UNIVERSIDAD DE DEUSTO

TRANSITIVE AND SCALABLE FEDERATION MODEL FOR REMOTE LABORATORIES

Tesis doctoral presentada por Pablo Orduña

Dirigida por Dr. Javier García Zubía

Bilbao, Abril de 2013

UNIVERSIDAD DE DEUSTO

TRANSITIVE AND SCALABLE FEDERATION MODEL FOR REMOTE LABORATORIES

Tesis doctoral presentada por Pablo Orduña
dentro del Programa de Doctorado en Sistemas de Información
Dirigida por Dr. Javier García Zubía

El doctorando El director

Bilbao, Abril de 2013

Author: Pablo Orduña
Advisor: Dr. Javier Garcia Zubia

The following web-page address contains up to date information about this dissertation and related topics:
http://paginaspersonales.deusto.es/porduna/phd/
Text printed in Bilbao
First edition, April 2013

A mis padres y a Sonia

Abstract

Education –as everything– is changing, is being digitalized. The Internet has become yet another tool for students. It does not only enhance the availability of educational resources, but also enables the collaboration among institutions and countries, transforming education –both formal and informal–.

In the STEM fields –Science, Technology, Engineering and Mathematics–, students need to access not only contents, but also laboratories. An experiment can be considered as a conversation with Nature: students ask and the Nature answers. The only way to learn the Nature's language is by experimenting, in a traditional way or through the Internet. That said, the design and maintenance of laboratories is expensive and complicated, which slows down their development and use in educational institutions, especially in those countries with limited resources.

In the beginning, and as a solution to the lack of availability, laboratories started being simulated with software tools. With the growth of the Internet, the research community has placed efforts to digitalize these laboratories, making them remote, enabling students to access real equipments through the Internet, where the conversation with Nature goes through the network. Once laboratories are available through the Internet, it is possible to share them with other educational institutions, as any other digitalized resource. Sharing laboratories and experiences between countries and institutions needs a proper technical support: it needs federation protocols that make this process automated.

The focus of this dissertation is to explore and evaluate a new model for scaling up this sharing –enabling a higher number of concurrent users to access equipment distributed among multiple institutions–, and managing more complex scenarios than those described in the literature –re-sharing or subcontracting accesses to laboratories, or distributing them through low cost devices–. Both the federation model and the technical viability have been implemented in the WebLab-Deusto remote laboratory management system, also described in this dissertation.

Resumen

La educación –como todo– está cambiando, está digitalizándose. Internet se ha convertido en una herramienta más para el alumno, que no solo favorece la disponibilidad de los materiales educativos, sino que también permite y fuerza la colaboración entre instituciones y países, haciendo que la educación -formal o informal- sea una fuerza integradora más.

En áreas científico-tecnológicas los alumnos no solo necesitan acceder a contenidos, también necesitan acceder a laboratorios. Un experimento puede ser visto como una conversación con la naturaleza: el alumno pregunta y la naturaleza responde. La única forma de aprender el lenguaje de la naturaleza es experimentando, ya sea de forma clásica o con la intermediación de Internet. Ahora bien, el diseño y mantenimiento de laboratorios presenciales es caro y complicado, lo que frena su desarrollo y uso en los centros educativos, especialmente en países con recursos limitados.

En un principio, y como solución a su falta de disponibilidad, se procedió a simular los laboratorios mediante herramientas de programación. Pero desde la aparición de Internet la comunidad investigadora se ha esforzado en digitalizar estos laboratorios, remotizándolos, permitiendo que los estudiantes accedan a laboratorios reales a través de Internet, donde la conversación fluye a través de la red. Una vez que estos laboratorios están disponibles en Internet, es posible compartirlos con otras entidades educativas, como cualquier otro recurso digitalizado en la red. Compartir experimentos y experiencias entre países e instituciones necesita de un soporte tecnológico adecuado, de protocolos de federación que automatizarán el proceso.

El foco de esta tesis es explorar y evaluar tecnológicamente un nuevo modelo de federación para escalar esta compartición –permitiendo un mayor número de estudiantes concurrentes accediendo a equipamiento distribuido entre múltiples entidades educativas– y gestionar escenarios más complejos de los disponibles en la literatura –recompartiendo o subcontratando accesos a laboratorios, o distribuyéndolos en dispositivos de bajo coste–. Tanto el modelo de federación como la viabilidad técnica de estos escenarios han sido implementados en el sistema de gestión de laboratorios remotos WebLab-Deusto descrito en esta tesis.

Acknowledgements

A dissertation is a journey which, even if it is an individual effort, can definitely not be done without the context of the individual and the effort of the fellow travellers. Therefore, I must acknowledge these travellers for making the end of this journey possible. First of all, I want to thank Dr. Javier Garcia-Zubia, who introduced me on research by giving me the opportunity to start working on *that project* back in October 2004, which was called WebLab-Deusto. He was my director during those years, the director of my end of degree project while I was an undergraduate, and my PhD advisor during the last ones... and a fantastic, close, generous and always motivating colleague and friend during all this time. I am fortunate of having spent all this time with Javier, always having this feeling that there is still so much I need to learn from him. Thank you, Zubia.

I also want to thank Diego López-de-Ipiña, head of the research unit where I've worked since 2007, for being in the difficult position of balancing and solving so many interests during the 6 years I've been working in his team. He has always been generous, prudent and reasonable, and much of this work would never have been possible without these abilities.

This work is also possible thanks to the rest of the WebLab-Deusto team, present and past. I want to thank Jaime, Ignacio, Unai, Olga, Luis, Fabricio, Iñigo, Martin, Gustavo for all the support, not only technically (WebLab-Deusto is far from being just a software project), but also for keeping up the motivation and for all the great moments we have spent together.

The motivation has also been positively influenced by the work environment. I owe much of this motivation to my labmates, both present in the SmartLab (the physical laboratory where the MORElab research group is established), and in the 505 while I was an undergrad, as well as the DeustoTech Learning group. I want to thank especially my colleagues and friends from the Internet Unit, who have always been supporting one each other for all these years, and not only in work-related stuff. They are definitely part of my life experience.

However, the context of the journey plays a critical role. A dissertation is written in a particular moment, not being possible to achieve those results before or later. In this sense, I want to thank the Global Online Laboratory Consortium community, which has been an important inspiration source, and which has enabled me meeting so many interesting people. I want to thank the team headed by Ingvar Gustavsson at the Blekinge Institute of Technology (BTH), and in particular to Johan Zackrisson, for developing such an interesting, well-designed and well implemented system as it is VISIR. It has played a major role in this dissertation, and the team has always been willing to help and create fruitful collaborations. I also want to thank the three research centers who hosted me during this time: first the University of Technology, Sydney (thank you David, Herbert, Michael, Tania, Michel) in 2010 for two weeks, where I could have interesting discussions that drove to the selection of this dissertation topic. Second the Center for Educational Computer Initiatives, where I could finish the implementation in those six weeks at the end of 2011 (thank you Jud, Phil, Kirky, Jim, Nico). Many design decisions were taken during that stay, after generously attending me with so many productive meetings and knowledge sharing. And finally the Departamento de Ingeniería Eléctrica, Electrónica y de Control of the UNED, where I could start writing this document in 2012 and received fruitful feedback (thank you Elio, Gabriel, Manuel, Sergio).

Last but not least, the individual context is not isolated, and I want to thank the inconditional support of those who were always close, including familly, friends and my girlfriend. I especially want to thank her, who has always supported me and understood the difficulties of the journey. Without your help, Sonia, I would not have had the required strength to cross it. Thank you. I also want to thank my familly for all the support they gave me. Muchas gracias, aita y ama, por todo el apoyo que me habéis dado, la comprensión del tiempo que no he pasado, y por todas las inversiones que habéis hecho en mí para que todo esto haya sido posible. Nunca os lo agradeceré lo suficiente.

Muchas gracias,

Pablo Orduña

April 2013

Contents

List of Figures **xiii**

List of Tables **xvii**

Acronyms **xxi**

1 Introduction **1**
1.1 Context 2
1.2 Motivation 3
1.3 Thesis Statement 4
1.4 Methodology 4
1.5 Contributions 5

2 Background **7**
2.1 Introduction 8
2.1.1 Remote Laboratories 8
2.1.2 Remote Laboratory Management Systems 11
2.1.3 Remote Laboratories Interoperability 12
2.1.4 Remote Laboratories Scheduling 13
2.1.5 Remote Laboratory Federation Models 15
2.1.6 Structure of this Chapter 16
2.2 Remote Laboratories 17
2.2.1 Aspects to Study 19
2.2.2 Remote Laboratories Analyzed in the Survey 23
2.2.2.1 WebLab-Deusto Xilinx Laboratories 25
2.2.2.2 MIT Microelectronics Laboratory 26
2.2.2.3 UTS Structural Beam and Couple Tanks Experiment 27
2.2.2.4 VISIR 28
2.2.2.5 OpenLabs Security Lab 31

2.2.2.6 ISILab . . . 32
2.2.2.7 RemotElectLab . . . 32
2.2.2.8 Fachhochschule Düsseldorf University of Applied Sciences (DUAS) . . . 33
2.2.2.9 Purdue University (CASPiE) . . . 33
2.2.2.10 Ilmenau University of Technology . . . 33
2.2.2.11 RLab . . . 34
2.2.3 Summary . . . 34
2.3 Remote Laboratory Management Systems . . . 35
2.3.1 iLab Shared Architecture . . . 36
2.3.1.1 Context . . . 36
2.3.1.2 Architecture . . . 36
2.3.1.3 Laboratory Servers . . . 39
2.3.1.4 Limitations . . . 40
2.3.2 Labshare Sahara . . . 41
2.3.3 Notes on the Convenience of Remote Laboratory Management Systems . . . 43
2.3.4 Other Systems . . . 45
2.4 Remote Laboratory Federation Models . . . 46
2.4.1 iLab Shared Architecture Federation Model . . . 46
2.4.2 Labshare Sahara Federation Model . . . 47
2.4.3 LiLa Sharing Model . . . 47
2.4.4 Comparison of Federation Models . . . 48
2.5 Interoperability of Remote Laboratories . . . 49
2.6 Related Systems . . . 51
2.7 Conclusions . . . 52

3 Remote Laboratory Management System: WebLab-Deusto 55
3.1 A Bit of History . . . 56
3.2 WebLab-Deusto Architecture . . . 60
3.2.1 voodoo: Underlying Server Management Middleware . . . 63
3.3 Transversal Features . . . 67
3.3.1 Scalability . . . 67
3.3.2 Security . . . 69
3.3.3 Authentication and Authorization . . . 70
3.4 Scheduling . . . 71
3.5 Experiment Development . . . 74
3.5.1 Managed Experiment Development . . . 76
3.5.1.1 Collaborative Managed Experiments . . . 80
3.5.2 Unmanaged Experiment Development . . . 81
3.5.2.1 LabVIEW Experiments . . . 82

3.5.2.2 Virtual Machine Experiments 84
3.5.3 Experiment Examples 86
3.5.3.1 Xilinx Laboratories 87
3.5.3.2 Aquarium Laboratory 87
3.5.3.3 Robot Laboratory 88
3.5.3.4 VISIR 89
3.5.4 Experiment Maintenance 89
3.6 Conclusions 90

4 Federation Model **93**
4.1 From Local Scheduling to Federation 93
4.2 Federation Model Internals 95
4.2.1 Introduction and Formalization 95
4.2.2 Federation Model Characteristics 98
4.2.3 Complex Scenarios 101
4.2.3.1 Transitive Federation 101
4.2.3.2 Distributed Load Balance 101
4.2.3.3 Concatenating Distributed Load Balance and Transitive Federation 106
4.2.4 Accounting 107
4.2.4.1 Accounting Scenarios 107
4.2.4.2 Sharing Approaches and Risk Management . . . 109
4.2.5 Design and Implementation 111
4.2.5.1 Design 112
4.2.5.2 Implementation 119
4.3 Applications of the Federation Model 121
4.3.1 Learning Management Systems and Content Management Systems 124
4.3.1.1 Integration LMS/CMSs Using a Federation Model 125
4.3.1.2 Case Study: Moodle 127
4.3.1.3 Case Study: Joomla 129
4.3.2 Interoperability 131
4.3.2.1 Case Study: WebLab-Deusto Running in MIT iLabs (ISA) 133
4.3.2.2 Case Study: iLab Batch Laboratories Running on WebLab-Deusto 135
4.3.2.3 Benefits of Interoperability 137
4.3.3 Ubiquitous Laboratories 137
4.4 Qualitative Evaluation 139
4.5 Conclusions 139

5 Evaluation **143**
5.1 Scenario Overview . . . 143
5.1.1 Reference Model . . . 144
5.1.2 WebLab-Bot . . . 145
5.1.3 Assumption Validation . . . 145
5.1.4 Notes on Concurrency and Stress . . . 149
5.1.5 Comparison Framework . . . 151
5.2 Non-federated Environments . . . 153
5.2.1 Aspects to Analyze in Non-federated Environments . . . 153
5.2.2 Protocols . . . 154
5.2.3 Database Engine . . . 155
5.2.4 Scheduling Backend . . . 159
5.2.5 Session Management . . . 164
5.2.6 Number of Copies of Experiments . . . 165
5.2.7 Number of Processes . . . 169
5.2.8 Summary of the Non-federated Environments Evaluation . 169
5.3 Federated Environment . . . 173
5.3.1 Aspects to Analyze . . . 173
5.3.2 Basic Federation . . . 174
5.3.3 Transitive Federation . . . 178
5.3.4 Federated Load Balancing . . . 182
5.3.5 Transitively Federated Load Balancing . . . 186
5.3.6 Summary of the Federated Environment . . . 194
5.4 Low Cost Environments . . . 195
5.5 Conclusions . . . 196

6 Conclusions **199**
6.1 Limitations . . . 201
6.2 Future work . . . 202
6.3 Final Remarks . . . 203

Bibliography **205**

List of Figures

1.1 Methodology used during the dissertation 5

2.1 Specific Purpose Remote Laboratory 8
2.2 Classification of remote and virtual laboratories as explained in (Dormido, 2004) 9
2.3 Complex SPRL example: VISIR, supporting multiple concurrent students 10
2.4 VISIR project: user's electronics board (left) and commutation matrix (right) 10
2.5 Remote Laboratory Management System 12
2.6 Local interoperability: VISIR integrated in a RLMS 12
2.7 Local interoperability: a remote lab integrated in an external tool . 13
2.8 Simple queue managed by a RLMS 14
2.9 Local load balance 14
2.10 Different levels to be analyzed in the state of the art 16
2.11 Technologies used in remote labs, by (Gravier et al., 2008) 20
2.12 VISIR Oscilloscope 22
2.13 Average time waiting in a queue depending on the number of concurrent students and average time per session 22
2.14 WebLab-Deusto CPLD laboratory 24
2.15 Users in queue during the 2008-2009 course in WebLab-Deusto Xilinx laboratories 24
2.16 Duration of the accesses in the CPLD laboratory of WebLab-Deusto in the course 2010-2011 (Garcia-Zubia et al., 2011) 25
2.17 Microelectronics WebLab 4.0-4.2. Retrieved from (del Alamo et al., 2002) 26
2.18 UTS structural beam experiment 28
2.19 VISIR breadboard 29
2.20 VISIR switching matrix 30
2.21 Servers of the BTH OpenLabs Security Lab 31

2.22 iLab Shared Architecture for batch experiments (Harward et al., 2008b) . . . 38
2.23 Internal queue management in the Lab Server for the Microelectronics WebLab . . . 38
2.24 iLab Shared Architecture for interactive experiments . . . 39
2.25 Sahara architecture (Lowe and Orou, 2012) . . . 42
2.26 Labshare Sahara Booking system . . . 43
2.27 Graasp screenshot, taken from (Gillet and Bogdanov, 2012) . . . 44
2.28 Sahara federation architecture (Diponio et al., 2012) . . . 48
2.29 LabConnector view from the Sahara users consuming iLab laboratories by (Yeung et al., 2010) . . . 50

3.1 WebLab-Deusto 0.1 User interface . . . 56
3.2 Comparison of the first 3 WebLab-Deusto versions . . . 58
3.3 Local architecture of WebLab-Deusto . . . 62
3.4 Protocols used by WebLab-Deusto through voodoo . . . 64
3.5 Multiple queue usage example . . . 73
3.6 Multiple queue usage second example . . . 74
3.7 Scheduling system . . . 75
3.8 WebLab-Deusto stack: client side (left) and server side (right) . . . 77
3.9 Sharing VISIR circuits among students with the collaboration API 81
3.10 LabVIEW Remote Panels integrated in WebLab-Deusto. Interaction screen. . . . 83
3.11 LabVIEW Remote Panels integrated in WebLab-Deusto. Configuration screen. . . . 83
3.12 Example of Virtual Machine laboratory integrated in WebLab-Deusto. Inside the Virtual Machine, a LabVIEW instance is running in a Ubuntu Desktop . . . 86
3.13 WebLab-Box, which typically contains an electronics laboratory . 88
3.14 Azkar robot, used by the Robot laboratory . . . 89

4.1 Federated remote laboratories . . . 96
4.2 Transitive federation . . . 99
4.3 Distributed load balance . . . 100
4.4 Exchanging remote laboratories across boundaries . . . 102
4.5 Possible schemes to federate the VISIR remote laboratory . . . 104
4.6 Ring of VISIR providers . . . 105
4.7 Distributed load balance with transitive federation (VISIR). Note: this is a theoretical situation. . . . 107
4.8 Federation sequence diagram . . . 113
4.9 Federation model class diagram: main interface . . . 114

4.10 Federation model class diagram: reservation status 114
4.11 Transitive federation sequence diagram 116
4.12 Federated load balance sequence diagram 118
4.13 List of experiments in the Colegio Urdaneta secondary school . . 120
4.14 WebLab-Deusto in Colegio Urdaneta: reserved experiment in remote system 121
4.15 VISIR experiment reserved in the University of Deusto 122
4.16 VISIR experiment reserved in ISEP 123
4.17 Federation of remote labs. Students registered in the federation . . 125
4.18 A LMS/CMS can act as a federated node consuming the federation protocol 126
4.19 If the federation is transitive, the LMS/CMS will still be able to consume remote laboratories in other universities 126
4.20 Admin granting permissions of an experiment on a course 128
4.21 Teacher adding a remote lab as an activity for a course 128
4.22 Student running an experiment 129
4.23 Available Joomla user groups to be edited 130
4.24 WebLab-Deusto manager in the Joomla Administration panel . . . 130
4.25 Permissions editor for each group 130
4.26 List of experiments available for current user 131
4.27 Heterogeneous federation 132
4.28 WebLab-Deusto robot laboratory running through an iLab system 134
4.29 WebLab-Deusto laboratories listed in iLab 134
4.30 MIT Microelectronics WebLab running through WebLab-Deusto in Facebook 135
4.31 Architectural overview of the iLab on WebLab-Deusto integration 136
4.32 Fishtank laboratory deployed outside the University but transitively being accessed by its students 138

5.1 WebLab-Bot simulated user life cycle. Real users submit more commands and files, but the bot does not stress this situation. . . . 146
5.2 Simple concurrent use case (not considered in this chapter). Blue points represent the reservation process, the purple ones the requests for reservation status and the yellow ones the commands submitted to the laboratory. As there are big delays between the requests of one user and the other, each method finishes very quickly 149
5.3 Concurrent use case as considered in this chapter. Blue lines represent the reservation process, purple lines the requests for reservation status. Reservation time is much longer since all the requests are concurrent on a single monothread SQLite database. 150

5.4 Sequential case (not considered in this chapter). Blue lines are the reservation process. As in Figure 5.2, even having 150 students in this way does not stress the system (since each previous student has already finished before the next one enters). 151
5.5 WebLab-Bot sample measurement. The green line (on the middle) represents the mean, while the vertical lines crossing it represent the standard deviation. The maximum and minimum values are represented by the red (on the top) and the blue (on the bottom) lines. 152
5.6 Ring configuration: every entity has access to each other through a chain of relationships 187
5.7 Star configuration: every entity knows each other 187
5.8 Distribution of students in the ring configuration 193
5.9 Low cost Raspberry Pi 195

List of Tables

2.1 Summary of the scheduling systems 35
2.2 Summary of existing federation models 49

4.1 Basic accounting schemes 108
4.2 Basic adjustments 108
4.3 Consumption abuse risks 109
4.4 Sharing strategies 110
4.5 Sharing strategies and risk management 110
4.6 Summary of the federation model 140

5.1 Measurements of a single, random execution with the WebLab-Bot client and the Google Chrome (version 23.0.1271.97 under Ubuntu 12.04) web browser, using a single local node with SQLite. This is not intended to provide an accurate comparison (since it will depend on the web browser and the operating system) but to show that the WebLab-Bot client is not very different to the real client . 148
5.2 Methods used by the student simulator and impact on the database, scheduling system and federation system 148
5.3 Different protocols, 4 processes with 1 copy of the laboratory using Redis for scheduling and MySQLdb. On the left column, the `reserve_method`; on the right column, the `send_command` method 156
5.4 Numerical summary of Table 5.3 157
5.5 Results with SQLite for scheduling and database with 1 core and 1 experiment. The vertical scale has been changed given the big difference among methods types (interaction methods are not affected by the database, reservation methods are affected by the scheduling database, `login` and `list_experiment` methods are affected by the regular database) 158

5.6 Results with MySQL for scheduling and database with 1 core and 1 experiment . . . 160
5.7 Results of `reserve_experiment` with different database engines 161
5.8 Numerical summary of Table 5.5, Table 5.6 and Table 5.7 . . . 162
5.9 `reserve_experiment` method using Redis and MySQL with 1 and 5 processes . . . 163
5.10 Numerical summary of Table 5.9 . . . 164
5.11 Results of the `reserve_experiment` method with MySQLdb, Redis, a single core and different numbers of copies . . . 166
5.12 Results of the `send_command` method with MySQLdb, Redis, a single core and different numbers of copies . . . 167
5.13 Numerical summary of Table 5.11 and Table 5.12 . . . 168
5.14 Results of the `reserve_experiment` method with MySQLdb, Redis, 80 copies of the laboratory and different number of processes 170
5.15 Results of the `send_command` method with MySQLdb, Redis, 80 copies of the laboratory and different number of processes . . . 171
5.16 Numerical summary of Table 5.14 . . . 172
5.17 Basic federation with a single copy of the laboratory, comparison of federated and non-federated environments. Scales are different per method . . . 175
5.18 Basic federation with 80 copies of the laboratory, comparison of federated and non-federated environments. Scales are different per method . . . 176
5.19 Numerical summary of Table 5.17 and Table 5.18 . . . 177
5.20 Results of 1 laboratory (single copy) shared through three entities transitively compared with the same situation in 2 entities. Scales are different per method . . . 179
5.21 Results of 1 laboratory (80 copies) shared through three entities transitively compared with the same situation in 2 entities. Scales are different per method . . . 180
5.22 Numerical summary of Table 5.20 and Table 5.21 . . . 181
5.23 Load balance 3 copies in 3 institutions, compared with basic federation. Scales are different per method . . . 183
5.24 Load balance 240 copies in 3 institutions, compared with basic federation with 80 devices. Scales are different per method . . . 184
5.25 Numerical summary of Table 5.23 and Table 5.24 . . . 185
5.26 Up to 200 students accessing a single entity, with laboratories in 6 entities . . . 189
5.27 Numerical summary of Table 5.26 . . . 190
5.28 Up to 420 students accessing 6 entities concurrently, with laboratories in 6 entities . . . 191

5.29 Numerical summary of Table 5.28 192
5.30 Results with raspberry using Redis and SQLite for scheduling. Scales are normalized per method 197

Acronyms

AJAX Asynchronous JavaScript And XML

API Application Programming Interface

CMS Content Management System

CPLD Complex Programmable Logic Device

FPGA Field Programmable Gate Array

GNU GNU's Not Unix

GOLC Global Online Laboratory Consortium

HTML Hyper Text Markup Language

HTTP Hyper Text Transfer Protocol

ID Identifier

IEEE Institute of Electrical and Electronics Engineers

IO Input/Output

IP Internet Protocol

ISA iLab Shared Architecture

JSON JavaScript Object Notation

LMS Learning Management System

MIT Massachusetts Institute of Technology

MOOC Massive Open Online Courses

ORM Object-Relational Mapping

RDP Remote Desktop Protocol

RLMS Remote Laboratory Management System

RPC Remote Procedure Call

SOAP Simple Object Access Protocol

SPRL Single Purpose Remote Laboratory

SQL Structured Query Language

SSH Secure SHell

STEM Science, Technology, Engineering and Mathematics

TEL Technology-Enhanced Learning

UML Unified Modelling Language

UNED Universidad Nacional de Education a Distancia

UTS University of Technology, Sydney

VHDL VHSIC Hardware Description Language

VHSIC Very-High-Speed Integrated Circuits

VISIR Virtual Instrumentation Systems In Reality

VM Virtual Machine

VNC Virtual Network Computing

WSDL Web Service Description Language

XML eXtensible Markup Language

If you can't come to the lab, the lab will come to you

Jesús del Álamo

CHAPTER

1

Introduction

EDUCATIONAL remote laboratories were born nearly two decades ago (Carisa et al., 1995; Aktan et al., 1996; Henry, 1996), and since then they have been adopted in more and more fields: chemistry (Coble et al., 2010; Cedazo et al., 2006), physics (Del Alamo et al., 2002; Gillet et al., 2001), electronics (Gustavsson et al., 2007; Nedic et al., 2008), robotics (Safaric et al., 2005; Torres et al., 2006) and even nuclear reactor (Hardison et al., 2008). Efforts have been placed on achieving a greater degree of collaboration (Machotka et al., 2010; Gillet et al., 2008), even with virtual world technologies (Marcelino et al., 2010; Callaghan et al., 2009; Scheucher et al., 2009; Garcia-Zubia et al., 2010c). With time, they have been adapted to be suitable in mobile devices (Rochadel et al., 2012; López-de Ipiña et al., 2006; Orduña et al., 2011a), or using augmented reality (Andujar et al., 2011). Indeed, they have been considered as part of the *Five Major Shifts in 100 years of Engineering Education* in the Special Centennial Issue of the Proceedings of the IEEE (Froyd et al., 2012), in respect to the influence of Information Communications and Computational Technologies.

In the beginning, the target was improving the laboratory experience by providing more flexibility, or achieving what could be done in a physical laboratory through the Internet, and improving it when possible. However, remote laboratory developers realized that once an institution had shared a physical laboratory with its students, it was technically possible to enable students of other institutions to access these labs, although several layers had to be changed to make this successfully. This sharing of remote laboratories is referred as federation of remote laboratories, which is the core of this dissertation.

The work described in this dissertation is a novel federation model, which focuses on enabling the basic market rules on the sharing of remote laboratories to achieve a wider range of inter-institutional collaboration, through two key novelties: a) scaling up the number of concurrent users which can use a single remote laboratory and b) enabling institutions to re-share laboratories.

The remainder of this chapter is structured as follows: Section 1.1 explains the context of the research. Section 1.3 presents the core statement of this dissertation, and finally Section 1.5 summarizes the rest of the contributions of this dissertation.

The dissertation is structured as follows: Chapter 2 describes the state of the art relevant to the dissertation, Chapter 3 describes the WebLab-Deusto system, on which the federation model has been implemented, Chapter 4 describes the proposed federation model, and Chapter 5 evaluates both the federation model and its implementation in WebLab-Deusto. Finally, Chapter 6 summarizes the dissertation and shows the related future lines of work.

1.1 Context

As Internet technologies became spread, more universities worldwide developed their own remote laboratories (Section 2.1.1). Instructors, particularly of engineering fields, have been developing new laboratories where the focus has traditionally been on the equipment itself, rather than on the management or communication layers. This has lead to many remote laboratories that may work but do not address security, scalability, communications, proper technologies (requiring users to use a particular browser or installing a particular software platform). Furthermore, since each remote laboratory started from scratch, it also provided certain technical efforts that had to be redone every time.

Aiming this situation, remote laboratory management systems (RLMSs, Section 2.3) were developed. These are software systems which are focused on providing a technical solution for developing remote laboratories, implementing common features and covering many of the common issues in remote laboratory development. For instance, RLMSs typically implement authentication (using different techniques: LDAP, OpenID, etc.), user tracking, scheduling (using queues or calendar-based booking, Section 2.1.4) or administration tools (adding users, assigning permissions, etc.). This way, a remote laboratory developer can focus on making the equipment available through the Internet, and the rest of the application life cycle can be managed by the RLMS. Additionally, new releases of these RLMSs can come with new features which are transparent for the existing remote laboratories: if a RLMS becomes integrated with a Learning Management System (such as Moodle), automatically all the remote laboratories developed with this RLMS will be integrated.

A novel feature that RLMSs started supporting was federating systems deployed in different universities (Section 2.4). This way, it became possible that one university shares with other a particular laboratory easily. The provider university will have the RLMS, which administrates a set of remote laboratories, and the consumer university will also have the RLMS deployed, managing or not a set of remote laboratories. The provider can this way share laboratories to the consumer generically, and the consumer can define which local users will be able to use which laboratories. Prior to this dissertation, the federation models were established in a 1 to 1 basis. They did not support balancing the load of users among multiple institutions, which provide copies of the same laboratory.

1.2 Motivation

Federating laboratories is interesting for several reasons. If two universities share a laboratory with each other, their students will be able to access twice more laboratories, increasing the experiential learning. Additionally, a characteristic generally present in remote and traditional laboratories is that they are long time unused. In the case of remote laboratories, they can be shared regardless of the city, country or continent where the student is. Federations of remote laboratories make it possible to share the costs of laboratories, either directly (by paying for the access) or indirectly (sharing back different complimentary laboratories, or other valuable contents, such as learning material or similar). While a remote laboratory may be more expensive than a traditional one depending on the particular setting (Lowe et al., 2012a), federations have the potential to decrease the cost or at least to increase the experiential possibilities with relatively low additional costs.

Indeed, the interest on federation of remote laboratories is growing. The Labshare project survey (Kotulski and Murray, 2010), made on all 34 Australian universities offering undergraduate engineering programs, reflects that the interviewed executives were more interested in getting involved for the pedagogic merits of the remote laboratories, and were more inclined on initially being laboratory consumers than providers. Indeed, the European Union Commission is investing 60 million euros in research actions, projects and network of excellences in Technology-Enhanced Learning (TEL), under the objective ICT-2011.8.1 of the call FP7-ICT-2011-8. One of the target outcomes is precisely *"Supporting European wide federation and use of remote laboratories and virtual experimentations for learning and teaching purposes"*[1]. Indeed, the IP project Go-Lab[2], funded by this call with 10 million euros, aims to support a wide federation of remote and virtual laboratories. Parallel and related efforts have been placed on systems that

[1] http://cordis.europa.eu/fp7/ict/telearn-digicult/telearn-objectives_en.html
[2] http://www.go-lab-project.eu/

index remote laboratories located at different institutions such as lab2go (Zutin et al., 2010) or even grant access to laboratories as LiLa (Richter et al., 2011b,a; Mateos et al., 2011).

At the same time, massive open online courses (MOOC) such as Coursera[3], edX[1] or Udacity[2] are supporting large scale courses with thousands of students. The experiential part of the technical courses that would typically require a laboratory is being managed through simulators. One of the targets of this dissertation is precisely to explore how remote laboratories can be scaled up to be used in this type of environments and under which restrictions. While studying the particular policies, load modeling and applying it to a real MOOC is outside the scope of this dissertation, keeping this type of situation in mind during the design, development and writing of this dissertation has been an important incentive. It is recommended that the reader keeps this type of situation while reading the document, too.

1.3 Thesis Statement

In the remote laboratory ecosystems (where there are multiple providers, consumers and potential consumers), the sharing of laboratories is limited by the existing federation mechanisms. By developing a federation model which supports features that resemble a market environment in a completely decentralized basis –where consumers can re-sell to others laboratory accesses and where consumers can freely choose between different providers of the same laboratory and even consume them all– it becomes technically feasible to support a remarkable volume increase of consumers. If this federation model is implemented in a remote laboratory management system, it becomes possible to use all the tools of this remote laboratory management system to create and maintain new remote laboratories, which will automatically be shared through this federation model.

1.4 Methodology

A research strategy has been defined in order to achieve the statement presented above. The strategy is defined as follows:

1. Update knowledge by reviewing the literature in the area of remote laboratories, as well as attend conferences of this area.

2. Design and develop different parts of the federation model, incrementally redefining them and adapting them to the new trends in the literature.

[3] https://www.coursera.org/
[1] https://www.edx.org/
[2] http://www.udacity.com/

3. Attend conferences and workshops to present partial results and validate them with the existing state of the art.

4. Experimentation and evaluation of the prototype at each particular stage.

5. Contact experts at conferences, project meetings and consortia (the author is an active member of the GOLC –Global Online Laboratory Consortium). Contact for particular details by e-mail or by visiting other research centers (the author was a visiting researcher in the *Center for Educational Computer Initiatives* in the Massachusetts Institute of Technology (*MIT*) for 6 weeks in 2011 and in the *Departamento de Ingeniería Eléctrica, Electrónica y de Control* of the Spanish Distance University (*UNED*) in 2012 for 6 weeks, as well as a 2-week visit to the Labshare software development team in the University of Technology, Sydney).

6. Redesign with feedback from all this network, as well as the literature.

7. Technical evaluation of the prototype.

8. Dissemination of the results obtained during the research process.

This methodology is illustrated in Figure 1.1.

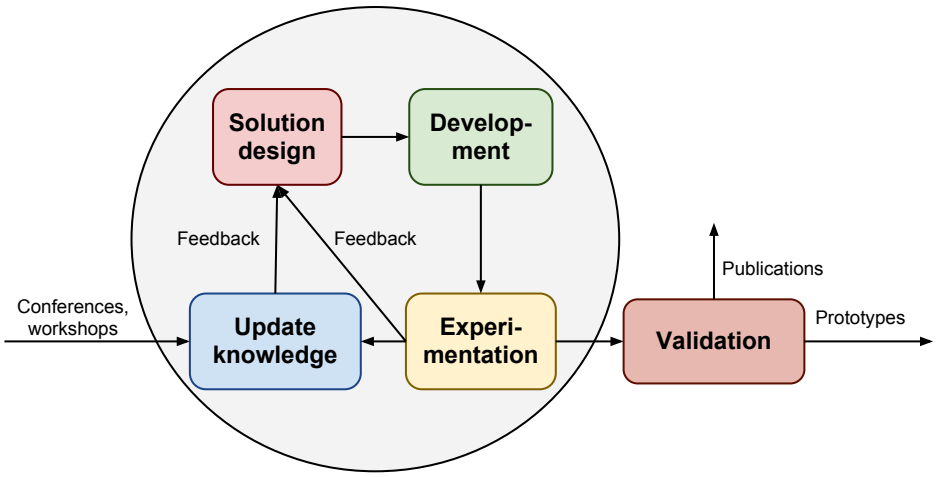


Figure 1.1: Methodology used during the dissertation

1.5 Contributions

The following major and minor contributions can be found in this dissertation:

- The proposed **federation model:** detailed in Chapter 4, it is the core contribution of this dissertation. Its key features are federated load balance (redirecting users to available slots in other institutions using a decentralized approach) and transitivity (being able to re-share laboratories as part of existing contracts). These features add a measurable quantitative performance cost, analyzed in Chapter 5, and always under low time boundaries (in stressful situations described in that chapter, simulations did not achieve one tenth of second in the interaction of students with the laboratories with 420 simultaneous students).

