning (ICWOAL), 2014 International Conference on*. IEEE, 2014, pp. 1–6.

[20] D. Gillet, T. De Jong, S. Sotirou, and C. Salzmann, "Personalised learning spaces and federated online labs for STEM education at school," in *Global Engineering Education Conference (EDUCON), 2013 IEEE*. IEEE, 2013, pp. 769–773.

[21] S. Govaerts, Y. Cao, A. Vozniuk, A. Holzer, D. G. Zutin, E. S. C. Ruiz, L. Bollen, S. Manske, N. Faltin, C. Salzmann *et al.*, "Towards an online lab portal for inquiry-based STEM learning at school," in *International Conference on Web-Based Learning*. Springer, 2013, pp. 244–253.

[22] L. Rodriguez-Gil, M. Latorre, P. Orduña, A. Robles-Gómez, E. Sancristobal, S. Govaerts, D. Gillet, I. Lequerica, A. C. Caminero, R. Hernandez *et al.*, "OpenSocial application builder and customizer for school teachers," in *Advanced Learning Technologies (ICALT), 2014 IEEE 14th International Conference on*. IEEE, 2014, pp. 31–33.

[23] M. Tawfik, E. Sancristobal, S. Martin, R. Gil, G. Diaz, A. Colmenar, J. Peire, M. Castro, K. Nilsson, J. Zackrisson *et al.*, "Virtual instrument systems in reality (visir) for remote wiring and measurement of electronic circuits on breadboard," *IEEE Transactions on learning technologies*, vol. 6, no. 1, pp. 60–72, 2013.

[24] J. Lester, B. Mott, J. Rowe, and R. Taylor, "Principles for pedagogical agent authoring tools," in *Design recommendations for intelligent tutoring systems: Authoring tools and expert modeling techniques*. Robert Sottilare, 2015, p. 151.

[25] J. K. Olsen, D. M. Belenky, V. Aleven, and N. Rummel, "Intelligent tutoring systems for collaborative learning: Enhancements to authoring tools," in *International Conference on Artificial Intelligence in Education*. Springer, 2013, pp. 900–903.

[26] R. Kumar and C. P. Rose, "Architecture for building conversational agents that support collaborative learning," *IEEE Transactions on Learning Technologies*, vol. 4, no. 1, pp. 21–34, 2011.

[27] C. Ray and S. Gilbert, "Bringing authoring tools for intelligent tutoring systems and serious games closer together: Integrating GIFT with the Unity game engine," in *AIED 2013 Workshops Proceedings*, vol. 7, 2013, p. 37.

[28] N. Matsuda, W. W. Cohen, and K. R. Koedinger, "Teaching the teacher: tutoring simstudent leads to more effective cognitive tutor authoring," *International Journal of Artificial Intelligence in Education*, vol. 25, no. 1, pp. 1–34, 2015.

[29] M. Virvou and E. Alepis, "Mobile educational features in authoring tools for personalised tutoring," *Computers & Education*, vol. 44, no. 1, pp. 53–68, 2005.

[30] V. Aleven, R. Baker, Y. Wang, J. Sewall, and O. Popescu, "Bringing non-programmer authoring of intelligent tutors to MOOCs," in *Proceedings of the Third (2016) ACM Conference on Learning@ Scale*. ACM, 2016, pp. 313–316.

[31] H. C. Lane, M. G. Core, M. J. Hays, D. Auerbach, and M. Rosenberg, "Situated pedagogical authoring: Authoring intelligent tutors from a student's perspective," in *International Conference on Artificial Intelligence in Education*. Springer, 2015, pp. 195–204.

[32] K. R. Koedinger, V. Aleven, N. Heffernan, B. McLaren, and M. Hockenberry, "Opening the door to non-programmers: Authoring intelligent tutor behavior by demonstration," in *International Conference on Intelligent Tutoring Systems*. Springer, 2004, pp. 162–174.

[33] T. Bickmore and L. Ring, "Making it personal: end-user authoring of health narratives delivered by virtual agents," in *International Conference on Intelligent Virtual Agents*. Springer, 2010, pp. 399–405.

[34] V. Aleven, B. M. McLaren, J. Sewall, and K. R. Koedinger, "The cognitive tutor authoring tools (CTAT): preliminary evaluation of efficiency gains," in *Intelligent Tutoring Systems: Proceedings of the 8th International Conference, ITS 2006, Jhongli, Taiwan, June 26-30, 2006*. Springer Berlin Heidelberg, 2006, pp. 61–70.

[35] R. A. Sottilare, K. W. Brawner, B. S. Goldberg, and H. K. Holden, "The generalized intelligent framework for tutoring (GIFT)," *Orlando, FL: US Army Research Laboratory–Human Research & Engineering Directorate (ARL-HRED)*, 2012.

[36] S. B. Blessing, "A programming by demonstration authoring tool for model-tracing tutors," in *Authoring Tools for Advanced Technology Learning Environments*. Springer, 2003, pp. 93–119.

[37] V. Aleven, B. M. McLaren, J. Sewall, M. van Velsen, O. Popescu, S. Demi, M. Ringenberg, and K. R. Koedinger, "Example-tracing tutors: intelligent tutor development for non-programmers," *International Journal of Artificial Intelligence in Education*, vol. 26, no. 1, pp. 224–269, 2016.

[38] B. A. Shawar and E. Atwell, "A chatbot as a novel corpus visualization tool." in *LREC*. Citeseer, 2004.

[39] J. Feng, S. Bangalore, and M. Rahim, "Webtalk: Mining websites for automatically building dialog systems," in *Automatic Speech Recognition and Understanding, 2003. ASRU'03. 2003 IEEE Workshop on*. IEEE, 2003, pp. 168–173.

[40] J. Maloney, M. Resnick, N. Rusk, B. Silverman, and E. Eastmond, "The Scratch programming language and environment," *ACM Transactions on Computing Education (TOCE)*, vol. 10, no. 4, p. 16, 2010.

[41] G. W. Johnson, *LabVIEW graphical programming*. Tata McGraw-Hill Education, 1997.

[42] J.-M. Sáez-López, M. Román-González, and E. Vázquez-Cano, "Visual programming languages integrated across the curriculum in elementary school: A two year case study using "Scratch" in five schools," *Computers & Education*, vol. 97, pp. 129–141, 2016.

[43] W.-H. Kuan, C.-H. Tseng, S. Chen, and C.-C. Wong, "Development of a computer-assisted instrumentation curriculum for physics students: Using LabVIEW and Arduino platform," *Journal of Science Education and Technology*, vol. 25, no. 3, pp. 427–438, 2016.

[44] F. Kalelioğlu, "A new way of teaching programming skills to K-12 students: Code.org," *Computers in Human Behavior*, vol. 52, pp. 200–210, 2015.

[45] D. Kumar, "Digital playgrounds for early computing education," *ACM Inroads*, vol. 5, no. 1, pp. 20–21, 2014.

[46] I. Iturrate, G. Martín, J. García-Zubia, I. Angulo, O. Dziabenko, P. Orduña, G. Alves, and A. Fidalgo, "A mobile robot platform for open learning based on serious games and remote laboratories,"

in *Engineering Education (CISPEE), 2013 1st International Conference of the Portuguese Society for*. IEEE, 2013, pp. 1–7.

[47] M. Á. Serna, C. J. Sreenan, and S. Fedor, "A visual programming framework for wireless sensor networks in smart home applications," in *Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2015 IEEE Tenth International Conference on*. IEEE, 2015, pp. 1–6.

[48] J. Wiriyakul and T. Senivongse, "A visual editor for language-independent scripting for bpmn modeling," in *Computer Science and Software Engineering (JCSSE), 2015 12th International Joint Conference on*. IEEE, 2015, pp. 156–161.

[49] C. E. Wieman, W. K. Adams, and K. K. Perkins, "PhET: Simulations that enhance learning," *Science*, vol. 322, no. 5902, pp. 682–683, 2008.

[50] M. Kaluz, J. Garcia-Zubia, M. Fikar, and L. Čirka, "A flexible and configurable architecture for automatic control remote laboratories," *IEEE Transactions on Learning Technologies*, vol. 8, no. 3, pp. 299–310, 2015.

[51] H. Crompton, D. Burke, K. H. Gregory, and C. Gräbe, "The use of mobile learning in science: a systematic review," *Journal of Science Education and Technology*, vol. 25, no. 2, pp. 149–160, 2016.

[52] L. J. Couse and D. W. Chen, "A tablet computer for young children? Exploring its viability for early childhood education," *Journal of Research on Technology in Education*, vol. 43, pp. 75–96, 2010.

[53] S. N. Şad and Ö. Göktaş, "Preservice teachers' perceptions about using mobile phones and laptops in education as mobile learning tools," *British Journal of Educational Technology*, vol. 45, no. 4, pp. 606–618, 2014.

[54] D. G. de la Iglesia, J. F. Calderón, D. Weyns, M. Milrad, and M. Nussbaum, "A self-adaptive multi-agent system approach for collaborative mobile learning," *IEEE Transactions on Learning Technologies*, vol. 8, no. 2, pp. 158–172, 2015.

[55] McAfee, "McAfee Labs Threats Report, May 2015," Intel Security, Tech. Rep., May 2015.

[56] J. Chacon, H. Vargas, G. Farias, J. Sanchez, and S. Dormido, "Ejs, jil and labview: How to build a remote lab in the blink of an eye," *IEEE Trans. on Learning Technologies*, vol. 8, no. 4, pp. 393–401, 2015.

[57] D. Lowe, S. Murray, E. Lindsay, and D. Liu, "Evolving remote laboratory architectures to leverage emerging internet technologies," *IEEE Transactions on learning technologies*, vol. 2, no. 4, pp. 289–294, 2009.

[58] L. Rodriguez-Gil, J. Garcia-Zubia, P. Orduna, and D. Lopez-de Ipina, "Towards new multiplatform hybrid online laboratory models," *IEEE Transactions on Learning Technologies*, 2016.

[59] J. Cassell, "Embodied conversational interface agents," *Communications of the ACM*, vol. 43, no. 4, pp. 70–78, 2000.

[60] C. Marrin, "Webgl specification," *Khronos WebGL Working Group*, 2011.

[61] K. Finstad, "The usability metric for user experience," *Interacting with Computers*, vol. 22, no. 5, pp. 323–327, 2010.

[62] M. I. Berkman and D. Karahoca, "Re-assessing the usability metric for user experience (UMUX) scale," *Journal of Usability Studies*, vol. 11, no. 3, pp. 89–109, 2016.

[63] ISO, *ISO 9241-11:1998 Ergonomic Requirements for Office Work with Visual Display Terminals (VDTs).: Guidance on Usability*. International Organization for Standardization, 1998.

[64] J. Brooke *et al.*, "SUS-A quick and dirty usability scale," *Usability evaluation in industry*, vol. 189, no. 194, pp. 4–7, 1996.

[65] A. Bangor, P. T. Kortum, and J. T. Miller, "An empirical evaluation of the system usability scale," *Intl. Journal of Human–Computer Interaction*, vol. 24, no. 6, pp. 574–594, 2008.

Luis Rodríguez-Gil is a PhD student at DeustoTech Internet group. He finished his studies of a double degree in Computer Eng. and Industrial Org. Eng. in 2013, and he completed a MSc in Information Security in 2014. Since 2009, he has been involved in the WebLab-Deusto Research Group, collaborating in the development of the WebLab-Deusto RLMS. He has published several peer-reviewed publications and contributed to some Open Source projects.

Javier García-Zubia (M'08-SM'11) holds a PhD in Computer Sciences by the University of Deusto. He is a full professor in the Faculty of Engineering of the University of Deusto, Spain. His research interest is focused on remote laboratory design, implementation and evaluation. He is the leader of the WebLab-Deusto research group.

Pablo Orduña (M'05) is a full time researcher and project manager at the MORElab Research Group at DeustoTech Internet. He finished Computer Engineering in 2007 and his PhD in 2013 in the University of Deusto. During his PhD he was a visiting researcher twice for 6 weeks each, in the MIT CECI in 2011 and UNED DIEEC in 2012. Since 2004, he has also been involved in the WebLabDeusto Research Group, leading the design and development of WebLab-Deusto.

Diego López-de-Ipiña is an associate prof. and P.R. of MORElab group and director of DeustoTech Internet unit, and of the PhD program within the Faculty of Eng. of the University of Deusto. He received his PhD from the University of Cambridge in 2002. Responsible for several modules in the BSc and MSc in Comp. Eng. degrees, he is interested in pervasive computing, IoT, semantic service middleware, open linked data and social data mining. He is taking and has taken part in several big consortium-based research european (IES CITIES, MUGGES, SONOPA, CBDP, GO-LAB, LifeWear) and Spanish projects, and has more than 70 publications in relevant int. conf. and journals, including more than 25 JCR-indexed articles.

APPENDIX

D

Paper IV

Hybrid laboratory for rapid prototyping in digital electronics

This paper was published as a book chapter in the book *'Online Experimentation: Emerging Technologies and IoT'*. The book was published by the International Frequency Sensor Association (IFSA) on 30 December 2015. Its ISBN is 978-84-608-5977-2. The full book can be purchased from: http://www.sensorsportal.com/HTML/BOOKSTORE/Online_Experimentation.htm. The author-submitted version is appended here.

HYBRID LABORATORY FOR RAPID PROTOTYPING IN DIGITAL ELECTRONICS

Luis Rodriguez-Gil[*], Javier Garcia Zubia[#], Pablo Orduña[*], Ignacio Angulo[#], Diego López-de-Ipiña[*]
[*] Deusto Institute of Technology, University of Deusto
[#]Faculty of Engineering, University of Deusto

{luis.rodriguezgil, zubia, pablo.orduna, ignacio.angulo, dipina}@deusto.es

Abstract

This chapter describes the integration of an educational electronic design tool in a Remote Laboratory, and the implementation and addition of a Hybrid (virtual and remote) laboratory. The goal of these integrations is to provide an extended educational process, which can improve the teaching and learning of Digital Electronics. The tools and workflow that have been integrated allow students to easily design and implement their own Digital System. Then, they can, in just a few seconds, program that system remotely into a real electronics board and test it. This is done through a Remote Laboratory. Furthermore, the real board can be used to control a virtual model. This allows for a significantly greater variety of possible exercises and for a more immersive and engaging experience. Through the Remote Laboratory, the whole process can be carried out by students in just a few minutes, without requiring them to purchase or setup any equipment.

Keywords

Remote labs, remote laboratories, hybrid laboratories, virtual laboratories, online laboratories, digital electronics, software design tools, electronic design tools

Introduction

Digital Electronics are important in most Engineering Degrees. There are different course structures and approaches to teach them. Most often, they involve learning about digital circuits design. Once students understand the basics, courses often rely on VHDL or other HDLs [1][2]. The course organization also varies, but often it is done mainly through two different courses: an introductory one, and a mostly practical one. The first one would focus on matters such as digital electronics, binary codes, Boole algebra, and the analysis and design of digital systems with integrated circuits. The second one would involve the design of digital systems through the VHDL or even C languages and they use FPGAs and microcontrollers as tools. While the first block mostly takes place in a classroom using pen and paper and simulators, and seeks to provide a general understanding of digital electronics, the second aims to teach the student how to design and implement industrial systems, and takes place in a laboratory, making use of more advanced tools such as Xilinx ISE (or other CAE tools).

Teachers understand that practical work in the laboratory is critical, because only through it can the student truly accept and understand the reality of things. Many works have been published on this matter [3][4]. However, the typical process of teaching-learning is not without issues. This work will focus on two of them. The first is due to the time that elapses between the design and testing stages. Students learn and design in the classroom, at home, or in the library, but need a laboratory for testing. These laboratories need to be reserved in advance, and most often students do not have the chance to test the system straightaway. They need to wait for hours or days. This delay harms the learning process. The second issue is the limitations of the exercises that are typically proposed in academic settings. Though these exercises try to simulate real-life industrial problems, and often use real industrial controllers such as FPGAs, they are generally implemented in practice using simplistic inputs and outputs such as switches and LEDs because real industrial equipment is too expensive to

buy and maintain, and hard to control. Because of this, students often feel that the exercises are not challenging or interesting enough. This work thus aims to help the Digital Electronics teacher in two ways.

First, using the Boole-WebLab-Deusto tool seeks to streamline the teaching-learning process through the use of an electronics design tool and a remote laboratory. Remote laboratories themselves, and FPGA ones in particular, have been used successfully for digital electronics education [5][6]. By integrating the laboratory with the design tool and by removing several client-side requirements and delays we can remove the aforementioned time gap.

It is noteworthy that this work does not suggest to fully replace hands-on laboratories with remote laboratories. Instead, it proposes to use the remote laboratories as a complement to traditional hands-on laboratories. The combination of both is likely to provide the most educational benefits.

The rest of work is organized as follows. Section 2 presents the subject Digital Electronics in the University of Deusto. The Sections 3, 4 and 5 describe the functionality of the proposed tool: WebLab-Boole-Deusto. Sections 6 and 7 are dedicated to describing the technical solution adopted to implement the hybrid laboratory. The paper ends with the Conclusions and Future Work.

Digital Electronics in the University of Deusto and other institutions

Digital Electronics in the University of Deusto is imparted through theory, classroom exercises and practical laboratory work. The stated goals are the following:

- To be able to analyze problems and design digital systems through the techniques of digital electronics.
- To be able to implement a digital system and to measure its signals.

The contents of the subject include the fundamentals of digital electronics and the nature of electronic signals; binary coding and arithmetic; Boole algebra and basic Boolean operators. From here, students move on to analyse (in the classroom), design (in the classroom) and implement (in a laboratory) digital systems.

We can distinguish two kinds of digital systems: combinational (those which have no memory and whose output is simply a combination of its inputs; this is the case of adders, subtractors, multiplexers, comparers, etc.) and sequential (those also known as finite state machines, whose next state depends on the current one). Of each of those, we can also distinguish between bit-level and word-level systems. This work focuses mostly on the former (bit-level systems).

In the Digital Electronics subject, students are taught through the steps involved in the design of an efficient bit-level digital system. The steps for a combinational system are, in short, the following (see also Figure 1):

1. Understand the “problem statement” or requirements provided by the teacher.
2. Specify the **truth table** that meets the description.
3. Create the **Karnaugh maps** from that truth table.

4. Obtain the simplified **Boolean expressions** that describe the system.
5. Design the **digital circuit** itself.
6. Implement that circuit with actual logic gates or, more commonly, in a higher level device through VHDL, Verilog, or even microcontroller code.

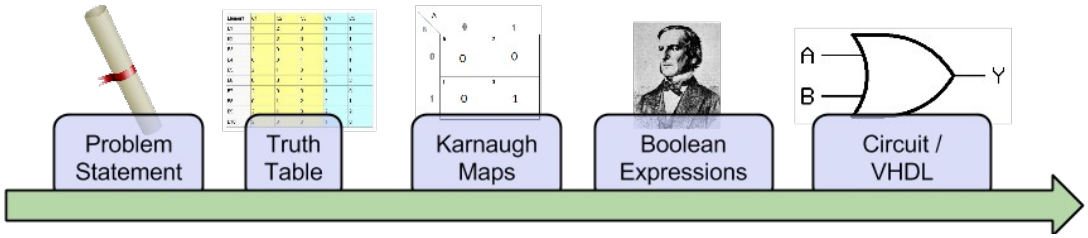

Figure 1. Educational combinational system design steps.

For a Finite State machine the steps are mostly similar. For both types of circuits, specifying the truth table correctly is critical. A wrong truth table will certainly yield a wrong circuit. However, that is relatively easy for students. Most problems come from the Karnaugh maps, because the simplification system is visual and mistakes are likely to be made. To face these challenges, teachers rely on experience and on the use of simulators and other design tools. Teachers and their students follow the process, they obtain the digital circuit, and simulate it. If the behavior is appropriate according to the original problem statement, then it can be implemented. Unfortunately, the circuit does not always behave as expected. In that case, the student often has trouble finding which step is wrong. This is particularly true because most professional simulators and design tools generally do not focus on the process details (which are the main concern of the students) but on obtaining an actual digital circuit, and for that purpose they do not expose some of the steps to the student. Yet another issue is that often teachers and students are hesitant to actually implement and test the obtained digital circuits. This is not because the process itself is too advanced, or because seeing the final circuit working would not provide academic value, but simply because it is often a long, tedious and time-consuming process, in which minor mistakes are likely. The result is that the student often does not have a clear picture of the design process and of what that design process really entails. This is very detrimental to the learning process.

To handle these issues, this work proposes the Boole-WebLab-Deusto system. It integrates two tools: One for the detailed design of combinational and sequential digital systems (Boole-Deusto) and other for the rapid prototyping and testing of digital systems through a remote laboratory (WebLab-Deusto). Additionally, WebLab-Deusto has been extended with a hybrid virtual reality layer, which makes the exercises more interesting and engaging for the student, and that is described in later sections of this work.

Boole-Deusto

Boole-Deusto is an Open Source, educational, electronic design tool. Its first version was released in the year 2000, and has been downloaded and used by thousands of students ever since. Oriented for an educational setting and not professional usage, it is easy to use and particularly useful for introductory courses because it covers every step of the design process. This is important because most professional design tools handle certain steps automatically, which is detrimental for learning purposes.

Students often start by naming their system and choosing the number of inputs and outputs that they want. From then on, they can input the truth table, simplify the system visually through Karnaugh diagrams, and make use of the other functionalities provided by Boole-Deusto.

Though describing every feature of Boole-Deusto is beyond the scope of this work, it is noteworthy that it covers rather thoroughly the process that a student would follow designing a digital system with pen and paper. This includes, for instance, the capability to simplify the truth table into simplified Boolean expressions by drawing circles on the Karnaugh diagrams themselves, as shown in Figure 2.

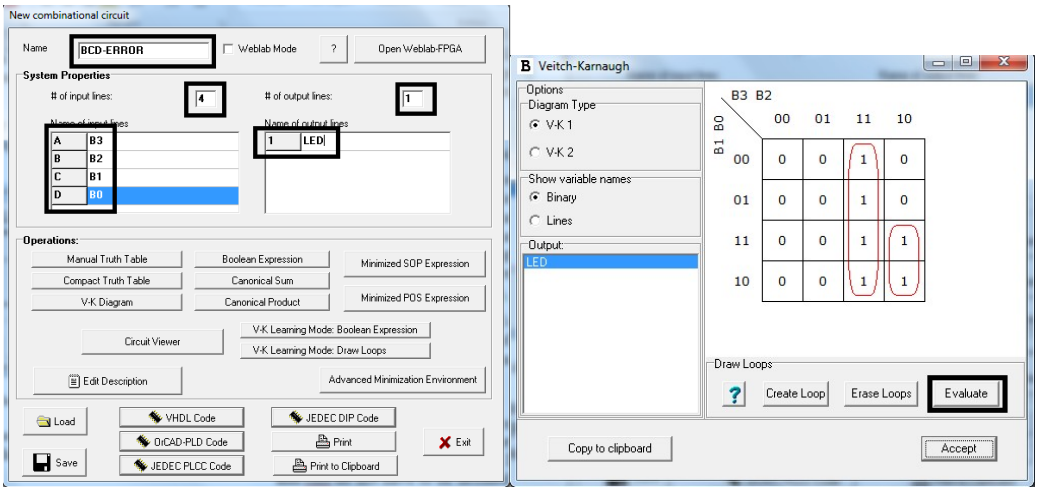

Figure 2. Description and Veitch-Karnaugh Map of a combinational system in Boole-Deusto

Boole-Weblab-Deusto

Originally, Boole-Deusto only had the offline capabilities described earlier. With them, Boole-Deusto guided the students through the design process but once it was time to test the system they were on their own. Through the work described here, Boole-Deusto has been integrated with the WebLab-Deusto remote laboratory, to form the WebLab-Boole-Deusto system. Through WebLab-Boole-Deusto, students can easily design a digital circuit and easily test it in real (remote) hardware, all in a few minutes.

To use it a special “Weblab Mode” has been added to Boole-Deusto. This mode provides certain functionalities and ensures that the system that is being designed is compatible with the WebLab-Deusto hardware (does not exceed the number of inputs and outputs, their names are chosen from a fixed list indicating the component they control, etc.).

Using the Weblab Mode is relatively straightforward. Users design their system as they normally would, with some minor differences: when in Weblab Mode, the maximum number of inputs and outputs is limited to the capabilities of the physical board that will host the program. Also, rather than typing the I/O names, the users must choose among a list of predefined inputs and outputs. The most common ones are switches and LEDs respectively.

Once they have specified the truth table, or made use of any other Boole-Deusto feature, and the system is ready, students only need to click the "Open WebLab" button and generate the VHDL code that describes their system's behavior (see Figure 3).

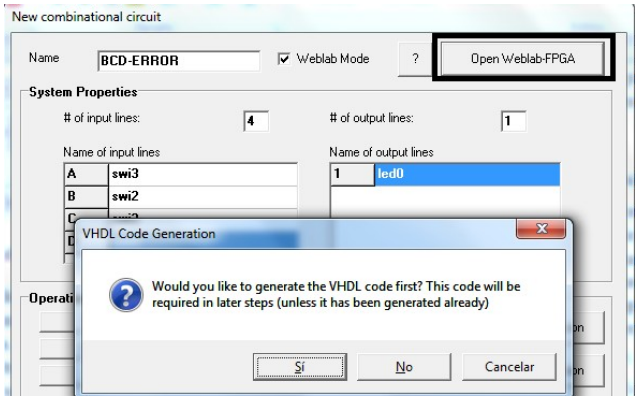

Figure 3. WebLab-Boole-Deusto. Opening Weblab and generating the VHDL code

The VHDL that Boole-Deusto generates in its WebLab mode is automatically made compatible with WebLab-Deusto. For this, the program takes special care when generating the code to make the signal names match those that WebLab-Deusto expects and to include certain specific headers and preprocessor statements which are Weblab-Specific.

After saving the VHDL file the Weblab-Deusto website will open in the default browser and it will request for the user's authentication credentials to gain access to the Weblab-FPGA experiment itself (as shown in Figure 4). Once there, all that is needed is to choose the VHDL file that has just been generated. This will send it to the server, which will automatically synthesize and program it into a physical board. This process (especially the synthetisation part) is relatively complex, and can take some time (though all the work is done by the server and no further user input is required).

Figure 4. Weblab-Deusto user login page

When the board has been successfully programmed, it will be shown through a webcam stream, along with a set of switches to interact with it. The output of the board can be seen through the webcam from the real LEDs. The experience is essentially the same as the real one, except that it tends to be much faster and more convenient (as students do not need to deal with cables, connections and board programming directly, which are conceptually simple processes but very time-consuming and error-prone). Figure 5 shows the webcam stream of a running instance of the experiment, and part of the user interface. In this case, the student's program is turning on the second LED and pressing two of the nine switches (of which only 3 are shown).

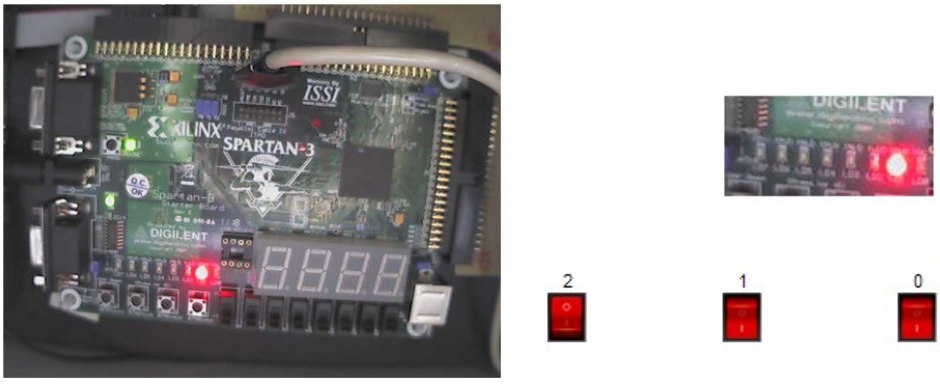

Figure 5. The user's system running on a real, remote FPGA board. (3 of the 9 switches are shown on the right).

Though so far the process that has been described is for combinational systems, the process is similar for sequential ones (Finite State Machines). In this last case, students can define the automata by drawing its State Transition Diagram (see Fig. 7), or optionally go through additional steps, such as manually specifying the truth table. Once this is done, VHDL code can be generated easily. For this, the main difference with combinational systems is that FSMs need a clock. Students can choose one to use among a list of different

ones. The procedure for FSMs (See Figure 6) and their architecture is, however, beyond the scope of this chapter.

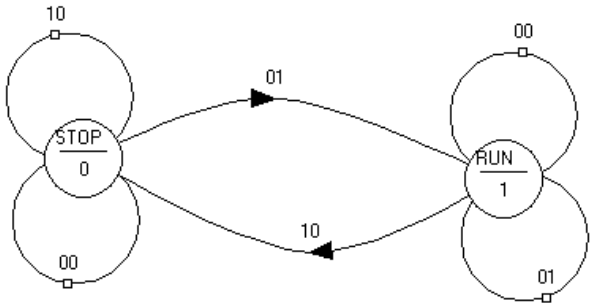

Figure 6. Boole-Deusto's FSM design tool. Students can design and define the system by drawing, as they normally would

Weblab-Deusto and Virtual Reality

The system and the approach that have so far been described solve the first issue that was mentioned at the start of this work. They make the full design-implementation-testing process easy and accessible for all students, with no interruptions in-between. With just a few minutes and an up-to-date browser, they can readily design and test in real hardware their own combinational or sequential digital systems.