- The **scheduling system** on which the federation model is based. As explained in Section 3.4 and Section 4.1, the federation model is implemented as a concrete scheduling plug-in in the WebLab-Deusto remote laboratory management system. Certain features, such as local load balance are indeed inherited from this system.

- The **remote laboratory management system** (WebLab-Deusto, Chapter 3) on top of which the scheduling system and the federation model are implemented.

- **Applications** of the federation model to other scopes: interoperability with other RLMSs (Section 4.3.2), in Learning Management Systems / Content Management Systems (Section 4.3.1), application of the federation model in ubiquitous laboratories (Section 4.3.3).

Those who cannot remember the past are condemned to repeat it

George Santayana

CHAPTER

2

Background

THIS chapter describes the state of the art of federation models for sharing remote laboratories. The first section (2.1) introduces the basic concepts that will be used during the whole dissertation. In the second section (2.2), different remote laboratories are detailed. This section is based on a survey developed as part of this dissertation to gather requirements from them that could affect the federation model. In the third section (2.3), different existing remote laboratory management systems are analyzed, given that the federation models are commonly implemented at this layer. In the next section (2.4), different existing federation models are analyzed. The following section (2.5) analyzes the interoperability of remote laboratory management systems, linking them with their federation models. Finally, other scopes (2.6) where federation has been addressed conceptually has been analyzed.

It is important to note that this layered structure, explained in Section 2.1.6, will split certain frameworks in the different sections. For instance, the iLab Shared Architecture (ISA) natively provides a federation model, so the federation model itself is analyzed in section (2.4). However, it is also a remote laboratory management system and as such it is analyzed in the previous section (2.3), without analyzing the federation models. Additionally, certain laboratories developed on top of the ISA are presented in the previous section (2.2), without focusing on the management system.

It is also worth remarking that while the context of this dissertation is remote laboratories, this chapter will not pretend to be a state of the art of the different approaches for developing remote laboratories, the different classifications available in the literature, or the different existing fields where remote laboratories are

applied. Instead, it is focused on exploring and explaining the existing federation models, detailing in which layers they are implemented and analyzing what are their strengths and limitations.

2.1 Introduction

This section describes the concepts on which the proposed federation model is based, and on top of them it defines the structure of this chapter in Section 2.1.6. The analyzed concepts are:

- Remote laboratories (Section 2.1.1).
- Remote Laboratory Management Systems (Section 2.1.2).
- Remote laboratories interoperability (Section 2.1.3).
- Remote laboratories scheduling (Section 2.1.4).
- Remote laboratory federation models (Section 2.1.5).

2.1.1 Remote Laboratories

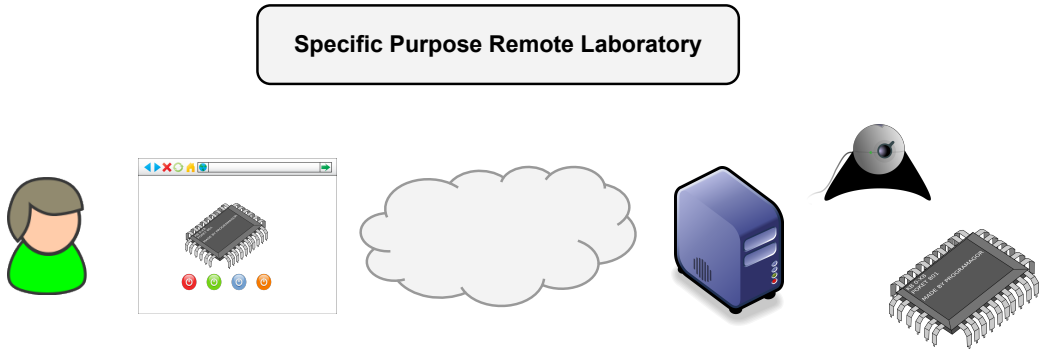


Figure 2.1: Specific Purpose Remote Laboratory

A remote laboratory is a software and hardware tool that allows students to remotely access real equipment located in the university (see Figure 2.1). It is important to remark that, as shown in Figure 2.2, this is different from virtual laboratories, which are usually simulations instead of real equipment.

Every remote laboratory manages at least a subset of the following features: authentication, authorization, scheduling users to ensure exclusive accesses -typically

through a queue or calendar-based booking-, user tracking and administration tools. However, most remote laboratories are developed aiming a particular setting (e.g., a laboratory for robots based on a microcontroller, an electronics board). So the focus is usually achieving that the particular setting is available through the Internet. For this reason, in this contribution these remote laboratories will be referred as *specific purpose remote laboratories* (SPRL). The variety of SPRLs in the literature is broad, as the range of technologies used (Gravier et al., 2008). Certain SPRLs are analyzed in Section 2.2.

		NATURE OF THE RESOURCE	
		Real	**Simulated**
ACCESS TO THE RESOURCE	**Local**	Hands-on lab	Mono-user virtual lab.
	Remote	Remote lab.	Multi-user virtual lab.

Figure 2.2: Classification of remote and virtual laboratories as explained in (Dormido, 2004)

While a SPRL depends on a particular setting, the complexity of its nature may be very high. An example of this is the VISIR project (Gustavsson et al., 2007, 2009; Tawfik et al., 2011b), widely described in Section 2.2.2.4. This SPRL enables students to build electronic circuits in a protoboard as Figure 2.4 shows, using different components such as resistors and capacitors, and link them to function generators, oscilloscopes, etc. VISIR provides a commutation matrix based on relays which makes it possible that every time a student attempts to take a measurement, the system builds the desired circuit, takes the measurement, and returns the data to the user in a fraction of second. This way, multiple students can be working with different circuits at the same time, multiplexing the amount of users in time (Figure 2.3). However, a big amount of users will degrade the user experience, since students will wait for several seconds before retrieving the desired value. For this reason, a maximum number of concurrent users is used, which might range between 16 and 60 depending on the components established. Six different universities across Europe have already deployed the system.

While there are many classifications in the literature (Gomes and Bogosyan, 2009; Gravier et al., 2008; Garcia-Zubia et al., 2007a), a particular classification which is especially relevant to this dissertation is the type of interaction of the remote laboratory. There are two main approaches: *interactive laboratories*, which guarantee a direct connection and an interaction with the laboratory during the

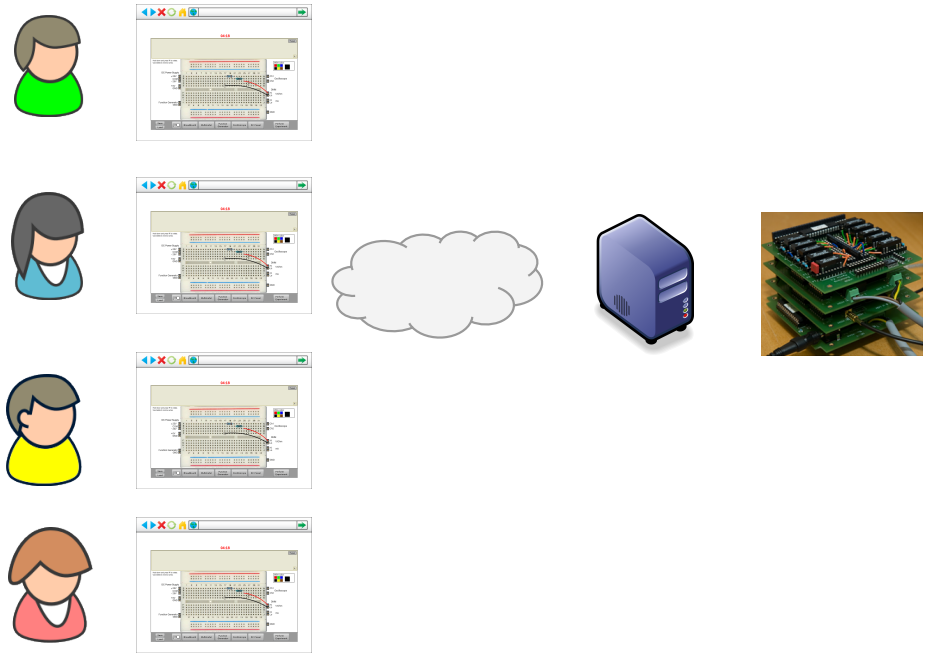


Figure 2.3: Complex SPRL example: VISIR, supporting multiple concurrent students

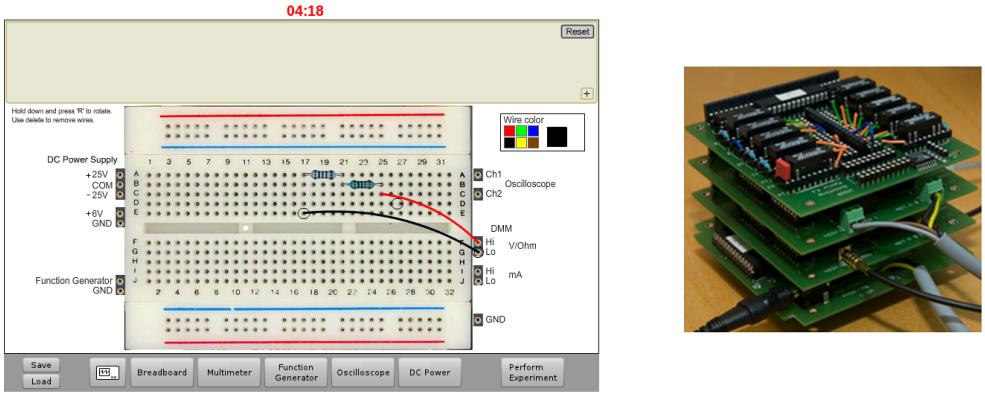


Figure 2.4: VISIR project: user's electronics board (left) and commutation matrix (right)

execution of the task, and *batch laboratories*, where students submit tasks that are executed in the laboratory and they are not able to modify the task during its execution. This classification has an impact on the scheduling schemes, since while batch laboratories typically use a queue, interactive laboratories may use a queue or a calendar based booking depending on the required session time. Therefore this concept will be present in different points of the dissertation.

Some cases, however, are not so clearly classified in interactive or batch laboratories. For example, VISIR itself seems to be an interactive laboratory, but when students are connecting resistors they are actually not connected with the server. It is once students *submit* a circuit that the task is executed in the server –without being modified by the user– and then the result is returned. So technically, it is a batch laboratory. However, the user cycle and perception is closer to an interactive laboratory, and it is usually treated as such.

2.1.2 Remote Laboratory Management Systems

Many of the features developed in a remote laboratory (authentication, authorization, scheduling, user tracking...) are reusable from one remote laboratory to another. Due to this reason, information systems that provide these services arose. Examples of these systems are: MIT iLabs[1] (Harward et al., 2008b), Labshare Sahara[2] (Lowe et al., 2012b, 2011, 2009), VLCAP (Raman et al., 2011) and WebLab-Deusto (Orduña et al., 2011b). These systems are independent of a particular setting, so they may be referred as *general purpose remote laboratories* in contrast to *specific purpose remote laboratories*. However, given that most of their work is adding management layers, these systems will be referred as Remote Laboratory Management Systems (RLMS; see Figure 2.5). RLMSs come with guidelines and software development toolkits to develop remote experiments. This way, RLMSs speed up the development process of remote laboratories: teachers aiming to create a remote laboratory do not need to work on scheduling, authentication, authorization, etc. but focus on making the experiment available through the Internet.

Additionally, RLMSs provide administration tools, so teachers can use them to add students, grant permissions on certain laboratories, track students, etc. The advantage of embracing these systems is that once they add a new transversal feature, all the experiments developed with that RLMS will automatically have it. For instance, if a RLMS did not support LDAP (a directory protocol common in universities which is used to authenticate and authorize users), and in the next release it supports it, automatically all the experiments developed with that RLMS will

[1] http://ilab.mit.edu/wiki/

[2] http://www.labshare.edu.au

support it. RLMSs are analyzed in Section 2.3, and Chapter 3 is focused on the WebLab-Deusto RLMS.

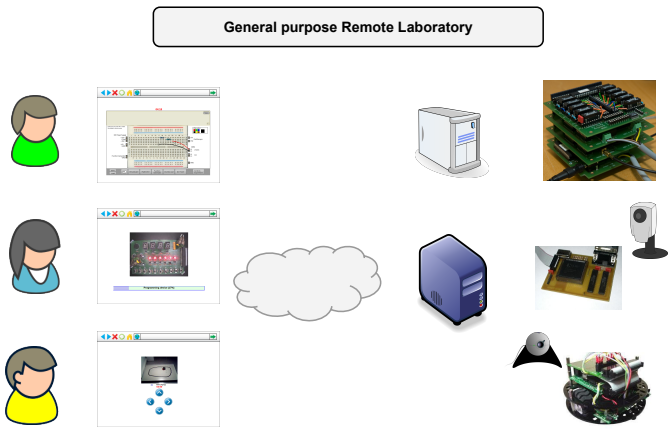


Figure 2.5: Remote Laboratory Management System

2.1.3 Remote Laboratories Interoperability

Remote laboratories interoperability refers to the ability of certain remote laboratories to integrate or be integrated in other remote laboratories. This interoperability can be local (connecting two local systems directly) or federated (using federation protocols). Examples of the former would be integrating a remote laboratory in a remote laboratory management system (e.g., the efforts to integrate the VISIR SPRL in the iLab Shared Architecture (Zutin and Auer, 2011), or in WebLab-Deusto (Orduña et al., 2011b), see Figure 2.6), or integrate it in other local tools, as in Figure 2.7. Examples of the latter are deeply explained in Section 4.3.2.

Integrating remote laboratories provides two advantages:

- The integrated system inherits the features of the integrator. For instance, VISIR does not natively support federation, but through WebLab-Deusto or iLabs, it inherits the provided federation mechanisms.

- Teachers and students centralize the administration in a single system. Students using WebLab-Deusto to access VISIR will use the same credentials in the same web application they may have used for other laboratories.

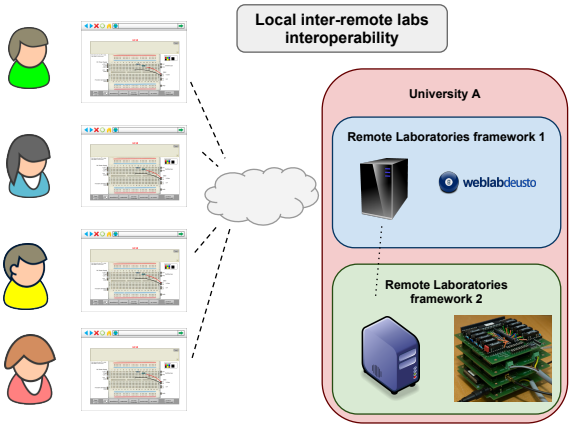


Figure 2.6: Local interoperability: VISIR integrated in a RLMS

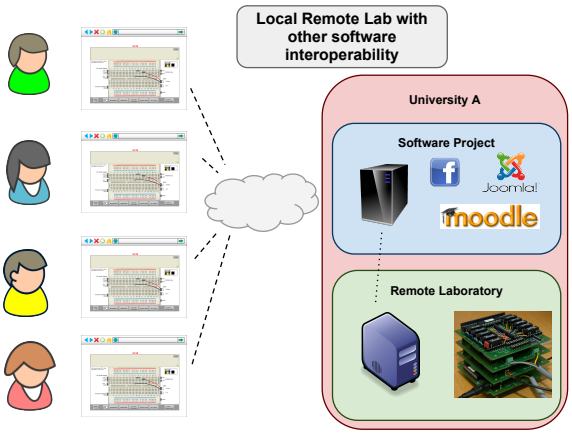


Figure 2.7: Local interoperability: a remote lab integrated in an external tool

2.1.4 Remote Laboratories Scheduling

Many remote laboratories need to guarantee exclusive access to laboratories for a certain amount of time. If a student submits a program to a microcontroller and then check if the program submitted works as expected, other students should not be able to submit programs during that time. Additionally, the system must control that students only have a certain amount of time, so they after some time, the current student cannot access anymore and other student can access.

Scheduling is one of the most challenging issues in remote laboratories. Indeed, it changes very much among different remote laboratories. RLMSs attempt to provide generic solutions, but there are always exceptions that cannot be easily integrated.

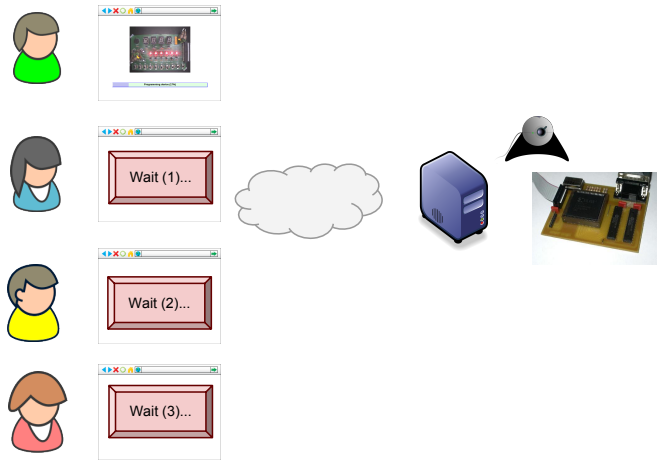


Figure 2.8: Simple queue managed by a RLMS

There are two main approaches: queueing (see Figure 2.8) or calendar-based booking. Nowadays, the iLab RLMS supports queueing for batch laboratories and booking for interactive laboratories. Labshare Sahara supports both booking and queueing for the interactive laboratories. WebLab-Deusto supports queueing for both interactive and batch laboratories (see Section 3.4). In order to offer more available slots in the calendar or to make the queue move faster, the load of users can be balanced among different copies of the experiment, as long as the experiment costs makes this possible. Figure 2.9 represents this concept: a RLMS is managing two copies of the same experiment, and four students attempt to use it, so the first two users will use the experiment, and the other two will wait in a common queue. This feature, called *local load balance*, is implemented in both Labshare Sahara and WebLab-Deusto.

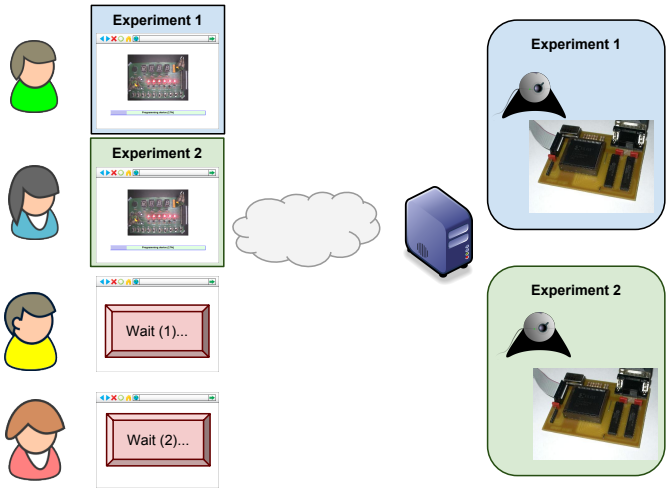


Figure 2.9: Local load balance

2.1.5 Remote Laboratory Federation Models

A unique characteristic of remote laboratories when compared to traditional laboratories is that the distance of the student to the real equipment is not an issue, so remote laboratories can be shared with other institutions. One entity can share a laboratory to other entity. We refer to *entities* rather than universities since they do not need to be universities: research centers may be interested in sharing local resources as part of an agreement, and secondary schools would reasonably be consumers.

This sharing can be managed in a direct, simple way: the *provider entity* (the one where the equipment is located) creates accounts of users of the *consumer entity* (the one interested in using the provider university's equipment for their students). Students of the consumer entity directly access in the provider entity and the provider entity does all the work: it authenticates the user, authorizes him to use the laboratory and provides the laboratory.

There are multiple problems with this solution. First, the provider entity must create and manage the user accounts of all the interested consumer universities. In a complex scenario, where a wide variety of consumers are involved -such as foreign universities and even secondary schools that simply do not speak the provider entity's language-, this approach does not scale. Second, the management of this approach is cumbersome: consumer universities would need to notify providers every change, and local databases or protocols such as LDAP would not be available. Third, the consumer entity cannot carry a proper accounting of the uses performed: it must trust the provider entity. If both institutions come to an agree-

ment where users of the consumer entity can access up to 10,000 times, there will be no way for the consumer entity to audit this if the provider entity at some point says "you have already reached the limit."

In order to handle these and other problems, a two-side model is required (see Figure 4.1), where both universities have the same software framework that manages this sharing. The consumer entity can authenticate and authorize local students, and once authorized, the local framework will contact the provider entity and request a slot. This way, the provider entity does not need to manage students and courses of the consumer entity, and the consumer entity can track all the requests performed to the provider entity, being able to track students and audit the overall use.

In this sense, MIT iLabs have been effectively sharing remote laboratories around the world for years (Harward et al., 2008b,a). Different universities can use the MIT iLabs framework to develop, maintain and share their remote laboratories with other universities, as addressed in Section 2.3.1. WebLab-Deusto also provides a federation model for sharing laboratory, which is the main contribution of this dissertation and is addressed on Chapter 4.

2.1.6 Structure of this Chapter

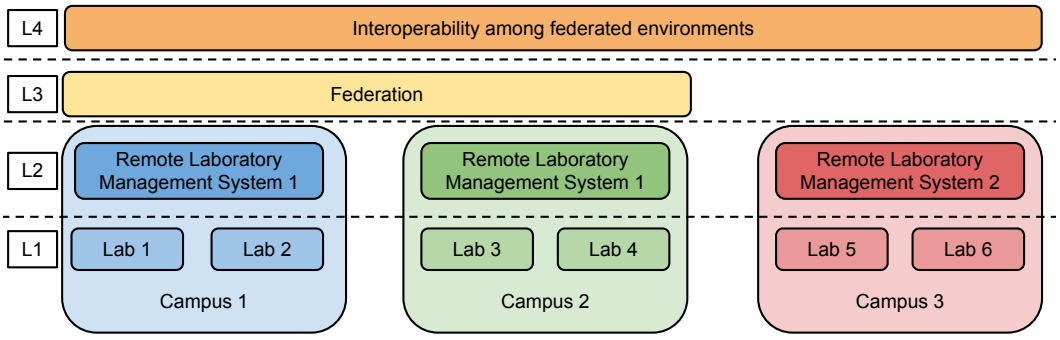


Figure 2.10: Different levels to be analyzed in the state of the art

Now that all the basic concepts have been explained, we can differentiate four different levels related to federation models, which are the main topic of this dissertation. The four levels are described in Figure 2.10, and they are:

- *L1:* Laboratories, which are connected to the equipment and manage all the logic and user interface for the target experience.
- *L2:* Remote Laboratory Management Systems, which manage one or more laboratories, providing features such as authentication, authorization,

scheduling, user tracking or administration tools. These systems can be dedicated software systems (such as WebLab-Deusto, iLabs or Sahara); otherwise those features will be implemented by the Laboratories themselves.

- *L3:* Federation of remote laboratories. Federation occurs when different educational entities are sharing accesses of laboratories with other entities. While this can technically be done on top of the laboratories directly, it is commonly done on top of RLMSs, being other feature provided by the system.

- *L4:* Interoperability among federated systems. Federation models tend to be defined for a particular RLMS. Therefore it is technically possible to create interoperability at federation level, relying on more than one federation models.

The requirements are passed from bottom to up: remote laboratories require certain features that are implemented by the RLMSs. The RLMSs support different generic ways to support these labs, such as common scheduling schemes or life cycles. These schemes and interfaces are pushed up so the federation models are defined on top of them. Finally, interoperability layers at federation level will have to deal with the requirements of the different types of federation models.

For this reason, this chapter has been organized analyzing each particular layer:

- In Section 2.2, several remote laboratories are explained. The focus of this part will not be the equipment attached or pedagogic aspects, but the requirements it propagates to the RLMSs. To do this, a survey was performed among multiple remote laboratory developers to see what requirements they had from the scheduling point of view.

- In Section 2.3, different RLMSs are analyzed, except for WebLab-Deusto, which is analyzed in Chapter 3.

- In Section 2.4, the federation models propagated from the analyzed RLMSs are described, except for the federation model of WebLab-Deusto, described in Chapter 4.

- Finally, in Section 2.5, certain ongoing efforts in the area of remote laboratories interoperability at federation layer are presented.

2.2 Remote Laboratories

This section describes a variety of remote laboratories. The focus of this section is not providing an exhaustive list of existing remote laboratories (Garcia-Zubia and

Alves, 2012), but a list of different remote laboratories with different features that may affect the federation model. The proposed federation model will not support all these remote laboratories, but this section will be used during the discussion of the limitations of federation models.

Given that this section is focused on some technical aspects which are usually not described in the literature, a survey was performed by the author to gather this data. This survey (published in (Orduña and Garcia-Zubia, 2011)) was performed during the summer of 2010, concerning the 2009-2010 course. The survey was submitted to 20 remote laboratory developers, and 14 of them provided accurate data. While the particular data has changed during this time, these changes will not be conceptually relevant to the discussion. The survey contained the following questions:

1. Relying Remote Laboratories Platform / Rig Management Framework used
    - If doubting, just say the name of your remote laboratory; some experiments are built on top of a Rig Management Framework. These could be MIT iLabs, Sahara, LiLa, VISIR, own one (specify name).
2. How many users used the experiment?
3. How many times was it used?
    - By all students and demo users if any.
4. What scheduling system does the laboratory provide?
    - None (if the system is available the user uses it, else an error message is displayed), a queue, a priority queue, a regular booking system, a mix of the previous options...
5. What is the time slot assigned to each student for a single usage (if any)?
    - Sometimes, users have access to the experiment for a fixed slot of time (60 seconds, 1 hour, etc.) or a dynamic one (10 minutes and if no one is waiting this can be expanded).
6. What is the distribution of time actually used by your students in each slot?
    - Maximum time used by a student for a single acquirement of the experiment, mean time used, mean deviation, etc. If you don't have accurate data, please make a guess and specify that it was a guess.
7. What is the required initialization time, if any?

- If it requires some time for programming the device and so on, say the number of seconds (or if variable explain how variable it is). Otherwise, say 0.

8. What is the disposing time, if any?
   - Once the student has finished using the experiment, does it require some time for cleaning the resources? How much?
9. If the experiment is virtual, what are the hardware requirements of it?
   - "N/A" if it requires hardware attached (electronic board, robot, etc.). Examples: a virtual experiment might require 512 MB in memory (so few instances can be deployed in a single server), and takes 2 minutes to run in a modern machine, etc.
10. If the experiment is real (instead of virtual), how many rigs ("copies") of the experiment were provided?
    - "N/A" if it is virtual. Was the load of users balanced among different physical installations of the experiment? How many?
11. If a queue is used, what was the distribution of number of students in different positions of the queue?
    - "N/A" if no queue is used. What was the longest queue formed, mean size of the queue when the queue was used, etc. If you don't have accurate data, please make a guess and specify that it was a guess.
12. Comments

The first subsection of this section will be a description of the technical aspects that will be analyzed in each solution and which motivated these questions. The rest of the subsections will be the remote laboratories themselves, including relevant answers to these questions.

2.2.1 Aspects to Study

Comparing two remote laboratories is as complex as comparing two traditional hands-on-lab laboratories, given that there are so many aspects that can be evaluated. It is possible to compare the costs (development, maintenance) (Lowe et al., 2012a), the technology used (Gravier et al., 2008), networking and deploying issues (Chen et al., 2010; Qiao et al., 2010), or its learning outcomes for a particular

subject (Bright et al., 2008; Nickerson et al., 2007; Corter et al., 2011; Garcia-Zubia et al., 2011).

However, these criteria are not critically relevant for the federation model. The costs of remote laboratories are not only a motivator for sharing remote laboratories (and therefore for building a federation model), but they are also the basis on top of which any business model could be develop on top of a federation model. However, whether a laboratory costs 10 euro or 100,000 euro, the model itself will not change. The particular implementation may need to be audited more carefully if that equipment is at risk, but the federation model remains the same.

The technology used by each remote laboratory is also not highly relevant for the federation model. It is important to acknowledge that as shown in Figure 2.11 (from (Gravier et al., 2008)), the number of technologies involved in existing remote laboratories is very high. This is particularly relevant for the implementation used in this dissertation (WebLab-Deusto, explained in detail in Chapter 3), which will need to target multiple frameworks and approaches. But the federation model should be able to be implemented in any technology (or, as it is the case when using a remote laboratory management system, it should implement it and the remote laboratories should remain in other technologies (Orduña et al., 2009)). The federation model remains independent of the particular protocols used and their impact on its deployment (Orduña et al., 2012a).

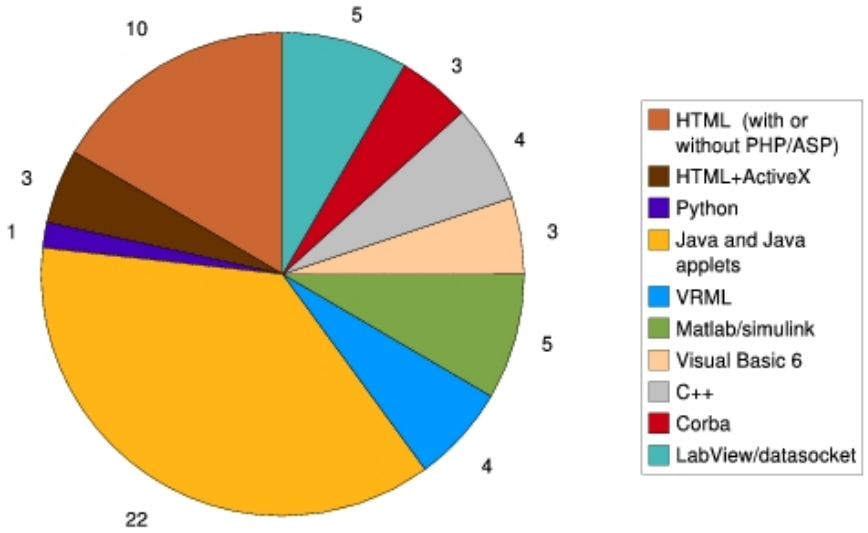


Figure 2.11: Technologies used in remote labs, by (Gravier et al., 2008)

However, there are other criteria which are highly relevant for the federation model itself, which are the aspects involved in selecting a proper scheduling mechanism. Examples of these aspects are: time used by each session, amount of potential users, average number of concurrent users, number of copies of the remote laboratory.

Scheduling could be considered a cross-laboratory requirement, since most remote laboratories will require or benefit from a scheduling system. Basic models of scheduling schemes in remote laboratories have already been studied in the literature (Maiti, 2010b). The functionality of the scheduling schemes is sometimes very domain dependent. For instance, there are experiments that can manage tens of concurrent users per second, where a calendar would not make sense, but even a queue-based API that requires several steps such as "start, check, stop" would add too much latency.

Additionally, when building a federation of remote laboratories, properly selecting scheduling schemes is crucial. Both the provider university (the university which hosts the physical experiment) and the consumer university (the university whose student is using the experiment) should support the same schemes if there is any plan to support load balancing among different institutions. If both universities use the same remote laboratory management system, this is not an issue. However, with different infrastructure providers, this would not be the case, and the scheduling schemes must be carefully considered.

In a first classification, the experiments can be classified into the following two categories:

1. Experiments where users are interacting with the experiment while it is running.

2. Experiments that do not require this interaction, letting the user submit a set of instructions that will run in the experiment and will return results that will be processed.

Most interactive experiments could be classified in the first category. The key factor is that the user will acquire a resource during a whole session (and, depending on the laboratory, also during the setup and tear down phases). Users will send commands, files, will press buttons or write texts in a remote desktop, and will see the results in real time, without anybody else using the experiment at the same time.

However, some experiments that might be classified as interactive should actually be classified as the second category. An example is VISIR (Gustavsson et al., 2007), developed by the BTH in Sweden. In VISIR, students see a breadboard that can be manipulated, and the configurations are submitted to the server. In the server, the system checks that the experiment is not harmful and with a set of relays it builds the circuit in a matter of hundredths of seconds. The measurement

is then performed and the result is returned, while the experiment can be used by other request. In Figure 2.12 the oscilloscope is shown. Since the time required by the components for providing reliable values tend to be a fraction of a second, the system can handle a number of students concurrently -in the University of Deusto, it is regularly used with up to 60 concurrent students- without losing the perception of having acquired the experiment. For instance, if the student uses the oscilloscope, one request will be performed after another, so the graphic in the screen will continuously show real measurements, even if other students are at the same time using the experiment.

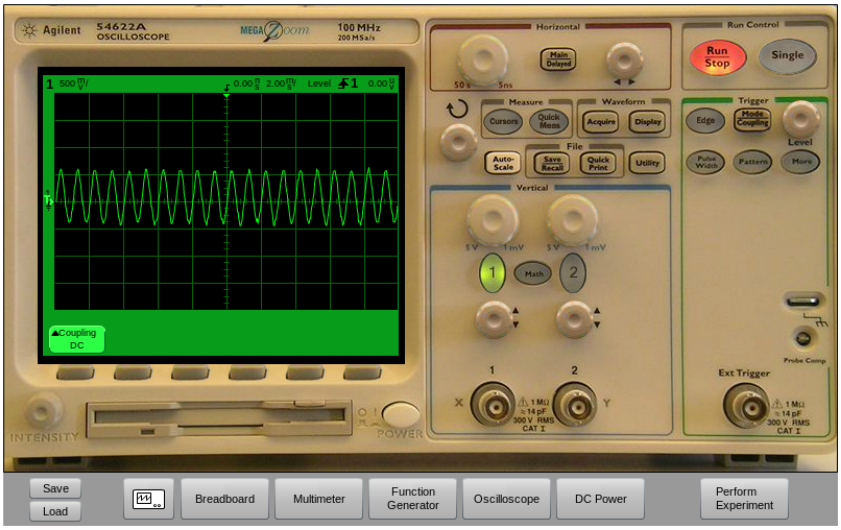


Figure 2.12: VISIR Oscilloscope

Batch experiments may all be classified in the second category. In these experiments, the user sends an experiment setup, and the user does not need to interact with the experiment while it is running. The experiment can take a long time, or it can take just fractions of seconds (as in VISIR), and the user will retrieve the results during the same session or in the future. The MIT Microelectronics WebLab (Del Alamo et al., 2002) is one of the first remote experiments, and it falls in this category. The scheduling scheme behind the Microelectronics WebLab have been a queue of users -before and after reimplementing it under the iLabs Shared Architecture-, which is the efficient approach for this category. Furthermore, no additional consideration is required in this category as long as the resources are independent (i.e., users can only request a single resource so no resource is spared while a user is in the queue), which is usually the case.