However, as explained, there are still some engagement issues. A common exercise that students are asked to do (just as an example) is to create the controller for an engine, or a water tank. If this was the case, students would design their system, and then test it on the remote hardware. However, the hardware only has LEDs as outputs. In practice, students would need to imagine that the LEDs are actually the engine or the water tank. Conceptually this makes sense, as, in principle, seeing how the LEDs are behaving, students can predict with relative ease whether it would really work on a water tank or not. However, in practice students do not find these exercises too engaging or realistic. As a result, they feel less interested and motivated, and the learning process is significantly affected.

There are at least two apparent solutions for this issue. The most obvious would be to connect an engine or a water tank to the FPGA board. However, this is not easy in practice. Real hardware of that kind can be expensive and hard to maintain. In addition, it would make the FPGA board useful only for that specific exercise, which, again, implies a large cost. Other, less obvious solution is to use virtual reality. By combining the real, physical FPGA controller, with a virtual reality layer that contains a model to control, it is possible to achieve a system that is both affordable and realistic enough, and which students can find engaging and immersing (There has been significant research on remote laboratories with virtual and augmented reality [7] with positive and interesting results).

The latter is the approach that has been chosen at Weblab-Deusto. Providing the virtual reality layer and integrating it with the existing architecture to form a hybrid laboratory in which both real and virtual components interact seamlessly is significantly complex. The next sections of this work will introduce hybrid laboratories, and describe the hybrid layer of WebLab-Deusto and Weblab-FPGA-Watertank (the specific water tank experiment) in more detail.

Remote, virtual and hybrid laboratories

Previous sections have described the functionality of the WebLab-Boole-Deusto system. The following sections provide a technical overview of the hybrid laboratory scheme that builds upon that system.

Throughout the last years multiple online laboratories have appeared. Though they all share some common traits, there are different types of them. Traditionally, online laboratories have been separated in two categories: remote laboratories and virtual laboratories. The former provide access to real hardware, which can be used remotely and which can often be monitored through a webcam stream. The latter provide access to a simulation, which can run either in the server-side or the client-side, and there is no real hardware behind it (beyond that needed by the infrastructure of the virtual laboratory server itself).

Though this distinction still applies today for most laboratories, there are now some "hybrid" laboratories, which try to leverage the advantages of both types of laboratory by combining their characteristics. Though in order to be considered "hybrid" a laboratory would need to include both virtual and real components [8], the interaction between these components will also vary.

In some cases, that interaction is sequential: the hybrid laboratory will first let users work with the virtual simulation. Once the users have finished working with the simulation, the laboratory will then let them test their work against the real hardware. The key about such a scheme is that there is only a limited interaction (if any) between the virtual and remote components of the laboratory.

Sometimes, however, the nature of the virtual-real interaction is more complex. That is the case of the laboratory that will be described in this work. In this case, the real components and the virtual components interact with each other in both directions, and the interaction with one component would not make sense without the other. The experience provided by the laboratory is precisely configured by that interaction.

The WebLab Hybrid System

Traditionally, the WebLab-Deusto remote laboratory management system has provided access to many electronics experiments, for devices such as FPGAs, PLDs or microcontrollers. In many of these cases, WebLab-Deusto provides remote access to a development hardware board with these components, which is then remotely programmed with the specified logic. Students can then use WebLab-Deusto to remotely program that logic into real hardware. This is the approach that has been described so far. This general scheme has been available at WebLab-Deusto for several years [9] and is similar to the one provided by other remote laboratories. It is well-proven and most appropriate for a variety of user needs, but it has certain limitations.

Ideally, a perfect remote laboratory would let users experiment with different output devices. Real logic devices (FPGAs, PLDs, microcontrollers...) are used to control systems as diverse as production lines, engines, water tanks, industrial machinery, etc. However, in practice, a laboratory with real hardware cannot provide so much. In the particular case of our traditional experiments, outputs are simple LEDs. It is up to the users to "imagine" that these LEDs are outputs acting on more advanced machinery. This often makes the experiments less engaging and realistic than would be desired. Providing varied instances of real, industrial hardware is not an option. That hardware is often very expensive, it takes a lot of physical space, it implies a maintenance effort and it can be dangerous. Sometimes, having certain equipment in a standard remote laboratory setting, such as a power plant, can be outright impossible.

In order to solve these issues, WebLab-Deusto has developed a Hybrid laboratory (see Figure 7), which is essentially an additional layer on top of some existing laboratories. Users are still provided a real hardware board (such as a FPGA) which they can program. However, that board no longer has only LEDs as outputs. Instead, the board can interact in both directions with a virtual model. Thus, an absolutely real hardware board, programmed by the user, can be used to control an arbitrarily complex (or a purposely simplified) virtual simulation.

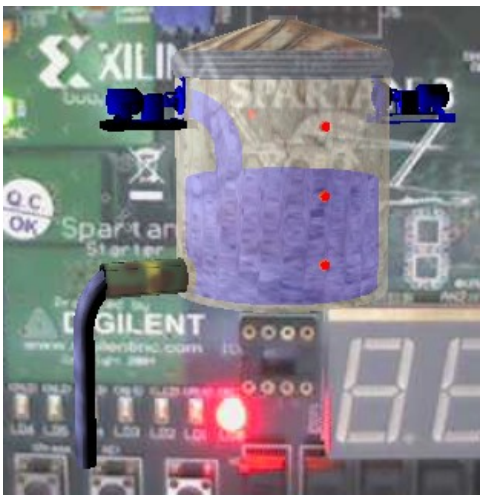

Figure 7. WebLab-FPGA-Watertank laboratory running. The virtual model is controlled by the real FPGA board, which is streamed in real time.

Advantages and Limitations

The advantages of this system have been outlined in the previous section, but will be detailed here. The first advantages that should be taken into account are those characteristic of any online laboratory. Users can connect remotely, from their homes or any other location. They no longer need physical access to the equipment. Because monitoring personnel is no longer required, the availability of the laboratory can be extended, and barring downtimes can potentially be 24/7. Usages can be spread in time, so less instances of

each equipment piece are needed. If applied properly, these advantages can result in a lower cost and a greater convenience.

However, the Hybrid system that is being discussed here has additional specific advantages, as it tries to leverage the particular strengths of remote and of virtual labs, by having a system that is made of both a real, physical controller board and of a virtual model.

One of the most significant ones is its affordability. Because it does not require hardware components beyond the standard remote laboratory infrastructure and the physical controller board, it is relatively inexpensive. A fully-real remote laboratory with, for instance, a real water tank to control, would require more money, resources, space, and maintenance. Because the controller board is still a physical FPGA (or similar), is physically programmed and runs the user's logic, not much realism is lost. As long as the virtual simulation (that the real, non-virtual board will control) is accurate enough, the experience will be nearly the same. Moreover, in some cases a realistic equipment to control is actually not desired. Because that equipment is often very complex and hard to control, sometimes a simplified model, which provides a smoother learning curve, is actually preferred.

The system is very extensible, and it is relatively straightforward to extend the laboratories with new controller devices (PLDs, microcontrollers) and new virtual models. Also, it is noteworthy that a single controller board could service many different experiments with different virtual models to control, which would otherwise (with a traditional remote laboratory) be impossible.

Implementation

The system is based on a physical controller board which runs the logic, which is defined by the remote student (normally, through VHDL) and automatically programmed into the board by the server. In order to be able to do this, the server receives the VHDL code from the remote student, synthesizes it using specialized Xilinx software, and programs it into the physical controller board.

The controller board has both inputs and outputs, which can be accessed through the users' logic. The inputs depend on the virtual model. For instance, if the virtual model features water level sensors, the output of these sensors will be received as inputs on the physical board. The outputs affect the virtual model as well. In a virtual model with actuators such as water pumps, the outputs on the board are used, for instance, to turn on the water pumps. In order to ensure that users understand that the physical board they are using is running their code and is fully real, outputs are also mirrored on some physical LEDs, which can be observed (along with the board itself) through a webcam.

The user interface -which displays the aforementioned webcam- is fully web-based. It also displays the virtual model in 3D. In order to provide a realistic experience, the virtual model is relatively complex and the graphics are three-dimensional and hardware-accelerated, with shader support. Only a few years ago this would have been impossible without using proprietary components such as Adobe Flash or Java Applets, but

nowadays, using modern Web technologies, WebLab-Deusto manages to provide these features requiring only a standard up-to-date browser.

Hardware

The WebLab-Deusto hybrid lab system is designed to be generic, and to support quite easily almost any type of physical controller.

When implementing a new physical controller there are two main tasks to accomplish. First, the server must support synthetisation for that controller. That is, the server must be able to receive the "source code" for a program or logic, and be able to synthesize it into a format that can be programmed on the physical board. Second, the physical board must be able to provide its outputs to the system, so that they can be used to control the virtual model.

The system currently deployed at Deusto fully supports FPGA controller boards, and has partial support for PLD boards. This is likely to be extended in the future. The physical controller boards are actually made of two different boards. One of them is the actual controller -in our case, the FPGA development board. The other one is a custom board powered by a PIC microcontroller, which reads the outputs of the FPGA board and passes them to the WebLab-Deusto system through an HTTP-based protocol (see Figure 8). This is necessary because, for the virtual model to behave as intended, it needs to receive the state of these physical outputs.

Figure 8. An instance of the WebLab-FPGA hardware, shared by the traditional WebLab-FPGA and the Hybrid one.

Client & user interface

As previously mentioned, one of the main goals of the client (see Figure 9) was to be fully web-based and portable. Though nowadays most laboratories have been ported to the web, traditionally those with advanced graphics requirements have had significant difficulty, and have relied upon non-standard, proprietary components [10].

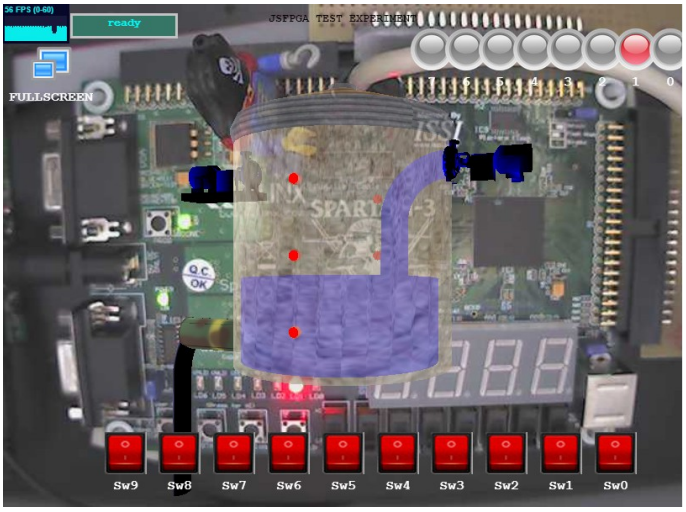

Figure 9. WebLab-FPGA-Watertank full user interface. The FPGA board is a video stream, not a static image.

The main reason for that has been the traditional lack of support for this kind of graphics in HTML and the web standards.

The most common components have been Adobe Flash and Java Applets, which have also been prevalent in other kind of RIAs (Rich Internet Applications). Some more specific components have also been popular, such as the LabView Remote Panel [11]. Reliance on these components is not without significant drawbacks. First, applications that use them are no longer fully web-based. Users need to install plugins, which is not always possible, especially on a network on which they are not administrators, as is the case in many schools and universities. These plugins, even when installed, impose a maintenance burden, because they need to be frequently upgraded and are a security risk. Throughout time, hundreds of thousands of computers have been infected through them. Because of these issues, relying on these non-standard components can deter users from using the labs [12].

Other issue which is nowadays more significant than ever is that these technologies are not supported on most mobile devices. In a world with an ever-increasing number of phones and tablets, and with an ever-decreasing number of laptops [13][14] mobility is an issue of outmost importance. Today, with the advent of HTML5 and of new web standards, these components are no longer required. One of the secondary goals of the system that is described in this work is to prove that, and to a base upon which to build new standards-based rich applications. Thus, the WebLab-Deusto-FPGA client is built on WebGL [15][16], which is not part of the HTML5 standard but which is very closely related, and supported by all modern browsers (including Chrome, Firefox, and Internet Explorer, among others).

The system also partially supports Canvas, but its performance on it is much slower for the current FPGA-Watertank laboratory. Canvas is part of the HTML5 standard, and provides certain graphics capabilities, but not 3D acceleration. It is, however, a useful tool for 2D graphics and data visualization and is being used already

by several academic projects for this purpose [17][18]. Still, because the Watertank laboratory currently features full 3D graphics and is not really designed to work without 3D hardware acceleration, for a proper user experience WebGL is required. In order to support both WebGL and (to a limited extent) Canvas, the system actually makes use of ThreeJS, an Open Source library which abstracts the rendering system. As mentioned, however, there are significant differences between the rendering systems it abstracts, as WebGL provides full 3D acceleration and shaders, which is required for maximum efficiency and for rendering top graphics. In our case, this was selected because we wanted the virtual model to be as realistic as possible.

WebGL is already used for several applications, including data visualization as well [19][20]. Support in mobile devices is steadily growing, so now the client can run perfectly on most modern tablets and smartphones with updated browsers. This would have been impossible using traditional proprietary components such as Adobe Flash or Java Applets. Beyond WebLab-Deusto's, there are already some interesting examples of laboratories which make use of WebGL. One of these is a collaborative learning system developed by UNED [21]. This framework features a collaborative environment in which WebGL simulations are automatically generated from 3D Easy Java Simulations applets [22] (based on Java Applets).

Synthesization System

A core part of the system is its ability to program the logic provided by users into a physical controller board. This logic is specified through VHDL, a hardware definition language. The original Weblab-FPGA system did not have that capability. Users had to provide an already synthesized binary file to program, which they synthesized on their own computers by using specialized Xilinx software. Though this approach has some minor advantages (users get to fully practice with the Xilinx development environment and can make use of all of its features to fix synthetisation errors), it also has major inconveniences: the Xilinx software needs to be installed on the computer. Administration privileges are needed to install and a free but explicit license is required. The size is over 6 gigabytes. Projects need to be configured with specific settings to be compatible. Also, it is available for some platforms only. With the new approach, the process is much easier for users and students. The synthetisation software is on the server and users only need to provide the VHDL source code. They do not need to have any specialized software on their machines, or install anything, but they can still do so if they wish to.

Controller and Model interaction

The system has two components (see Figure 10). The first is a real controller board, with inputs that users can read and outputs that users can control through the program they provide. The second is a virtual model, which is essentially a simulation. The model in that simulation, which also has inputs and outputs, is controlled through the physical board. Though the system is designed to be extended, currently the standard laboratory is based on a FPGA board and provides a virtual water tank to control. The water tank has two inputs. Each one represents the state of a water pump. Thus, users, through the logic they specify for the (real) FPGA board, can turn the water pumps on and off.

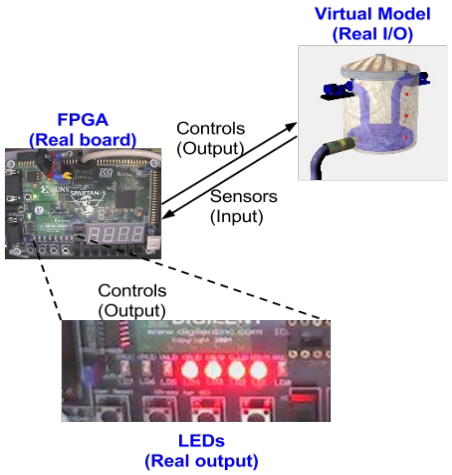

Figure 10. WebLab-FPGA-Watertank architecture and interaction between components.

Depending on the specific exercise, the students will design the logic so that the pumps are turned on or off at a certain moment. The key here is that the virtual model also has outputs which can be received on the FPGA board (and hence on the logic that is running on it). In this particular case, the water tank model has five virtual sensors. Three of them are water level sensors. They are located at different heights of the virtual water tank. The first one is at 20% height, the second at 50%, and the third at 80%. When the virtual water reaches each sensor, the sensor turns on and the real, physical board can read that. The other two outputs are temperature sensors. They are only used for an "advanced" water tank mode. In this mode, the water pumps cannot work continuously. If they do, their temperature raises until they breakdown. Each pump has an associated sensor whose output is turned on if the temperature is high. This way users can design the logic in such a way that whenever a pump is getting too hot it is stopped and the other pump starts working. Of course, if users fail to design the logic appropriately in this advanced mode, the virtual pumps will stop working and they will fail the experiment.

In the future, it would be quite easy to add new models, with a completely different set of inputs and outputs. The physical FPGA boards support at least up to 8 lines of each.

Conclusion and Future Work

The main goals of the WebLab-Boole-Deusto system were two:

- Allow the rapid prototyping of digital systems by providing a seamless workflow in which a student can design, implement and test a digital system fast and easily.
- Making experiments more engaging while maintaining realism.

As it stands now, the goals are met: students, provided only with an Internet connection, an up-to-date browser, and the Boole-Deusto system, can implement a digital circuit, of either the combinational or sequential kind, in just a few minutes and a few clicks. Immediately, these students can open WebLab-Deusto

and test that digital circuit on a real FPGA board. Furthermore, (if appropriate) they can choose to access the FPGA-Watertank experiment instead of the conventional FPGA laboratory, and make their circuit control the virtual water tank. Students are finding these experiments more interesting than traditional ones on which they can only control and see some LEDs, because they can get the feeling of controlling more complex and challenging equipment.

Several challenges that arose during the implementation stage have all been cleared successfully, and the scheme that has been described in this chapter is deployed and working, and has been tested with many students already, accounting for hundreds of users. Though for now the only deployed and tested hybrid laboratory is the water tank that has been described throughout this work, more laboratories are likely to be added. The WebLab-Deusto hybrid architecture is intended to be generic and to provide a straightforward way to create engaging and affordable experiments. These are easy to develop and maintain because, being based upon a virtual model, they can all use the same hardware base (the FPGA board, or even a different controller board). As stated, at the same time they are realistic because the controller that users act upon -which controls the virtual model- is a real, physical board.

In the future, the system is likely to be made more usable for mobile users (phones and tablets). The importance of these platforms for students and remote laboratories is likely to grow, and as depicted in Error: Reference source not found11, the system is already functional (with some limitations due to requirements such as file uploading). Though a few years ago this would not have been possible (at least, not with the kind of 3D graphics support that is displayed), the WebLab-Deusto hybrid system architecture and its use of modern web technologies such as WebGL make mobile support relatively simple.

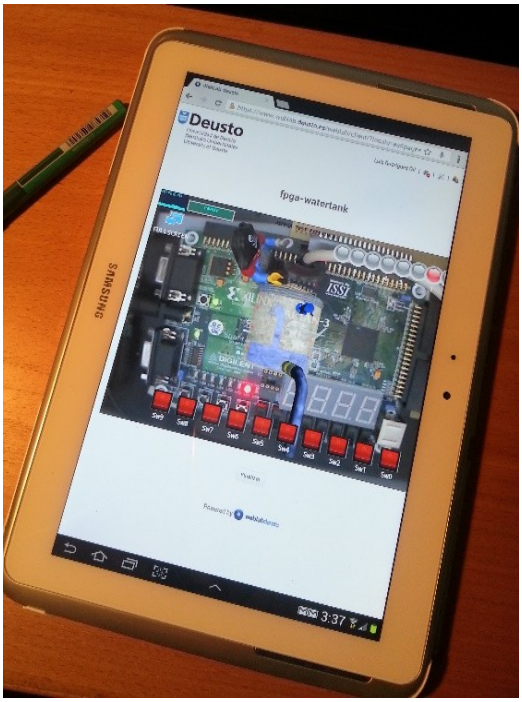

Figure 11. WebLab-FPGA-Watertank running on a tablet on the mobile Chrome browser.

References

[1] Boluda, J.C, Peiro, M.A, Torres, M.A.L, Girones, R., Palero, R.J.C. "An Active Methodology for Teaching Electronic Systems Design." IEEE Transactions on Education, vol. 49, no 3, pp. 355-359. 2009.

[2] Azcondo, de Castro, and Brañas, "Course on Digital Electronics Oriented to Describing Systems in VHDL." IEEE Transactions on Industrial Electronics, vol. 57, no. 10, pp. 3308-3316. 2010.

[3] Feisel, Lyle D., and Albert J. Rosa. "The role of the laboratory in undergraduate engineering education." Journal of Engineering Education, vol. 94, no. 1, pp. 121-130, 2005.

[4] Dym, C. L., Agogino, A. M., Eris, O., Frey, D. D., and Leifer, L. J. "Engineering design thinking, teaching, and learning." Journal of Engineering Education, vol. 94, no. 1, pp. 103-120, 2005.

[5] Orduña, Irurzun, Rodriguez-Gil, Zubia, Gazzola, López-de-Ipiña. "Adding New Features to New and Existing Remote Experiments through Their Integration in WebLab-Deusto." International Journal of Online Engineering. Vol. 7, no. S2, pp. 33-39. 2011.

[6] Lobo, J. "A Remote Reconfigurable Logic Laboratory for Basic Digital Design." 1st Experiment@ International Conference. 2011.

[7] Callaghan, V., Gardner, M., Horan, B., Scott, J., Shen, L., and Wang, M. "A mixed reality teaching and learning environment." In Hybrid Learning and Education" (pp. 54-65). Springer Berlin Heidelberg. 2008.

[8] Gomes, Luís; Bogosyan, Seta. "Current trends in remote laboratories." IEEE Transactions on Industrial Electronics, 2009, vol. 56, no 12, pp. 4744-4756.

[9] García-Zubia, J., López-de-Ipiña, D., Hernández, U., Orduña, P., and Treba, I. "An approach for WebLabs analysis". International Journal of Online Engineering, 3(2). 2007.

[10] Rodríguez-Gil, L., Orduña, P., García-Zubia, J., Angulo, I., López-de-Ipiña, D. "Graphic Technologies for Virtual, Remote and Hybrid laboratories: WebLab-FPGA hybrid lab." In Proceedings of the 10th International Conference on Remote Engineering and Virtual Instrumentation (REV). February, 2014.

[11] Orduña, P., Garcia-Zubia, J., Rodriguez-Gil, L., Irurzun, J., López-de-Ipiña, D., and Gazzola, F. "Using LabVIEW remote panel in remote laboratories: Advantages and disadvantages." In Global Engineering Education Conference (EDUCON), 2012 IEEE (pp. 1-7). IEEE. April, 2012.

[12] Stieger, S., Göritz, A. S., & Voracek, M. "Handle with care: the impact of using Java applets in web-based studies on dropout and sample composition." Cyberpsychology, Behavior, and Social Networking, 14(5), 327-330, 2011.

[13] El-Hussein, Mohamed Osman M.; Cronje, Johannes C. "Defining Mobile Learning in the Higher Education Landscape." Educational Technology & Society, 2010, vol. 13, no 3, p. 12-21.

[14] Barnes, J., and Herring, D. "Using Mobile Devices in Higher Education." In Society for Information Technology & Teacher Education International Conference (Vol. 2013, No. 1, pp. 206-211). March, 2013.

[15] Marrin, C. “WebGl specification.” Khronos WebGL Working Group, 2011.

[16] Leung, C., and Salga, A. “Enabling webgl.” In Proceedings of the 19th international conference on World Wide Web (pp. 1369-1370). ACM. April 2010.

[17] Miller, C. A., Anthony, J., Meyer, M. M., & Marth, G. “Scribl: an HTML5 Canvas-based graphics library for visualizing genomic data over the web.” Bioinformatics, 29(3), 381-383. 2013.

[18] Boulos, M., Warren, J., Gong, J., & Yue, P. “Web GIS in practice VIII: HTML5 and the canvas element for interactive online mapping.” International Journal of Health Geographics, 9(1), 14. 2010.

[19] Congote, J., Segura, A., Kabongo, L., Moreno, A., Posada, J., & Ruiz, O. “Interactive visualization of volumetric data with webgl in real-time.” In Proceedings of the 16th International Conference on 3D Web Technology (pp. 137-146). ACM. June 2011.

[20] Bochicchio, M. A., Longo, A. and Vaira, L. “Extending Web applications with 3D features.” In Web Systems Evolution (WSE), 2011 13th IEEE International Symposium on (pp. 93-96). IEEE. September 2011.

[21] Jara, C. A., Candelas, F. A., Torres, F., Salzmann, C., Gillet, D., Esquembre, F. and Dormido, S. “Synchronous Collaboration between Auto-Generated WebGL Applications and 3D Virtual Laboratories Created with Easy Java Simulations.” In Proceedings of the 9th IFAC Symposium Advances in Control Education", ISBN (pp. 978-3). June 2012.

[22] Esquembre, F. “Easy Java Simulations: A software tool to create scientific simulations in Java.” Computer Physics Communications, 156(2), 199-204. 2004.

APPENDIX

E

Paper V

Towards new multiplatform hybrid online laboratory models

This paper was published in the journal '*IEEE Transactions on Learning Technologies*'. It was first online on 18 July 2016. Its DOI is: 10.1109/TLT.2016.2591953. It can be accessed online through the IEEE Xplore Digital Library at: http://ieeexplore.ieee.org/document/7515014/. The published version is appended here.

Towards new multiplatform hybrid online laboratory models

Luis Rodriguez-Gil, Javier García-Zubia, *Senior Member, IEEE,* Pablo Orduña, *Member, IEEE* and Diego López-de-Ipiña

Abstract—Online laboratories have traditionally been split between virtual labs, with simulated components; and remote labs, with real components. The former tend to provide less realism but to be easily scalable and less expensive to maintain, while the latter are fully real but tend to require a higher maintenance effort and be more error-prone. This technical paper describes an architecture for hybrid labs merging the two approaches, in which virtual and real components interact with each other. The goal is to leverage the advantages of each type of lab. The architecture is fully web-based and multiplatform, which is in line with the industry and the remote laboratory community trends. Only recently has this become technically feasible for graphic-intensive laboratories due to previous limitations in browser-based graphical technologies. This architecture relies on the recent HTML5 and WebGL standards to overcome these limitations, and makes use of the Unity technology. To ensure that the proposed architecture is suitable we set requirements based on the literature, we compare it with other approaches and we examine its scope, strengths and weaknesses. Additionally, we illustrate it with a concrete hybrid lab and we evaluate its benefits and potential through educational experiments.

Keywords—remote laboratories, virtual environments, architectures, web-based architectures, hybrid laboratories

1 Introduction

Online laboratories have traditionally been classified in two large and distinct groups: virtual and remote ones [1], [2], [3]. The former provide access to a simulation while the latter provide access to real equipment over the Internet. Educators have long debated over how those laboratories compare to hands-on ones, and over which type of lab is more effective. Though no consensus has been reached, research suggests that certain virtual laboratories can be as effective as hands-on laboratories [4] and that some types of laboratories are more or less adequate depending on the set learning objectives [5]. Today, in practise, the line between *virtual* and *remote labs* is more blurry, because there are laboratories which have characteristics of both.

Traditionally, these types of lab have certain aspects in common which result in a different set of advantages and trade-offs [6], [7]. Traditional virtual laboratories require no physical space, are highly scalable and can adapt reality to fit the teaching needs, such as by simplifying it or by displaying unobservable phenomena. Traditional remote laboratories rely only on real equipment so they provide real data and include authentic delays and unanticipated events such as measurement inaccuracies, through which students can learn about the complexities of science [8].

Currently, efforts are being dedicated to the research of advanced forms of lab that not only mimic a hands-on experience but provide their own new features and advantages, which may not exist in a real hands-on lab. One of these models are *hybrid* labs. The definition of *hybrid* lab is not clearly established in the literature, but, in the context of this work, a *hybrid* lab is simply a lab which mixes virtual and real components [9]. Some authors do not use the word *hybrid*, but simply refer to these labs as either *virtual* or *remote*. Figure 1 shows where this lab model fits within the traditional characterization of labs. Thus, hybrid laboratories try to leverage some of the advantages of virtual and remote ones, mainly by being able to provide realism, cost-effectiveness and additional features such as gamification or virtual environments.

In this line, some works have applied augmented reality (AR) to a robotics remote lab [10], [11]. Researchers from UNED have applied AR to a hybrid lab for the control of a thermal process [12], [13] and to a hybrid lab for the control of a three-tanks system [14]. Researchers from the University of Deusto have integrated remote laboratories into the SecondLife[1] virtual world [15] and developed a hybrid FPGA lab which overlays a virtual watertank model over a real FPGA controller, allowing bidirectional interaction with the board. Other works from the University of Ulster have integrated an electronics remote lab in a gamified virtual world [16], [17] and in a serious game [18]. Authors from the Polytechnic University of Madrid have integrated electronics remote laboratories in a virtual environment [19]. Other

- *L. Rodriguez-Gil, P. Orduña and D. López-de-Ipiña are with the Faculty of Engineering, University of Deusto, Avda. Universidades, 24, 48007, Bilbao, Spain and DeustoTech - Deusto Foundation, Avda. Universidades, 24, 48007, Bilbao, Spain. E-mail: luis.rodriguezgil@deusto.es.*
- *J. García-Zubia is with the Faculty of Engineering, University of Deusto, Avda. Universidades, 24, 48007, Bilbao, Spain.*

1. http://www.secondlife.com

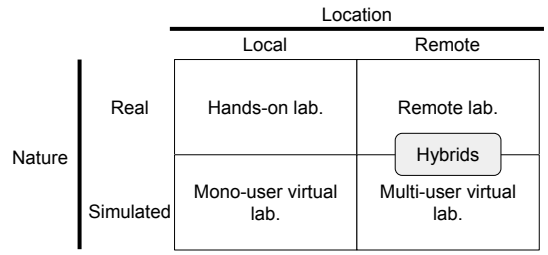

Fig. 1. Hybrid laboratories within a characterization of labs (adapted from [1] and [9])

researchers from TU Graz and MIT have integrated a simulation and a remote lab to build a collaborative Wonderland-based virtual world [20].