While the use of a queue is common in experiments of the second category, experiments of the first one offer a wider range of scheduling schemes. Therefore,

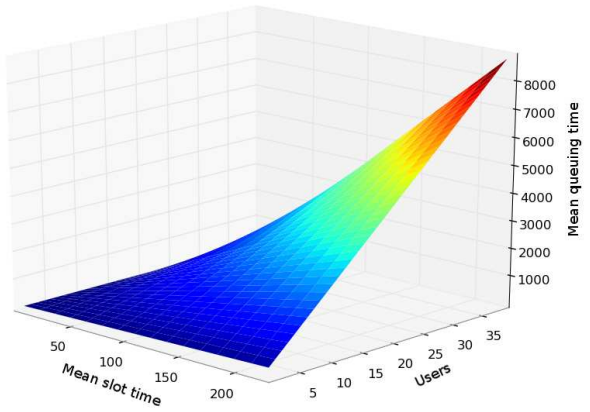


Figure 2.13: Average time waiting in a queue depending on the number of concurrent students and average time per session

a second classification within the first category can be established, referring to the scheduling scheme used by the experiment. Experiments in the first category may use queuing, booking, or a mixture of them.

The decision depends mainly on three factors, as stated in Figure 2.13:

1. Number of expected concurrent users

2. Number of available copies of the laboratory

3. Duration of each user

If there is a single copy which uses a queue, users take an average of 1 minute per session, and 15 concurrent users are expected, then the latest user will have to wait the queue for 15 minutes, which is acceptable. If 120 concurrent students are expected, or if the average time is 8 minutes, then the latest user will have to wait for 2 hours, which might cause a worse user experience, and a booking system will become more suitable. If 7 copies of the experiment are added and the system can balance the load of users among the different copies automatically, the latest user will again have to wait for only 15 minutes. On the other hand, if the number of concurrent users is low and the average of time per session is also low, a booking system becomes particularly inconvenient: users would have to state when they want to use it even if the experiment is free at that moment. Therefore, the same experiment can find more suitable one scheduling scheme or other depending on the usage of the experiment rather than on the nature of the experiment.

It is important to remark that the number of concurrent users may differ considerably from the number of total users. One class of 100 students may have access to a laboratory, and if each experiment session takes one minute, the chances of many students using it at the very same time are very low.

2.2.2 Remote Laboratories Analyzed in the Survey

This subsection explains several remote laboratories, focusing on the aspects detailed in the previous section. As previously stated, it is the result of a survey performed during the summer of 2010. The laboratories detailed here are therefore only the ones which participated in the survey. There are four remote laboratories in particular where more focus has been placed: these are the Xilinx laboratories of WebLab-Deusto (used as an example during the dissertation), the MIT Microelectronics laboratory (used in Section 4.3.2), the UTS structure beam (used when discussing federation models and remote laboratory management systems) and finally the VISIR system, extensively used in Chapter 4 for complex scenarios and in Chapter 3 as an example of a remote laboratory integrated in WebLab-Deusto.

2.2.2.1 WebLab-Deusto Xilinx Laboratories

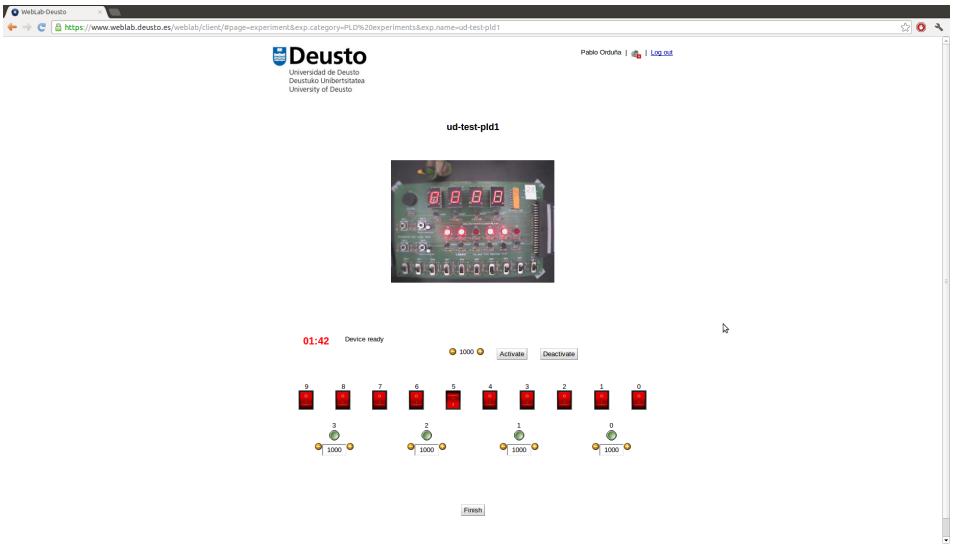


Figure 2.14: WebLab-Deusto CPLD laboratory

WebLab-Deusto counts with two remote laboratories based on two Xilinx devices: CPLD (2 copies nowadays, but only one during the 2008-2009 course) and a FPGA. Both laboratories are essentially equivalent: students learn how to

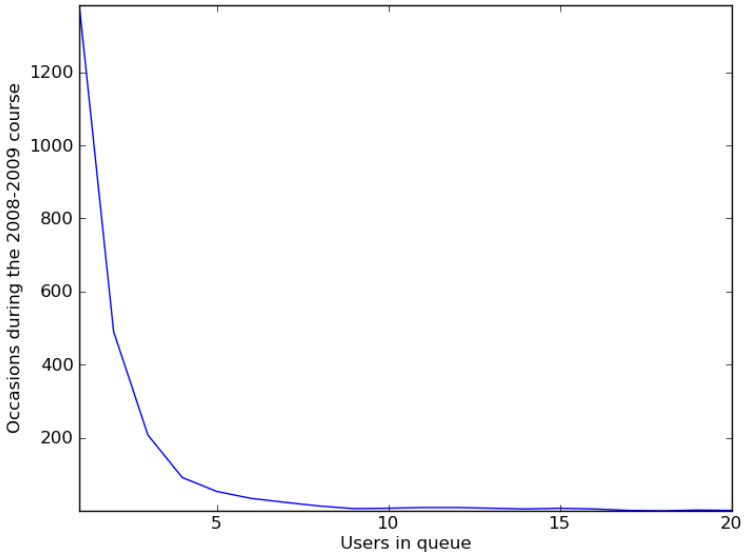


Figure 2.15: Users in queue during the 2008-2009 course in WebLab-Deusto Xilinx laboratories

program VHDL for FPGAs and CPLDS. In order to do this, students at home program a code, they compile it and upload it to WebLab-Deusto. There, they can use a set of switches and buttons (see Figure 2.14) to interact with a real CPLD or FPGA with the program uploaded. Students can see if the behavior is the expected very quickly, so they can log out (and let other students come in), change the code and re-upload it in other session. The maximum assigned time was inferior to 4 minutes (200 seconds).

These two laboratories had together 97 students during the course 2008-2009. Both experiments are managed with a priority queue. The students used these experiments a total of 4541 times during that course. However, in 2169 times (47% of the times), the student directly started using the experiment, without entering in the queue, so 47% of the times the number of concurrent students was actually 1. The distribution of the remaining is represented in Figure 2.15, showing the highest position in the queue achieved by each session. As the figure shows, 1383 times the user was in the position 0 of the queue (this is, there was one student using the experiment and nobody else in the queue); 491 times the position was 1; 1208 times the position was 2; 92 times position 3; and the number of times descents quickly, achieving 2 times where the user entered in the position 20 of the queue.

Therefore, 89.0% of the times the user had to wait less than 7 minutes (400 seconds), and 95.6% of the times the user had to wait less than 800 seconds (13

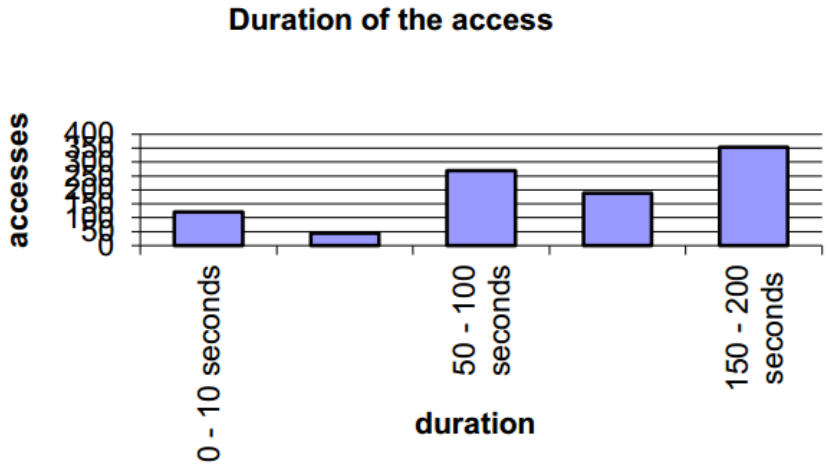


Figure 2.16: Duration of the accesses in the CPLD laboratory of WebLab-Deusto in the course 2010-2011 (Garcia-Zubia et al., 2011)

minutes and 20 seconds) to acquire the desired experiment. These numbers consider the maximum assigned time (200 seconds), but surely most of the students finished their work before that time so the queuing time was even shorter. Indeed, in the course 2010-2011, the duration of each session in the CPLD laboratory was measured (Garcia-Zubia et al., 2011) as shown in Figure 2.16, demonstrating that the mean time per session that year was actually closer to half the assigned time than to the maximum assigned time.

2.2.2.2 MIT Microelectronics Laboratory

The MIT Microelectronics laboratory was the first iLab laboratory developed. It was lead by Jesús del Álamo(Harward et al., 2008b), who back in 1998 wanted to let his students access semiconductor devices. A device suitable to be used was available in a research laboratory, but it was logistically not feasible to bring students to the crowded lab.

A grant funded by Microsoft permitted MIT to explore the potential of remote access to devices. The first version was developed, called *Microelectronics WebLab*, based on a Java applet that enabled students to submit descriptions to a server connected to the equipment. The server was developed in Microsoft ASP. In autumn 1998 it was used by students, and by spring 1999 it had already been used by almost 100 students.

Since the end of 1999, an alliance between MIT and Microsoft called iCampus was built, and within this alliance, more experiments were added to the *Microelectronics WebLab*. After adding a switching matrix, several devices could con-

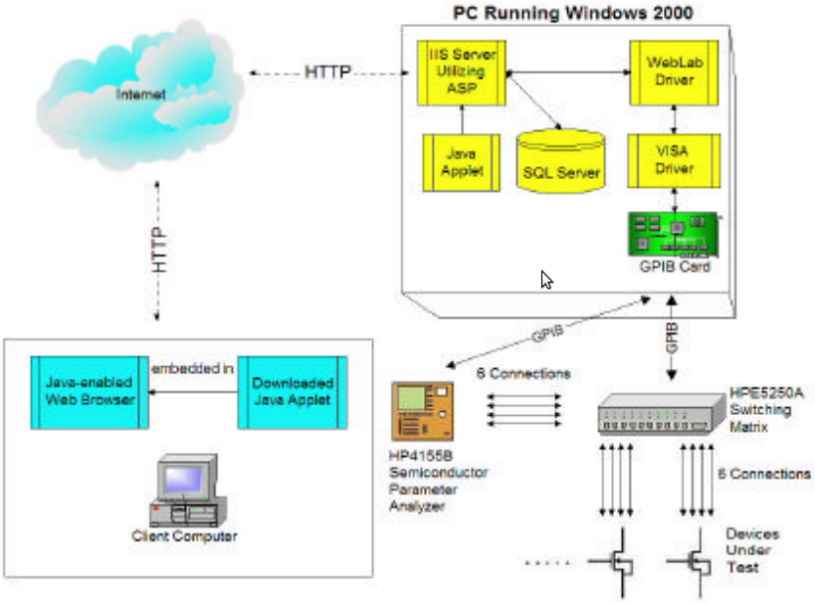


Figure 2.17: Microelectronics WebLab 4.0-4.2. Retrieved from (del Alamo et al., 2002)

currently be used. This way, in 2002 the version 4 was running with 6 different experiments running on the same equipment (del Alamo et al., 2002), as it can be seen on Figure 2.17. Nowadays, its version 7 is active and it is used by several other universities (including the University of Deusto). This laboratory is an example of batch laboratory, which relies on a priority queue.

2.2.2.3 UTS Structural Beam and Couple Tanks Experiment

Labshare[1] was a project backend by the Australian Government's Diversity and Structural Adjustment Fund, brought by five Australian universities: the University of Technology Sydney, Curtin University of Technology, Queensland University of Technology, Royal Melbourne Institute of Technology, and the University of South Australia. The target was to build a network of remote laboratories at Australian level that can be shared among different universities.

Within the project, a software solution for developing and managing remote laboratories is provided, called Sahara. Sahara is a remote laboratory management system, more deeply detailed in Section 2.3.2. Sahara Release 1 is considered the original set of remote laboratories developed in the UTS (University of Technology, Sydney), which was not publicly available. Sahara Release 2 is considered the first

[1] http://www.labshare.edu.au/

public Labshare system that actually rewrites Sahara Release 1. It was released in late May 2010, under a BSD license. These two versions were the ones referred during the survey.

Regarding scheduling, Sahara Release 2 used a priority queue for interactive experiments, where the priorities were based on the origin of the user and the type of user. Other scheduling systems, such as booking, were later introduced in Sahara Release 3.

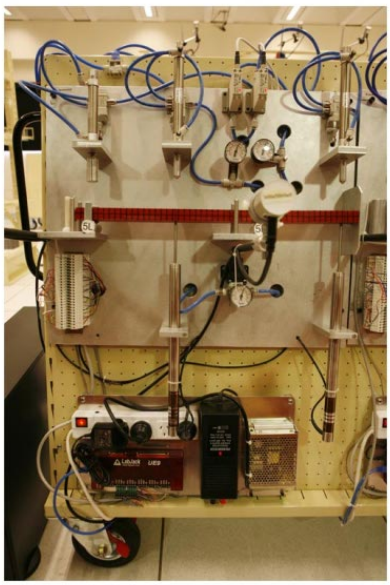


Figure 2.18: UTS structural beam experiment

However, Sahara provides a key feature, which is the ability of reusing a resource for different types of experiments. For example, an available experiment is a structural beam that can be loaded. This loaded beam can be in a horizontal or a vertical position (Figure 2.18). Depending of the exercise, the inclination does or does not matter. Therefore, if there are 2 vertical beams and 2 horizontal beams working, and all of them are being used by different students, when a new student wants to use a vertical beam, his request will be added to a queue for the vertical beams. If another student comes, requesting any beam, his request will be handled as soon as any beam is available.

This feature is very interesting, since experiments are commonly seen as resources themselves, instead of something that can be performed with a resource that may be reused for other experiments.

Sahara Release 1 was used with different experiments during the course 2009-2010. One of these experiments was the coupled tanks experiment. In this experi-

ment, users are allowed to use the experiment for 1 hour (half for demos), although during the last course they used it a mean of 37 minutes. 91 users were registered, which used the experiment 616 times. Additionally, there were 166 demo users. Since UTS had three instances of the experiment, the estimations of users in queue are 0 to 2 during the course and 5 to 10 during the last 3 days before the assignment are due.

2.2.2.4 **VISIR**

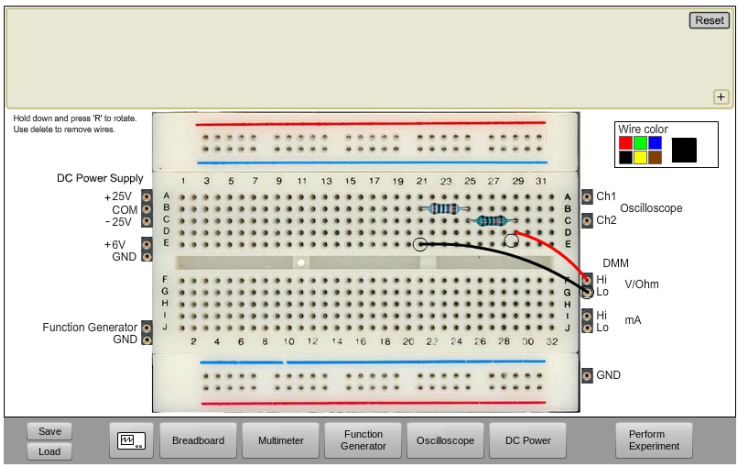


Figure 2.19: VISIR breadboard

VISIR[1] -Virtual Instrument Systems in Reality- is an Open Source remote experiment developed by the BTH (Blekinge Institute of Technology) in Sweden.

VISIR is composed by three major software components:

1. The client, developed in Adobe Flash, which provides the user interface.
2. The measurement server, developed in C++, which processes the requests from the client, validates them (ensuring that the circuits can be built and that the equipment will not be damaged) and submits them to the equipment server.
3. The equipment server, developed in LabVIEW, which controls the final equipment.

In terms of hardware, the equipment server will interact with different components (multimeters, oscilloscope, power supply) directly through PXI (a hardware

[1] http://openlabs.bth.se/

interface), and will build the requested circuits through a switching matrix based on a set of boards with relays (Figure 2.20).

Additionally, efforts have been placed (Hernández-Jayo and Garcia-Zubia, 2012; Hernandez-Jayo et al., 2010; Hernández-Jayo, 2012) by Unai Hernandez to reduce the costs of the deployment of VISIR as well as to increase the range of experiences supported by the system. So as to achieve this, the PXI component is replaced by a set of LXI components, which permits the university to use components of different companies.

As previously detailed, several concurrent users can use the experiment feeling that the hardware is only controlled by them. This is so because the requests are processed sequentially very fast. The default limit of concurrent users is 16, but this number depends on two factors. The first one is the time that the particular attached components (resistors, capacities...) require to take a valid measure. The second one is the latency that is considered acceptable for students. For instance, if a capacity requires half a second to provide a valid measurement, and the maximum number of seconds that a user should wait is 8, then the maximum amount of concurrent users should be set to 16. On the other hand, if there is no such slow capacity, the amount of users can be increased quickly. In the University of Deusto, it is regularly used with 60 concurrent students. However, once again, it is unlikely that all these students will perform the request on the same second, so the policies can be simpler. According to VISIR developers, the typical session lasts for 10 minutes approximately, while in other locations sessions tend to last more, even achieving some hours.

The VISIR experiment has been deployed in 7 universities worldwide. It is available in the original Blekinge Institute of Technology (Sweden), as well as in the University of Deusto (Spain), in the FH Campus Wien in Vienna (Austria), in the Carinthia University of Applied Sciences in Villach (Austria), in the Polytechnic Institute of Porto – Higher School of Engineering in Porto -ISEP- (Alves et al., 2011) (Portugal), in the Spanish National University of Distance Education (Tawfik et al., 2011c, 2012, 2011a) -UNED- (Spain) and in the Indian Institute of Technology Madras (IIT-M). Additionally, it is going to be deployed in the University of Stuttgart (Germany), in Qatar and in Brazil. This makes VISIR the remote laboratory with more copies around the world.

During the course 2009-2010, the VISIR deployed in the BTH had 1128 sessions. The University of Deusto, which shared it with UNED, had around 80 users, which in total used the system 480 times. The FH Campus Wien had 17 users. The usage of the system has increased and indeed in the University of Deusto during the course 2011 – 2012, it was used 1991 times by 4 different classes with a total of 104 students. ISEP in Porto has even used it in courses with 500 students enrolled (Alves et al., 2011).

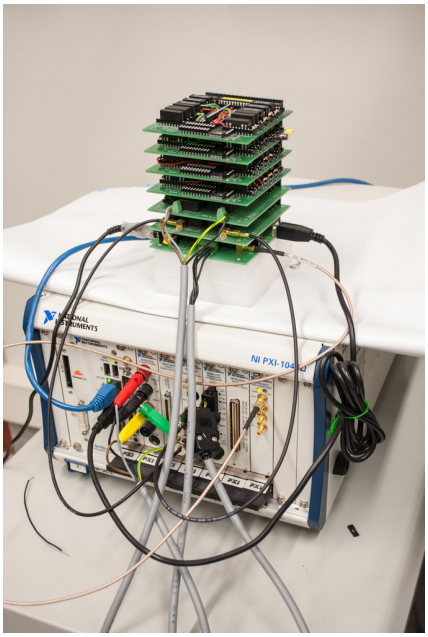


Figure 2.20: VISIR switching matrix

2.2.2.5 **OpenLabs Security Lab**

The Blekinge Institute of Technology, authors of VISIR, has other interesting remote laboratory, called OpenLabs Security Lab (Zackrisson and Svahnberg, 2008). In this laboratory, the user selects a set of machines that his experiment will require, performing booking requests such as "3 Microsoft Windows machines, 2 FreeBSD machines and 1 Linux system" for a certain moment. The system has a pool of 32 machines, and each machine has a modified GRUB boot loader installed. When the system is forced to be rebooted, the boot loader requests to a global controller what image to deploy. If the system has reserved a FreeBSD, the controller will reply that the loader must deploy a FreeBSD image. The boot loader downloads that image and boots it. The user can then connect to the experiment through Remote Desktop, VNC or SSH. When time is finished, the controller can restart the system by resetting the power distribution unit used by the machine. Then the hard disk will be overwritten to make sure that there is no original data left in the next session.

During the last course, 1403 reservations were performed. Each reservation usually does not take longer than 3 hours. However, the system requires around 10 minutes (depending on the hard drives and on the bandwidth) for downloading the image, and around 40 minutes to fully erase the 80 GB hard disks. The time slots per machine are in an hourly basis, but one additional hour is always reserved for

Figure 2.21: Servers of the BTH OpenLabs Security Lab

initialization and another for cleaning up.

This experiment is especially interesting from the scheduling point of view, since the users actually require more than one resource at the same time. If a user requests 4 machines, he will start the experiment only if there are 4 machines at the same time available. If there are 3 machines available at 2 pm, and 4 machines available at 3pm, then this user will have to wait until 3 pm. If later another student only requires 3 machines, this user will be able to access the system at 2pm.

In the case of BTH Security Lab this is not a problem, since the users see a timetable and select among the available dates. However, if this characteristic of acquiring more than one resource at the same time appeared in an experiment based on queues, well known complex situations would arise.

2.2.2.6 **ISILab**

ISILab[1], which stands for Internet Shared Instrumentation Laboratory, is an electronics remote laboratory developed in the University of Genoa (Ponta et al., 1861). It is compounded by a many independent experiments which have been developed on top of ISILab. An exhaustive list of these experiments would be: active derivator circuit, common emitter single stage amplifier, common emitter single stage amplifier with a resistance in the emitter, active integrator circuit, comparison between two types of common emitter single stage amplifiers, transistor working as a switch, complementary-symmetry stage, 8 bit AD/DA converter, 4 bit AD/DA converter, High-pass filter, low-pass filter, half-Wave active rectifier, switch with speed-up,

[1] http://isilab.dibe.unige.it/

inverting amplifier, non inverting amplifier, complementary-symmetry stage with cross-over distortion compensation and inverting amplifier with bipolar class B output stage.

All these experiments have the same scheduling system, based on queues. As in VISIR or in the user enters in a session but does not acquire the experiment. During this session, the user will submit configurations that will be queued and run in the physical experiments. These submissions usually take 500-700 milliseconds, but might achieve the 2 seconds in a single experiment. The scheduling system used is therefore a queue. During the course 2009-2010, the system was used between 800 and 1000 times by 30-40 students, using all the experiments listed above. The queue has achieved queues of less than 5 users.

2.2.2.7 **RemotElectLab**

The RemotElectLab is an electronics remote laboratory developed in the Polytechnic Institute of Porto – Higher School of Engineering, Portugal. The experiment uses an Elvis breadboard and National Instruments software both in client and server side (Sousa et al., 2010).

The experiment takes around 1 minute in complete all the measurements, and it has no initialization or disposing time. Since it was used by tenths of users, and a single Elvis setup (although they have planned to use more in the future), the system uses a priority queue.

2.2.2.8 **Fachhochschule Düsseldorf University of Applied Sciences (DUAS)**

The Dusseldorf University of Applied Sciences (Fachhochschule Düsseldorf) in Germany provides a set of industrial remote experiments (Langmann, 2004, 2005).

During the 2009-2010 course, 588 users accessed these systems. Each user's session lasted for around 5 to 30 minutes, depending on the experiment, although some experiments actually have no time limit. The system currently does not support any scheduling system at the moment, but since there is a single instance of each experiment, users just request the use in case that the experiment is not busy.

2.2.2.9 **Purdue University (CASPiE)**

The Purdue University in Indiana, United States, offers different experiments to its students under the CASPiE[1] (Center for Authentic Science Practice in Education) effort.

In both the HPLC (High Performance Liquid Chromatography) and the Raman experiments, the user will connect to a remote application that will permit the user

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

to monitor the experiment and gather generated data. This data gathering may take 2.5 to 9 minutes, with an additional time to dispose the resources (1-2 minutes). However, the scheduling system is handled through a booking system managed by the lab administrator. The Lab administrator will book a slot of 3-4 hours to each class, and during this time the users will be able to use it. Therefore, the scheduling system works at group level, supporting only some students at the same time. This model is also unique compared with the rest of experiments studied, since it establishes exclusivity: the students that can use the experiment during those 3-4 hours cannot use it during any other hour.

During the course 2009-2010, the HPLC experiment was used 784 times by 200 students, while the Raman experiment was used 17 times by 25 students.

2.2.2.10 **Ilmenau University of Technology**

The Ilmenau University of Technology, Germany, runs a hybrid interactive laboratory (Wuttke, 2006). The user will be able to use a real experiment. However, if the experiment is busy, students can use a virtual version that supports multiple students concurrently. In fact, students can practice with the virtual version and then check their algorithm with real world conditions in the physical experiment.

During the last year, it was used around 200 times. The experiment lasts 10 minutes, and it requires 30 seconds for synchronizing with the model and other 30 seconds for disposing the resources. At the moment, no scheduling system is used: the users can see that the experiment is busy or not and they can request using it.

2.2.2.11 **RLab**

RLab, a remote laboratory developed in the University of Maribor, Slovenia. The experiment consists in controlling and monitoring a real robotic arm (Safaric et al., 2005).

During the course 2009-2010, 2 robotic arms were used by 50 students, which used it in total 200 times. The average session lasts for an hour, so a booking system is used. No disposing time is required, and only 30 seconds are required to download the Java applet the first time that it is used.

2.2.3 **Summary**

This section has described a set of remote laboratories. As pointed out before, the only ones presented here are those who participated in a survey during summer 2010. The purpose of this survey was describing the wide range of scheduling systems used in remote laboratories. For this reason, instead of focusing on pedagogy or technological issues, the survey was focused on the scheduling systems used.

Table 2.1: Summary of the scheduling systems

Remote Laboratory	Type		Prio. Queue	Booking			No system
	Batch	Interact.		Reg.	Shared	Impos.	
WebLab-Deusto	✓	✓	✓	✗	✗	✗	✗
MIT iLabs	✓	✓	✓	✓	✗	✗	✗
Labshare Sahara	✗	✓	✓	✓	✗	✗	✗
VISIR	✓	✗	✓	✗	✗	✗	✗
BTH Security Lab	✗	✓	✗	✗	✓	✗	✗
ISILAB	✓	✗	✓	✗	✗	✗	✗
RemotElectLab	✗	✓	✓	✗	✗	✗	✗
CASPiE	✗	✓	✗	✗	✗	✓	✗
Ilmenau	✗	✓	✗	✗	✗	✗	✓
DUAS	✗	✓	✗	✗	✗	✗	✓
RLab	✗	✓	✗	✓	✗	✗	✗

With this data, it is possible to classify experiments in two types: batch and interactive:

- Batch experiments (iLabs batch experiments, WebLab-Deusto, VISIR, ISILab)
- Interactive experiments (WebLab-Deusto, Labshare Sahara, RemotElectLab, iLabs interactive experiments, RLab, BTH Security Lab, Purdue University, Ilmenau University of Technology, Fachhochschule Düsseldorf University of Applied Sciences)

Among interactive experiments it is also possible to classify the following schemes:

- Queues and priority queues (WebLab-Deusto, Labshare Sahara, RemotElectLab)
- Regular booking (iLabs interactive experiments, Labshare Sahara, RLab)
- Shared resources booking (BTH Security Lab)
- Imposed group booking (Purdue University)

- No system (Ilmenau University of Technology, Fachhochschule Düsseldorf University of Applied Sciences)

This denotes the diversity of scheduling systems used nowadays by remote laboratories. These results have been summarized in table Table 2.1. Note that in three cases (WebLab-Deusto, MIT iLabs, Labshare Sahara), it is summarizing what they support rather than the particular laboratories explained above. This is explained with more detail in the next section.

2.3 Remote Laboratory Management Systems

A remote laboratory management system, as explained in Section 2.3, is a software system which provides a set of toolkits to:

- Develop new remote laboratories.
- Manage common features such as authentication, scheduling, etc.

In this section, two major RLMSs are detailed: the iLab Shared Architecture, developed in the MIT (Section 2.3.1), and Labshare Sahara (Section 2.3.2), lead by the University of Technology, Sydney as part of the Labshare project. In Section 2.3.3, a discussion on if these systems are useful is presented. In Section 2.3.4, other systems which may not be RLMSs are explained. Finally, note that the WebLab-Deusto RLMS is not explained in this section because, being part of the contribution of this dissertation, Chapter 3 is dedicated to it.

2.3.1 iLab Shared Architecture

The iLab Shared Architecture (ISA) is the remote laboratory management system developed mainly by the Center for Educational Computer Initiatives (CECI) of the Massachusetts Institute of Technology (MIT). It is the remote laboratory management system more spread in the world, with different initiatives to share laboratories in Europe, Australia, Asia, and the United States. It is also the first RLMS to support federation as a native feature.

2.3.1.1 Context

The origins of the iLab project was started by Jesús del Alamo as detailed in Section 2.2.2.2 with the Microelectronics WebLab. Given the success of the results, it started being used with other universities. This way, MIT and two Singapore universities crated the Singapore-MIT Alliance (SMA) in Autumn 2000. In 2003, the University of Chalmers, Sweden, also started using the project with 350 students.

Meanwhile, in 2001, a new architecture based on Web Services was started, which would later provide a common infrastructure for all the laboratories, with a development leaded by the *Center for Educational Computing Initiatives* of the MIT. This architecture, called ISA (iLab Shared Architecture) would support natively *sharing* the laboratories among different universities. The Microelectronics WebLab would in 2004 become part of the ISA.

This architecture in the very beginning only supported batch laboratories, where students submitted tasks that would be executed and results would be returned. Students would not be able to interact with the laboratory during the assigned time. However, in 2007 the ISA started supporting interactive laboratories with a different scheduling scheme (Hardison et al., 2008).

In June 2008, 5400 students had used the system in 9 countries in four continents (Harward et al., 2008b). Nowadays the 5 continents have universities using iLab and the number of students has increased considerably, being the remote laboratory management system more spread in the world.

2.3.1.2 **Architecture**

The iLab system splits the experiments in two major categories:

- Batch laboratories (managed through queues)
- Interactive laboratories (managed through booking)

The batch architecture is represented in Figure 2.22. In the figure, there are two *campuses* which could be two different universities. Each campus has a *Lab Server*, which is the software system attached to the final equipment that will be used by students, and it contains the laboratory dependent logic. Furthermore, each campus has a *Service Broker*, which is the server that performs all the user, permissions and access management.

Students log in the *Service Broker* of their campus, where a set of credentials and permissions are associated to their accounts. Once logged in, students will be able to use those experiments they have permissions of, regardless of where they are located. This way, if a student of *University A* wants to use a laboratory located in *University B*, the student logs in *Service Broker* of *University A* and submits the requests for the laboratory to this *Service Broker*. The *Service Broker* of *University A* will forward those requests to the *Lab Server* of *University B* directly, and will return to the student the response generated by this *Lab Server*. All these communications are implemented with SOAP.

The queue is managed by the *Lab Server*, since it can be contacted by students from different *Service Brokers* at the same time. The *Lab Server* offers a Web

Service for queues which could be implemented in different ways by different *Lab Servers* as long as they maintain the same interface.

In the case of the Microelectronics WebLab, the queue is managed through an intermediate database. As seen on Figure 2.23, the *Lab Server* validates the experiments before adding them to the queue. This way, it will only queue the valid experiments, so students submitting invalid experiments will not have to wait for a time slot. Internally other process will gather and process the experiments from the database.

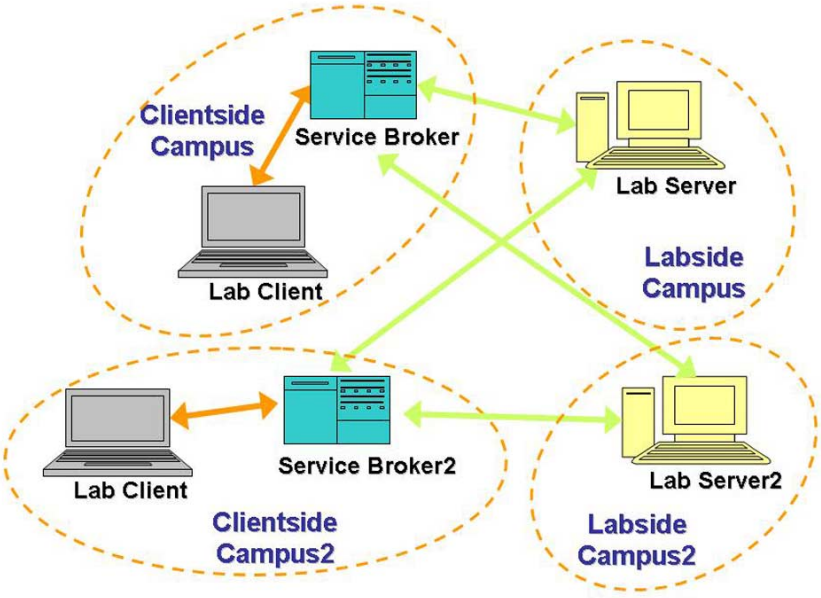


Figure 2.22: iLab Shared Architecture for batch experiments (Harward et al., 2008b)

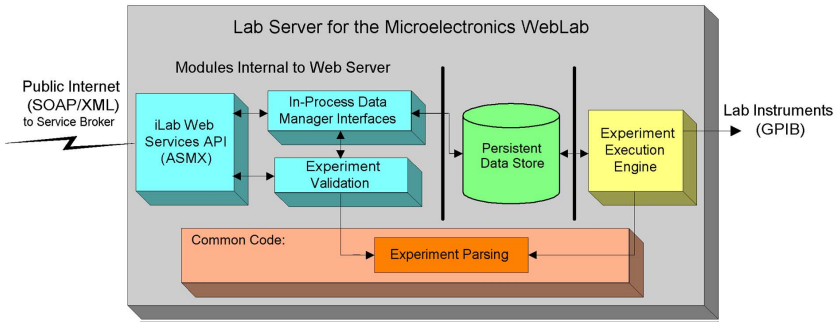


Figure 2.23: Internal queue management in the Lab Server for the Microelectronics WebLab

This interactive architecture is represented in Figure 2.24. The target of this version is to optimize the communication between client and *Lab Server*. In the batch architecture this is irrelevant, since no communication between them existed. In order to optimize this communication, the direct connection between both nodes was enabled, using the protocol chosen by the *Lab Server* developer. The communication between the different servers involved at ISA level is still based on SOAP.

However, this approach required further changes. In the batch architecture, user tracking is managed by the *Service Broker*, given that all the requests go through it before achieving the *Lab Server*. However, if clients directly talk with the *Lab Server*, it is not possible to achieve this degree of user tracking. Therefore a new component, called *Experiment Storage Service* was required. While students are performing the assignment, the *Lab Server* is submitting to the *Experiment Storage Service* the information.

Given that the interactive experiments in iLab take around 20 minutes, they do not support queues for users, so a booking system was developed. The booking system takes into account the setup and tear down times (for cleaning resources, etc.). Since these laboratories are also federated, different actors may influence in the reservations. For instance, administrators of the student side university may need to change the reservations. Administrators of the lab side university may also need to do this (e.g., for a planned or unplanned laboratory change). Given that students may reserve a laboratory in advance, the iLab Shared Architecture split the reservation in two different systems: the *Lab-side Scheduling Service* and the *User-side Scheduling Service*. This way, both sides can contact the other to manage changes done after student booked the experiment.

2.3.1.3 **Laboratory Servers**

The iLab project aims to be laboratory independent, to share the features among the different laboratories automatically.

In order to do this, the architecture separates the role of the *Lab Server* and the rest of the roles. Furthermore, since the communication is managed by lab developers, it does not need to be based on messages.

The client, in the case of the batch laboratories, can in theory be developed in any client-side technology, such as AJAX, Java applets or Flash. However, these new clients must include not only the user interface of the laboratory, but also all the communications with the server and the queue management. These communications are calls to a pure SOAP web service. In the case of the interactive laboratories, the client is left as part of the *Lab Server*, so any technology can be used (LabVIEW Remote Panels, pure HTML web pages, etc.).

To develop the *Lab Server*, Microsoft .NET skeletons are already provided.

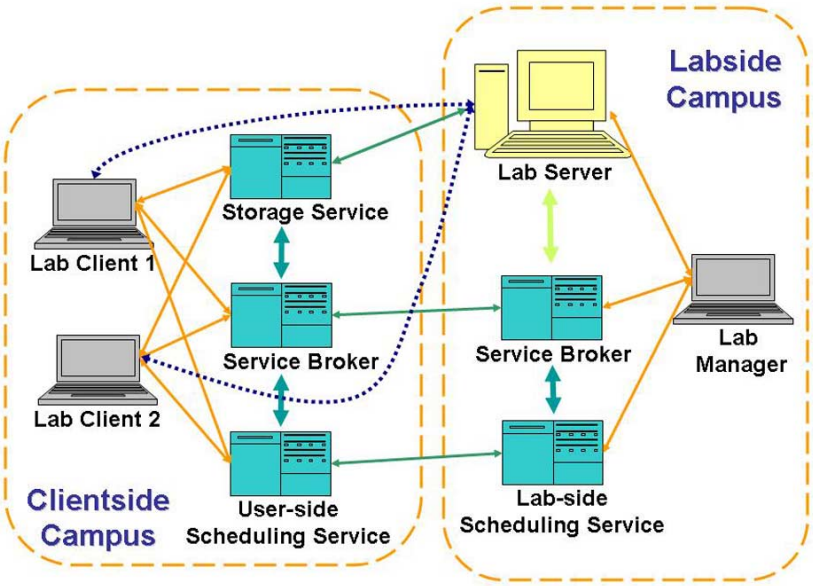


Figure 2.24: iLab Shared Architecture for interactive experiments

Other technologies could be used if they implemented the entire web service interface. In the case of the batch laboratories, this includes the queue management. That is a relevant reason for keeping laboratories in the same technology: if other implementations were pursued, a considerable amount of work would be required.

Security has been analyzed in the ISA. However, it is the *Lab Server* who submits the experiment data to the *Experiment Storage Server*. A potential security flaw in the *Lab Server* might make it difficult even to audit it, since it may even provide wrong data of what the student actually submitted. For instance, if a set of malicious commands can be executed in the *Lab Server*, it could not only do the activity, but also inform the *Experiment Storage Server* that something else was done. Given that the *Lab Server* should be implemented by teachers who are experts on the final equipment -and not security experts-, auditing the final code is especially relevant when using the ISA. However, the alternative design is that some kind of component checks the commands submitted by the student, which is also a problem in terms of latency, so the solution achieved is the better of two evils.

2.3.1.4 Limitations

Just as every remote laboratory management system (and every software system), ISA has been designed with certain use cases (in this case laboratories) in mind, and therefore it does not cover certain situations. This section points out those

situations.

As described in the previous section, the iLab Shared Architecture defines two modes:

- Batch laboratories, based on queues.
- Interactive laboratories, based on a booking system.

Therefore, it becomes difficult to manage interactive laboratories which take short time (e.g., 1-5 minutes), since it would require a booking system. Students would need to book the laboratory for that time (e.g., 5 minutes) and then perform the laboratory assignment during the time they want. Additionally, students coming in minute 3 or 4 would need to book it once the previous session is finished (and not when the previous user finished, since he could have finished in minute 2).

Additionally, one of the issues with more impact on this dissertation is the lack of load balancing. Each laboratory is assigned to a *Lab Server*. If two copies of the same *Lab Server* provide the same laboratory, the system cannot manage it, neither in the batch nor in the interactive architectures. They could be added, but presented as two different, independent laboratories. So if there are 5 copies and one of them is free at the time a student wants, the student has to manually enter in every laboratory to check which one is available. When the number of copies is increased, the problem is higher. For instance, one VISIR system could be considered a set of up to 60 independent Lab Servers which supports 60 concurrent students. In the ISA, students would need to go to the 60 laboratories to find a slot. Indeed, supporting concurrent access to the VISIR system through iLabs is a pending issue of the current integrations (Garbi Zutin, 2011).

A minor limitation is that the reference implementation of the iLab Shared Architecture, which is Open Source, has been implemented in Microsoft technologies, so it requires Microsoft products such as Microsoft Windows, SQL Server or IIS. This has an impact on the price of the overall solution when licenses are taken into account, but there is ongoing work to support both SQL Server and MySQL, to re-implement certain parts in Java and to implement certain *Lab Servers* in Linux.

Finally, while the iLab Shared Architecture is based on SOAP Web Services, many basic operations are not supported through web services, and those which are, are not accessible for final user clients. Internally, the ISA uses SOAP to communicate the different services. However, the security scheme implemented makes that if a *Service Broker* submits a request to a *Lab Server*, the *Lab Server* will contact the *Service Broker* back to check "did you really submit this request?," and whenever the *Service Broker* replies, the *Lab Server* processes the request. If third-party developers want to consume any operation, they must be first registered in the server and they must have a public IP address. Furthermore, all the regular interactions performed by students (such as logging in, and in the case of the interactive

architecture, all the booking process) are performed through a regular web user interface, which cannot be consumed so easily by third party applications. While it is possible to work around this (and indeed, applications have been developed), it is a problem when consuming it from other platforms, as reported in (Deaky et al., 2012) when integrating it in Android, and it is one of the reasons why only the batch laboratories were targeted. At the time of this writing, this is planned to be solved for authentication in the first release of iLabs of 2013.

2.3.2 Labshare Sahara

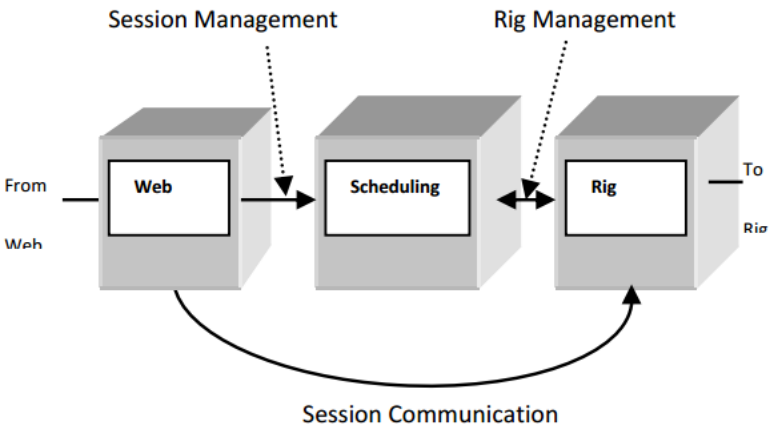


Figure 2.25: Sahara architecture (Lowe and Orou, 2012)

The Labshare Sahara project[1], as already introduced in Section 2.2.2.3, is a remote laboratory management system developed by the UTS (University of Technology, Sydney) as part of the Labshare project. Prior to the Release 2, it was actually a set of tools that was spread for each remote laboratory, not being a RLMS. In Release 2, in May 2010, it became a Release 2 on top of which several remote laboratories were migrated to provide a common centralized administration. Release 3 was released in January 2011, and incorporated several changes, such as the support for calendar-based booking, and in 2012 the Release 3.1 was released with further changes. In this section, we are going to discuss only the latest version.

The Sahara architecture is composed by three major components. The interactions among these components are described in Figure 2.25:

1. The Web Interface (WI), developed in PHP, manages the basic user interface: authentication, administration tasks, shows the permissions, etc.

[1] http://labshare-sahara.sourceforge.net

2. The Scheduling Server, developed in Java using the OSGi framework, manages the allocation of resources, contacts the laboratories to check for broken systems, and performs all the scheduling process.

3. The Rig Client, in plain Java, which is the software component attached to the final equipment.

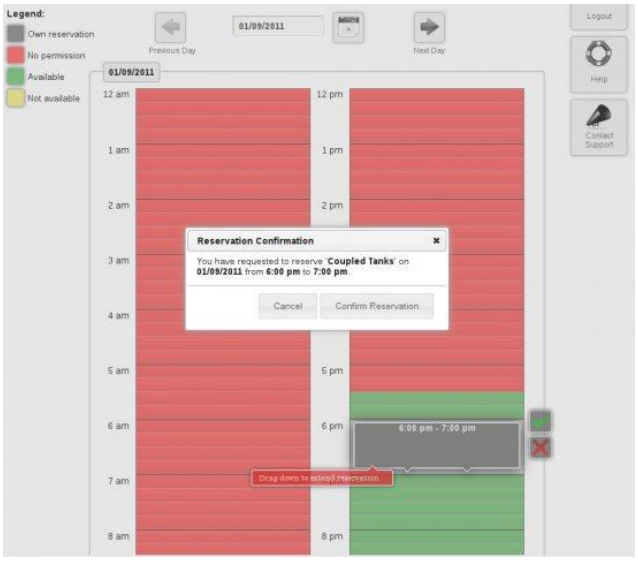


Figure 2.26: Labshare Sahara Booking system

Labshare Sahara has one of the most complex, advanced and powerful scheduling systems in remote laboratories. Already in Release 2 it supported local load balance, so the system could manage more than one copy of a laboratory. Indeed, in UTS several copies of laboratories (called rigs) were deployed, and students were balanced among them. Additionally, the scheduler split resources from requested laboratories, so students could ask for a laboratory called *load beam*, while other students may ask for a *vertical load beam* and others for a *horizontal load beam*. This way, one resource can serve more than one type of laboratory. For instance, a vertical load beam can serve students requesting *vertical load beam* and *load beam*.

In Release 3, a booking system was added (as seen on Figure 2.26). However, this booking system was not mandatory: students choose whether they want to use it through queueing (i.e., "I want to use the experiment as soon as possible") or through booking (i.e., "I want to use it tomorrow at this time"). It uses a hybrid algorithm on which the queue will not let a student use a resource if the requested time is higher than the time remaining before a booked slot. This approach collects the benefits of both worlds, although it also creates new problems analyzed in (Lowe and Orou, 2012).

Additionally, Labshare Sahara supports multiple user authentication mechanisms, including not only storing credentials in a database, but also Shibboleth and LDAP (since Release 3.1).

The main limitations for the scope of this dissertation are two: at the moment of this writing, Sahara does not support a federated scheme for sharing laboratories, and it does not support batch laboratories.

2.3.3 Notes on the Convenience of Remote Laboratory Management Systems

It is possible to question if RLMSs are really required or not. While much of the GOLC (Global Online Laboratory Consortium) community relies on one or other RLMS, or similar approaches, there is one initiative which uses a very different approach. This approach is lead by the REACT group of the EPFL (Lausanne, Switzerland).

Back in the late 90s (Gillet et al., 2001), these efforts moved in the same direction that most remote laboratory developers do in their first stages: they required log-in access, supported scheduling services (queue based) to access the equipment, etc. Furthermore, these features were considered requirements for remote laboratories (Salzmann et al., 1998). However, the evolution of this work went in a completely different direction when compared with those who moved to the RLMS-like approaches.

Instead of reusing these common *features* to develop a common software framework that could be used for developing similar remote laboratories, they considered if it was possible to remove these "*features*." Removing them makes laboratory development easier.

Indeed, they have made remote laboratories which at laboratory level cannot be broken by students, so authentication is not required. Furthermore, the laboratories have been designed to be collaborative, and multiple users can use them at the same time, therefore not requiring scheduling mechanisms. Avoiding all the layers provided by RLMS, it becomes easier to support direct, simple calls between low cost devices and the final users.

Additionally, this disruptive research line has also developed a software system based on OpenSocial called Graasp[1] (Li et al., 2012; Bogdanov et al., 2012a,b) (see Figure 2.27). On this system, it becomes possible to share easily these laboratories, or even just parts of them in the form of widgets.

This research line represents a highly interesting call to RLMS communities to re-think how many of their laboratories actually *need* those features provided by their RLMSs (and these features are not used just because their RLMS provides

[1] http://graasp.epfl.ch

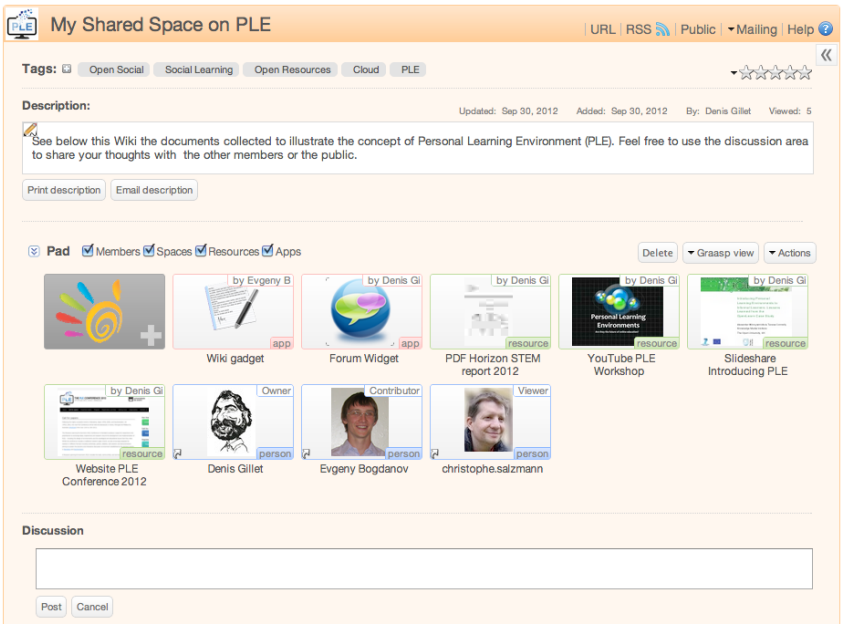


Figure 2.27: Graasp screenshot, taken from (Gillet and Bogdanov, 2012)

them). The approach used in this dissertation is to match the requirements of those laboratories detailed in Section 2.2 at RLMS level.

Additionally, the federation model proposed in this dissertation is compatible with certain Graasp features, such as the use of anonymous accounts by the consumer system, or the simultaneous access to the laboratories to promote collaboration. The research line described in this section is an interesting topic for the scope of this dissertation.

2.3.4 Other Systems

There are other systems which, while may not be a remote laboratory management system or have the impact of the ones mentioned above, are worth being noted in before continuing with Section 2.4.

Lab2go (Zutin et al., 2010) is a semantic index of remote laboratories, based on ontowiki. The project could resemble some features of remote laboratory management systems, since it shows the list of laboratories available in different universities. However, granting the access or managing the scheduling of those laboratories is outside the scope of Lab2go, since its purpose is to maintain an updated list of existing laboratories, with the documentation, locations, and semantically classified. It is a powerful tool for searching laboratories among the community of remote laboratories useful for a particular class.

Finally, there are two projects that fit in the definition of remote laboratory management systems and have been used with real student, but which while technically possible, have not developed more than one laboratory. The first one would be the NETLab project (Maiti, 2010a) (note that it is not NetLab (Nedic et al., 2003; Nedic and Machotka, 2007), the electronics remote laboratory developed in the University of South Australia), which has developed a deep research on scheduling systems (Maiti, 2011). The second one is RVLab (Muros-Cobos and Holgado-Terriza, 2012), which has been deployed with a domotics laboratory but has been designed to support further laboratories through XML-RPC APIs.

2.4 Remote Laboratory Federation Models

At the time of this writing, the only existing federation model for educational remote laboratories (apart from the proposed in this dissertation) used in production is the federation model designed in the iLab Shared Architecture (ISA). Additionally, during the writing of this dissertation, Labshare Sahara has very recently announced that it is supporting federation in future releases (Diponio et al., 2012). The overall architectures have already been addressed in Section 2.3.1 and Section 2.3.2, so here only the federation models are discussed in Section 2.4.1 and Section 2.4.2.

There are other initiatives targeting laboratory sharing but which are not federation models themselves. The closest one to a federation model is the sharing model of the LiLa project is discussed, and it is analyzed in Section 2.4.3 why it is not considered a federation model in the context of this dissertation. Other initiatives include World Wide Student Laboratory (Arodzero, 1998) –which was used by two universities directly, without any automatic process– or PEMCWebLab (Bauer et al., 2008) –PEMC stands for *Power Electronics and Motion Control*, and this project aims to share laboratories in this particular domain directly relying on a common Moodle Booking block–.

2.4.1 iLab Shared Architecture Federation Model

The federation model of the ISA relies on sharing a set of independent laboratories of one university with other universities. The sharing model has been designed to be one-to-one (in terms that the consumer university cannot re-share the laboratory to a third university).

From the client-side campus perspective, the backbone component is the Service Broker. It is the responsible for enabling other universities to use the local resources, while in that process the rest of the servers are involved.

In the interactive architecture, the Service Broker contacts the equivalent servers in the lab-side campus (as well as other servers). In the batch architecture, however, the Service Broker contacts the Lab Server directly. Therefore, there is no single point for administrating the federation system in the case of the batch architecture. If there are 4 Lab Servers and they want to be consumed by other university, they must be registered one by one.

From the scheduling perspective, it supports queues for batch laboratories and booking for interactive laboratories. In the case of the interactive architecture, the scheduling is managed by dedicated servers in both sides, so nothing needs to be implemented by the laboratory developer. In the case of the batch architecture, the queuing is managed by the Lab Server. While normally laboratory developers just reuse the existing source code from the iLabs project, it is not a separated component updated by the iLab team. Furthermore, if other technology was selected, this logic must be rewritten (as it has been done by the University of Queensland in Java (Payne and Schulz, 2013)).

Load balancing is not supported in any of the architectures. It could be implemented locally in the batch laboratories, but it becomes more complex in the interactive architecture. Federated load balance is also not supported in any of them.

2.4.2 Labshare Sahara Federation Model

The plans for supporting federation by Labshare Sahara have been described in (Diponio et al., 2012). While at the time of this writing it has not been released, the advanced analysis made by the development team in the past (Lowe and Orou, 2012), and the complex scheduling systems supported by Labshare Sahara (detailed in Section 2.3.2) makes this work especially interesting and worth being commented in this section.

The architecture currently presented is shown in Figure 2.28. In this architecture, the different scheduling servers of Labshare Sahara are communicated each other using SOAP. This way, the scheduling system used is the native of Labshare Sahara, which supports both scheduling and queuing (being able to manage both at the same time, or enable only one of them) for their laboratories (all interactive).

Essentially, and as in the case of the iLab Shared Architecture, the architecture has no centralized element, but supports that multiple organizations contact each other directly. Being a 1 to 1 architecture, it does not support re-sharing of laboratories or balancing the load of users among different universities.

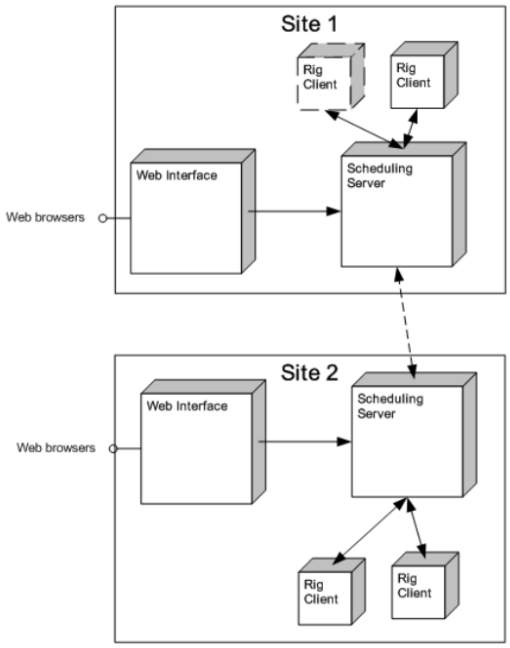


Figure 2.28: Sahara federation architecture (Diponio et al., 2012)

2.4.3 LiLa Sharing Model

The European project called LiLa (Library of Labs)[1] (Richter et al., 2011a) provides an interface based on SCORM for adding content to an LMS. Among the additions that LiLa provides, it makes it possible to contact an external server using a technique called CORS (Cross-Origin Resource Sharing), to check if the student has a currently active slot available in the final laboratory. Otherwise, students can book in advance in the system (Mateos et al., 2011).

However, the LiLa scheduling server will not contact the final laboratory server and therefore there is no way to ensure that a malicious user could go directly to the laboratory server and access it without going to the scheduling server first. There are solutions to restrict this, such as using Shibboleth to verify that at least the student is a member of the network of universities that is granted to use the laboratory, but still does not imply that the user booked the laboratory in advanced. This kind of scenarios are outside the scope of the LiLa project, given that it is a meta-framework on top of which different RLMS could be plugged rather than a traditional RLMS itself.

Indeed, LiLa does not provide tools for creating remote laboratories, since it is focused on indexing and granting access to them. For this reason, while concep-

[1] http://www.lila-project.org

Table 2.2: Summary of existing federation models

	MIT iLabs	Labshare Sahara
Federation	✓	✓
Queueing on interactive	✗	✓
Booking on interactive	✓	✓
Queueing on batch	✓	✗
Booking on batch	✗	✗
Mixed booking and queuing	✗	✓
Local load balance	✗	✓
Transitive federation	✗	✗
Federated load balance	✗	✗

tually the focus is the same (sharing laboratories), the system does not fall in the same category as other RLMSs, and neither the sharing model can be classified and compared with the other federation models presented in this section.

2.4.4 Comparison of Federation Models

In Table 2.2, the models presented (except for the LiLa model) above are presented. Both systems support federation, however in a very different way. Labshare Sahara does not support batch laboratories, while MIT iLabs supports both types. Labshare Sahara supports both booking and queueing on interactive laboratories (and even supports mixing both approaches), while MIT iLabs leaves queueing only for batch laboratories, requiring booking in the interactive laboratories. One important difference between both systems is that Labshare Sahara supports local load balancing: there can be multiple copies of a laboratory (called *rigs* in their terminology), and students are balanced among the different copies. Finally, both systems use a 1-1 basis (where one entity shares laboratories to other, but this other cannot re-share the laboratory to other entities), not supporting transitive federation or federated load balance.

2.5 Interoperability of Remote Laboratories

As analyzed on Section 2.3, each remote laboratory management system has been designed with a set of laboratories and use cases in mind. This makes them focus on certain audiences, specializing on them while trying to support the rest. The approaches taken by these systems are different, and even key features of some of them are not supported on the rest. This is common given the wide background

differences in remote laboratories in terms of technologies (Gravier et al., 2008) and approaches to create the laboratories. In order to build an ecology of remote laboratories (Harward et al., 2008a), not only a software infrastructure is required, but also a deep understanding of the student audiences. Since each system has been influenced by different student audiences, building bridges between two systems, when feasible, make it possible for each system to consume laboratories designed for other audience.

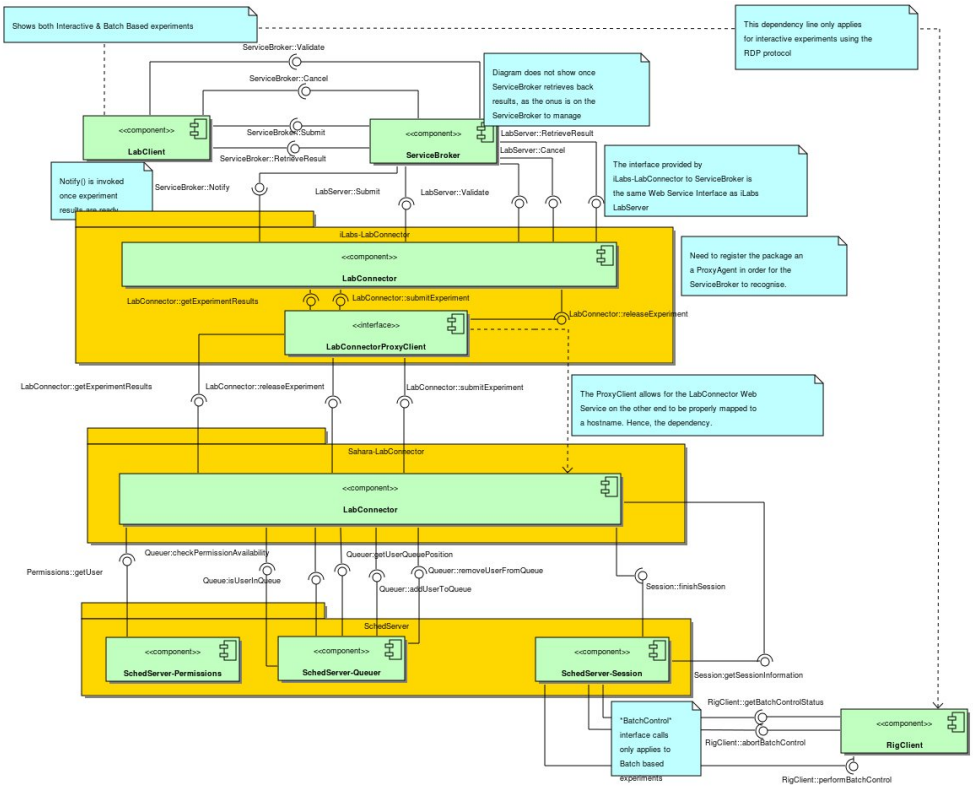


Figure 2.29: LabConnector view from the Sahara users consuming iLab laboratories by (Yeung et al., 2010)

In this context, (Yeung et al., 2010) proposed the LabConnector application protocol interface (API) as a bridge between iLabs and Labshare Sahara focused at federation protocol level. Essentially, the LabConnector mapped concepts of the iLab Shared Architecture to the Labshare Sahara and made it possible to communicate both systems, as seen in Figure 2.29. This mapping was developed with a set of SOAP calls between both systems.

The LabConnector was evaluated with the radioactive laboratory developed by the University of Queensland using iLab (detailed in (Jona and Vondracek, 2013)),

being this laboratory consumed by Labshare Sahara. While the bridge itself might have technical difficulties in becoming adopted by other systems (due to the particular mappings of both systems), it represented a clear step forward in the interoperability of remote laboratory systems.

Other major effort in this line was implemented in integrating WebLab-Deusto with the iLab Shared Architecture. This effort is described in Section 4.3.2.

Finally, efforts are being held as part of the GOLC (Global Online Laboratory Consortium) (Lowe and Yeung, 2011), under the name of ReLaSIS (Remote Laboratory System Interface Specification). These efforts aim to develop a common interface for different RLMSs, which if it is implemented by major RLMSs, it would enable them (and third parties) to communicate each other.

2.6 Related Systems

When discussing scheduling and federation of remote laboratories, it is common to think in Grid Computing frameworks. Indeed, the concept of Virtual Organizations (VOs) (Foster and Kesselman, 2004), commonly used in the Grid Computing literature, resembles the idea of distributed load balance in remote laboratory federation. There have indeed been approximations from the Grid Computing world to the educational remote laboratories world and vice versa.

The CIMA (Common Instrument Middleware Architecture)(Atkinson et al., 2006) middleware, which provides a middleware for sharing instrumentation equipment remotely to remote users, was based on Grid Computing technologies. While this definition could resemble an educational remote laboratory, in the case of CIMA the focus were the resources, not the users, and the audience was closer to researchers worldwide using remote instruments than students learning to use them. This subtle difference becomes clear when analyzing the particular use cases. An educational remote laboratory has a set of requirements, such as:

- Small setup should be required in the client side $\rightarrow$ when learning to program a CPLD, it is acceptable to install CPLD-related tools, but not a complex system to connect to other nodes. Ideally, a simple web page should be all what is required (Garcia-Zubia et al., 2009).
- User tracking: whatever students have done must be stored somehow so the instructor can later audit it and even evaluate it in certain cases.
- Roles of *administrator*, *teacher* and *student*. Teachers should be able to see what students have done, while students should only see it.
- The scheduling features tend to be lower in educational remote laboratories. In Grid Computing, due to the preemptive nature of certain processes

it is possible to split the jobs (especially when working in virtualized environments(Sotomayor Basilio, 2011)) to achieve a higher throughput. In an interactive remote laboratory where students must be present, where they cannot be re-scheduled for 10 minutes later, and where execution cannot be split, this is not possible.

Therefore, while the approaches are even compatible (and indeed, bridges could be built to access the Grid for provisioning resources while relying the learning management in the RLMS (Harward et al., 2008b)), the particular use cases are very different and therefore applying one on the other is complicated.

On the other hands, efforts to use Grid Computing in educational remote laboratories have been placed (Bagnasco et al., 2006), but they have not been consolidated and indeed in the next release of the ISILab project the Grid features were discarded.

2.7 Conclusions

This chapter presented the state of the art of remote laboratories federation models. First, a set of basic concepts used in the whole dissertation are presented (Section 2.1). Using these concepts, the structure of the chapter is described in Section 2.1.6, defining what parts are going to be defined in each layer. Using this structure, the first point is presenting a few examples of existing remote laboratories in Section 2.2. The target was not showing a deep analysis of the existing remote laboratories but a small subset that shows some special features, and they were only analyzed from the perspective of scheduling point of view.

This dissertation is focused on federation models. Federation models are not usually implemented by single purpose remote laboratories, given that all that effort would only be available for one remote laboratory. Instead, existing federation models are implemented in remote laboratory management systems. Therefore, before explaining federation models, the underlying remote laboratory management systems are analyzed (Section 2.3), and with that information it becomes possible to analyze the federation models (Section 2.4).

The existing federation models are compared in Section 2.4.4. In this comparison, sharing models for remote laboratories which are not based in providing a certain API that manages the sharing or which are not related to federation have been excluded. While a big effort has been placed on supporting multiple scheduling schemes to federate remote labs, there is currently no support for transitive federation (re-sharing laboratories) neither for federated load balance (locating multiple copies of a laboratory in multiple universities and balancing the copies among them). These two features are indeed the focus of this dissertation.

Given that part of the contribution of this dissertation is supporting interoperability through the proposed federation model, the existing interoperability bridges (Section 2.5) have also been analyzed. Finally, other related federation systems have been presented (Section 2.6), not being able to be directly applied to the scope of this dissertation.

When done well, software is invisible. There are several processors in my little camera, but I don't think of my camera as a computer. There are dozens of computers in a modern car, but we still think of it as a car.

Bjarne Stroustrup

CHAPTER

3

Remote Laboratory Management System: WebLab-Deusto

THIS chapter describes the WebLab-Deusto remote laboratory management system (RLMS). The proposed federation model, described in Chapter 4, emerged from this RLMS, and it has been implemented and evaluated on this RLMS. Therefore, it is worth immersing into the detail of this RLMS as part of this dissertation to explain what requirements this RLMS was aiming and therefore which requirements the federation model will prioritize. The chapter is divided in six sections:

- Section 3.1 shows an overview of WebLab-Deusto, detailing the history of the project to explain the design decisions taken.
- Section 3.2 describes the architecture of WebLab-Deusto.
- Section 3.3 analyzes the transversal features of WebLab-Deusto.
- Section3.4 details the scheduling mechanisms supported in WebLab-Deusto, except for the federated ones (described in Chapter 4).
- Section3.5 describes how to develop and manage experiments in WebLab-Deusto, as well as examples of experiments.
- Section 3.6 summarizes this chapter and draws the extracted conclusions.

3.1 A Bit of History

WebLab-Deusto is an open source RLMS developed by the University of Deusto since October of 2004. The WebLab-Deusto team, leaded by Javier Gacia-Zubia, had been involved in educational remote experimentation since 2001, even developing the very first version of a CPLD WebLab back in 2003.

This first version was developed by two industrial electronics engineering students, Rafael Luquin and Arkaitz Gil, aimed to enable students program a CPLD through the Internet. Starting from scratch, this first solution was focused on the hardware part of the remote laboratory, attending to the peripherals and the automation of the required tasks: programming a program on a CPLD and interacting with the CPLD inputs. As shown on Figure 3.1, the student user interface was not a web page but a Windows MS-DOS application; it was based on a TCP socket and it did not support any kind of scheduling. Internally, the WebLab-Deusto team has referred to this version as the WebLab-Deusto 0.1 version.

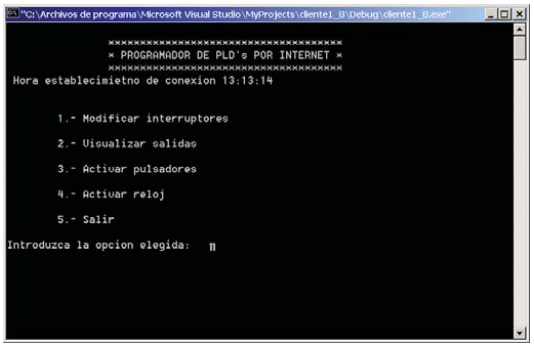


Figure 3.1: WebLab-Deusto 0.1 User interface

On October 15th, 2004, I was invited to work on the WebLab-Deusto project, with the target of making it available through a web browser. Back then, starting my 3rd year of Software Engineering, it was a very interesting challenge. By the end of December 2004, an ad hoc solution for CPLDs was developed, which was considered WebLab-Deusto 1.0. The server was developed in Python, although still using some legacy C code to interact with the serial port, and the client was a Java applet (written in Jython) embedded in the web page. The server provided a basic authentication system, a queue system to queue students, and all the communications were managed through a TCP socket. The Java applet was running outside the sandbox in order to access the student's hard disk to retrieve the selected file.