In Section 2 this paper proposes a set of requirements and criteria for a software architecture for the development of advanced hybrid online laboratories, according to the industry and the remote lab community trends. The most significant of these requirements are *universality* and intensive graphics. The architectures that previously published works implicitly or explicitly describe are analyzed. Section 3 proposes an architecture that builds upon that specification and knowledge. This is achieved by relying on the new HTML5 [21] and WebGL [22] standards (and on the Unity3D [23] technology), which has only recently become technically feasible due to advances in browser-based graphical technologies. Afterwards, in Section 4, the suitability of the proposed architecture is analyzed according to the previously set criteria, it is compared against other approaches, and its strengths and weaknesses are examined. The architecture is illustrated, in Section 5, with an implementation of a concrete hybrid lab. Its benefits are evaluated in Section 6 through educational experiments.

2 MOTIVATION

2.1 Virtual, remote and hybrid laboratories

Research works have concluded that hands-on, virtual and remote laboratories can all be educationally effective, though there is no consensus on which one is the *most* effective; and that is likely to depend on the specific circumstances, educational goals and other factors [4], [5].

Table 1 provides a comparative summary of some of the characteristics of each type of lab. These apply to the *average lab* of its kind but not necessarily to all of them. Realism indicates how close to reality the lab is, and how realistic experimentation with it is felt. Remote laboratories tend to be very realistic because by their nature they rely on real equipment and provide real data, but even then, some works report that only properly designed ones are indeed perceived as realistic [6]. *Can alter reality* indicates whether these laboratories can purposefully simplify reality or even modify it, in order to make certain concepts easier to understand and to

TABLE 1
Comparative summary of characteristics of traditional hands-on, virtual and remote labs, and of hybrid laboratories

	Hands-on	Virtual	Remote	Hybrid
Realism	Very high	Low	High	Medium
Can alter reality	No	Yes	No	Yes
Recurring costs	High	Low	High	Medium
Augmented features	No	Yes	No	Yes
Depends on rich media	No	Yes	No	Yes

hide complexities that are out of a specific educational scope. *Recurring costs* indicates how expensive the lab tends to be in time. The initial investment is not taken into account here because it depends too much on the concrete lab. Hands-on and remote laboratories tend to have the highest recurring costs because they require significant maintenance —including the repair or replacement of hardware components— and supervision. Virtual laboratories have the lowest because, for the most part, once they are developed they require little to no infrastructure. *Augmented features* refers to the fact that some laboratories admit additional software features to enhance or extend the learning experience. Examples of such features would be overlaying a non-visible physical reality on the display so that it can be understood more easily —for instance, an electromagnetic field [20]— or being integrated into a serious game. *Depends on rich media* indicates whether the lab needs to use media such as non-basic computer graphics, videos, or sound. Hands-on laboratories quite predictably don't have such a dependency. Remote laboratories often require only a webcam and basic components to interact with the equipment, so they don't need it either. The average virtual and hybrid lab, however, will need to depict or extend the experimentation reality through graphics, and sometimes additional media.

2.2 Requirements

This section lists the different goals and requirements that the proposed architecture should meet and the criteria and rationale that has been used for establishing them:

1) **Universality**: Accessible to as many as possible.
2) **Security**: Minimize the security risk that its users are exposed to.
3) **Power**: Support relatively advanced technical features such as rich media (graphics, sound and video) or low-latency bidirectional communications.

The importance of universality and security for an online lab was supported by a group of experts and discussed in [24].

2.2.1 Universality

Research suggests that this is the characteristic that experts in the Remote Laboratory community value most,

and has long been the industry trend. Some criteria can be used to evaluate how universal an architecture is:

- Support in desktop browsers
- Support in mobile browsers
- Reliance on HTTP and HTTPS ports only
- Non-dependency on plugins

2.2.2 Security

Remote laboratories are usually hosted by institutions such as universities. Their IT teams are often hesitant to offer intrusive technologies to students because they do not want to expose them to security risks, for which the university itself could be liable [24]. *Non-intrusive* technologies are thus preferred. Browser plugins increase the system's attack surface and have been a source of vulnerabilities in the past, so reliance on them also tends to lower security.

Additionally, it is more convenient if the existing infrastructure does not need to be modified to support the online laboratories. For this, they should ideally rely on HTTP or HTTPS, so that no special ports need to be opened or non-standard protocols allowed.

There tends to be some positive correlation between *universality* and *security*, which is why some features discussed here are present in both subsections.

Thus, in summary, these criteria can be used to evaluate how security-oriented an architecture is:

- HTTP or HTTPS reliance
- Non-intrusive
- Non-dependency on plugins

2.2.3 Power

Most remote laboratories, in terms of power and of rich media, need only a webcam stream (e.g., [25]). Hybrid laboratories, however, have similar requirements at this respect to virtual laboratories, and require advanced graphics or other media such as audio. Many rely on 2D or even 3D graphics to provide or enhance the experience. It is also common to require relatively low-latency bidirectional communication with the server.

The power —understood in relation to the number of technical features that it can offer— of a hybrid lab architecture can be evaluated through the following:

- Hardware acceleration
- Audio & video

The network protocol that it is based on will also influence the capabilities of the lab significantly, in terms of power.

2.3 Previous experiences

The University of Deusto has had several previous experiences regarding hybrid laboratories. The original WebLab-FPGA-Watertank [26], [27], [28] was a WebGL-based hybrid lab that allowed students to program a physical FPGA board to control a virtual watertank. That proof-of-concept laboratory, which was based in WebGL, is a precursor to the implementation example that is described in Section 5. The experience suggested that hybrid laboratories do indeed have potential and that it is technically possible to control a virtual water tank with a real physical FPGA. At the same time, however, the difficulties encountered in the creation, implementation and deployment of the lab highlighted the potential benefits of a careful architectural design and analysis and the appropriateness of the criteria and requirements that have been described in this work. Also, it raised the question of whether users would find using such a lab satisfactory.

2.4 Difficulties and goals

Creating hybrid laboratories, especially multiplatform ones, is currently a significant challenge. Though remote laboratories and virtual laboratories on their own are well-established technologies, and much literature exists on the topic, hybrid laboratories are less thoroughly explored, at least in part because until recently graphic technologies in the browser were limited, and often relied on non-standard plugins [29]. There are very few well-documented models, architectures or guidelines for the creation of new hybrid laboratories; and the few hybrid laboratories that exist are often proofs-of-concept that are not maintained or made widely available to the public.

The goal of the architecture that is proposed in this work is thus to establish a model —which leverages the modern technologies and standards that have appeared— upon which hybrid laboratories can more easily and effectively be created. The architecture thus aims to be useful and reusable, and for that purpose an attempt will be made:

1) To rely on open standards and on free technologies.
2) To not be bound to specific Remote Laboratory Management Systems —even though the example remote lab will have to depend on one—.

2.5 Related works and architectures

Hybrid labs are relatively new and the literature dedicated to their architectures and models is scarce. There are, however, significant examples of hybrid labs —either in production or as proof-of-concepts—. In this section some of those labs and their architectures, extracted from their published works, from their websites or from other sources, are described.

2.5.1 eLab3D

The *eLab3D* system is an educative virtual world developed by researchers from the Polytechnic University of Madrid. It provides access to both virtual and hybrid labs [19], [30]. In this virtual environment, students can access a virtual campus, which offers labs of different kinds [31]. As of now, most of these labs are simulations (virtual labs) but there are also some electronics hybrid

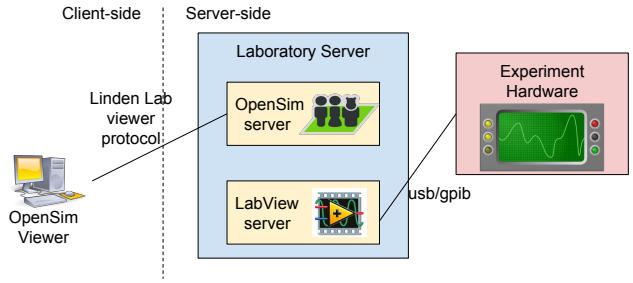

Fig. 2. Architecture of the eLab3D system

labs available. Students use seemingly virtual electronics equipment through the 3D virtual environment, and can then obtain *real* measurements (from real hardware). The system is designed with collaboration in mind, and students are able to see each other.

The *eLab3D* makes use of an OpenSim-based technology, which is a Java-based technology to help build virtual worlds. Heavily inspired on the SecondLife architecture and protocol, its feature-set is similar. There are several viewers available, though eLab3D recommends a specific one. That one, and most others, are Java-based desktop applications.

The students install Java and the OpenSim viewer, which is configured to connect to the eLab3D servers, they run it, authenticate against the eLab3D servers, and access the virtual campus, from which they can *move* or teleport to the lab they want to use.

Figure 2 summarizes the *eLab3D* architecture. There is a central *Laboratory Server* which runs different modules. The students use a desktop computer to connect directly, through an OpenSim Viewer client, to a standard (not modified) OpenSim server. A LabView server is connected to the hardware (electronic boards) through USB or GPIB. A custom HTTP-based Web Service component communicates the OpenSim server and the LabView server. Table 5 summarizes the technical characteristics of the *eLab3D* architecture.

2.5.2 iLab-TEALsim

The iLab Project [32] is a RLMS developed at MIT. TEALsim [20] is an open-source simulation toolkit to illustrate physical concepts. Using Wonderland [33] —an open-source toolkit for creating collaborative 3D virtual worlds— researchers have created a collaborative virtual environment for hybrid physics experiments [20], [34].

Figure 3 summarizes the architecture. The virtual environment, multi-user and collaboration features are provided by a Wonderland server. Clients connect to the server through a custom java-based Wonderland Client. The client includes TEALsim-based simulations, and lets students interact with LabVIEW –which provides access to the real hardware— through a VNC client that is included in Wonderland. Table 5 summarizes the technical characteristics of the project's architecture.

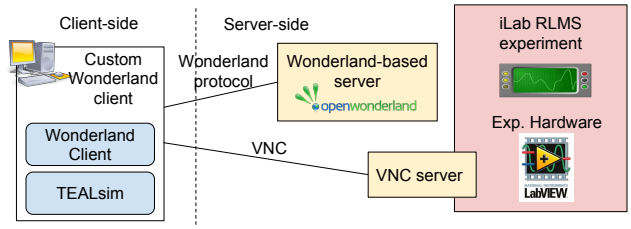

Fig. 3. Architecture of the iLab-TEALsim system

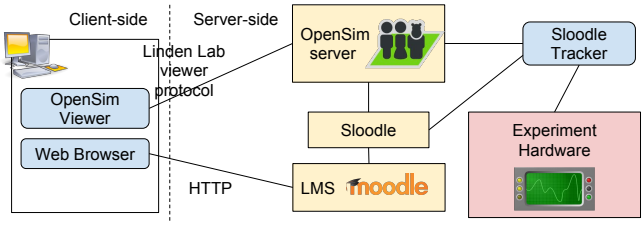

Fig. 4. Architecture of the virtual world version of Circuit Warz

2.5.3 Circuit Warz (virtual world versions)

The Ambient Intelligence and Virtual Worlds Research Team from the University of Ulster has published several related works in this area. In 2009 they describe the *Engineering Education Island* [35], which is an educational virtual world based on Second Life [36], [37]. Students can use the standard Second Life client to view a virtual increased-scale CPU, learn how it works and perform exercises. Their behaviour and results can be tracked through Sloodle and a purposedly-developed extension for Sloodle (*Sloodle Tracker*). Sloodle is an Open Source technology that integrates Second Life with Moodle [38], [39]. The Sloodle Tracker adds additional capabilities, such as tracking user position. The use of Second Life guarantees that several students can be present at the same time and interact with each other, though no further specific collaboration or gamification capabilities are provided at this stage.

Later works extend this design and name it *Circuit Warz*. The scheme that is described in [40] adds gamification capabilities and integrates real electronics hardware into the virtual world. Students compete in teams to achieve the highest score in resolving exercises, some of which include resolving circuits that are backed by real hardware. Figure 4 provides a high-level view of this latter scheme. The server is no longer Second Life itself; but OpenSim, an Open Source project which aims to provide mostly protocol-compatible capabilities, which is open and thus easier to build upon [41], [42]. Moodle, Sloodle and the specific-purpose Sloodle Tracker are still used to register and display user performance, but it is also used to integrate real hardware circuits into the virtual world. Thus, some of the experiments that may be offered to the user are hybrid: they interact with a

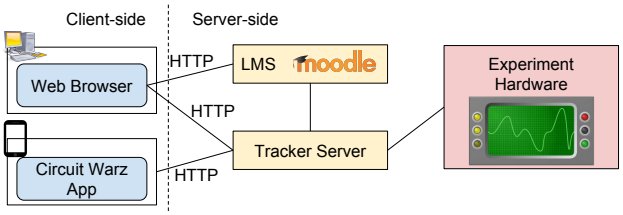

Fig. 5. Architecture of the Circuit Warz 2 system

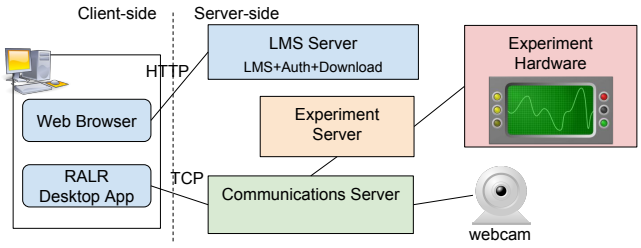

Fig. 6. Architecture of the ARL system

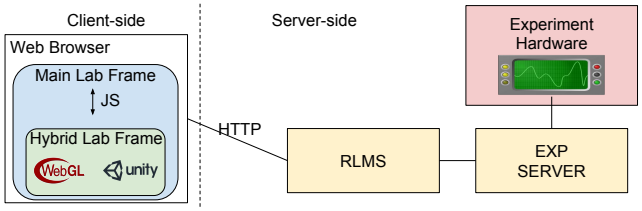

Fig. 7. Proposed architecture

circuit in the virtual world (for instance, by choosing the value of a resistance) and the system calculates the result through a real-hardware circuit. Table 5 summarizes the technical characteristics of the virtual-world version of the Circuit Warz architecture.

2.5.4 Circuit Warz (serious game versions)

Later alternative versions of Circuit Warz [18], [43] fundamentally changed the approach, architecture and technologies. The OpenSim-based architecture was abandoned in favour of using a custom-made Unity3D-based engine. This implies that there is no longer a Java dependency. Unity3D can easily be exported to different platforms, including desktop, mobile, or in recent versions, even the Web. However, it also implies that multi-user and other virtual-world features are no longer provided by default. This version of Circuit Warz also changes the goal and user experience. It is no longer a multi-user gamified virtual world, but instead a single-user serious game with a backstory in which the player needs to repair a space station by resolving circuits, which may be connected to the remote lab.

Figure 5 provides a high-level view of this new approach. The lab is no longer a multi-user virtual world —it is now a serious game with a single-user virtual environment— so it no longer relies on the OpenSim server or protocol. Instead, it uses a tracker server to register user actions into Moodle and to obtain results from the hardware circuits. A most significant advantage of this approach —and one of the main reasons of the architecture change— is that Unity3D can generate browser apps, mobile apps and (in later versions) even WebGL, so it can be accessed online from a web page without requiring Java-based viewers[2]. The serious game can be integrated into Moodle (by being displayed within a Moodle course in the browser) or can be played on its own as a mobile app. Table 5 summarizes the technical characteristics of the serious game version of the Circuit Warz architecture.