WebLab-Deusto 1.0 was the first version used with students in the University of Deusto, starting on February 2005. While it was successfully used by students and feedback was encouraging, certain problems were noticeable:

1. Plug-ins → Relying on Java applets required students to install a certain version (1.4) of the Java Runtime Environment in their computer. In the Faculty of Engineering there were 3 versions deployed during that term in different computer rooms (1.3, which was required for other software of other courses, 1.4 and 1.5). In certain computer rooms (those with 1.3) students were not able to use it, and given that they did not have administration privileges, they could do nothing about it but go to other computer room.

2. Firewalls and HTTP proxies → Relying on a TCP port was a source of issues when users were behind a firewall or behind an HTTP proxy.

With these two problems in mind, WebLab-Deusto 2.0 was developed, aiming to avoid these two problems. In February 2005, Jesse James Garrett coined the term AJAX (Garrett et al., 2005) (Asynchronous JavaScript and XML) for a set of technologies that were being used to build interactive web applications in applications such as Google Maps or Google Search. This way, it was possible to develop applications that would run on web browsers using only HTTP and therefore crossing firewalls and proxies.

In order to quickly embrace this approach, a small wrapper was developed in Mono (an open source implementation of the Common Language Infrastructure on which .NET is based) that mapped SOAP requests to the older TCP protocol. The client was rewritten in pure JavaScript using AJAX, consuming the SOAP protocol offered by the Mono application(Garcia-Zubia et al., 2005). This way, users' interaction was sent through HTTP to the Mono application, which would convert it to a socket to the WebLab-Deusto 1.0 server. This version was in production since November 2005.

The first deployment of WebLab-Deusto 2.0 was with a FPGA instead of a CPLD. Since the tools used were the same (Xilinx Impact, serial port), as well as the electronics board, no software change was required. However, it was necessary to have one server for each device (one for the FPGA and the other for the CPLD). However, plans for using WebLab-Deusto with other experiments were built, so the software architecture had to change to enable the administration of multiple remote laboratories. Additionally, maintaining two layers of software with two protocol versions was complicated.

Therefore, the design and early implementation for WebLab-Deusto 3.0 was started in December 2005. During the period of December 2005 – November 2007, WebLab-Deusto 3.0 was being developed and, at the same time, WebLab-Deusto 2.0 was being maintained and used in more courses. One revision of WebLab-Deusto 2.0 was developed to control instruments with a GPIB bus (Garcia-Zubia et al., 2007b). With the growing base of mobile device users, and the increasing quality of web browsers, it was desirable and possible to adapt WebLab-Deusto to run on mobile devices back in 2006 (López-de Ipiña et al., 2006).

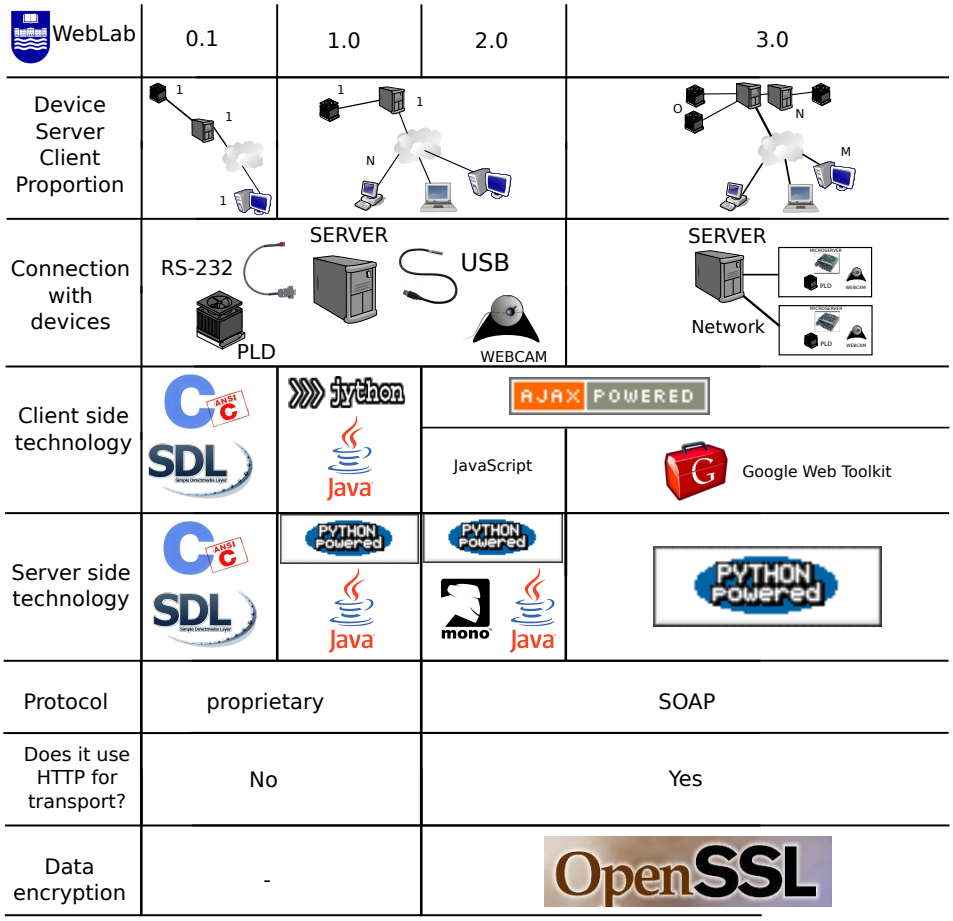


Figure 3.2: Comparison of the first 3 WebLab-Deusto versions

WebLab-Deusto 3.0 (see Figure 3.2) was finished by mid-2007 and it was deployed in November 2007. This version was a major change for the architecture of WebLab-Deusto, indeed reimplementing it from scratch, since it is the first version where WebLab-Deusto was a RLMS (Remote Laboratory Management System). Previous versions aimed to support a single type of laboratory. In order to support 3 laboratories (e.g., GPIB, PLD and FPGA laboratories), it was required to deploy three different, independent systems. However since WebLab-Deusto 3.0 (Garcia-Zubia et al., 2008), most of the tasks that these independent systems performed (authentication, authorization, user tracking, deployment configuration, scheduling...) were centralized to a single software server. This server was designed to be scalable, so multiple processes or machines can run it, increasing the system throughput.

WebLab-Deusto 3.0 was my *end-term project*. In Spain, the studies of engineering lasted for 5 years and during the last year an *end-term project* was commonly required. This project was done while attending classes, and depending on the university, it took 1 or 2 semesters. I was awarded with the *best end-term project* of the University of Deusto, as well as the *best end-term project* in Spain on the field of education by the CESEI (Spanish Chapter of the IEEE Education Society by its acronym in Spanish).

Current WebLab-Deusto is still based on the code base of WebLab-Deusto 3.0, with major changes in both architecture and code, so some current components were still designed and developed in 2005.

During the next years, the aim was to improve the system, making it easy to develop new experiments. Different programming languages were iteratively supported (Orduña et al., 2009), more authentication protocols were added, more experiments were developed, and several internal changes were performed at architecture level to improve the performance. During these years, more people have been involved in the development system. Jaime Irurzun worked since October 2007 to December 2010 in different tasks, from system administration to the redevelopment of the entire database to migrate from raw SQL to the sqlalchemy ORM, as well as experiment development and starting the administration panel. His end-term project was the inclusion of WebLab-Deusto in Second Life(Garcia-Zubia et al., 2010b). Luis Rodriguez-Gil has been working since September 2009, also working on different tasks such as the administration panel, the support of Flash in WebLab-Deusto and the integration of the VISIR project.

In January 2010, the version of the system was released codenamed 3.9, as well as the 4.0 release in April 2011. The rest of the sections of this chapter will detail the details of the current version of the repository, which will become the 5.0 release at some point of 2013, once ongoing detailed documentation is finished.

3.2 WebLab-Deusto Architecture

This section describes the current architecture of WebLab-Deusto. WebLab-Deusto supports federation of instances of WebLab-Deusto, but the design and implementation of this feature is deeply detailed in Chapter 4. Therefore this section will only explain the *local architecture* of WebLab-Deusto.

WebLab-Deusto uses a layered architecture. Users provide their credentials to an isolated server called *login server*. This server will validate the credentials and if they are valid, it will contact the *core server* requesting for a new session. From this point, users will interact with the *core server*. These credentials can be a username and a password, but also other authentication mechanisms described in Section 3.3.3.

The *core server* performs all the business logic operations. It checks if users are authorized to use a particular laboratory, lists the personal information of the user, will contact other federated environments, will track users, etc. Additionally, most of the interactions with the experiments will go through this server, and it will forward them to the proper server managing the requested or acquired laboratory.

Experiments can be spread through the university. While developing WebLab-Deusto 3.0, there were experiments in 3 different rooms of the Faculty of Engineering of the University of Deusto. In one particular case, some students had access to the physic experiments, so it could not be considered secure. WebLab-Deusto is built on the following four related assumptions:

1. Experiments can be spread through the University, in different rooms that may not have public IP addresses, so acting as a proxy for the requests may be required.

2. Experiment code cannot be trusted. Experiment developers are not always IT experts, so they may have insecure settings, or be located in insecure networks.

3. The physical security of the experiment might become compromised, so the architecture cannot rely on the experiment servers for secure information.

4. There is a wide range of technologies that can be useful for developing remote laboratories. Sometimes a Java application might become useful given that it is possible to control the sandbox on which applications are run. Sometimes it is easier to develop the application in LabVIEW. Some experiment developers may prefer .NET. Supporting security in all these technologies becomes a major problem.

With these assumptions in mind, two more servers were defined: the first one is the *experiment server*, and the second one called *laboratory server*. The *experiment server* refers to the particular laboratory that will be in direct contact with

the hardware equipment. This laboratory will be developed by the experiment developer. The *laboratory server* refers to a server deployed in the same laboratory (in the same physical room) of one or multiple *experiment servers*. The *laboratory server* is provided by WebLab-Deusto, and grants some security to the attached *experiment servers*.

Therefore, the assumptions defined above are addressed with these separate components as follows:

1. All the user requests are sent to the *core server*, which sends them securely to the *laboratory server*, and this one sends them to the final *experiment server*. This way, neither the *laboratory server* nor the *experiment server* needs to have a public IP address.

2. Even if the *experiment server* is located in an untrusted network, the *laboratory server* guarantees that the messages are sent securely from the *core server* to the room where it is located. From this point, it is technically possible to use a physical cable to add a security layer to the *experiment server*.

3. Since all the interaction is managed through the *core server*, it registers all the interactions. If a malicious user goes to the *experiment server*, he may break it, but there is no way to alter or access information sent by other users. It is also not possible that a wrongly programmed *experiment server* or manipulated equipment stops the *core server* from logging the interaction.

4. The required security layers can be adopted by the *laboratory server*, which will interact with the *experiment server*. This *experiment server* can be developed in any programming platform.

These interactions are shown in Figure 3.3, which summarizes the local architecture of WebLab-Deusto. In this figure, it is possible to see how students connect to the *core server*, which forwards the requests to the *laboratory server* of *room 2*, which forwards the requests to the particular *experiment servers*.

However, adding more layers adds two disadvantages:

1. Latency $\rightarrow$ the more layers are added, the more time it will require for a message to be sent to the final hardware equipment.

2. Flexibility $\rightarrow$ some protocols (such as RDP, VNC or SSH) simply cannot be forwarded as a set of command messages.

A special middleware called *voodoo* was designed and developed to address the first disadvantage. This middleware is described in Section 3.2.1. All WebLab-Deusto has been developed on top of this middleware, distributed as part of WebLab-Deusto. This middleware adds configuration management and manages

the server's life cycle. Developers using *voodoo* will define conceptual servers, which implement certain interfaces, and they will define which conceptual servers will be in which processes (operating system processes) and which processes in which machines. If one conceptual server requests *voodoo* to send a message to other server which is in the same process, there will be no network communication. Instead, the message will be sent directly through the process memory. Thanks to this, it is possible to even bundle all the servers (*login*, *core*, *laboratory* and *experiment servers*) in a single process, and the communication among the different systems will be as fast as if no middleware was located. Applications to this have been described in Section 4.3.3.

Therefore, as shown on Figure 3.3, it is possible the room 1 has a single server called *laboratory and experiment server*. If this server can be trusted, both conceptual servers can be deployed as a single process and avoid the added latency.

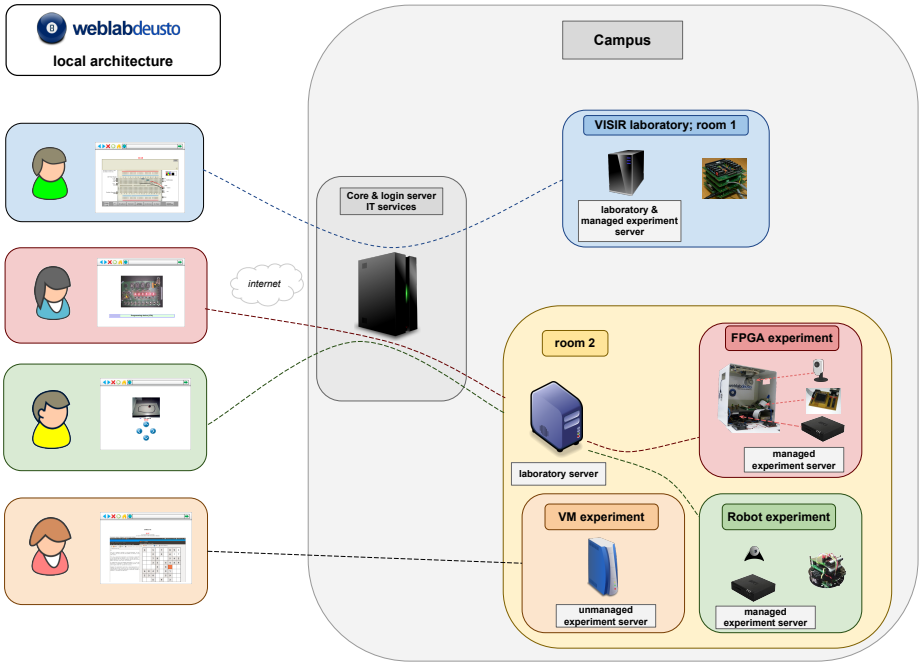


Figure 3.3: Local architecture of WebLab-Deusto

Regarding the second disadvantage (flexibility), two types of experiments are defined within WebLab-Deusto:

1. Managed experiments → those experiments which are developed with the API of WebLab-Deusto and all the messages are sent through the *core server*.

2. Unmanaged experiments → those experiments with communications not

managed by WebLab-Deusto but directly from the user. For instance, in Figure 3.3, the latest user will connect to a virtual machine experiment directly through RDP or VNC connection. This is the same case as LabVIEW remote panel laboratories.

The difference of these two types of experiments is detailed in Section 3.5.

3.2.1 voodoo: Underlying Server Management Middleware

During the design of WebLab-Deusto 3.0, a major issue was balancing between two incompatible directions:

- Support a secure, layered architecture at the cost of latency.
- Support fast connections at the cost of security.

The main problem was that the deployment of WebLab-Deusto 2.0 showed that both requirements were going to be common. This motivated the development of a middleware solution that made the system administrator choose the proper deployment on each situation. This section describes this middleware, called *voodoo*.

Using *voodoo*, developers can define a service interface, and identify it with a unique and qualified name. From this point, developers can implement it and export it using *voodoo* tools, and also develop other components which will call this interface. The communication between client and server depends on a set of requirements, such as what protocols talk each one and where they are deployed. Thus, system administrators can optimize the system to bundle services in a single process to increase speed, or to separate them in different processes or machines for security purposes.

In order to do so, in *voodoo* three different layers are defined:

1. **Server:** each software component that offers a service.
2. **Instance:** each operating system process that runs one or multiple *servers*.
3. **Machine:** each computer that will run a set of *instances*.

In the same way, five different protocols have been developed. These protocols have been represented in Figure 3.4. The protocols are:

- **Direct:** this protocol refers to a direct memory call if both client and server are running on the same *instance*.
- **UNIX socket:** this protocol uses a UNIX socket, and it serializes the request through it. It will only be available if both client and server are running on the same *machine* and this machine is using a UNIX system (such as Linux or Mac OS X).

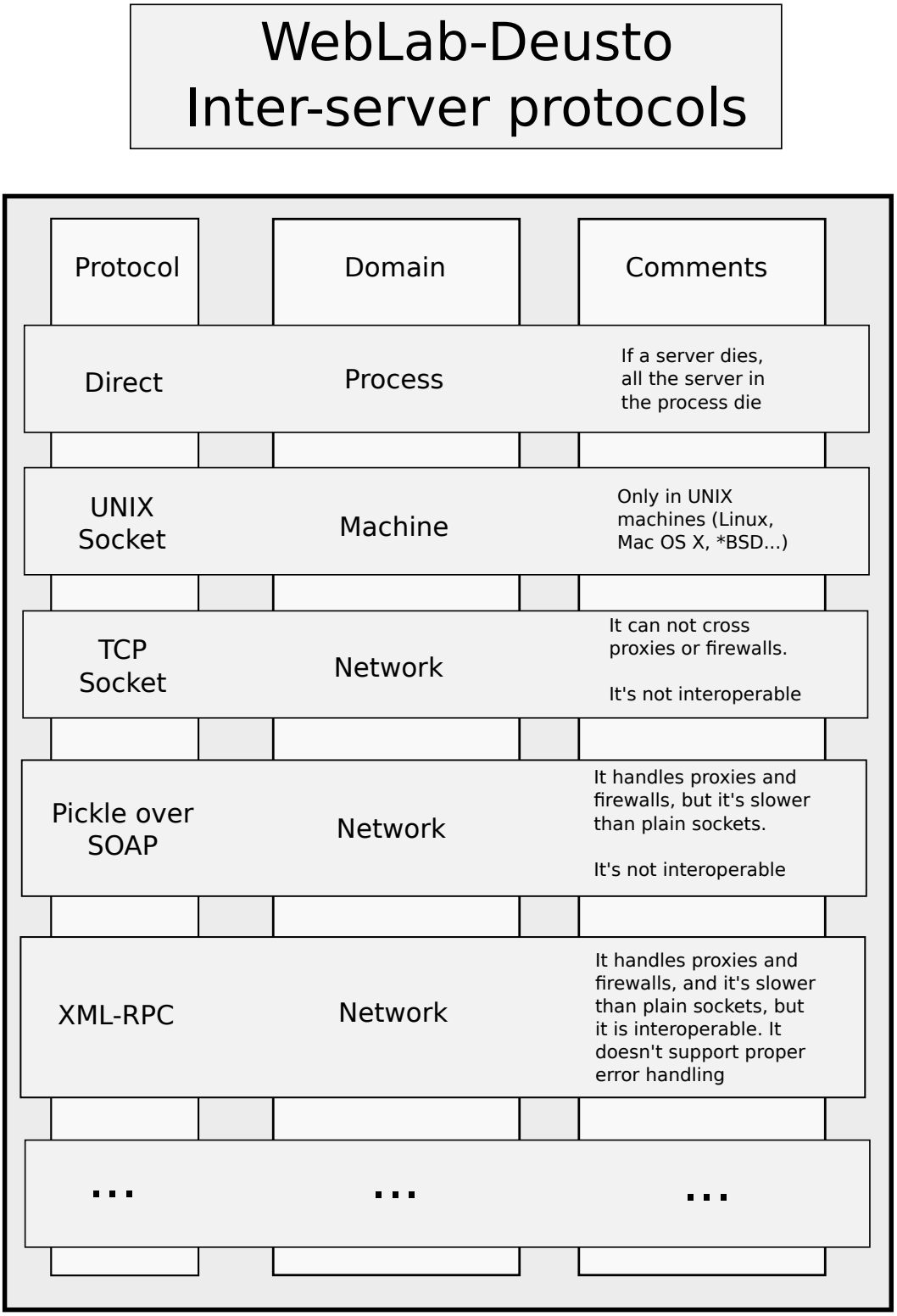


Figure 3.4: Protocols used by WebLab-Deusto through voodoo

- **TCP socket:** this protocol uses a TCP socket, and it serializes the requests through it. It will be available if both client and server are using the same network and both support it.
- **SOAP:** this protocol uses SOAP to serialize the messages. However, the messages are first serialized using a Python dependent library to support complex structures and classes, so it will not be possible to interoperate with other applications when using this protocol. It is available if both client and server are using the same network and both support it.
- **XML-RPC:** this protocol uses XML-RPC to serialize the messages. It can be used to interoperate with other platforms, but it limits the messages sent, since they cannot use complex structures.

The system administrator will be able to define which servers are bundled in which instances (processes), and which instances in which machines. This way, the administrator can define that one *experiment server* developed in Java is listening in a particular port and IP address, and a Python *laboratory server* will consume it. At the same time, it can define that in other machine both the *experiment server* and the *laboratory server* are in the same process, so the *Direct* protocol is used and no latency is added. Indeed, as explained in Section 4.3.3, it is possible to place all the servers in a single process in embedded devices to have a single, light WebLab-Deusto process.

Example: the following code shows how the *login server* calls the *core server*:

```
up_server = self._locator.get_easy_server(ServerType.UserProcessing)
session_id, server_route = up_server.reserve_session(db_session_id)
```

The interface of this method is defined as follows in the ServerType module:

```
UserProcessing = [
    'reserve_session'
]
```

Finally, it is implemented as follows in the *core server*:

```
def do_reserve_session(self, db_session_id):
    session_id = self._session_manager.create_session()
    # ...
    return session_id, self._server_route
```

Neither the developer of *login server* or the developer of *core server* need to know how these two servers are going to communicate each other. The system administrator will later define that the *login server* is configured as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<server
    xmlns="http://www.weblab.deusto.es/configuration"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.weblab.deusto.es/configuration
                                        server_configuration.xsd"
>

  <configuration file="server_config.py" />

  <type>weblab.data.server_type::Login</type>
  <methods>weblab.methods::Login</methods>

  <implementation>weblab.login.server.LoginServer</implementation>

  <!-- <restriction>something else</restriction> -->

  <protocols>
    <!-- This server supports both Direct calls, as SOAP calls -->
    <protocol name="Direct">
      <coordinations>
        <coordination></coordination>
      </coordinations>
      <creation></creation>
    </protocol>
    <protocol name="SOAP">
      <coordinations>
        <coordination>
          <parameter name="address" value="127.0.0.1:10025@PLD" />
        </coordination>
        <coordination>
          <parameter name="address" value="192.168.0.1:10025@LAB_NETWORK" />
        </coordination>
        <coordination>
          <parameter name="address" value="130.206.100.16:10025@INTERNET" />
        </coordination>
      </coordinations>
      <creation>
        <parameter name="address" value=""     />
          <parameter name="port"    value="10025" />
      </creation>
    </protocol>
  </protocols>
</server>
```

And that the *core server* is configured as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<server
    xmlns="http://www.weblab.deusto.es/configuration"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.weblab.deusto.es/configuration
                                        server_configuration.xsd"
>

  <configuration file="server_config.py" />

  <type>weblab.data.server_type::UserProcessing</type>
  <methods>weblab.methods::UserProcessing</methods>
```

```
  <implementation>weblab.core.server.UserProcessingServer</implementation>
  <!-- <restriction>something else</restriction> -->

  <protocols>
    <!-- This server supports both Direct calls, as SOAP calls -->
    <protocol name="Direct">
      <coordinations>
        <coordination></coordination>
      </coordinations>
      <creation></creation>
    </protocol>
    <protocol name="SOAP">
      <coordinations>
        <coordination>
          <parameter name="address" value="127.0.0.1:10026@FPGA" />
        </coordination>
        <coordination>
          <parameter name="address" value="192.168.0.1:10026@LAB_NETWORK" />
        </coordination>
        <coordination>
          <parameter name="address" value="130.206.100.16:10026@INTERNET" />
        </coordination>
      </coordinations>
      <creation>
        <parameter name="address" value=""     />
        <parameter name="port"    value="10026" />
      </creation>
    </protocol>
  </protocols>
</server>
```

This way, if both servers are in the same instance, they will communicate through the *Direct* protocol. If they are not in the same process, they will communicate through SOAP. In order to generate the SOAP endpoint URL, the client will check the coordination arguments. If both are using the same network (e.g., LAB_NETWORK), it will use the detailed IP address and port to contact the server.

3.3 Transversal Features

This section details information of a set of features that are transversal to the system. While scheduling might also be considered a transversal feature, it has been maintained in its own section (Section 3.4) due to the impact of this dissertation.

3.3.1 Scalability

WebLab-Deusto is designed to be scalable both in terms of number of users and number of experiments.

In order to face a bigger number of concurrent users, the load of users can be balanced among different processed or servers. This means that if there are 100 users concurrently using WebLab-Deusto and there are 4 servers deployed, each server would only have to handle 25 users, performing much better and decreasing the latency of each request. This concept is usually referred as scaling horizontally (scale out): adding more servers makes the system support more users.

Since WebLab-Deusto is a stateful server (users maintain a session in the servers, storing information such as the usage of the experiment, the position of the queue, etc.), the main problem is how to handle the sessions of the users. If there are two servers and a load balancer (which can be a dedicated hardware or a regular Apache server using the mod_proxy_balancer module, for instance) which forwards the user requests to any of the servers, and the session of a particular user is stored in the first server, when this user performs another request, it could be forwarded to the second server, which would not have the session and would therefore return an error.

To tackle this problem, there are traditionally two solutions:

- Storing the session in a database: if there is a server such as MySQL, which is accessible by both servers, dedicated to store the sessions, it will not matter which server the user is forwarded to. Both servers will handle properly the session, since every request will retrieve the session from the database and store it there again.

  - Additionally, this approach provides fault tolerance, since if one server is down, the other can still handle users' requests correctly, as long as the database servers are still working.
  - However, this approach requires the retrieval and storage of the session in the database in every single request. This causes a higher latency in each request, and a worse performance in general.

- Session affinity/sticky sessions: HTTP load balancers can check the cookies sent by the clients looking for a certain argument, such as the session identifier. Apache proxy_balancer can check for a customizable cookie and if it contains a certain string it will forward it to one server or to other. If this string is not provided or it is not recognized, it will forward it to the server which is less busy (attending to different Apache customizable policies). This way, the login method can create a new session in a particular server and set a cookie defining that this user should be forwarded to this server while using this session. If the login method is always forwarded randomly to the server less busy, users may use different servers in different sessions; however they would always use the same server during a particular session.

  - This approach supports that the session is stored in memory, thus maintaining a proper latency.
  - However, it relies on the HTTP load balancer to support this type of configuration.

WebLab-Deusto supports both approaches, while in production in the University of Deusto we use the second approach (session affinity/sticky sessions) since the latency and performance is better. However, it is possible to test the behavior of both approaches under certain constraints using the WebLab Bot as detailed here.

3.3.2 Security

Security has always been considered a mandatory feature in the design of WebLab-Deusto.

First, as already detailed in Section 3.2, WebLab-Deusto does not trust the experiment servers. They can be developed by the instruction designer, who may not be aware of many security aspects. Therefore, all the messages are passed and stored by the core server first, so as to be able to audit attacks. This is not the case in the unmanaged laboratories, and it is one of the drawbacks that teachers must be aware of when developing laboratories with that scheme.

For all the transversal layers, WebLab-Deusto natively supports common security aspects:

- It natively supports SSL to encrypt the communications with students, with other federated environments, and with LDAP servers which support it.
- Passwords are not stored in plain text anywhere: even for those students which do not support LDAP, OpenID or OAuth 2.0, and therefore the password must be stored in the database. In these cases, 4 random characters plus the hash of the concatenation of those characters with the password are stored. The system generates those characters when creating the user. For example, if the 4 random characters are “abcd”, and the password is “password”, the database stores “abcd{sha}0a4d24...”, where “0a4d24...” is the SHA hash of “abcdpassword”. This way, a malicious user managing to access the database cannot use rainbow tables or precalculated hash digests based on dictionaries to reveal passwords. Additionally, it will not be possible to see if two users have the same password, since the random characters are the random characters make the hash be different.
- SQL injection attacks are a type of injection attack on which a malicious user manages to send a message that breaks an existing SQL sentence in the system and injecting its own code. In WebLab-Deusto they are treated by using

an Object-Relational Mapping (ORM) which internally uses parameterized queries to avoid possible injections.

- XSS (Cross-Site Scripting) attacks consist in managing to insert malicious HTML code in the client to run it in the web browser of other users, and therefore being able to access private information or access. These attacks are avoided by the use of GWT (Google Web Toolkit) as development toolkit for the whole client. It already escapes all the messages inserted by the user or coming from the server. GWT supports injecting HTML code directly, but WebLab-Deusto only uses this for static or safe content.

- CSRF (Cross-Site Request Forgery) attacks are based on the ability of web browsers to submit information from one website to another, even if the response cannot be processed. This way, a malicious website could be sending messages to WebLab-Deusto such as "perform reservation, submit this malicious file." Any valid user of WebLab-Deusto who is logged in WebLab-Deusto and goes to this website could be attacked using this attack. However, all the WebLab-Deusto requests (except for the log-in method, which requires the password) require a session identifier not only in the cookie but also (and primarily) in the request itself. Therefore the attacker should have access to that session identifier, which is not possible thanks to the web browser security restrictions.

3.3.3 Authentication and Authorization

WebLab-Deusto provides an extensible authentication mechanism. This way, all users are stored in the database, but different UserAuth mechanisms can be used for each user. The system will check for each user what mechanisms are available, and will check the credentials with each system. If any of the mechanisms say that the user is valid, the authentication mechanism will understand that it is a valid user.

For instance, if a password is provided by 'student1', who has two UserAuth, one providing a password hash stored in the database, and another one detailing a certain LDAP server that is valid for this user, then the system will check one system and then the other. If any of them says that it is correct, it does not check more systems. The order of these systems is detailed in the database, so it will first check local passwords and then it will check LDAP servers, for instance.

The following authentication mechanisms are provided:

- Database: Local database stored password. The default and simplest option: A hash of the password is stored in the database.

- Trusted IP addresses: When an external entity, such as another university, needs to be able to log in as one of their students -stored locally in the WebLab-Deusto database-, a trust relationship can be built on a certain entity's IP address. For instance, a Learning Management System in the other university could have a public IP address, and WebLab-Deusto would let the LMS to connect as any of the students managed by this LMS.

- LDAP: Some LDAP servers are setup in the database, and the system will delegate to these servers the decisions. LDAP is an application protocol for reading and writing directories. Through these protocols it's possible to gather information of students from a LDAP infrastructure of the University, and it is possible to use LDAP to authenticate users. WebLab-Deusto uses LDAP to register users and to check that the password provided by the user is the password used in the system. Therefore, for a certain amount of time, the university credentials are handled by WebLab-Deusto. It does not store it in any format, but if the WebLab-Deusto server is hacked, the credentials of those users using the system during that time are in danger. This is the approach used with students in the University of Deusto.

- OpenID: WebLab-Deusto supports OpenID. OpenID is an open standard that enables the decentralized authentication. The authentication process, which consists on a user demonstrating the system that he really is who claims to be, can be handled by remote servers in a transparent way.

- Facebook: WebLab-Deusto students can log in through Facebook[1]. This integration was developed using OAuth 2.0, so the same code could be used to use other systems supporting OAuth, such as Google Accounts.

3.4 Scheduling

WebLab-Deusto has traditionally used a priority queue, so different students were queued and balanced among the different instances of the copies available. For instance, if there are 3 different copies of a certain type of laboratory (CPLD, for instance) available, and 4 users attempt to use a CPLD, the first 3 users will use it for a limited amount of time and then the fourth one will be queued and it will use the first available copy of the laboratory once any user frees it.

Prior to 4.0, these queues were built for each experiment type. Therefore, if a certain experiment could run on more than one device, it would not be supported, or two queues would be required. For instance, in the University of Deusto there are different Xilinx experiments deployed (CPLDs and FPGAs). Since most of

[1] http://apps.facebook.com/weblab-deusto/

the code (both in client and server) is common, it was possible to perform some common experiments. However, since there was a queue which could solely be used by CPLD copies and another for FPGAs copies, it was not possible to schedule experiments that would use any of them. It was also not possible to establish that a particular copy (such as one of the CPLDs) was present by more than one queue.

In WebLab-Deusto 4.0 these situations have been implemented, by splitting the "Experiment Type" and the new concept of "Resources." This way, there is a queue for the resource "CPLDs," and another queue for the resource "FPGAs." When a user attempts to reserve an experiment, the reservation will be queued in all the possible queues that might be valid for what the user requested. If a reservation request can be handled only be CPLDs, then it will only be queued in the queue of the "CPLDs." But if it can also be handled by CPLDs or FPGAs, it will be queued in both systems. When the student asks for the state of the reservation request, the system always reports the best result possible (if there are 10 reservations for FPGAs before and only 5 for CPLDs, the system will report that there are 5 users before).

This way, a single device can be reused for many experiments at the same time that an experiment can be built on top of different devices. For the experiment developer this is handled automatically. If a device is used by 3 different experiments, the software of the three systems is deployed and running, but the scheduler of WebLab-Deusto will handle that no student will use a certain server if other one is being used.

In order to clarify this, a set of exemplary situations are analyzed.

First, in Figure 3.5, there are three resources which are all Xilinx devices (Resource 1 and Resource 2 being CPLDs and Resource 3, which is a FPGA), which at the beginning are available. When a first user requests a CPLD, Resource 1 is assigned. Next, other user requests a CPLD, and Resource 2 is assigned. Then, a student requests "Any Xilinx device," so the Resource 3 is assigned. Finally, a student requesting a FPGA will be in queue in position 1.

In Figure 3.6, in the same three resources are in the beginning busy (already assigned to somebody else), and the queues are empty. When the first user requests a CPLD, he is in queue in position 1. A second student requesting a CPLD will be in position 2. While there are two copies of the laboratory, there is somebody else (the first student) in queue before him. When a third student comes in and requests "Any Xilinx device," he will internally be appended to both queues: the queue for CPLDs -where he will be in position 3- and the queue for FPGAs -where he will be in position 1-. He will be kept in both queues until he is assigned one laboratory, but every time the student requests the position in the queue, he will be returned the best result, which in this case will be 1. Similarly, if a fourth student requests a FPGA, he will be appended to the FPGA queue, and he will be in position 2. All these queues internally support priorities. Therefore, if the fourth student had

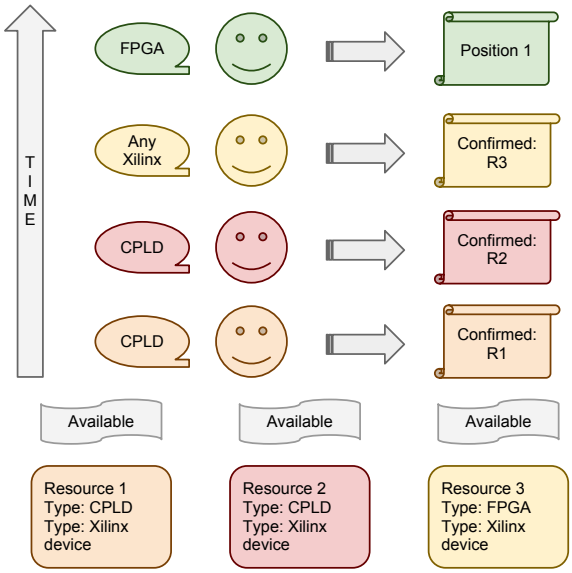


Figure 3.5: Multiple queue usage example

a higher priority than the third one, the fourth student would become in position 1. The third student would be in position 2, since the position in the FPGA queue is still lower than the position -3- in the PLD queue.

So as to implement this, WebLab-Deusto has a plug-in based scheduling system with a common API. As it is described in Chapter 4, both the federation protocol and the interoperability with iLab (batch laboratories being consumed by WebLab-Deusto) are indeed implemented as plug-ins of this system.

The plug-in system is described in Figure 3.7. For each resource type (e.g., *microcontrollers*), there will be a scheduler implementation (e.g., a queue). Once all the schedulers have been created, for each laboratory identifier (e.g., *ud-logic* of category *PIC experiments*), there is an `IndependentSchedulerAggregator`, that will aggregate multiple schedulers. For example, one scheduler could be *laboratories in other university*, and other could be *the local queue of PICs*. If *ud-logic* is configured with an aggregator of both systems, then it will perform a reservation in both systems and whenever the user requests the current position, both will be checked and the one in a better situation will be returned. For instance, if it is in the first position of the queue in one of the schedulers, and it is in the third position in the other, it will return that it is in the first position. However, the schedulers are global and can be shared among different `IndependentSchedulerAggregator`. This way, other laboratory could be using the same exact queue in other independent aggregator.

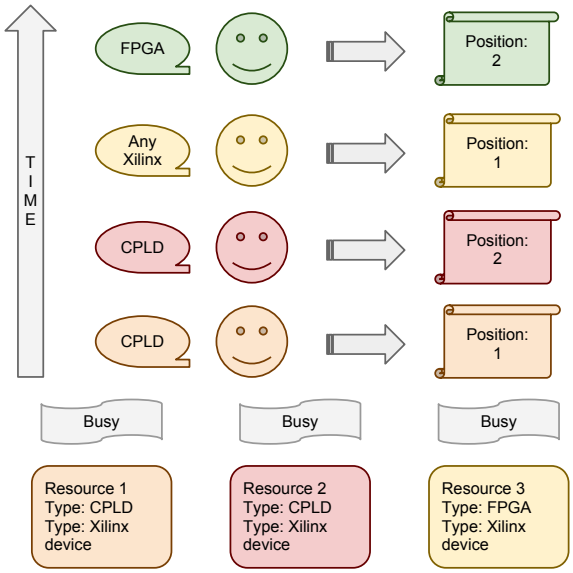


Figure 3.6: Multiple queue usage second example

3.5 Experiment Development

As already introduced in Section 3.2, WebLab-Deusto defines two types of experiments:

- Managed experiments → those experiments for which all the communications are managed and tracked by WebLab-Deusto.
- Unmanaged experiments → those experiments which manage their own communications outside WebLab-Deusto, commonly using external protocols such as VNC, RDP, SSH or LabVIEW Remote Panels.

The target of both systems is to make it possible to other developers of other institutions to create their remote laboratories using WebLab-Deusto.

In the initial design of WebLab-Deusto, only managed experiments were supported. This was so for a set of reasons: from the security perspective, in the managed scheme, all the communications are going to be based on HTTP, crossing firewalls and HTTP proxies, and being secured by the system administrator through HTTPS. From a deployment perspective, WebLab-Deusto manages the networking in the server side (e.g., if the experiment server is in other server in a private network). From teachers perspective, every interaction is going to be stored

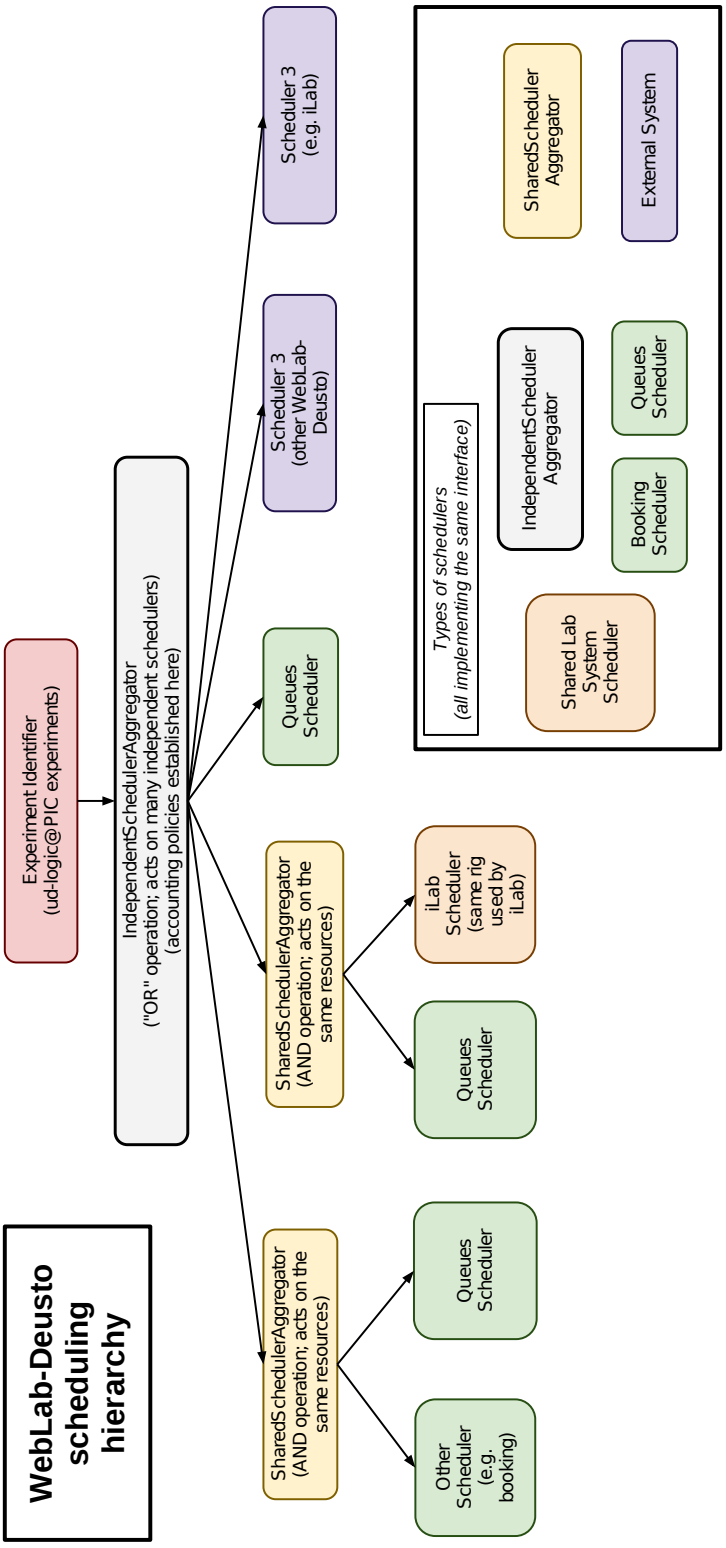


Figure 3.7: Scheduling system

so fine-grained user tracking is possible. And the support for mobile devices increases, since it is possible to make it only dependent on standard web browser technologies. These reasons are analyzed in (Garcia-Zubia et al., 2009).

However, managed experiments are based on a request/response basis, so essentially, the client submits a set of commands and parses the responses. While this is fine in a number of scenarios, the experiment development becomes a rather difficult task. It is not possible to let students to access a virtual machine through VNC, RDP or SSH, neither to access LabVIEW Remote Panels, nor to build a standalone web application outside the WebLab-Deusto API.

For this reason, release 4.0 introduced the unmanaged experiments. In this scheme, the communications are managed by the experiment developer. This way, a trade-off is achieved between supporting more deployment situations -both in server and client sides: more devices supported, clients can be behind firewalls or proxies...- and making the development easier. The experiment developer must be aware of this trade-off and he must take into account that, especially when it is shared with other universities, networking problems typically arise and it is not always possible to request IT services to open certain ports. However, if at their own risk they want to use a technology that does not suit in the managed scheme, WebLab-Deusto provides the unmanaged scheme. This is the reason for which WebLab-Deusto laboratories are all developed with the managed scheme, while the unmanaged is supported.

Internally, the line between a managed and an unmanaged experiment is very thin. Unmanaged experiments are internally experiments developed with the managed experiments API, but ready to be configured to support other systems without programming, such as virtual machines or LabVIEW Remote Panels. Additionally, most managed laboratories do not use the managed experiments API to manage the webcams, so they could be considered "partially unmanaged." Therefore, the split between managed and unmanaged experiments is closer to a conceptual barrier than to a strict barrier.

3.5.1 Managed Experiment Development

WebLab-Deusto supports experiments developed in other technologies (both the client and the server). The idea is that the client will be a web-based client technology (Java, Flash, JavaScript, etc.) that will call functions of the framework, and the framework transparently will call methods of the server using standard protocols. At the moment, the only standard protocol supported in the server is XML-RPC. Anyway, WebLab-Deusto provides libraries that can be used by the developer of the experiment in a number of platforms, as seen on Figure 3.8.

This way, the client must implement a subset of the following methods. They are all optional and manage the life cycle of the experiment. They are expressed

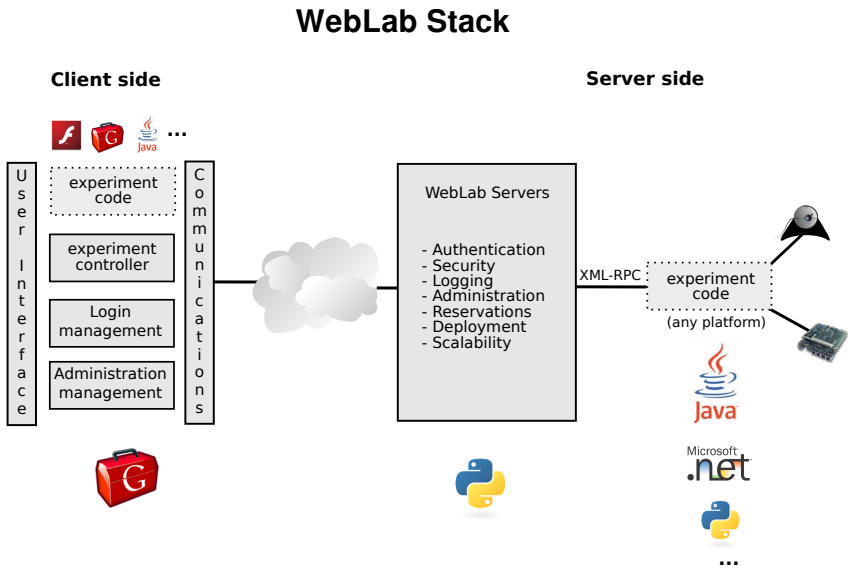


Figure 3.8: WebLab-Deusto stack: client side (left) and server side (right)

and documented in Java, but similar APIs are provided for Flash and Java applets.

```
/**
 * User selected this experiment. It can start showing the UI. It can
 * load the VM used (Adobe Flash, Java VM, Silverlight/Moonlight, etc.),
 * or define requirements of the (i.e., require 2 files, etc.). It should
 * also show options to gather information that will be sent to the
 * initialization method of the experiment server, that later will be
 * retrieved through the {@link #getInitialData()} method.
 */
public void initialize(){}

/**
 * Retrieves information sent to the experiment when reserving the
 * experiment. It might have been collected in the UI of the
 * {@link #initialize()} method.
 */
public JSONValue getInitialData(){
    return null;
}

/**
 * User is in a queue. Thetype filter text typical behavior will be to
 * hide the UI shown in the {@link #initialize()} method.
 */
public void queued(){}

/**
 * User grabs the control of the experiment (in the server side, the
 * experiment is already reserved for the user).
 *
 * @param time Seconds remaining. This time is the maximum permission time.
 * @param initialConfiguration Data sent by the experiment server in the
 * initialization method.
```

```
     */
    public void start(int time, String initialConfiguration){}

    /**
     * User experiment session finished. The experiment should clean
     * its resources, or notify the user that it has finished. It may still
     * wait for the {@link #postEnd(String)} method to be called so as to
     * receive the information sent by the experiment when disposing
     * resources.
     */
    public void end(){}

    /**
     * Returns if this experiment is expecting a {@link #postEnd(String)}
     * or not. Between {@link #end()} and {@link #postEnd(String)}, there
     * must be some polling to the server, especially if the experiment
     * server takes some time to end. However, users most of the times
     * don't wait for a result, so they can start a queue somewhere else.
     * Other times, it is interesting to receive something at the end of
     * the experiment. Those times, it is required to implement this method
     * returning true.
     *
     * @return if the experiment expects a {@link #postEnd(String)}
     */
    public boolean expectsPostEnd(){
        return false;
    }

    /**
     * The experiment finishes cleaning the resources in the server side.
     * This can be helpful when the experiment does anything in the end,
     * such as storing a result.
     *
     * @param initialData Information sent by the server when finished the
     * initialization of the experiment. It is the same information obtained
     * in the {@link #start(int, String)} method, but this method will not
     * always be called if the experiment life is too short (such as in the
     * batch experiments).
     *
     * @param endData Information sent by the server when finished cleaning
     * resources
     */
    public void postEnd(String initialData, String endData){
        this.boardController.clean();
    }

    /**
     * How much time does will the user have the experiment.
     *
     * @param time Time in seconds remaining
     */
    public void setTime(int time){}
```

In order to interact with the server, other methods are already implemented so the experiment developer can use them. Once again, this is expressed in Java but it is provided and adapted to Adobe Flash and Java applets:

```
    /**
     * Is the user accessing through facebook?
     */
```

```
    public boolean isFacebook();

    /**
     * What is the session id of the user? It is useful when using other type of
     * communications, such as iframes in the LabVIEW experiment.
     */
    public SessionID getSessionId();

    /**
     * Send a command, don't care about the result
     */
    public void sendCommand(Command command);

    /**
     * Send a command, notify me with the result
     */
    public void sendCommand(Command command, IResponseCommandCallback callback);

    /**
     * Send a string command, don't care about the result
     */
    public void sendCommand(String command);

    /**
     * Send a string command, notify me with the result
     */
    public void sendCommand(String command, IResponseCommandCallback callback);

    /**
     * Finish the experiment.
     */
    public void finish();
```

In the server side, the API is very similar. The following methods are provided and the experiment developer can implement them. The experiment developer knows that the usage is already isolated by the scheduler, so no more than one student will call these methods concurrently, the end method will be called before calling the start method again, etc. Once again, these are expressed and documented in Python, but they are also available in .NET, LabVIEW (without remote panels), C, C++ and Java.

```
class Experiment(object):

    def __init__(self, *args, **kwargs):
        super(Experiment, self).__init__(*args, **kwargs)

    def do_start_experiment(self, client_initial_data, server_initial_data):
        """ do_start_experiment(client_initial_data, server_initial_data)
            -> initial_configuration

        This method indicates that a student has been assigned to use this
        laboratory. client_initial_data will provide the data (typically a
        JSON-serialized string) that the experiment client submitted (if any),
        and server_initial_data is a JSON-serialized string with the data passed
        by the core server. This includes the time slot available for the
        current user, the priority, etc.

        This method must return a JSON-serialized string which can be an empty
```

```
        object ("{}"), but it can state that it is a batch experiment (and
        therefore the scheduler will mark it as free once the start method has
        finished), and it can provide information that the client will receive
        (such as "the URL for the camera in this copy of the laboratory is this
        one").
        """
        return "{}"

    def do_get_api(self):
        """
        do_get_api() -> api_version

        Reports the api version that the experiment uses. The default api level
        is the current one. Experiments may override this method to return a
        different one.
        """

    def do_send_file_to_device(self, file_content, file_info):
        """do_send_file_to_device(file_content, file_info)
        raises (FeatureNotImplemented, SendingFileFailureError)
        """

    def do_send_command_to_device(self, command):
        """do_send_command_to_device(command)
        raises (FeatureNotImplemented, SendingCommandFailureError)
        """

    def do_should_finish(self):
        """
        Should the experiment finish? If the experiment server should be able to
        say "I've finished", it will be asked every few time; if the experiment
        is completely interactive (so it's up to the user and the permissions of
        the user to say when the session should finish), it will never be asked.

        Therefore, this method will return a numeric result, being:
          - result > 0: it hasn't finished but ask within result seconds.
          - result == 0: completely interactive, don't ask again
          - result < 0: it has finished.
        """
    def do_dispose(self):
        """
        Experiment should clean the resources now, and optionally return data.
        Default implementation: yes, I have finished.
        """
```

3.5.1.1 Collaborative Managed Experiments

The WebLab-Deusto scheduler permits system administrators to define that there is a fixed number of scheduler slots for a single laboratory. For example, the administrator can define that 4 scheduler slots are assigned to a laboratory. In this case, if 4 students request a laboratory, they can be forwarded to the same laboratory, and a fifth student will be in a queue. This makes it possible to develop new collaborative experiments (for example, based on games) and increase the collaboration in those experiments where different students indeed access the same experiment concurrently (such as VISIR).

However, it was not possible to properly use this collaboration with the existing

managing API, so a new one has been added targeting this feature. The API has been called the concurrent managed experiments API. This API is essentially the same one as the managed experiments API, but the laboratory server guarantees that the experiment server receives a unique identifier for each student during a particular session. This way, experiment developers can use the previous API, or the following one, with the same methods but a new argument called "client_id":

```
class ConcurrentExperiment(object):

    def do_start_experiment(self, client_id, client_initial_data,
                                server_initial_data):
    def do_get_api(self)
    def do_send_file_to_device(self, client_id, file_content, file_info)
    def do_send_command_to_device(self, client_id, command)
    def do_should_finish(self, client_id):
    def do_dispose(self, client_id):
```

The management of the collaboration relies on the experiment developer. As an example, Luis Rodriguez-Gil modified the VISIR client to support to enable students to share their current circuit, as seen on Figure 3.9, and then the server side managed the collaboration. Further information is available in (Orduña et al., 2012b). This shows how the development process enables even to enhance existing laboratories.

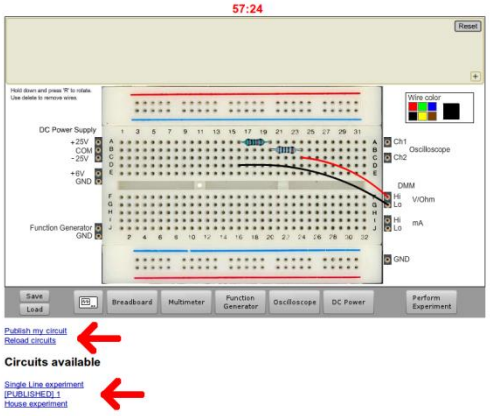


Figure 3.9: Sharing VISIR circuits among students with the collaboration API

3.5.2 Unmanaged Experiment Development

As previously stated, unmanaged experiments came from the necessity of making development easier to develop new types of laboratories. In contrast to the

managed experiments, the unmanaged experiments do not require regular development in any programming language using a particular API. All the communications between client and server is managed by the experiment developer.

At the moment of this writing, there are two main unmanaged experiment types: LabVIEW Remote Panels experiments and Virtual Machine experiments, explained in the following sections. However, the managed API could be used to develop new types of remote laboratories.

3.5.2.1 **LabVIEW Experiments**

WebLab-Deusto supports LabVIEW Remote Panels, as deeply explained in (Orduña et al., 2012a). This integration was a joint effort between Fabricio Gazzola (Federal Institute of Santa Catarina, Brazil) and Pablo Orduña. The target was to support that experiment developers using LabVIEW could integrate their laboratories in WebLab-Deusto automatically.

A key feature of LabVIEW for remote laboratories is the support for "Remote Panels." With this feature, experts on LabVIEW can create an intuitive user interface for instrumentation and publish it to the web. LabVIEW comes with a web server that automatically deploys the application, and in the client side the user will be able to see the user interface. This makes LabVIEW a suitable and efficient tool for remote laboratory developers, and this feature has frequently been used in the literature, used in major remote laboratories as LabShare Sahara (Sarukkalige et al., 2010) and MIT iLabs (Harward et al., 2008b).

However, the use of this tool not only requires the user to install a plug-in in his web browser, but also introduces some limitations in terms of security (limited authentication), deployment (no support for HTTP proxies, firewalls restrictions), latency (user might see data that has not yet been sent), scalability (for duplicating the experiment in order to balance the load among different copies) or administration.

With the growing user base of LabVIEW Remote Panels, its integration in WebLab-Deusto was considered an important interest. The target of the project was to support remote laboratory developers to create LabVIEW Remote Panels projects inside WebLab-Deusto, benefiting of the additional features provided by WebLab-Deusto.

In order to do so, a file-based protocol was established between a LabVIEW generic project and a new WebLab- Deusto server. Within this protocol, the LabVIEW project will store in the file certain information detailing messages meaning "load C:/path/file.vi," and the LabVIEW project responding "loaded" whenever it finishes loading it.

One of the main problems was that the authentication management. At application level, it is possible to require a user to manually provide a token provided by

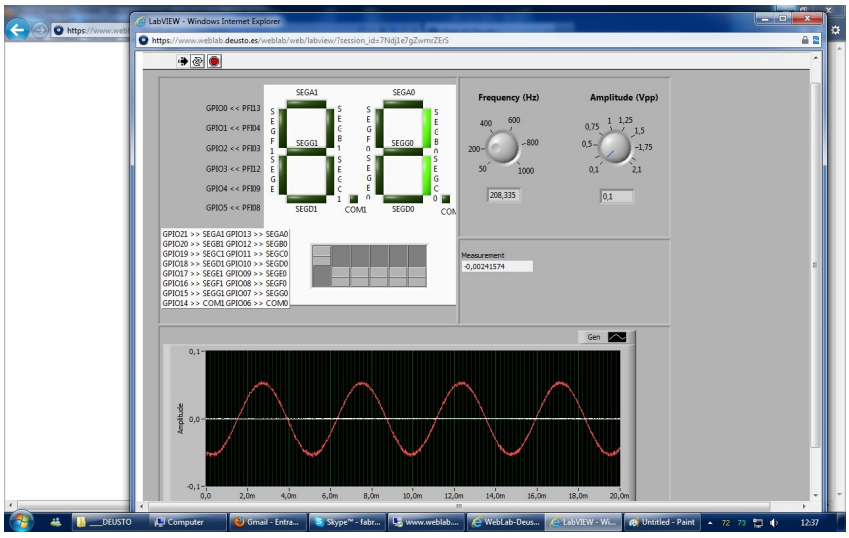


Figure 3.10: LabVIEW Remote Panels integrated in WebLab-Deusto. Interaction screen.

Figure 3.11: LabVIEW Remote Panels integrated in WebLab-Deusto. Configuration screen.

the platform. However, once a user logs in, any user with the URL might be able to interact with the system. In order to avoid this problem, LabVIEW provides a basic HTTP authentication mechanism, relying on an external file with usernames and passwords. It was not possible to programmatically modify this file, given that the hash algorithm used for the passwords was not a standard one. It was also not possible to rely on this file, since it would not be possible to use externals authentication systems such as OpenID or the LDAP system, and it would require managing authentication at two places.

Given that the communication starts with HTTP, another approach was to establish a HTTP proxy between the client and the server, managing the authentication at the proxy. However, whenever the Remote Panel is loaded and starts, it does not use HTTP anymore but a TCP/IP socket that cannot be used through any HTTP proxy. Furthermore, the communication sent through this socket does not contain any token that WebLab-Deusto can use to validate that the user is who claims to be if a socket proxy was established.

The approach taken in WebLab-Deusto is to create a complete copy of the project to be loaded with a random token in the name (e.g., myproject-asdf13414fd). This token is sent through a secure communication to the student's browser, which will open it in a new window with this token, and therefore it will be able to access. Whenever the student is finished, WebLab-Deusto deletes the copy of the project and sends a message to stop the current connection to the LabVIEW generic project, so the student will not be able to access the system after the assigned time has passed. The result is that the student, once logged in, selects the particular experiment if he has privileges to use it. Then, the user waits in the queue until he is granted the access to the real equipment, and a button is shown. Whenever the student clicks on the button, a new window is opened, accessing the real remote equipment, as showed in Figure 3.10 and Figure 3.11.

While WebLab-Deusto itself supports mobile devices, and can be run in any standards-compliant web browser without requiring any plug-in, laboratories developed with LabVIEW Remote Panels will inherit the restrictions already detailed of this technology. Therefore, those particular laboratories will not run in mobile devices, neither in operating systems where LabVIEW is not available. However, these laboratories will benefit from the common features provided by WebLab-Deusto, such as not requiring to develop a scheduling mechanism, neither handling security, or developing administration tools that already come in with WebLab-Deusto.

3.5.2.2 **Virtual Machine Experiments**

Not all experiment developers have experience with web technologies or LabVIEW, and therefore they will not be able to develop a remote laboratory in

WebLab-Deusto using the managed API or the unmanaged LabVIEW module. In order to make it easier to create a new remote laboratory, Luis Rodriguez-Gil supported Virtual Machines (VMs) in WebLab-Deusto. A virtual machine is a complete machine which is running *inside* other machine. There are different systems (depending on the technology used they will be called virtualization software or emulators), which provide the virtual hardware on which the virtual machine will run, such as VirtualBox, VMWare, KVM/Qemu, etc. Nowadays, the most common way to do this is by using certain features of the modern microprocessors which are actually designed for this, making it possible that the guest virtual machine runs as fast as the host operating system, as long as the architecture is the same or compatible.

Virtual machines guarantee a security layer since it is possible to control what the virtual machine can access (e.g., certain components and not certain others), and especially since it is possible to securely clean all what users did in the computer. This way, each student will see exactly the same that the previous student saw, as long as the system supports recovering the equipment to the initial state easily.

As an example, one can have a regular computer, running Microsoft Windows. The target equipment is connected through USB. This Microsoft Windows will be referred as the *host* operating system. On it, it is possible to install a virtualization software, in the example VirtualBox. This virtualization software will resemble a typical application (while certain low level operations are being processed by it), and it supports creating a new virtual machine. This virtual machine will have a virtual disk (which will be a file in the host operating system), a virtual DVD (which may be the host DVD, or other file in the host operating system), certain virtual networking capabilities, etc. On this virtual machine, the same or other operating system can be installed, in the example a GNU/Linux system, which will be referred as the *guest* operating system.

The virtualization software can attach the USB of the *host* machine to *guest* machine, and therefore on it is possible to access the equipment directly. It is therefore possible to run an application in the *guest* machine that uses the equipment. If a snapshot (a *picture* of the all the guest system: memory, disk, etc.) is taken using the virtualization software with this application running, it will be possible to stop the *guest* machine and every time a user logs in, the *host* operating system can start the *guest* machine using that snapshot, change the credentials of this machine by a randomly generated ones, and let the student to access it. This way, all the management is done in the secure *host* operating system, and students only have access to the *guest* machine. If students leave new icons in the desktop, and even install a new program, it will not matter since that state will be discarded and the next user will see the same state it was in the snapshot.

This is exactly what WebLab-Deusto does. A special Experiment Server has

been developed which starts virtual machines, changes their credentials (supporting a wide range of protocols: remote desktop, SSH, VNC, in Microsoft Windows and GNU/Linux), and supports all the communication to the WebLab-Deusto client. Therefore it is possible to do new experiment servers without programming, since the experiment developer only needs to install an application in the virtual machine, and students will automatically have access as seen on Figure 3.12.

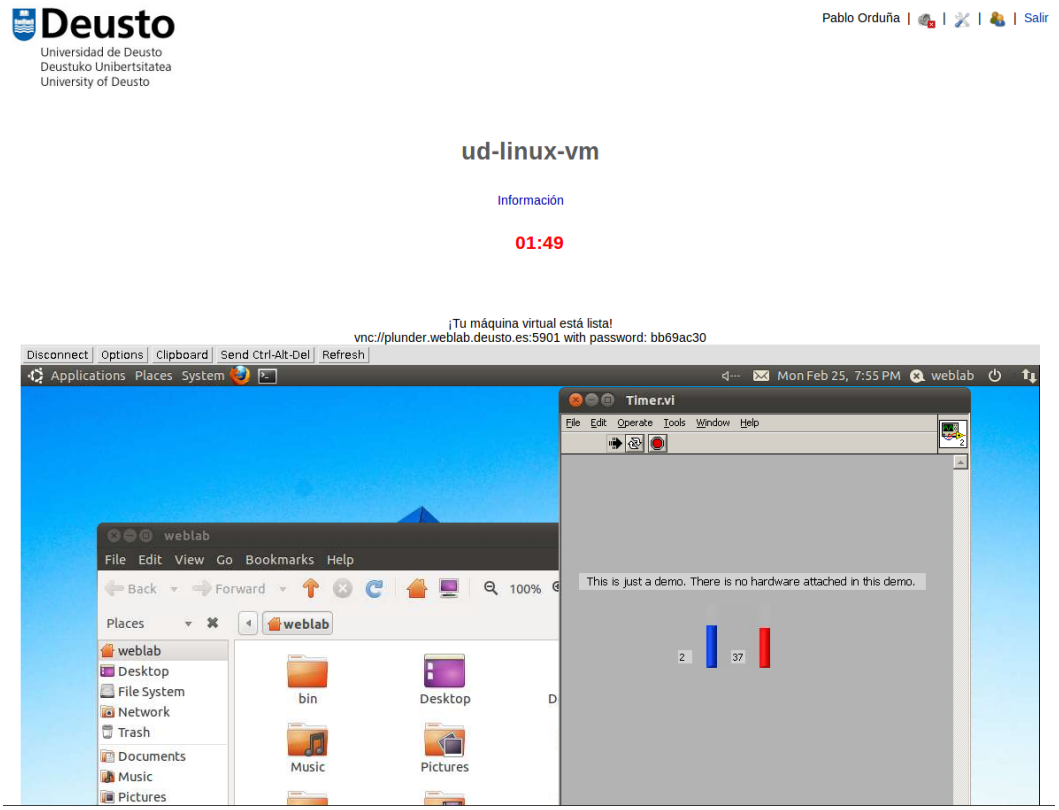


Figure 3.12: Example of Virtual Machine laboratory integrated in WebLab-Deusto. Inside the Virtual Machine, a LabVIEW instance is running in a Ubuntu Desktop

3.5.3 Experiment Examples

In this subsection, several examples are shown. Ignacio Angulo is the designer, developer and maintainer of all of them (except for VISIR, created by the Blekinge Institute of Technology and deployed in the University of Deusto by Unai Hernández). The purpose of these examples is not showing part of this dissertation's work, but to illustrate what can be done with the developed platform.

3.5.3.1 Xilinx Laboratories

The Xilinx laboratories are the oldest in WebLab-Deusto. The first laboratory used with students, back in February 2005, was the CPLD laboratory. It was the only laboratory run by WebLab-Deusto 1.0. Few months later, in November 2005, WebLab-Deusto 2.0 was used by students, first with FPGAs and then with CPLDs.

Both FPGA (*Field Programmable Gate Arrays*) and CPLDs (*Complex Programmable Logic Devices*) are considered *Xilinx laboratories* in WebLab-Deusto, and indeed, the devices used in WebLab-Deusto are manufactured by Xilinx.

The FPGA laboratory lets students remotely interact with FPGA devices. Through the Xilinx software, students can write a FPGA program locally as they normally would. Once the program is compiled, and ready to be tested, they upload the binary ".bit" file through WebLab-Deusto.

The FPGA laboratory will automatically program the FPGA board with the binary uploaded, and will start running it. To see the results, a webcam is provided. Students may also interact with the board remotely, by using the provided widgets. The widgets will send a signal to the board just like their physical counterparts would.

The FPGA laboratory, as the CPLD laboratories and the PIC laboratories, is developed within the WebLab-Box(Garcia-Zubia et al., 2010a) (Figure 3.13), designed and developed by Ignacio Angulo. On the WebLab-Box, the device, as well as a fit-pc, a PIC microcontroller, a camera, lighting system and networking materials is installed, to make it easier to create and deploy new laboratories. WebLab-Box was awarded as the Best Educational Tool in the ICELIE 2009 conference.

The CPLD works in the exact same way but with a different board, and with few different commands.

3.5.3.2 Aquarium Laboratory

The Aquarium laboratory grants access to a real aquarium located in the University of Deusto. On it, it is possible to feed the fish, turn on and off the lights, and, if the submarine is in the water and it is charged, control it.

Regarding feeding the fish, it may seem dangerous, but it is not. The system feeds them automatically three times a day, every 8 hour. If a user feeds them, then it does not let any other user to feed them before the next shift, guaranteeing that they are only fed three times.

The initial rationale behind this laboratory is that groups of primary school students are responsible of the life of these fish (even if they are not under a real danger). Teachers may know which groups of students have feed them correctly, which students did not forget and which students coordinated correctly so no one overfed the fish.

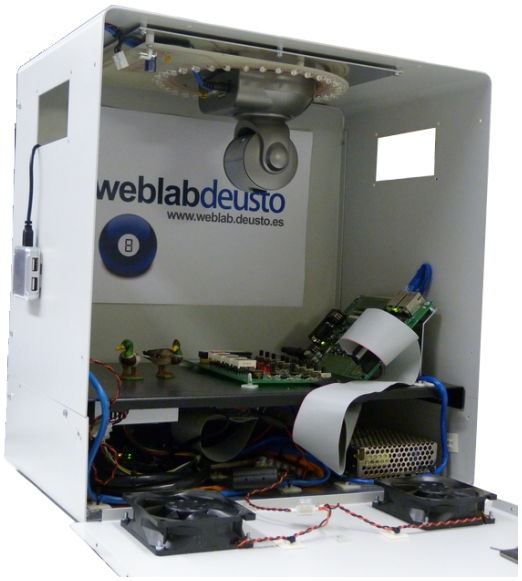


Figure 3.13: WebLab-Box, which typically contains an electronics laboratory

However, at the time of this writing ongoing work is being done for adding more sensors to this laboratory.

From a technical perspective, the whole laboratory is deployed in http://fishtank.weblab.deusto.es/, which uses a low cost ARM microprocessor called IGEPv2[1].