2.5.5 Augmented Remote Laboratory

A. Mejías, from the University of Huelva describes in his PhD thesis the *augmented laboratory* [11], [10]. The model he describes focuses on the use of augmented reality to *augment* electronics remote labs. The original design provides access to a programmable FPGA board and to a programmable robot. Through the virtual parts of the lab, modules can be added which can interact with that hardware bidirectionally. For example, users can program a robot which interacts with virtual obstacles and a virtual maze. Also, a FPGA-controllable virtual watertank is described, which is similar in purpose to the one that will be described in further sections of this paper.

The *augmented laboratory*, whose architecture is summarized in Figure 6, relies on Python-based augmented reality libraries and on a desktop applications. The student's workflow includes visiting the LMS server and downloading the required desktop software, authenticating through the Moodle-based LMS, and accessing the lab itself through the desktop software. Table 5 summarizes the technical characteristics of the original ARL architecture. It is noteworthy that newer, modified versions of this architecture exist, which rely on Java Applets and later, on JavaScript.

3 Proposed Architecture

The previous sections show that there are many different valid approaches to design hybrid labs. Most of them have different strengths, weaknesses and capabilities. A comparison can be found in Section 4. The architecture that we propose in this paper takes into account these previous works and ensures that the specific requirements that were listed in Section 2.2 are met.

Figure 7 shows an overview of the proposed architecture. The front-end of the architecture is web-based. The client is browser-based and it communicates with

2. The referenced implementation uses the *Unity3D web plugin*, which implies that users require that plugin to run the content. Currently, Unity3D can deploy to WebGL almost just as easily, so that should no longer be a limitation.

the server only through a single HTTP channel. In the server-side there is a Remote Laboratory Management System which hides the complexity of the actual physical deployment and which can provide access to different labs. The experiment server provides the logic of the experiment itself and controls the interaction with the hardware. This architecture and the design and technology choices are explained in more detail in the following sections. Table 5 summarizes the technical characteristics of the architecture.

3.1 Client-side

Two of the main goals that were established for this architecture were *universality* and *conformance* with the industry trends. Therefore, the architecture, at least from a client-side perspective, needs to be fully web-based. Nowadays this is expected for new applications unless there are restrictions that make it impossible, it is in line with the industry trends and offers many advantages in terms of development, deployment, maintenance, user experience, and security.

Porting online labs and other applications to the Web is not new. The issues this involves, applied to remote labs, were described in [24]. That work also summarizes the strengths and weaknesses of the different technologies that in 2009 were available to meet the different criteria. This paper updates some of them to reflect the technological changes that have since occurred, and partially bases its client-side architecture on the one that results from its analysis.

Also, more recent works in remote lab architectures, such as the one that describes the LaboREM [44] architecture, analyze the advantages and disadvantages of different front-end technologies and propose the use of web-based ones (HTML5) as the most appropriate technical choice.

3.1.1 Classification of technologies

The client of an online lab can be developed in a wide range of technologies. Two of the large groups they could be classified into are:

1) Desktop clients
2) Web-based clients

Desktop clients run *natively* in the user's OS, and are thus very powerful: they can generally do anything that the computer itself can do. Unfortunately, they also tend to be less portable and are often bound to a platform (be it, for example, the OS itself in the case of C or C++ applications, or the Java Virtual Machine in the case of Java applications). They are also more intrusive and thus less secure, which especially in the case of an online lab is a problem [24]. Online labs are hosted by Universities, who prefer to avoid exposing their students to such risks, and who may have some responsibility if their students' computers are breached because of them.

TABLE 2
Analysis of the different possible communication technologies

	Low latency	Requires plugins	Firewall traversal
Raw sockets (UDP)	✦✦✦✦✦	Yes	✦
Raw sockets (TCP)	✦✦✦✦	Yes	✦✦
Basic HTTP	✦	No	✦✦✦✦✦
AJAX (HTTP)	✦✦	No	✦✦✦✦✦
Web Sockets (HTTP)	✦✦✦	No	✦✦✦

Web-based clients run within a browser. By themselves they can run in any platform that has a compliant browser, they demand no installation and they are subject to the browser's security frame. However, traditionally certain features have been provided by browser plug-ins, which were less restrictive but which have many of the disadvantages of desktop software.

3.1.2 Communication technologies

An online lab that supports interaction requires bidirectional communication —it needs to be able to send the actions of the user as well as receive the response or the changing state of the experiment—. Table 2 summarizes the different methods that could be used. The *latency* and *firewall traversal abilities* have been analyzed comparatively (more stars is better). Latency refers to the amount of time that sending, completing and replying to a request will typically take. Firewall traversal ability refers to the fact that some protocols are blocked by firewalls more often than others.

The table shows that the lower the latency of a technology, the higher the chance of firewall issues. Particularly, protocols that are not based on HTTP will be blocked by most firewalls, especially if they rely on ports different from 80 (HTTP) and 443 (HTTPS). Basic HTTP and AJAX will have the least issues. Web Sockets are now also part of the HTTP standard and their usage is fast-growing, so eventually they should have good support, but, as of now, many firewalls and proxies do not support them and they often use blocked ports. Considering these characteristics, the client-side communication technology that the proposed architecture relies on will be *AJAX*.

3.1.3 Client technologies

Online labs are what used to be called RIAs (Rich Internet Applications). They depend on rich media such as graphics or even audio and video. Traditionally these capabilities were not supported by HTML. With new standards such as HTML5 and WebGL, this is no longer the case. The use or not of plug-ins and the support or not of certain features greatly affect universality and security. The choice of client-side technology is thus particularly important for the architecture.

Table 3 summarizes the capabilities of each technology. HTML4 refers to basic HTML, without HTML5-related features. *Canvas* and *SVG* refer, respectively, to the *canvas*

TABLE 3
Analysis of supported features in browser technologies

	HTML4	Canvas & SVG	WebGL	Flash	Java
2D acceleration	Limited	✓	✓	✓	✓
3D acceleration	✗	Limited	✓	✓	✓
Video	✗	✓	✓	✓	✓
Audio	✗	✓	✓	✓	✓
Plugin independency	✓	✓	✓	✗	✗
Mobile	✓	✓	✓	✗	✗

TABLE 4
Analysis of browser support for different technologies

	HTML4	Canvas	WebGL	Flash	Java
MS IE	✓	✓	✓(recent)	Plugin	✓
MS Edge	✓	✓	✓	Included	✗
Chrome	✓	✓	✓	Included	Dropped
Firefox	✓	✓	✓	Plugin	Dropping
Safari	✓	✓	✓	Included	✓
Mobile (misc)	✓	✓	Partial	✗	✗

and the *svg* elements, which are part of the HTML5 standard, and provide different ways to draw graphics in a web page. These graphics are not accelerated, so its 3D capabilities are limited. WebGL is a standard [22] by the Khronos group, which is related to HTML5 but not strictly part of it. It provides an OpenGL-based API through JavaScript, which gives access to fully-accelerated graphics in the browser. Flash and Java are technologies that have historically been used to provide advanced media features in web pages. Traditionally they have relied on external browser plugins. This has led to significant security issues. The May 2015 McAfee Labs Threats Report states that "Adobe Flash has long been an attractive attack surface for cybercriminals" [45], and that even today the number of Adobe Flash malware samples is growing. Because of this and other issues, the main browsers are dropping or limiting support for these technologies and for addons with native access in general. As of now, newest Chrome does not support them and Firefox is dropping support. Some browsers such as Chrome include their own Flash plugin. Support in different browsers is summarized in Table 4.

Other browser-based technologies that can provide these capabilities exist. *Microsoft Silverlight* is a technology by Microsoft which has capabilities that are similar to Flash. It has been much less popular, is less widely supported, and is no longer under development. *CSS3's* new features could be used to render 2D and 3D graphics, though they are not intended for this use. *Unity Web* is a plugin to run Unity 3D projects on a browser. Newest Unity versions can export to WebGL, which is expected to eventually replace it completely. *CSS 3D* additions to CSS are a powerful part of the HTML5 standard *LabView Remote Panels* can run LabView from a browser. It is a domain-specific technology and requires a plugin with native access in all browsers, in a way similar to Java Applets. It has no mobile support.

3.1.4 Choosing the client-side technologies

As established in previous sections, one of the main goals of the proposed architecture is **universality**. Relying on browser plugins would be contrary to that. This discards most technologies, including *Adobe Flash*, *Java Applets*, *LabVIEW Remote Panels*, *Microsoft Silverlight* or *Unity Web Player*. None of these technologies are consistently supported in mobile phones either, which is also a requirement. After these considerations, the client-side technologies that could be used are *basic HTML*, *HTML5 canvas*, or *WebGL*. It also discards raw sockets (UDP or TCP) as communication technologies, leaving *basic HTTP*, *AJAX* and *Web Sockets* as possible options.

Of the three communication technologies, *basic HTML* would be the most universal, because it is certainly supported by every browser and server and it can be deployed without any particular firewall consideration. However, nowadays AJAX is just as prevalent and much more flexible. It is also a proven technology, and the most popular communications technology in modern web applications. AJAX is also the choice of some recent remote lab architectures [46]. Thus *AJAX* or the more modern *web sockets* are technologies to consider. *Web sockets* are now part of the HTML standard and are already supported by most important browsers, including mobile ones, and by most HTTP servers. They can provide a bidirectional stream communication channel and a lower latency than request-based AJAX, so they would theoretically be perfect for an interactive remote lab application. Unfortunately, as of now many firewalls and institutional proxies do not yet support them. Because online labs are often deployed behind such systems, *web sockets* are also discarded in favour of *AJAX*, though this choice could likely be re-evaluated in the future, once *web socket* support increases.

Regarding RIA technologies, the most universal would, again, be HTML4. Unfortunately, as Table 3 shows, it is also very limited feature-wise, and not really an option for a virtual or hybrid lab that relies on interactive graphics. The realistic choice would thus be among *Canvas* and *WebGL*. *Canvas* is very widely supported. It is part of the core HTML5 standard, and is supported in every major browser, including mobile. *SVG* is similar, but generally slower for fast-changing animation, and declarative in nature. *WebGL* is more powerful, but its support is not as wide. Because this architecture aims to support labs with advanced graphics, WebGL will be the proposed graphics technologies. Its 3D acceleration capabilities grant it significantly more power, and there are many appropriate development tools and frameworks available. Even though it is slightly less supported than standard Canvas, modern desktop and mobile browsers support it already, and support is increasing steadily.

In summary, the proposed architecture will rely on *WebGL* to provide advanced multimedia features and on AJAX for client-server communication. As a result, online labs that use such an architecture:

- Will be supported in modern desktop and mobile browsers
- Will be able to be deployed behind most firewalls and institutional proxies
- Will be able to use modern accelerated 3D graphics and have real-time communication with a relatively low latency

3.1.5 Development technologies

One of the advantages of historically popular plugin-based systems such as *Adobe Flash*, *Java Applets* or *LabVIEW Remote Panels* is that they have a powerful associated set of development tools. The choice of WebGL is convenient at this respect, because being a popular and powerful standard, it has many tools available. One of such tools is *Unity 3D*. *Unity 3D* is a set of tools that can be used to develop 3D interactive applications and deploy them easily into many different platforms, one of which is WebGL.

3.2 Server-side

The client-server communication takes place through HTTP. The server-side will thus need to have a web server to handle these requests. Beyond that, internally, there are no particular restrictions. Because the proposed architecture aims to be extensible and to support different labs and hardware, it will rely on a Remote Laboratory Management System, which can act as a front-end and that can easily provide capabilities such as federation and integration with other lab frameworks. The RLMS (Remote Laboratory Management System) has a module for the particular lab, which also handles any lab-specific interaction which may be required.

Additionally, although the proposed architecture does not require it, implementations could also rely on the *smart device* paradigm [47]. The *smart device* paradigm aims to decouple the client and the remote lab by establishing a common specification that is shared among different remote labs. This could be used to allow the laboratories to easily use the same virtual model with different types of remote hardware, or even to 'plug and play' remote hardware. It would also help integration into Massive Open Online Laboratories (MOOLs) [48].

4 Comparison with other architectures

As described in Section 2.5 several different architectures exist and have been used for the creation of different types of hybrid online labs. Table 5 compares the characteristics of these architectures and of the proposed one.

A key aspect that the table shows is that the chosen technologies have a very significant effect on *universality* and *security*. The architectures that are based on Opensim (eLab3D, CW1) or on specific desktop technologies (ARL) tend to be less universal because they tend to rely on native desktop applications which do not run on a browser, are intrusive —they require native access to the Operative System—, do not run on mobile devices and rely on non-standard protocols and ports which are often blocked by institutional firewalls. On the contrary, those that are designed for the newer web standards, including the proposed architecture, can provide advanced features without compromising universality or security.

The main advantage of the proposed architecture over some of the listed ones is thus that it provides significantly greater *universality* and *security*, which were indeed the main goals set in Section 2.2.

It can be observed that in these terms the proposed architecture is similar to the CW3 architecture. CW3 was designed with similar goals and also relies on modern Web standards to provide those features without compromising *universality* or *security*. The approach, however, is significantly different. While CW3 is a single-player serious game that integrates domain-specific remote hardware (circuits to be solved) and whose interaction with the hardware is *batch* in nature —the interaction is discrete; a request is sent to the hardware, which returns the result once calculated— the proposed architecture is designed for augmented remote labs which are interactive and most likely hosted within a RLMS environment. The virtual environment is added upon a traditional remote lab experiment (which is thus *augmented*) and a continuous interaction takes place between the virtual environment and the real hardware: users reserve the hardware for a time, and through that time the virtual environment affects the physical model, and the physical model affects the virtual environment.

Which approach is more appropriate to teach a specific subject (an augmented remote lab, a serious game, or a virtual world) is an interesting question but it is most likely dependent on the subject itself, on the preferences of the teacher and of the students, on the particular context and on other factors. It is thus beyond the scope of this paper.