3.5.3.3 Robot Laboratory

The robot laboratory uses the commercial robot Azkar-bot (see Figure 3.14), with an attached microcontroller. WebLab-Deusto manages to establish that three different learning activities are using the same equipment, so the scheduling system will queue other users internally.

The *robot-proglist* lets users choose one among a few of predefined programs to program the bot with. The programs currently available are the following:

- **Follow black line:** The robot will first move randomly while avoiding obstacles (walls) until it finds the black line. It will then position itself on the line and follow it using its infrared sensors.
- **Walk alone:** It will walk around while avoiding any obstacles in its way.
- **Turn left and right:** Rotates left and right, non-stop.

[1] http://igep.es/index.php?option=com_content&view=article&id=46&Itemid=55

Figure 3.14: Azkar robot, used by the Robot laboratory

The *robot-movement* lets users control a bot remotely. The bot can move forward or backward, and turn to both sides.

To make the bot move, simply click on the appropriate button. Alternatively, users can control the bot by using the arrows shown. The bot will move according to its own position, and not to the position of the camera.

The bot will not obey the user if it finds a wall in its way, in which case it will try to avoid it.

Finally, the *robot-standard* lets students program the bot directly.

The bot uses a PIC processor, so the program should be written using the Microchip PIC compiler. It is noteworthy that the bot has, among other things, infrared sensors, to which the developer has access.

The microcontroller model of the robot is a PIC 18F4550. It has two different motors for each wheel. The motors can go either forward or backward. It also has two obstacle sensors, which can be used to avoid the walls, and two infrared sensors, which can be used to detect the line.

3.5.3.4 **VISIR**

The VISIR experiment lets you access the BTH OpenLabs VISIR through WebLab-Deusto. It has been described in Section 2.2.2.4, but it is still relevant to explain that it has been integrated in WebLab-Deusto. Indeed, VISIR has had an important impact on the design of several federation features, as well as some of the internal APIs in WebLab-Deusto (such as the collaboration API).

3.5.4 **Experiment Maintenance**

WebLab-Deusto also manages the laboratory status. Periodically (by default once a minute), it checks certain conditions, configured for each Experiment Server. For instance, the following configuration code every time checks that it can retrieve the defined image and that it is actually an image, and that the established port of that IP address is listening to connections:

```
'exp1:ud-fpga@FPGA experiments' : {
  'coord_address' : 'fpga:process1@box_fpga1',
  'checkers' : (
    ('WebcamIsUpAndRunningHandler',
      ("https://www.weblab.deusto.es/webcam/proxied.py/fpga1",)),
    ('HostIsUpAndRunningHandler',
      ("192.168.0.70", 10532)),
  ),
  'api'      : '2',
},
```

This way, if for any condition the laboratory is not working (e.g., the camera is broken, there has been a shutdown, etc.), the system will automatically remove the laboratory from the scheduling system, and the system administrators, as well as the laboratory administrators, will be notified by e-mail. If it works again, then it will be added to the scheduling system and administrators will be also notified.

This feature is very relevant for the load balance system. Given that WebLab-Deusto supports that students are automatically balanced among different copies of a laboratory, if one of these copies is broken and students are assigned to it, then they will have a very bad experience, since they will randomly be assigned a broken system. In some laboratories, students can choose between "any available copy" or "this particular copy," but the first option would not be useful if this maintenance system is not properly configured.

3.6 Conclusions

As detailed through this chapter, WebLab-Deusto is a robust remote laboratory management system, developed during the last 8 years. On top of this system, different remote laboratories have been built. The system will provide a set of features that will be automatically available to all these laboratories.

In Section 3.2, it is shown how WebLab-Deusto implements a distributed but flexible architecture. Different components can be spread in different servers, while it is still possible to centralize all the functions in a single process in a single server for testing or for supporting low cost scenarios. This way, it can be adapted to different scenarios. In production in the University of Deusto there are multiple servers providing different laboratories, while in smaller deployments a single node has been provided. This will enable novel low cost scenarios later described in Section 4.3.3.

The transversal features provided are detailed in Section 3.3. They include scaling up, guaranteeing certain security level by dealing with certain common security problems, and providing different mechanisms for authentication, such as LDAP, OpenID or OAuth 2.0. In addition, it provides a complex scheduling system, described in Section 3.4. This scheduling system is extensible through plug-ins, where the priority queuing system used in most laboratories is a plug-in,

the federation model (described in Chapter 4) is implemented as other plug-in and interoperability layers described in Section 4.3.2 have been implemented as other plug-in. The main plug-in -the queuing system- supports load balance, so users are balanced among multiple copies of the laboratory, if available. WebLab-Deusto provides an experiment tracking system to remove from the queue those laboratories which are not working (e.g., if the webcam is failing, if the particular lab is down or similar).

All these features are used by any laboratory that is implemented using the WebLab-Deusto stack. This is the case for all the sample laboratories deployed in the University of Deusto (and explained in Section 3.5.3). Basically, this stack is split in client code (where Adobe Flash and Java applets are supported while not recommended), and server code (where libraries for Python, Java, .NET, C, C++ and LabVIEW are provided). These libraries, explained in Section 3.5, provide message driven libraries, where client submits commands to the experiment server, and WebLab-Deusto manages the communication and storing of these commands. While using these libraries and AJAX is recommended to support mobile devices and avoid problems with firewalls, HTTP proxies and other constraints, the reality of remote laboratories is complex and experiment developers may not even be familiar with these technologies or they may want to reuse existing code which cannot be easily integrated. For this reason, WebLab-Deusto provides alternative mechanisms, such as LabVIEW Remote Panels and Virtual Machines. While they are not recommended due to the constraints they impose the final user, they are still supported to develop new labs.

To sum up, WebLab-Deusto is a flexible and scalable remote laboratory management system. The target of this flexibility is not to be yet another component to be learned by instructors, but a rather hidden and humble software component that communicates securely and efficiently the different roles present in remote laboratories: students, instructors, laboratory designers, IT services. The support of LDAP, OpenID or OAuth 2.0 is focused on relying on existing and well maintained systems for authentication. The support of multiple APIs for development is focused on supporting multiple profiles of laboratory designers, as well as existing remote laboratories (such as VISIR). Ideally, the authorization will be managed where possible by other systems already used by students, such as LMSs or CMSs (see Section 4.3.1). This way, WebLab-Deusto would become an *underlying* component which guarantees the distribution, secure sharing, and development framework of remote laboratories.

As a final note, this chapter describes the WebLab-Deusto remote laboratory management system, which is the system on top of which the federation model has been designed and implemented. Valuable contributions from other individuals have been explained, identifying their authorship in the particular components. However, it is important to remark that the author of this dissertation has still been

involved in many of the design decisions, has participated in the development of them, and has later been responsible of the integration of these components and their deployment and maintenance.

Everything should be made as simple as possible, but no simpler

Albert Einstein

CHAPTER

4

Federation Model

THIS chapter describes and analyzes the proposed remote laboratory federation model. This model has been developed and implemented on top of WebLab-Deusto -deeply explained in Chapter 3-. While the quantitative evaluation of the federation model is addressed in Chapter 5, a qualitative analysis is provided in this chapter.

The chapter is divided in three sections:

- Section 4.1 details the impact of local scheduling strategies on the proposed federation model.
- Section 4.2 describes the federation model.
- Section 4.3 shows the impact of the federation model on different areas of the remote laboratories ecosystem.
- Section 4.4 provides a qualitative analysis of the federation model.
- Section 4.5 summarizes the chapter and outlines the conclusions.

4.1 From Local Scheduling to Federation

This section explains the impact of local scheduling on federation models. As detailed in Chapter 2, there are different scheduling mechanisms, primarily queues and calendar based booking. And as stated in Chapter 3, WebLab-Deusto provides

a pluggable scheduling system, where different scheduling services can be implemented (such as priority queues, or consumption in other systems). So this section only focuses on showing how federation can be implemented as a particular scheduling system inside the pluggable system. In the rest of this chapter, the proposed federation model has been formally described.

In remote laboratories, any scheduling system focuses on granting *exclusivity* on *a set of resources* to *a set of users*:

- This *exclusivity* will depend on a number of students: CPLD enabling a single user to access the equipment or VISIR enabling 60 students to the same system.

- The *set of resources* is commonly one but it may vary over time. If the system supports load balancing (such as Sahara or WebLab-Deusto), more copies of the same laboratory can be added or removed and the system will manage to balance the load of users among these copies.

- The *set of users* will also vary over time, as new groups use the system and groups of previous years stop using them.

Federation of remote laboratories, on the other hand, aims to enable that at least two systems located in different entities (universities, secondary schools, research centers...) can share laboratories. This means that *a set of users* of the consumer system, registered only in that system, will be using *a set of resources* (i.e., laboratories) located in the provider system in an *exclusive* basis, without being registered there.

This way, a federation could be considered a particular case of local scheduling. Indeed, in the federation model proposed in this chapter, it is treated as such. However, there are relevant differences, such as authentication, authorization or user tracking, and where each scheduling system is located. Most of them can be targeted by passing enough information from one system to another and with trust chains among systems.

Being the resources located in the provider entity, the only system which can guarantee exclusive access is the provider system. Therefore the management of the *set of resources* and *exclusivity* is located in the provider system. This management can be exactly the same as the one provided in non-federated environments, and it is independent of the scheduling system.

The consumer system may apply its own policies, overriding the permissions granted by the provider entity. The provider system defines the maximum permissions. For instance, the provider system will grant the consumer system to enable students to use a laboratory during 10 minutes. The consumer system, therefore, will never be able to let its own students access the remote laboratory for 20

minutes. However, it can request the provider system a session that will last only 5 minutes, or with a lower priority.

However, what is clear is that both systems must support the same exact type of scheduling interfaces. This problem arises when two systems use different remote laboratory management systems (RLMSs), or different versions of the same RLMSs. For example, both RLMSs may support queueing, but not using the same interface. The iLab Shared Architecture (ISA), in its batch architecture, provides information about the estimated waiting time, which is something that WebLab-Deusto does not provide. Consuming the ISA from WebLab-Deusto is easy since that data can be ignored. However, consuming WebLab-Deusto from ISA (as explained in Section 4.3.2), forces WebLab-Deusto to make up that information. Labshare Sahara supports that when a student is using a laboratory, if there is no need to leave (since other student is waiting), the student can stay longer using the laboratory. This is not possible by WebLab-Deusto, so particular fixes which will limit the potential of the laboratory would be required.

Supporting different scheduling systems is a major issue. As stated in Chapter 2, there are many types of scheduling systems in current remote laboratories. Taking into account the research held on this area (Maiti, 2010b; Lowe and Orou, 2012; Maiti, 2011), it is arguable that this is an aspect of remote laboratories subject to change. If a specific purpose remote laboratory uses a scheduling system that is not supported by a RLMS, it will be considerably limited for its students. For example, if WebLab-Deusto did not support concurrent access to a single laboratory, then it would be possible to integrate VISIR (which supports concurrent access to up to 60 students), but the user experience would be poor (only one user would be able to access). Therefore, in order to support more types of laboratories, federation models must support a wide range, or be extensible enough to support them.

4.2 Federation Model Internals

This section shows the internals of the federation model. It formalizes the federation model (Section 4.2.1), defining the characteristics (Section 4.2.2), explaining complex scenarios that have been aimed (Section 4.2.3) the impact of accounting (Section 4.2.4) and the design and implementation (Section 4.2.5).

4.2.1 Introduction and Formalization

As previously stated and described in Figure 4.1, a remote laboratories federation enables distinct entities (such as universities, research centers or even secondary schools) to share laboratories transparently. The following vocabulary is defined to

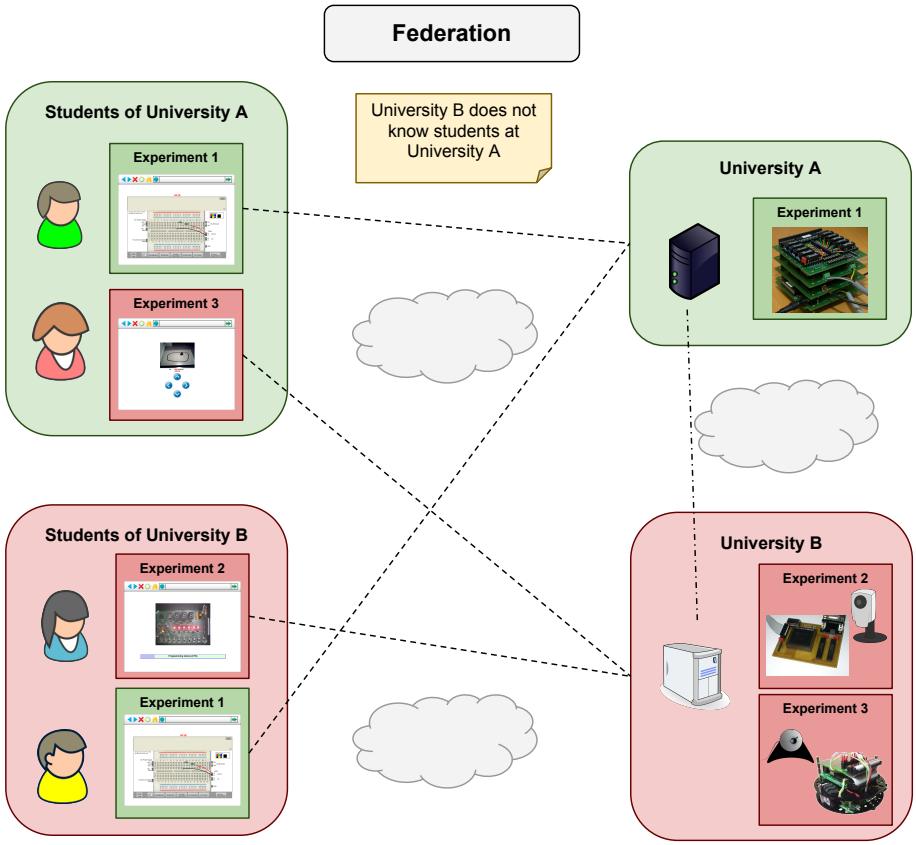


Figure 4.1: Federated remote laboratories

formerly detail this federation model:

- An entity will be denoted by E_x, being $_x$ the entity identifier.
- The shared remote laboratory will be considered a *resource*, and will be denoted by $\Gamma_{x,T,N}$, being $_x$ the entity identifier, $_T$ the resource type and $_N$ the resource identifier.
  - The resource type and the resource identifier will only be explicitly detailed when necessary.
- The provider entity E_α sharing $\Gamma_{\alpha,T,N}$ to the consumer entity E_β, granting a maximum priority ρ and a maximum slot of t time, will be denoted by:

$$E_\beta \xleftarrow{\Gamma_{\alpha,T,N},(\rho,t)} E_\alpha$$

  - The priority and the size of the maximum slot will only be explicitly detailed when necessary.
- The entity E_α sharing $\Gamma_{\alpha,T,N}$ to a student Σ_α of E_α with a priority ρ for t time:

$$\Sigma_\alpha \xleftarrow{\Gamma_{\alpha,T,N},(\rho,t)} E_\alpha$$

Example: The University of Deusto (UD) counts with three different types of remote laboratories: FPGAs (3 copies), PICs (3 copies) and CPLDs (3 copies). The Urdaneta School (CU for its initials in Spanish) is a secondary school which has a contract with the University of Deusto, so the University of Deusto shares two of the laboratories (3 FPGAs and 2 PICs) with the Urdaneta School. This way, the University of Deusto ensures that there will always be at least one PIC available for its students. We may state that:

$$E_{CU} \xleftarrow{\Gamma_{UD,fpga}} E_{UD}$$
$$E_{CU} \xleftarrow{\Gamma_{UD,pic,1}} E_{UD}$$
$$E_{CU} \xleftarrow{\Gamma_{UD,pic,2}} E_{UD}$$

Note: priority (ρ) and time (t) are not defined in this example.

4.2.2 Federation Model Characteristics

The proposed federation model introduces two novel features not addressed in existing remote laboratory federation models:

1. It satisfies the *transitive property*.
2. It supports *distributed load balancing*.

The former feature, satisfying the *transitive property*, means that for three distinct entities E_α, E_β and E_γ, where E_α shares a resource $\Gamma\alpha$ with E_β, automatically E_β *can re-share* it with E_γ, with the same or lower conditions in terms of priority and amount of time per slot:

$$E_\beta \xleftarrow{\Gamma_\alpha,(\rho,t)} E_\alpha \implies E_\gamma \xleftarrow{\Gamma_\alpha,(\rho',t')} E_\beta \wedge \rho' \leq \rho \wedge t' \leq t$$

The emphasis on the verb *can re-share* denotes that E_β does not automatically re-share resources of E_α to any related entity. When E_β is granted to use Γ_α with a maximum amount of time t and a priority ρ, E_β will select if it will re-share this resource to which entities, and under which conditions (priority ρ' and t' time, as long as they are not greater than original ρ and t). Figure 4.2 describes this feature.

Example: The National Distance Education University (UNED) counts with a PIC laboratory. UNED and the University of Deusto (UD) come to an agreement that lets UD access 1000 times per year to the PIC laboratory at UNED. The University of Deusto has another agreement with the Urdaneta secondary school (CU), and it offers the PIC laboratory to this high school. Given the transitive property of the federation model, it is possible to re-share these slots. We therefore may state that:

$$\begin{array}{c} E_{UD} \xleftarrow{\Gamma_{UNED,pic}} E_{UNED} \\ \Downarrow \\ E_{CU} \xleftarrow{\Gamma_{UNED,pic}} E_{UD} \end{array}$$

The latter feature, supporting *distributed load balancing*, means that for three distinct entities E_α, E_β and E_γ, both E_α and E_β can share two different copies of the same type of resource with E_γ, and E_γ will be able to load balance its own users Σ_γ between both distributed copies. In Figure 4.3, both universities have copies of the same laboratories, and students of one university can use laboratories of the other if they are busy.

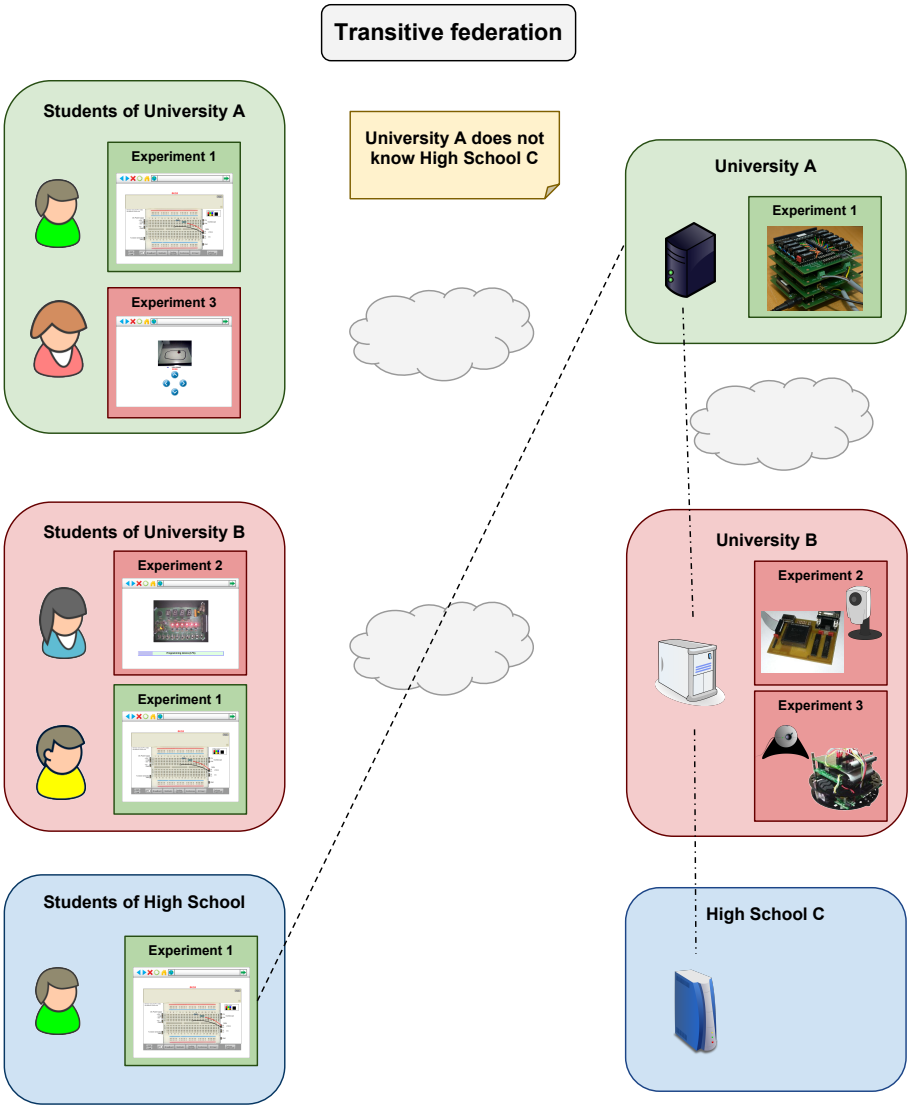


Figure 4.2: Transitive federation

$$E_\gamma \xleftarrow{\Gamma_{\alpha,T,1},(\rho_{\alpha\to\gamma},t_{\alpha\to\gamma})} E_\alpha \wedge E_\gamma \xleftarrow{\Gamma_{\beta,T,1},(\rho_{\beta\to\gamma},t_{\beta\to\gamma})} E_\beta$$

$$\Downarrow$$

$$\Sigma_\gamma \xleftarrow{\Gamma_{\alpha,T,1},(\rho'_{\alpha\to\gamma},t'_{\alpha\to\gamma})} E_\gamma, \rho'_{\alpha\to\gamma} \leq \rho_{\alpha\to\gamma} \wedge t'_{\alpha\to\gamma} \leq t_{\alpha\to\gamma}$$
$$\wedge\Sigma_\gamma \xleftarrow{\Gamma_{\beta,T,1},(\rho'_{\beta\to\gamma},t'_{\beta\to\gamma})} E_\gamma, \rho'_{\beta\to\gamma} \leq \rho_{\beta\to\gamma} \wedge t'_{\beta\to\gamma} \leq t_{\beta\to\gamma}$$

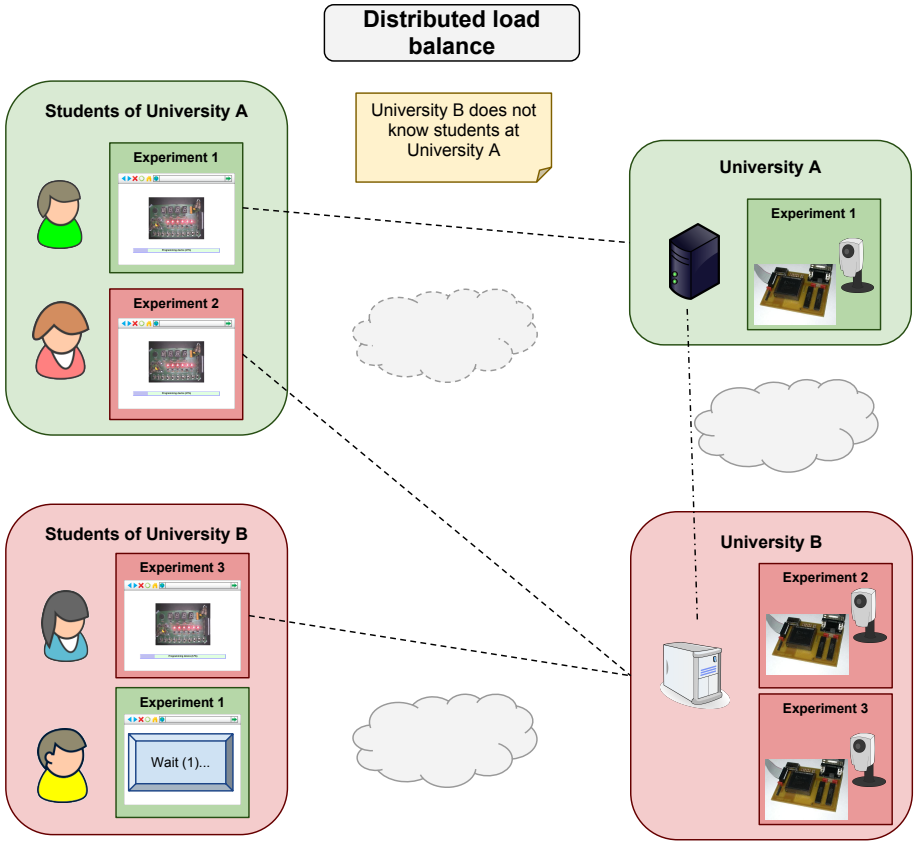


Figure 4.3: Distributed load balance

Example: The National Distance Education University counts with a PIC laboratory. The University of Deusto counts with a copy of the same laboratory. Both universities have an agreement with the Urdaneta secondary school so it can use the PIC laboratory. Therefore, students of the Urdaneta school will automatically use one laboratory or the other, depending on the availability at each moment.

$$E_{CU} \xleftarrow{\Gamma_{UNED,pic}} E_{UNED} \wedge E_{CU} \xleftarrow{\Gamma_{UD,pic}} E_{UD}$$
$$\Downarrow$$
$$\Sigma_{CU} \xleftarrow{\Gamma_{UD,pic}} E_{CU} \wedge \Sigma_{CU} \xleftarrow{\Gamma_{UNED,pic}} E_{CU}$$

4.2.3 Complex Scenarios

This subsection defines complex scenarios that can be built using the described federation model.

4.2.3.1 Transitive Federation

Transitivity itself can be used to bind consumers with foreign providers. As shown in the theoretical situation described in Figure 4.4, a local farm in Spain could share a remote laboratory with a local university (University of Deusto). The local university might share this laboratory with the surrounding secondary schools, interested in using the laboratory provided by the local farm.

If secondary schools in other regions would be interested in using it, complex chains could be built. For instance, the local university would be able to re-share the farm laboratory to a partner university in other country, which would once again re-share it to a local secondary school. This way, complex but feasible chains would be built between one farm whose employees may not speak other languages than Spanish with a high school whose employees may not speak other languages than German.

4.2.3.2 Distributed Load Balance

Distributed load balance is only used when there is more than one provider of the same exact remote laboratory. This is generally uncommon in the literature: those universities providing a laboratory are usually those who have developed the laboratory, focusing on an existing requirement by the academic staff, which may differ from one university to another. However, there are two major relevant exceptions: arising remote laboratory appliances and VISIR.

During the last decade, different solutions for exporting appliances of remote laboratories have been designed and implemented. An example of these appliances is WebLab-Box, designed by Ignacio Angulo (University of Deusto) and shown in Section 3.5.3.1. As seen in Figure 3.13, this box comes with a camera, a set of LEDs, a fit-PC (a small, fan-less computer) and material to control devices. On this box, in the University of Deusto different types of laboratories have been

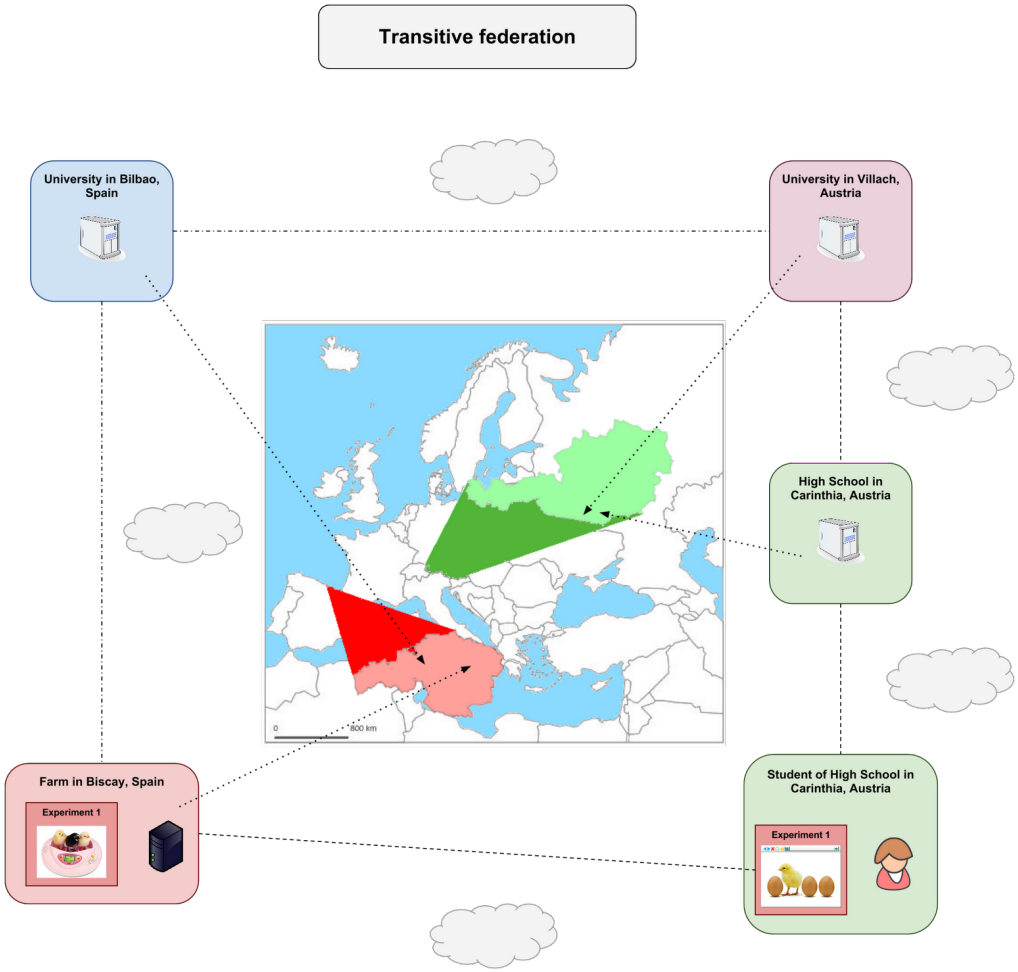


Figure 4.4: Exchanging remote laboratories across boundaries

deployed, including FPGA, CPLD, and Microchip PICs. An entity may develop a remote laboratory that fits in the box, build this type of appliance box, and send or sell the laboratory to other entity (such as other university or high school). In a similar way, the CoNeT Mobile Labs (CML) proposed in (Lyalina et al.) provides all the requirements to send the whole appliance to other entity and start using it in a remote basis.

This type of situation, where developer entities send the equipment to the consumer entities instead of granting remote access could seem uncommon, especially when talking about federations of remote laboratories. However, maintaining a remote laboratory has certain advantages over *consuming* a remote laboratory. If the load of users is high in certain moments, provider universities may be interested in granting a higher privilege to its students. Therefore, owning a remote laboratory guarantees this quality of service. Administrators can assign a higher priority on the local devices or even keep some of them only for local users. Even if universities defined a selfish behavior and wanted to have their own resources, sharing them to be able to access a wider range of experiments or a higher number of students would be desirable. So the selfish behavior of acquiring and owning remote laboratories and at the same time sharing them in order to access resources provided by others are essentially compatible. And in order to exploit these scenarios properly, distributed load balance is required.

The other relevant exception is VISIR. As detailed in Chapter 2, it is the most spread remote laboratory in the world. The same exact equipment has been deployed in six European universities and one in India. While the particular components integrated in VISIR boards are usually different, it is affordable to come to an agreement to use the same components during certain weeks of the year. Using the proposed federation model, it would become possible for two provider universities to let their users have a virtual VISIR system composed by both systems. Given that each VISIR system supports up to 60 concurrent users, the virtual VISIR system would support 120 concurrent users, balanced between both systems. If the six European VISIR providers federated each other and the same configuration was used, up to 360 concurrent users may work. If the number of concurrent users was less interesting than the number of possible circuits available, the federation model would still permit to define each VISIR system to be considered different. It is even possible to create partial views, with 3 different configurations deployed in 6 systems to enable 120 users per configuration, as shown in Figure 4.5.

Example: Six universities with VISIR have the following fictional configurations. For the sake of simplicity, it has been considered that VISIR has only 2 slots for components (such as resistors or capacitors; the names have been simplified to X, Y and Z), which are connected. The reality of VISIR is

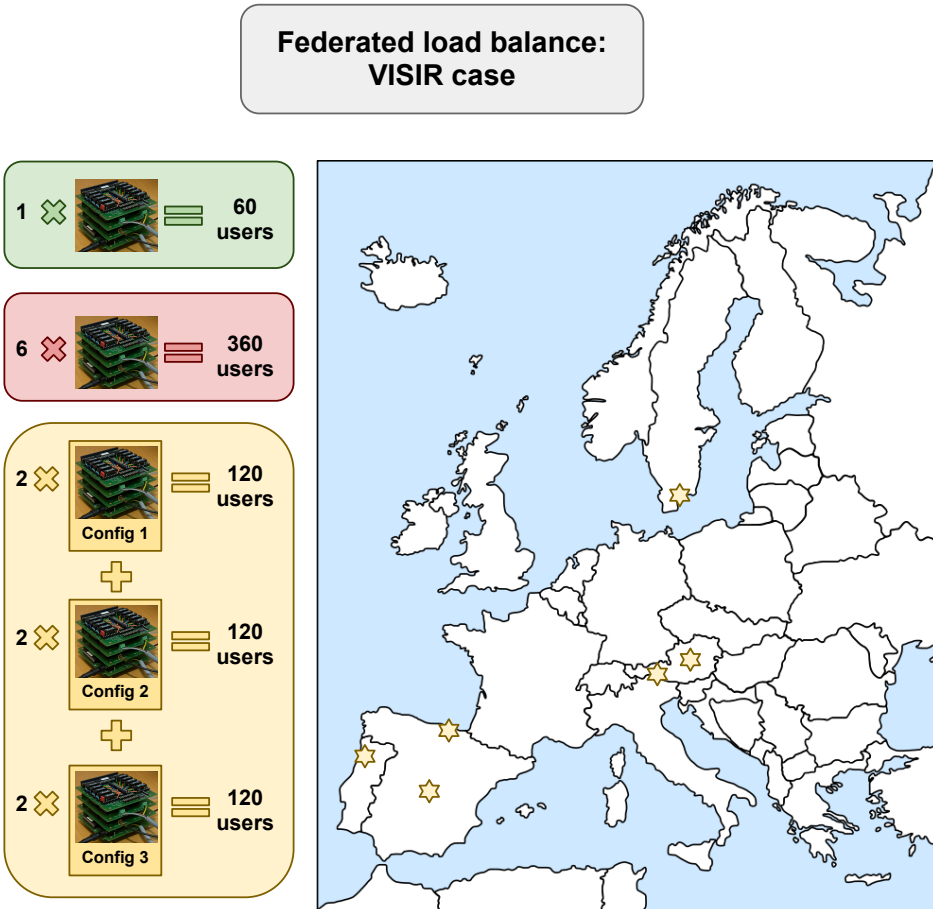


Figure 4.5: Possible schemes to federate the VISIR remote laboratory

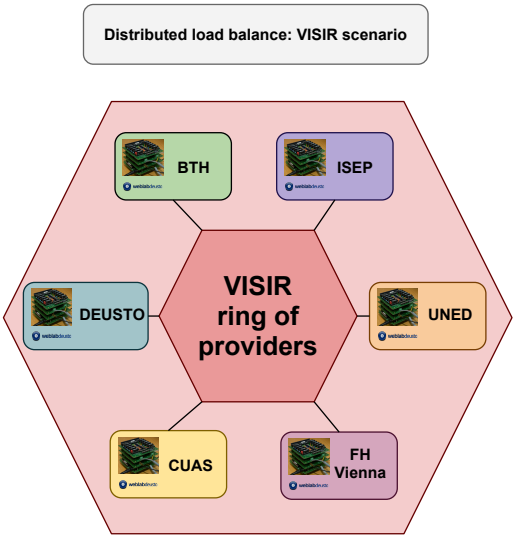


Figure 4.6: Ring of VISIR providers

more complex: it may happen that two components cannot be connected for deployment constraints, and it supports more slots (limited by the number of element in the switching matrix).