5 Implementation example: Watertank FPGA laboratory

5.1 Limitations of some traditional remote labs

Many remote labs that exist provide access to a particular hardware development board, such as a FPGA device [49], [50], a PLD [51], a PLC [52] or a microcontroller [53]. These labs are useful because these boards require specific training, and are themselves relatively expensive and hard to setup. The standard FPGA remote labs at the University of Deusto [54] provides access to a Xilinx FPGA Development Board. Students design the VHDL logic and can then program it into the board, interact with it through virtual switches and other inputs

TABLE 5
Comparison of the architecture characteristics

Approach		elab3D	iLab-TEALsim	CW1	CW3	ARL	Proposed
	Application type	Virtual world	Virtual world	Virtual world	Serious game	Augmented RL	Augmented RL
	Remote lab type	RLMS	RLMS	Domain-specific	Domain-specific	Domain-specific	RLMS
	Interaction type	Batch	Interactive	Batch	Batch	Interactive	Interactive
Universality	Desktop browsers	✗	✗	✗	✓	✗	✓
	Mobile browsers	✗	✗	✗	✓	✗	✓
	HTTP/HTTPS	✗	✗	✗	✓	✓	✓
	Built-in tech.	✗	✗	✗	✓	✗	✓
Security	HTTP/HTTPS	✗	✗	✗	✓	✓	✓
	Non-intrusive	✗	✗	✗	✓	✗	✓
	Built-in tech.	✗	✗	✗	✓	✗	✓
Power	Hardware acceleration	✓	✓	✓	✓	✓	✓
	Audio & video	✓	✓	✓	✓	✓	✓
	Network protocol	Opensim	Wonderland	Opensim	HTTP	Custom	HTTP

that are mapped to real, physical ones, and control a set of LEDs that act as outputs and that can be seen remotely through a webcam.

Although this scheme is by itself very useful, and has been successfully used by the University of Deusto for years, it has some limitations. The outputs are very simplistic. Although the hypothetical exercises that students need to resolve often include devices such as heaters, storage tanks, or industrial devices, to know whether their logic works they need to imagine its results through the LEDs, which are the only actual outputs available. As a result, exercises tend to not be particularly engaging. Additionally, it is often very hard to get a good idea of how a particular logic would perform under real conditions by just seeing the outputs.

The most straightforward solution would be to connect the FPGA device to industrial hardware. Students would be able to program the logic and see how it behaves when applied to a real model. This approach is taken by some remote labs, but it also has several drawbacks. Real industrial hardware and realistic scale models tend to be expensive to purchase. If a straightforward approach is taken, each industrial model will need to be linked to its controller. As a result, to give support to several models, several controller boards would be required as well. Those can normally be used by a single user at the same time. They also require a significant amount of physical space and maintenance efforts. As a result, the costs are often too high, especially because some industrial hardware can also be a security risk.

5.2 The FPGA-Watertank laboratory

The FPGA-Watertank labs aims to use the hybrid labs architecture described in this paper to provide a hybrid lab that partially resolves some of the aforementioned issues. The lab provides access to a real FPGA board which can control virtual industrial models rather than real, physical ones. This has several advantages. The user

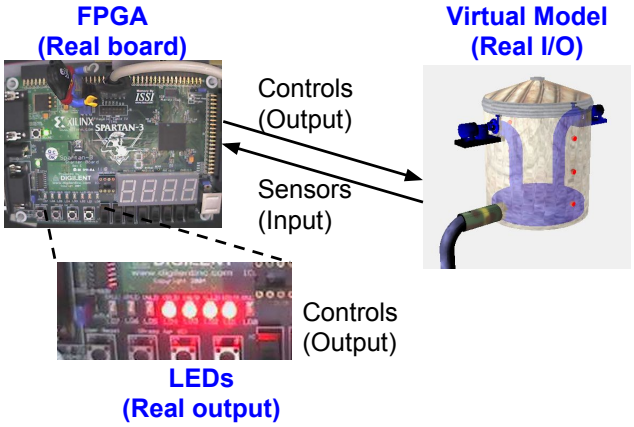

Fig. 8. Relationship between components

can see how their logic behaves under potentially realistic conditions, but at the same time the costs are driven down. Developing and maintaining a virtual industrial model tends to be much cheaper than purchasing and maintaining real industrial equipment. Also, a single controller board could potentially be used with an arbitrary number of different virtual models (e.g., engines, semaphores, robots). As a result, with several users and models, instances of the controller board could be used interchangeably and a much lower number of boards would be needed.

5.3 Components and scheme

Figure 8 shows how the main components of the lab relate to each other. The controller is a Xilinx FPGA board, which is the same controller board that is used for the standard FPGA remote lab at the University of Deusto. This board still has the LEDs as outputs, and these LEDs can still be watched by the students through

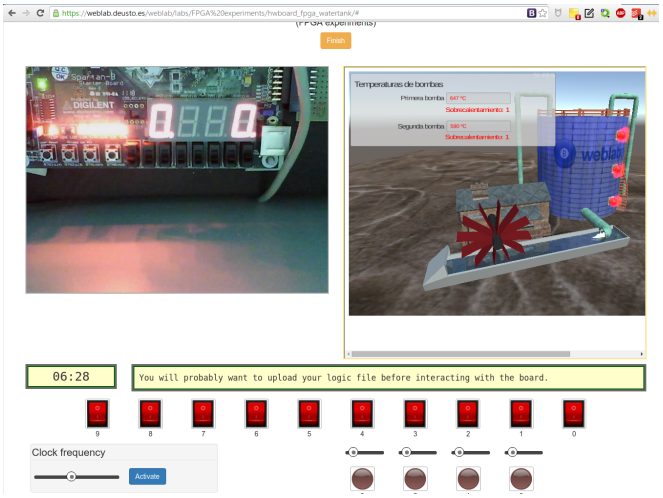

Fig. 9. Watertank-FPGA laboratory running in Chrome

a webcam. However, these outputs are also mapped to a virtual industrial model, which in this case is an industrial water tank. The virtual water tank has two actuators —the water pumps, mapped to the LEDs— . When the users' logic turns on a physical LED on the board, the virtual water pump will simultaneously be turned on. The virtual water tank also has several sensors: three water level ones, and two overheating ones. These sensors are mapped into the physical board as input signals. Finally, the water tank model also has a water output whose rate randomly increases and decreases, to simulate a varying water demand. Through this scheme, the real, physical hardware board can be used to control the virtual model.

Figure 9 shows a screenshot of the Watertank-FPGA lab running in the Chrome browser. The lab is integrated within the WebLab-Deusto [54] RLMS. In the upper left a webcam stream displays the real, physical board that is running the logic that the student has provided. In the upper right the virtual environment, created with Unity3D, is rendered in WebGL, without requiring any plugin or addon. In the lower side, virtual controls can be used to interact with the board.

5.4 Software design

The lab software is developed according to the architecture that was described in this paper. The client has two parts. First, a web based framework —based on Angularjs— that is integrated within the WebLab-Deusto RLMS and through which students can see the physical board, interact with it and send their logic. Second, a WebGL view that integrates within that aforementioned framework, which displays the industrial model that the board is controlling. This scheme is designed to provide a consistent user experience across user devices —PCs, mobiles and tablets— and controlled virtual devices —though at the time only the watertank is provided—.

Because the virtual model relies only on standard technologies such as WebGL, it is currently compatible with all modern browsers, including mobile ones. Because raw WebGL development is costly and time-consuming —WebGL is essentially an OpenGL for the Web—, the Unity3D toolset has been used. Unity3D provides an integrated graphical editor that greatly accelerates the development of such applications, and which relies on C#, a complex editor, and other technologies. As an additional advantage, applications can easily be built to different platforms, so apart from WebGL, it would be straightforward to build it as a native mobile application.

The lab server is also integrated with WebLab-Deusto, a RLMS. In WebLab-Deusto, labs such as this one have a separate *experiment server*, which could be implemented in any language but which is most often implemented in Python. Though the graphics are client-side, the virtual model simulation itself runs server-side here. This module is also the one with interacts with the hardware —the physical FPGA controller board—.

5.5 Technical evaluation

A technical analysis has been performed to verify that the lab that implements the architecture conforms to the requirements that were previously established and that it supports important characteristics.

1) *The lab runs on modern browsers without requiring plugins:* The lab relies only on HTML5 and WebGL. It has been successfully tested under recent versions of Google Chrome, Mozilla Firefox, Internet Explorer, Microsoft Edge, Safari, Mobile Firefox (*webgl needed to be explicitly enabled*) and Mobile Chrome.
2) *The lab can be deployed behind institutional proxies and firewalls:* The lab has been tested in the University of Deusto and has been deployed behind its proxy and firewall. It works through standard HTTP traffic in port 80, or through HTTPS in port 443.
3) *Deploying the virtual models is straightforward:* Client-side the virtual models can be developed in Unity3D and built to WebGL.
4) *The system relies on open-source or free technologies:* All the technologies, including the tools, that have been used are open-source or free. Specifically, the significant technologies are: GNU/Linux —open source, for the server-side deployment—, Python —open-source, for the server-side development—, Unity3D —closed-source but free—, HTML5 and WebGL —open standards—, WebLab-Deusto —open-source, a RLMS—.

6 Prospects and Satisfaction Survey

6.1 Procedure

This study, conducted in December 2015, aims to evaluate whether the model has indeed a significant potential for education. Although online labs are sometimes evaluated by measuring their educational effectiveness

TABLE 6
First and second questionnaires

First questionnaire	
Q1.1	*How much interest would you have in using each of the following technologies to promote your learning?* 4 point scale (1 to 4) for pen and paper, simulators, remote labs and augmented remote labs
Q1.2	*Do you think that using a remote lab (normal or augmented) can help your learning?* 4 point scale (1 to 4) from *helps very little* to *helps very much*
Second questionnaire	
Q2.1	*How much value do you think that using the lab has added to your learning?* 4 point scale (1 to 4) from *very little* to *very much*
Q2.2	*How could it have added more value?* Free comments
Q2.3	*How much value does the virtual environment add to the remote lab?* 4 point scale (1 to 4) from *very little* to *very much*
Q2.4	*Would you have preferred to have used a different system?* Multiple choice (Satisfied with the ARL, pen and paper, simulator, standard RL)
Q2.5	*How much interest would you have in using each of the following technologies to promote your learning?* 4 point scale (1 to 4) for pen and paper, simulators, remote labs and augmented remote labs
Q2.6	*How did the lab work? Check the statements you agree with* Check 1 or more: *There were technical issues*; *Technically worked with no issues*; *I barely learned anything*; *I am satisfied with what I have learned*

—for instance, by comparing the knowledge gain against other methods such as hands-on labs— this would not be particularly appropriate for this purpose because it aims to evaluate the model itself and not the particular example implementation. Instead, a survey-based study has been conducted. The goal is to ensure that the students find such a model useful and attractive, and that they are reasonably satisfied after using a prototype of a lab based on that model. The surveys have relied on a 4-point Likert-style scale with no neutral.

Firstly, students that had never used or been exposed to a remote lab were briefly described different types of online labs: remote labs, virtual labs, augmented remote labs, simulators. A first survey measured the initial interest that they had in using those technologies to learn. The students were not informed of the steps that would follow or of whether they would later use any of these technologies. Once the survey was filled, the students were asked to solve an electronics problem using the FPGA-Watertank lab. Finally, they took a second survey to measure their satisfaction and whether they believe to have learned effectively. The content of both surveys is included in Table 6, translated to English. The original questionnaires were in Spanish.

All the students in class returned the questionnaire, but not all questions were always filled. Empty or incomplete responses to individual questions were discarded.

6.2 Participants

The participants of the study were 58 first-year students of the Double Degree in Business Management and Industrial Engineering. They were all enrolled in a digital electronics course, and the test was conducted within that subject's context. They completed the tests in-class. None of them had used a remote lab before.

TABLE 7
Results of the initial survey's Q1.1: interest on the technologies (n=56)

Technology	Mean	S.D.
Pen and paper	2.500	0.981
Simulators	3.107	0.795
Remote Labs	3.250	0.543
Augmented Remote labs	3.357	0.789

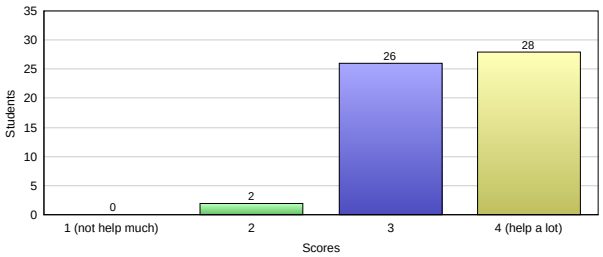

Fig. 10. Results of Q1.2: perceived learning potential of RLs / ARLs (n=56; mean: 3.464)

6.3 Initial test

The purpose of the first test was to assess the initial interest of the students in each type of tool, and particularly to determine how the interest and learning expectation with the lab model that is proposed in this work compares against other approaches.

That survey had only two questions. The first to measure their initial interest, and the second to measure their expectations regarding remote labs in general. Table 7 shows the result of the first question, which asks students to grade the interest they have in using different learning

TABLE 8
Results of the posterior survey Q2.5 — interest on the technologies (n=53; scale: 1-4)

Technology	Mean	S.D.
Pen and paper	2.604	0.809
Simulators	3.189	0.646
Remote Labs	3.189	0.585
Augmented Remote labs	3.359	0.676

TABLE 9
Results of the second survey: Q2.1 and Q2.3 — perceived value of learning and of the augmented features (n=57; scale: 1-4)

Question	Mean	S.D.
Perceived learning value (Q2.1)	3.614	0.522
Perceived augmented features value (Q2.3)	3.158	0.615

technologies in a four-grades scale. Figure 10 shows the result of the second question, which asks the students whether they believe that a normal or augmented remote lab can help their learning, and which also uses a four-grades scale.

6.4 Second test

A second survey was conducted after the students had used the FPGA-Watertank lab. 58 students returned the questionnaire. Questions that were not filled were discarded. Its main goals were the following:

- To measure their satisfaction with the lab.
- To check whether their opinion on the potential of the different technologies had changed after using the ARL.
- To evaluate whether they believe the lab model is useful.

Table 8 shows the interest in different technologies (Q2.5), which is exactly the same question as the Q1.1 in the first survey (see table 7). Table 9 shows the results of Q2.1 and Q2.3, which asks students to grade in a scale of 1 to 4 how they value the contribution of the lab to their learning, and how much they value the addition of a virtual environment to the remote lab.

Q2.4 asked the students whether they would have preferred to use a different technology instead of an Augmented Remote Lab. 43 students out of 56 who answered the question (76.8

Q2.6 asked the students to check the statements that they agreed with. 40 students out of 55 who answered the question (72.7

Additionally, the questionnaire included a space (Q2.2) for students to include free comments and suggestions to improve the system. Some of the provided ones were the following:

"I agree with the system"

"I think that the system is fine as it is"

"I think it's fine as it is, and it is possibly what has helped me the most to study the subject"

Several students also expressed that they would have liked more practical sessions.

6.5 Discussion

The surveys suggest that the students are definitely interested in learning through tools such as simulators, RLs, and ARLs. Of those, the ARL ranks the highest. This suggests that the architecture that is described in this work to facilitate the creation of such labs not only meets the technical requirements that are set, but can lead to labs that are found useful and possibly more engaging by students. It is remarkable that the opinion of the students did not vary significantly after using the Augmented Remote Laboratory: the interest that they had in using each technology remained mostly the same.

The other questions of the second survey also show that the students, in general, find the lab satisfactory. The students believe that it added value to their learning (Q2.1). Determining whether this perception is true, and to what extent, would require further educational experiments. Also, they find that augmenting a RL with a Virtual Environment is valuable (Q2.3). Questions that measure their general satisfaction (Q2.4, Q2.5, Q2.6) are also positive.

7 Conclusion and Future Work

This paper has described a new model for the creation of Augmented Remote labs which leverages recent advances in web technologies to support the advanced RIA features that hybrid online labs require (3D graphics, communications) while maximizing universality. By relying on HTML5 and WebGL, which are available on all modern browsers, it is possible to support the required technical features without using non-standard plugins (such as Java or Adobe Flash) that are not necessarily deployed everywhere and that, in fact, are not supported in many mobile systems. The concrete implementation that has been described suggests that the architecture meets the established requirements. Also, the tests conducted with students suggest that the described hybrid lab model is interesting and engaging to the users, and that it has educational potential.

In the future, several lines of work remain open. Hybrid labs are relatively new, and their possibilities are still being explored. The possibility of adding features such as multi-user collaboration, gamification, or new input mechanisms to the architecture will be explored. Additionally, once the presence of web sockets is wider, their suitability as a communications technology for interactive labs should be re-evaluated. Also, the interest on *smart devices* and MOOLs is growing, so exploring how those paradigms can be incorporated into laboratories that use the proposed model remains an interesting line of work.

An additional benefit of hybrid labs based on this model or on similar ones, which may be explored in the future, is that they have a very complete knowledge of their own state. For instance, in the implementation example, the lab keeps full track of the water level and of the board's output state. Thus, it is relatively easy to keep a full record of user activities, which could be used to augment the learning experience even further through learning analytics, automatic assessment, or even intelligent tutors based on that real-time data.

ACKNOWLEDGMENTS

This work has received financial support by the Department of Education, Language policy and Culture of the Basque Government through a Predoctoral Scholarship granted to Luis Rodriguez-Gil.

REFERENCES

[1] S. Dormido Bencomo, "Control learning: present and future," *Annu. Rev. in Control*, vol. 28, no. 1, pp. 115–136, 2004.

[2] M. Auer, A. Pester, D. Ursutiu, and C. Samoila, "Distributed virtual and remote labs in engineering," in *2003 IEEE Int. Conf. Ind. Technol.*, vol. 2. IEEE, 2003, pp. 1208–1213.

[3] C. S. Tzafestas, N. Palaiologou, and M. Alifragis, "Virtual and remote robotic laboratory: comparative experimental evaluation," *IEEE Trans. Educ.*, vol. 49, no. 3, pp. 360–369, 2006.

[4] 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.

[5] J. Ma and J. V. Nickerson, "Hands-on, simulated, and remote laboratories: A comparative literature review," *ACM Computing Surveys (CSUR)*, vol. 38, no. 3, p. 7, 2006.

[6] Z. Nedic, J. Machotka, and A. Nafalski, "Remote laboratories versus virtual and real laboratories," in *Frontiers in Education, 2003. FIE 2003 33rd Annu.*, vol. 1. IEEE, 2003, pp. T3E–1.

[7] J. V. Nickerson, J. E. Corter, S. K. Esche, and C. Chassapis, "A model for evaluating the effectiveness of remote engineering laboratories and simulations in education," *Comp. & Edu.*, vol. 49, no. 3, pp. 708–725, 2007.

[8] E. E. Toth, B. L. Morrow, and L. R. Ludvico, "Designing blended inquiry learning in a laboratory context: A study of incorporating hands-on and virtual laboratories," *Innovative Higher Education*, vol. 33, no. 5, pp. 333–344, 2009.

[9] L. Gomes and S. Bogosyan, "Current trends in remote laboratories," *IEEE Trans. Ind. Electron.*, vol. 56, no. 12, pp. 4744–4756, 2009.

[10] J. M. Andujar, A. Mejias Borrero, and M. A. Marquez, "Augmented reality for the improvement of remote laboratories: an augmented remote laboratory," *IEEE Trans. Educ.*, vol. 54, no. 3, pp. 492–500, 2011.

[11] A. Mejías Borrero, "Aportaciones a los laboratorios remotos en los estudios de ingenieriá. interaccioń de elementos virtuales y reales mediante realidad aumentada: El laboratorio remoto aumentado," Ph.D. dissertation, Universidad de Huelva, December 2011.

[12] H. Vargas, J. Sánchez-Moreno, S. Dormido, C. Salzmann, D. Gillet, and F. Esquembre, "Web-enabled remote scientific environments," *Comp. in Sci. & Eng.*, vol. 11, no. 3, pp. 36–46, 2009.

[13] H. Vargas, J. Sanchez Moreno, C. A. Jara, F. A. Candelas, F. Torres, and S. Dormido, "A network of automatic control web-based laboratories," *IEEE Trans. Learning Technol.*, vol. 4, no. 3, pp. 197–208, 2011.

[14] N. Duro *et al.*, "An integrated virtual and remote control lab," *Computing in Science & Eng.*, vol. 10, p. 50, 2008.

[15] J. García-Zubia *et al.*, "Developing a second-life-based remote lab over the weblab-deusto architecture," in *Proc. REV 2010 Conf., Stockholm, Sweden*, pp. 171–176.

[16] M. Callaghan, K. McCusker, J. L. Losada, J. Harkin, and S. Wilson, "Using game-based learning in virtual worlds to teach electronic and electrical engineering," *IEEE Trans. Ind. Informat.*, vol. 9, no. 1, pp. 575–584, 2013.

[17] ——, "Circuit warz, the games; collaborative and competitive game-based learning in virtual worlds," in *Proc. REV 2012 Conf.* IEEE, pp. 1–4.

[18] M. Callaghan, N. McShane, and A. G. Eguiluz, "Using game analytics to measure student engagement/retention for engineering education," in *Proc. REV 2014 Conf., Porto*. IEEE, pp. 297–302.

[19] A. Carpeno, S. Lopez, and J. Arriaga, "Using remote laboratory elab3d for a broader practical skills training in electronics," in *Proc. REV 2014 Conf., Porto*. IEEE, pp. 98–99.

[20] B. Scheucher, P. H. Bailey, C. Gütl, and J. V. Harward, "Collaborative virtual 3d environment for internet-accessible physics experiments." *iJOE*, vol. 5, no. S1, pp. 65–71, 2009.

[21] "Html5 specification," W3, Tech. Rep., Jan. 2016, available: https://www.w3.org/TR/html5.

[22] "Webgl specification," Khronos WebGL Working Group, Tech. Rep., Oct. 2014, available: https://www.khronos.org/registry/webgl/specs/1.0/.

[23] Unity3d website. Last accessed 2016-01-19. [Online]. Available: https://unity3d.com

[24] 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, 2009.

[25] M. T. Restivo, J. Mendes, A. M. Lopes, C. M. Silva, and F. Chouzal, "A remote laboratory in engineering measurement," *IEEE Trans. Ind. Electron.*, vol. 56, no. 12, pp. 4836–4843, 2009.

[26] J. García-Zubía, I. Angulo, L. Rodriguez-Gil, P. Orduña, O. Dziabenko, and M. Güenaga, "Boole-weblab-fpga: Creating an integrated digital electronics learning workflow through a hybrid laboratory and an educational electronics design tool," *Int. J. of Online Eng. (iJOE)*, vol. 9, no. S8, pp. pp–19, 2013.

[27] L. Rodriguez-Gil, P. Orduña, J. García-Zubia, I. Angulo, and D. López-de Ipiña, "Graphic technologies for virtual, remote and hybrid laboratories: Weblab-fpga hybrid lab," in *Proc. REV 2014 Conf., Porto*. IEEE, 2014, pp. 163–166.

[28] L. Rodriguez-Gil, J. García-Zubia, P. Orduña, I. Angulo, and D. López-de-Ipiña, "Hybrid laboratory for rapid prototyping in digital electronics," in *Online Experimentation: Emerging Technologies and IoT*, M. Restivo, A. Cardoso, and A. M. Lopes, Eds. Barcelona: International Frequency Sensor Association Publishing, 2015, ch. 9, pp. 179–194.

[29] J. Behr, P. Eschler, Y. Jung, and M. Zöllner, "X3dom: a dom-based html5/x3d integration model," in *Proc. 14th Int. Conf. 3D Web Technol.* ACM, 2009, pp. 127–135.

[30] S. Lopez, A. Carpeno, and J. Arriaga, "Laboratorio remoto elab3d: Un mundo virtual inmersivo para el aprendizaje de la electrónica," in *Proc. REV 2014 Conf., Porto*. IEEE, pp. 100–105.

[31] S. Lopez, A. Carpeno, and J. Arriaga, "Remote laboratory elab3d: A complementary resource in engineering education," *Tecnologias del Aprendizaje, IEEE Revista Iberoamericana de*, vol. 10, no. 3, pp. 160–167, 2015.

[32] V. J. Harward *et al.*, "The ilab shared architecture: A web services infrastructure to build communities of internet accessible laboratories," *Proc. IEEE*, vol. 96, no. 6, pp. 931–950, 2008.

[33] J. Kaplan and N. Yankelovich, "Open wonderland: An extensible virtual world architecture," *IEEE Internet Comput.*, vol. 15, no. 5, pp. 38–45, 2011.

[34] F. R. dos Santos, C. Guetl, P. H. Bailey, and V. J. Harward, "Dynamic virtual environment for multiple physics experiments in higher education," in *Global Edu. Eng. Conf. (EDUCON)*. IEEE, 2010, pp. 731–736.

[35] M. Callaghan, K. McCusker, J. Lopez Losada, J. Harkin, and S. Wilson, "Engineering education island: Teaching engineering in virtual worlds," *Innovation in Teaching and Learning in Information and Computer Sciences*, vol. 8, no. 3, pp. 2–18, 2009.

[36] Second life website. Last accessed 2015-12-18. [Online]. Available: http://secondlife.com

[37] M. N. K. Boulos, L. Hetherington, and S. Wheeler, "Second life: an overview of the potential of 3-d virtual worlds in medical and health education," *Health Information & Libraries Journal*, vol. 24, no. 4, pp. 233–245, 2007.

[38] Sloodle website. Last accessed 2015-12-18. [Online]. Available: http://sloodle.org

[39] J. W. Kemp, D. Livingstone, and P. R. Bloomfield, "Sloodle: Connecting vle tools with emergent teaching practice in second life," *British J. Edu. Technol.*, vol. 40, no. 3, pp. 551–555, 2009.

[40] K. McCusker, M. Callaghan, J. Harkin, and S. Wilson, "Intelligent assessment and learner personalisation in virtual 3d immersive environments," in *European Conf. on Games Based Learning*. Academic Conf.s Int. Limited, 2012, p. 591.

[41] Opensim website. Last accessed 2015-12-18. [Online]. Available: http://opensimulator.org

[42] S. L. Delp *et al.*, "Opensim: open-source software to create and analyze dynamic simulations of movement," *IEEE Trans. Biomed. Eng.*, vol. 54, no. 11, pp. 1940–1950, 2007.

[43] M. Callaghan and M. Cullen, "Game based learning for teaching electrical and electronic engineering," in *Advances in Computer Science and its Applications*. Springer, 2014, pp. 655–660.

[44] F. Luthon and B. Larroque, "Laborem—a remote laboratory for game-like training in electronics," *IEEE Trans. Learning Technol.*, vol. 8, no. 3, pp. 311–321, 2015.

[45] "McAfee Labs Threats Report, May 2015," Intel Security, Tech. Rep., May 2015.

[46] M. Kaluz, J. Garcia-Zubia, M. Fikar, and L. Cirka, "A flexible and configurable architecture for automatic control remote laboratories," *IEEE Trans. Learning Technol.*, vol. 8, no. 3, pp. 299–310, 2015.

[47] C. Salzmann and D. Gillet, "Smart device paradigm standardization for online labs," in *Global Eng. Edu. Conf. (EDUCON)*, 2013.

[48] C. Salzmann, D. Gillet, and Y. Piguet, "Mools for moocs: A first edx scalable implementation," in *Proc. REV 2016 Conf.* IEEE, pp. 246–251.

[49] W. M. El Medany, "Fpga remote laboratory for hardware e-learning courses," in *Computational Technologies in Electrical and Electronics Engineering, 2008. SIBIRCON 2008. IEEE Region 8 Int. Conf. on*. IEEE, 2008, pp. 106–109.

[50] J. Lobo, "A remote reconfigurable logic laboratory for basic digital design," in *1st Experiment@ Int. Conf.–Remote & Virtual Labs–exp. at*, vol. 11, 2011.

[51] J. Garcia-Zubia *et al.*, "Application and user perceptions of using the weblab-deusto-pld in technical education," in *Frontiers in Education Conf. (FIE), 2011*. IEEE, 2011, pp. GOLC1–1.

[52] E. Besada-Portas, J. A. Lopez-Orozco, L. De La Torre, and J. M. de la Cruz, "Remote control laboratory using ejs applets and twincat programmable logic controllers," *IEEE Trans. Educ.*, vol. 56, no. 2, pp. 156–164, 2013.

[53] M. Gilibert *et al.*, "80c537 microcontroller remote lab for e-learning teaching," *Int. Journal of Online Engineering (iJOE)*, vol. 2, no. 4, 2006.

[54] P. Orduña, J. Irurzun, L. Rodriguez-Gil, J. G. Zubía, F. Gazzola, and D. López-de Ipiña, "Adding new features to new and existing remote experiments through their integration in weblab-deusto." *iJOE*, vol. 7, no. S2, pp. 33–39, 2011.

Luis Rodríguez-Gil is a PhD student at DeustoTech Internet group. He finished his studies of a double degree in Computer Eng. and Industrial Org. Eng. in 2013, and he completed a MSc in Information Security in 2014. Since 2009, he has been involved in the WebLab-Deusto Research Group, collaborating in the development of the WebLab-Deusto RLMS. He has published several peer-reviewed publications and contributed to some Open Source projects.

Javier García-Zubia (M'08-SM'11) holds a PhD in Computer Sciences by the University of Deusto. He is a full professor in the Faculty of Engineering of the University of Deusto, Spain. His research interest is focused on remote laboratory design, implementation and evaluation. He is the leader of the WebLab-Deusto research group.

Pablo Orduña (M'05) is a full time researcher and project manager at the MORElab Research Group at DeustoTech Internet. He finished Computer Engineering in 2007 and his PhD in 2013 in the University of Deusto. During his PhD he was a visiting researcher twice for 6 weeks each, in the MIT CECI in 2011 and UNED DIEEC in 2012. Since 2004, he has also been involved in the WebLabDeusto Research Group, leading the design and development of WebLab-Deusto.

Diego López-de-Ipiña is an associate prof. and P.R. of MORElab group and director of DeustoTech Internet unit, and of the PhD program within the Faculty of Eng. of the University of Deusto. He received his PhD from the University of Cambridge in 2002. Responsible for several modules in the BSc and MSc in Comp. Eng. degrees, he is interested in pervasive computing, IoT, semantic service middleware, open linked data and social data mining. He is taking and has taken part in several big consortium-based research european (IES CITIES, MUGGES, SONOPA, CBDP, GO-LAB, LifeWear) and Spanish projects, and has more than 70 publications in relevant int. conf. and journals, including more than 25 JCR-indexed articles.

This dissertation was finished writing in Bilbao on June 26^{th}, 2017