University	Slot 1	Slot 2	Maximum concurrent users
University A	X	X	60
University B	X	Y	30
University C	Y	Y	50
University D	Z	Z	40
University E	Z	Z	10
University F	Z		20

If the six European universities were federated through the federation model, any of these universities may define their own lessons. The system administrator would then define that internally *lesson 2* will run on the VISIR system of the University A or University B, load balancing between both systems. This way, the names or number of lessons do not need to be shared among institutions. In the following table, an example of lessons for one university is described. For each lesson, the required components are defined, and with these requirements, the universities supporting that lesson are enumerated. The

maximum concurrent users is the sum of the maximum concurrent users of each university that supports the lesson. The *fault tolerance* column refers to the fact that if a system crashes (temporarily –a network issue– or permanently –a component is broken), students can still use other system.

Lesson name	Required components	Universities supporting it	Maximum concurrent users	Fault tolerance
Lesson 1	X & X	A	60	false
Lesson 2	X	A, B	90	true
Lesson 3	Y	B, C	80	true
Lesson 4	X & Y	B	30	false
Lesson 5	Z	D, E, F	70	true
Lesson 6	Z & Z	D, E	50	true

It is important to remark that this table is defined by a single university for its own students, and that each university can have similar tables. Furthermore, the configuration of the federation model is simpler than this: the federation layer only understands the third column, which is *lesson 2 is implemented in university A and B*; the rest is automatically managed by the system. For instance, if 60 users started using *lesson 1*, but at the same time there were already 40 users of other university using the VISIR system of University A, 20 students would enter and 40 would wait in a queue.

4.2.3.3 Concatenating Distributed Load Balance and Transitive Federation

These characteristics (distributed load balance and transitive federation) are not exclusive and therefore they can be concatenated to build even more complex and more useful scenarios. As described in Figure 4.6, the load of users can be balanced among the ring of providers. However, given the transitive property of the federation model, it is possible that any of these universities could re-share access to any member of the ring to other entities. As described in the fictional situation of Figure 4.7, each of these members of the ring of providers could re-share the access to other laboratories. This way, it would become possible that a student of an associated university of one of the universities could use any of the VISIR systems deployed by the ring of providers, fulfilling the priority and access constraints imposed by certain systems to other systems.

While the establishment of more intermediary nodes adds some latency to the reservation process, the final communication is direct from the user to the final institution, so the degradation is not high. As described in Chapter 5, this latency is notably low. This way, more nodes can be added if required. For instance, a

Learning Management System could be connected to a secondary school which is being connected to one of the universities in the ring that provides access to other university with the particular desired configuration. There would be 4 steps, but this is contemplated and enabled by the federation model.

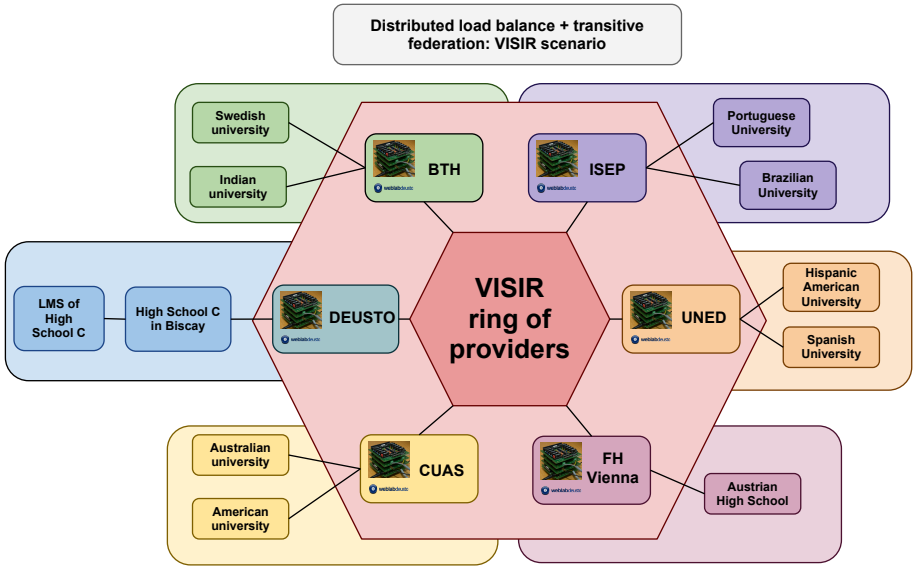


Figure 4.7: Distributed load balance with transitive federation (VISIR). Note: this is a theoretical situation.

4.2.4 Accounting

While a detailed accounting model is outside the scope of the federation model, it has a clear impact on its design. A federation model permits entities to share their own resources, and a transitive federation model permits entities to enable other entities to re-share their own resources. Given that any provider entity is not only interested but also in charge of the resources located in its own institution, the system must have some type of control on how their resources are shared. Additionally, any economic agreement arranged between a provider and a consumer must let the consumer entity audit the consumption of the resources.

4.2.4.1 Accounting Scenarios

In order to summarize the possible accounting scenarios that the federation model needs to aim, a set of scenarios has been defined. First, Table 4.1 shows the possible types of contracts that have been considered.

Table 4.1: Basic accounting schemes

Id	Name	Description
1	Cost per access	Consumer pays an amount per each access, independently of how long it waits
2	Fixed rate per number of accesses	Consumer pays an amount of money for a number of accesses (e.g., 10000 accesses), regardless of the time accessed
3	Fixed rate per time	Consumer pays a fixed amount of money for accessing during a particular amount of time (e.g., 100 hours), distributed in as many different sessions as desired

Table 4.2: Basic adjustments

Id	Name	Description
1	Priority	Consumer pays per month or semester, and gets access only during certain hours (e.g., 1:00 AM – 11:00 AM), and a lower priority or no access during the rest of the day
2	Definition of access	Interactive (and not batch) experiments might consider a minimum amount of time prior to billing an access. This highly depends on the nature of the experiment.

On top of this table, adjustments can be performed, such as "do not consider a session if it takes less than 10 seconds" in certain laboratories. These adjustments are summarized in Table 4.2.

It is remarkable that nowadays the price is usually zero and the amount of time is unlimited. Indeed, the WebLab-Deusto framework has not implemented mechanisms to limit the access of uses, except for disabling accounts that make notably unfair use.

Once the rules are defined, there are risks of improper use or abuses by consumers and providers. The simplest example of these abuses would be that the provider stated that the consumer has already consumed all the accesses when the consumer has not. Different federation models and approaches for sharing remote laboratories can address these abuses at the cost of performance and complexity of the final solution, and that is why these risks have an impact on the federation

Table 4.3: Consumption abuse risks

Id	Description	Example
1	Provider states consumer contract finished too early	Provider university abuses stating that the consumer has already achieved the maximum level of accesses or the total session of time.
2	Provider states consumer stated for too long	When returning usage information, provider abuses stating that the users stayed for longer than actual.
3	Consumer abuses using too much	A consumer may violate the agreement and let its students access more times than contracted.
4	Consumer abuses re-sharing	A consumer may re-share a laboratory and the final consumer plus the consumer may end up using the system longer than agreed.

model design. Table 4.3 lists the identified risks.

4.2.4.2 Sharing Approaches and Risk Management

As stated in Section 2.4, there are different approaches for sharing a remote laboratory. These approaches are summarized in Table 4.4, and their relation with the risks analyzed in Table 4.3 is detailed in Table 4.5.

As described, in strategies 1 and 2, the provider entity is the only one that knows when a student has requested access. Therefore, there is no way to properly manage risks 1 and 2.

In the case of the strategy 2, students will contact the provider system requesting a experiment, and the provider system will contact the Identity Provider (IdP) at the consumer university, verifying that the student is who claims to be and that he is granted to use the requested laboratory in the provider system. The IdP, which is commonly managed by the IT services of the consumer university, will authenticate the student and will generate a response to the provider system. Therefore in this case, there is a communication with the consumer Identity Provider, but this communication is not reliable to know how many requests are performed by the final student.

For example, a student Σ_α of the university E_α may request a laboratory in the university E_β. Using the strategy 2, the student will go directly to the E_β remote laboratory and request the experiment. E_β will redirect Σ_α to the E_α IdP to perform an authorization request. Σ_α will interact with the IdP, and the IdP will sign that Σ_α

Table 4.4: Sharing strategies

Number	Name	Description
1	Adding users in provider system	Consumer entity does not need a remote laboratory system. It provides a set of usernames to the provider entity which are added and they access directly.
2	OAuth / Shibboleth solution	Consumer entity does not need a remote laboratory management system. It needs an Identity Provider (IdP) that also authorizes consumer students in the provider system to use a remote laboratory.
3	Federated remote laboratory with direct connection	Both provider and consumer have deployed a system which manages the federation. Users of the consumer are handled at the consumer side system. The consumer and provider systems talk and when a reservation is performed the student is redirected to the final system.
4	Federated remote laboratory with proxied connection	Same as strategy 3, but all the communications between the student and the final remote laboratory are managed by the consumer side system and forwarded to the provider system.

Table 4.5: Sharing strategies and risk management

Strategy \ Risk	1	2	3	4
1	No	No	Yes	Yes
2	No	No	Yes	Yes
3	Yes	No	Yes	Yes
4	Yes	Yes	Yes	Yes

may use that particular remote lab. Therefore there is a message exchange between E_β and E_α, but this message exchange is not reliable for accounting. First, this communication does not guarantee that Σ_α will use the experiment: if Σ_α accesses the experiment, and there is a queue of 100 users, he might reconsider trying again later. In this case, E_α does not have a reliable process to know if it should count it as an access. Second, if the student enters again in the E_β remote laboratory, the remote laboratory might not contact again the IdP in E_α since the authorization could be cached. Once again, E_α should not rely on this authorization process to account accesses. Furthermore, this process is usually performed by IT services, which may not provide this data to the laboratory managers.

In strategies 3 and 4, consumer entity manages all the student reservation requests. Additionally, in strategy 4, all the communication sent by the student to the remote laboratory is also proxied through the consumer system, with the latency this approach adds. In both cases, risk 1 is covered in terms of number of accesses. Since scheduling is also managed by the consumer system, the consumer system becomes aware of when the provider system confirms that the user is accessing or not. In the previous example, if Σ_α sees the queue of 100 users and quits the queue, the consumer system is aware of this fact, so E_β cannot account too many requests, and if they do, E_α can compare the reservation results. However, in strategy 3 there is no way to control for how long the student used the system, since the communication with the final remote laboratory is direct. Therefore it would not cover risk 1 if the agreement is based on time, since it does not cover risk 2. The balance between strategies 3 and 4 is a matter of latency versus trust on the provider side. Nowadays, when most universities share their laboratories at zero cost with other partner universities, the balance is tipped in favor of strategy 3. The federation model detailed in this chapter supports both strategies, depending on the consumer system configuration.

4.2.5 **Design and Implementation**

The proposed federation model supports:

- Transitive federation
- Local load balance
- Distributed load balance
- Interactive experiments
- Batch experiments

As detailed in Section 4.1, the federation model is agnostic of the scheduling system used, although the design does not support booking.

4.2.5.1 **Design**

The proposed federation model does not distinguish between users and external entities. A federated system logs in as a regular user, providing credentials (usually a username and a password). The advantages of this approach are that:

- The administration tools are the same.
  - Student Σ_α can access experiment Γ_α 1000 times with priority ρ
  - Entity E_β can access experiment Γ_α 1000 times with priority ρ
- Debugging the protocol becomes simpler.
  - An external administrator can log in with the entity credentials and use the experiments.
- The model becomes simple.
  - Entity E_α grants access to student Σ_α:

$$\Sigma_\alpha \xleftarrow{\Gamma_\alpha,(\rho,t)} E_\alpha$$

  - Entity E_α grants access to entity E_β:

$$E_\beta \xleftarrow{\Gamma_\alpha,(\rho,t)} E_\alpha$$

The sequence diagram shown in Figure 4.8 shows the sequence performed in a simple federated environment. A detailed description of the interface in UML is represented in Figure 4.9, with the ReservationStatus represented in Figure 4.10.

As commented above, the behavior of the provider system is practically the same for a regular student than for a consumer system. The only difference for the provider system is that the consumer system will send additional information to the provider in the reservation process. This way, the provider system is aware of information of the original request, such as the IP address, the user agent (which is the web browser identifier, such as Mozilla Firefox 13.0 under Ubuntu GNU/Linux 12.04 or Microsoft Internet Explorer under Windows 7), and an identifier for the consumer system. The provider system will take into account this information if it is registered as an external entity instead of a regular user. This identifier could be easily anonymized by the consumer system, since it is not used by the provider system. In this line, particularly important information exchanged is a list of federated systems that have been tested. For instance, if E_α shares a copy of a laboratory type with E_β, and E_β shares a copy of the same laboratory type with E_α it might happen that a loop is created. This loop is broken once the remote system knows that a remote system is forwarding a local request.

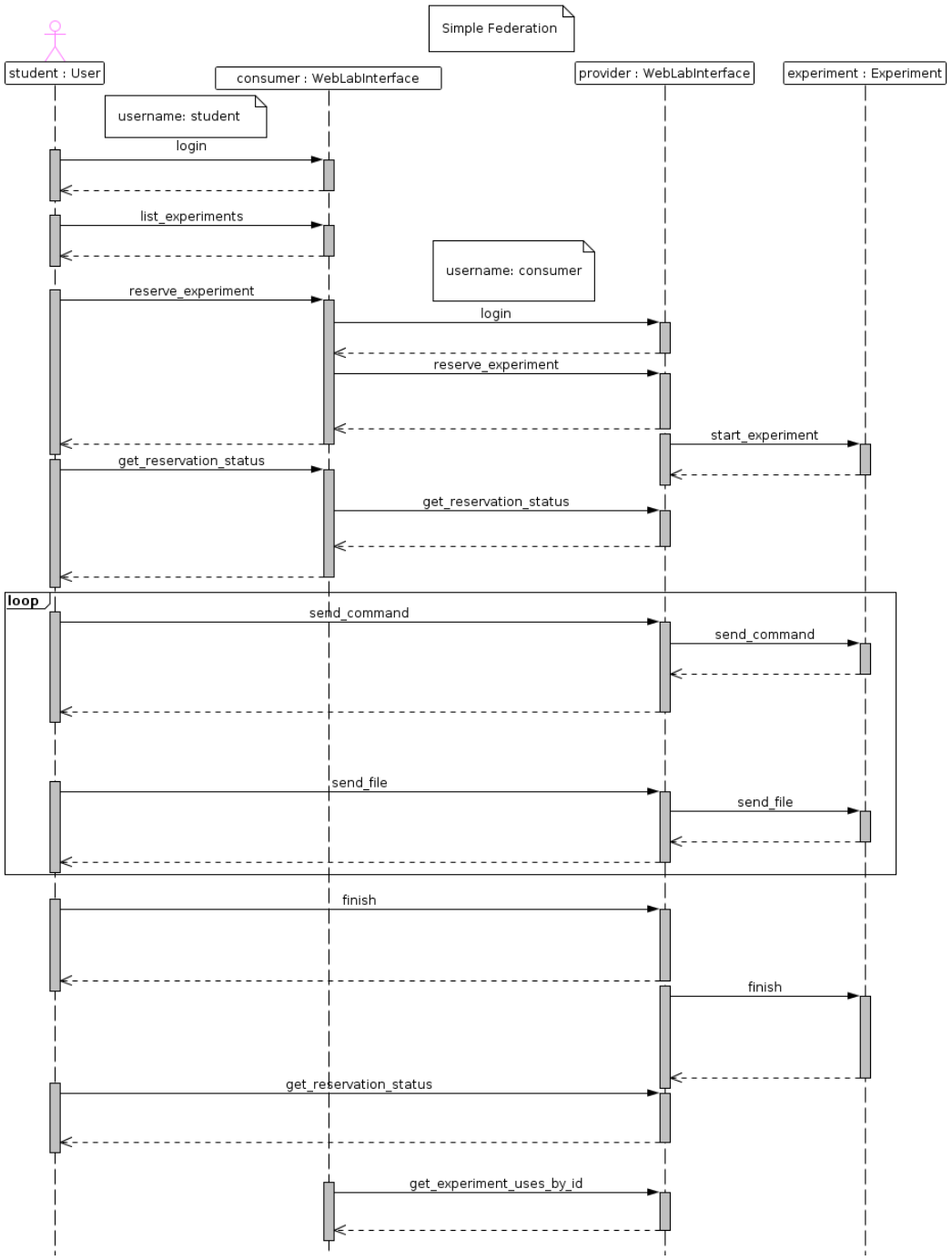


Figure 4.8: Federation sequence diagram

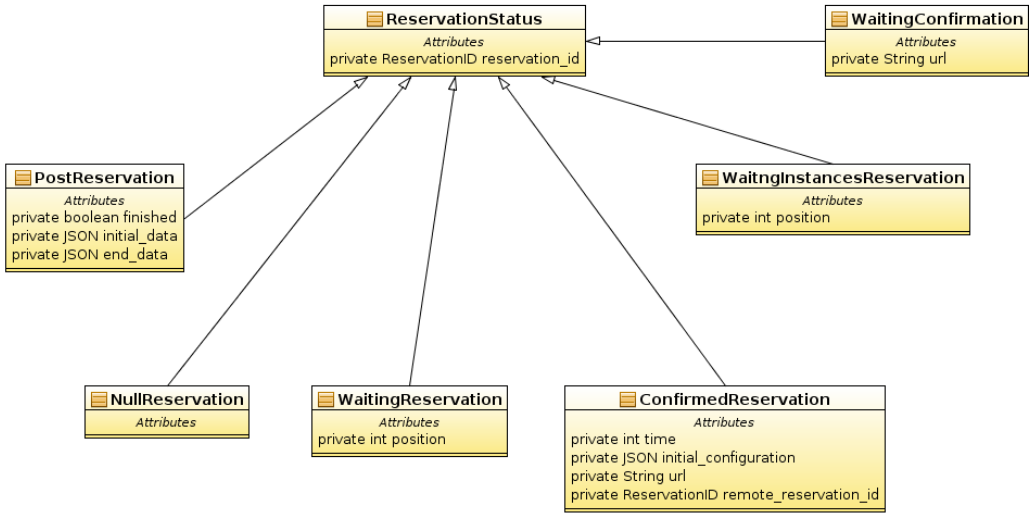


Figure 4.9: Federation model class diagram: main interface

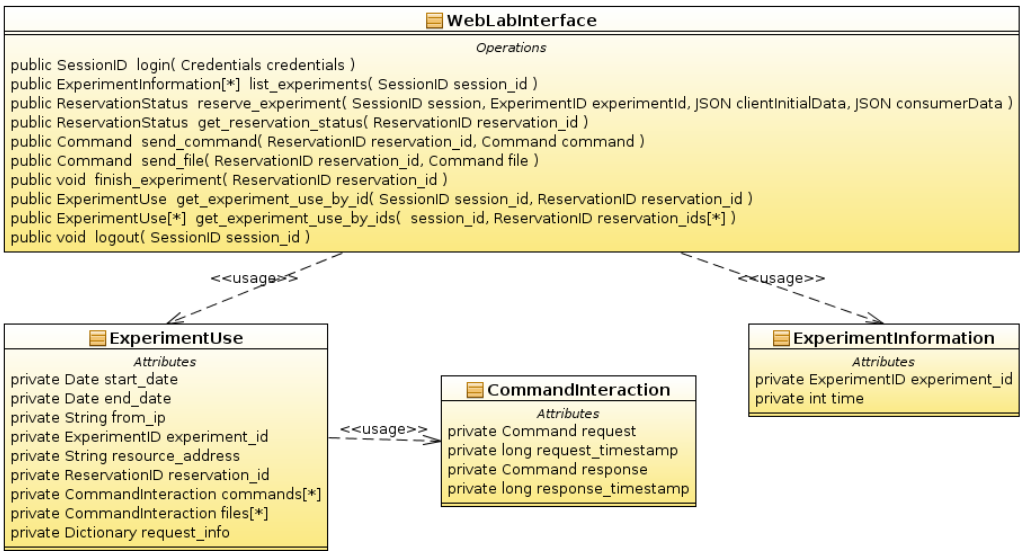


Figure 4.10: Federation model class diagram: reservation status

For the client, the mechanism is also very similar. The only difference is that when the system is reserved, the confirmation message that the consumer system sends to the user comes with an URL pointing at the provider system, and the reservation ID that must be used in the provider system. The client is aware of this, so if these fields are not empty, it will attempt to connect with that reservation ID to the remote system.

The main difference is for the consumer system. When students request a reservation to the consumer system, it will look up in the local scheduling system, described in Section 4.1. The consumer system may have local resources and federated systems matching the requested experiment. In the simple case of Figure 4.8, it will be assumed that no local resource exists that implements the system, and there is a single federated system which counts with matching resources. In this case, when the reservation method is called, the consumer system will call the provider's reservation method (prior authentication with the credentials of the consumer system), providing additional information such as the user agent and the user's IP address. When the student requests the status of the reservation, the provider system will check in its scheduling system and the consumer system will forward the requests, changing the reservation IDs. This process will continue until the remote experiment states that it has been finally reserved. At this moment, the consumer system will add the provider reservation ID and the URL so the client can connect directly to the provider system and start interacting with it. From this moment, the consumer system will be polling the provider system asking for the experiment usage done by that reservation. The provider will detail that the experiment does not finish until the student finally finishes or the assigned time expires. At this moment, the provider will return all the commands and files exchanged between the student and the provider.

The same process applies for transitive federation. As shown in Figure 4.11, there is an entity E_α called *Consumer* in the diagram, an entity E_β called *Prosumer* in the diagram (since it is acting as a provider and a consumer at the same time), and finally an entity E_γ called *Provider* in the diagram. The relationship is:

$$E_\beta \xleftarrow{\Gamma_\gamma} E_\gamma$$
$$E_\alpha \xleftarrow{\Gamma_\gamma} E_\beta$$
$$\Sigma_\alpha \xleftarrow{\Gamma_\gamma} E_\alpha$$

The sequence of the process resembles very much to the previous example, with an extra layer represented by E_β. If both Figure 4.8 and Figure 4.11 are compared, the process done by the consumer system in both diagrams is exactly duplicated in Figure 4.11 by the *prosumer* system. Indeed, since the *consumer* E_α does not need to be aware that there is a third entity E_γ involved in the process and the entity

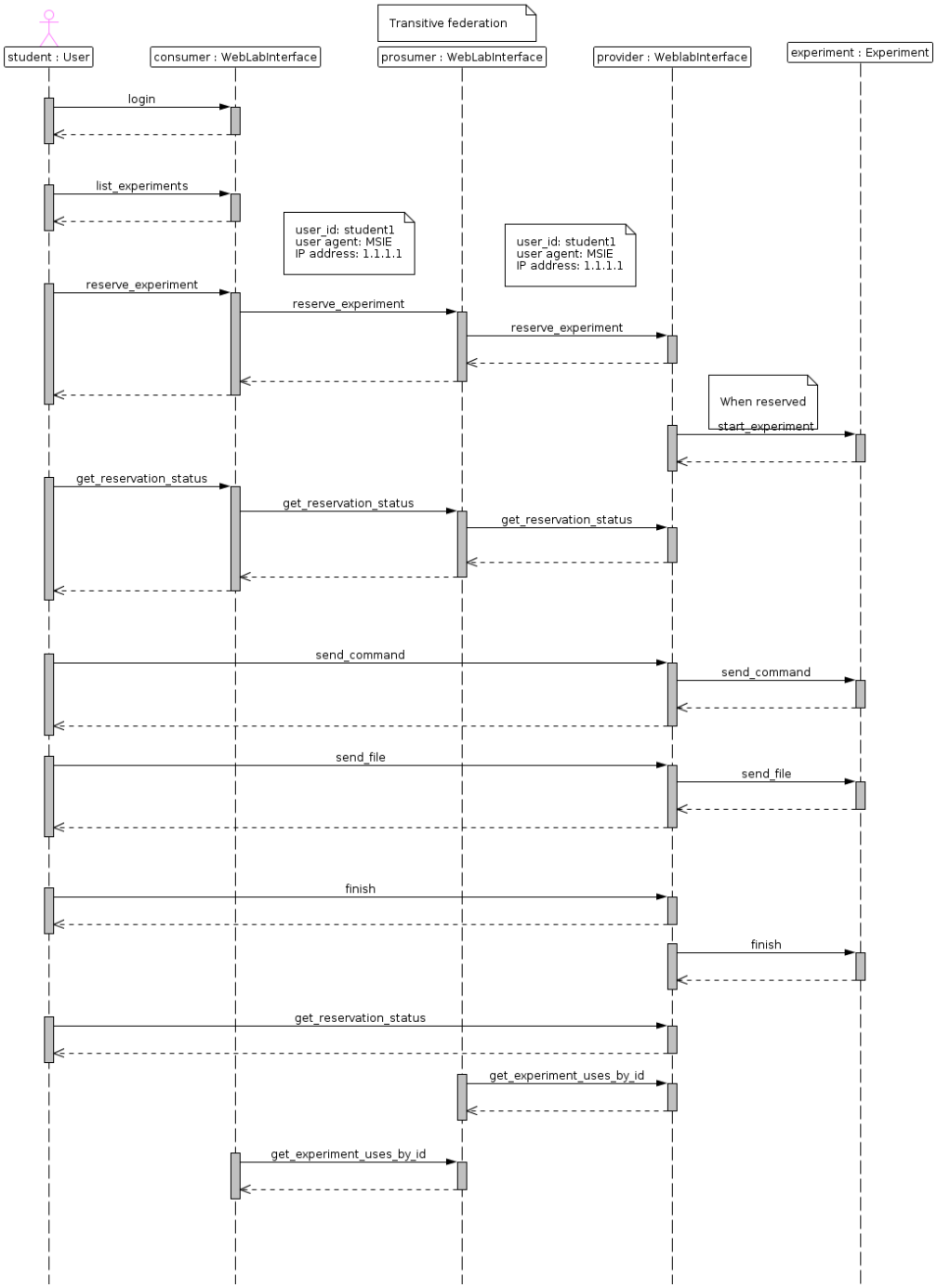


Figure 4.11: Transitive federation sequence diagram

E_β does not treat differently a regular student or an external entity, the process is exactly the same.

The process slightly changes in the case of federated load balance. This sequence is described in Figure 4.12. In this diagram, there are two independent providers (E_β called *provider1* and E_γ called *provider2*) which directly share a resource with the consumer system (E_α). Therefore, the relationship in this case is:

$$E_\alpha \xleftarrow{\Gamma_\beta} E_\beta$$
$$E_\alpha \xleftarrow{\Gamma_\gamma} E_\gamma$$
$$\Sigma_\alpha \xleftarrow{\Gamma_\beta} E_\alpha$$
$$\Sigma_\alpha \xleftarrow{\Gamma_\gamma} E_\alpha$$

In the reservation method at E_α, it will look up again the requested experiments in the local scheduling system, but this time it will find both E_β and E_γ. The system will perform a reservation method in each of these systems, but for each external entity, if the external entity responds that a resource has been reserved, E_α will cancel the rest of the reservations it previously did and it will stop doing new reservations. In the same way, for each request performed by the client asking for the reservation status, it will ask the other systems, and if a system responds that a resource was reserved, the rest of the reservations will be cancelled.

The reservation returned to the user is the best reservation gathered among the systems. For instance, if E_β returns that it is in position 1 and E_γ returns that it is in position 2, the system will return the user that it is in position 1. This is done this way given that the consumer system does not know what the priorities in the federated environments are. For instance, being in position 2 at E_β and in position 5 at E_γ does not mean that there are more chances to go to E_β, since that depends on the number of copies of the laboratory deployed. If E_β has deployed a single copy of the laboratory and E_γ has deployed 20, the queue might be faster in E_γ.

While the described algorithm is the one used for experimentation in the current dissertation, the federation model permits changes on the underlying algorithm keeping the same interface. For instance:

- The consumer system may cancel the reservations whenever a system returns that it is in position 1, 2 or any fixed number in the queue. This would be a more conservative policy that would make it more difficult for the same reservation to get reserved by two resources from different institutions.

- The consumer system may request all the pending reservations to each federated system on its own, and cache the responses for a few seconds, regardless of whether users are performing requests or not.

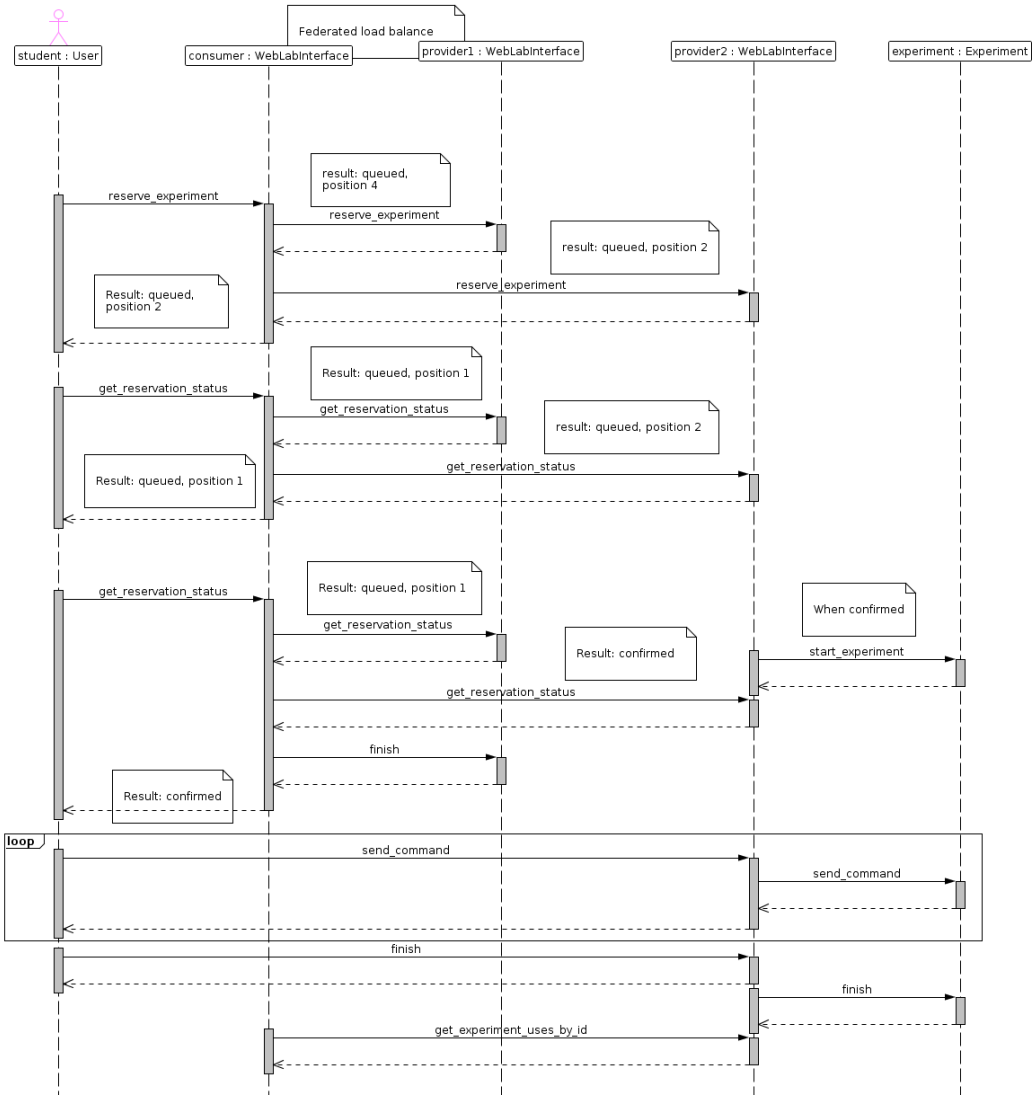


Figure 4.12: Federated load balance sequence diagram

- The consumer system may make a priority in the way it chooses the federated environments in order to reduce the global number of interactions. If a ring of 6 providers with copies of the same laboratory is built, different providers could build different priorities to balance the number of users and therefore reduce that all systems manage all the requests.

These changes have not been considered relevant for the final evaluation neither have an impact on the design of the federation model, so they have not yet been implemented in the reference implementation.

4.2.5.2 **Implementation**

The federation model has been implemented on top of WebLab-Deusto, a remote laboratory management system described in Chapter 3. Given that the federation model is based on the fact that external entities are almost treated as regular students, the core of the required change is in how WebLab-Deusto manages internally the resources, detailing which resources are local and which ones are federated.

As previously detailed in Section 4.1, the core of the scheduling infrastructure of WebLab-Deusto is a plug-in system, where different implementations can be developed. The most important implementation is a priority queue manager. However, other implementations coexist, including one to consume iLab batch systems, described in Section 4.3.2, and one for consuming external WebLab-Deusto instances, which allows federation.

This federation plug-in, as all the scheduling system, counts with two versions: a cross-platform version, developed in pure SQL which has been tested on SQLite and MySQL, and a Linux-aimed version developed using Redis, which is much faster, as detailed in Chapter 5.

Given that it is a plug-in of the scheduling system, the same key concepts still apply in terms of experiment IDs. Multiple experiment IDs can be associated to the same scheduler. For instance, an experiment ID could be *physics-1* and other experiment ID could be *physics-2*, and both experiment IDs could be mapped to the same federated physics remote laboratory. Furthermore, the plug-in extends this concept allowing the administrator to establish more complex associations. The provider university E_α could have an experiment called *visir*, shared with the university E_β. The university E_β can still create its own independent mappings, so its students can see two different experiment identifiers such as *physics-1* and *physics-2*. The administrator can still establish:

```
1  weblabdeusto_fed = ("EXTERNAL_WEBLAB_DEUSTO", {
2      'baseurl'  : 'http://www.weblab.deusto.es/weblab/',
3      'username' : 'weblabfed',
4      'password' : 'password',
5      'experiments_map' : {
6          'physics-1@Physics' : 'visir@VISIR',
```

```
7           'physics-2@Physics' : 'visir@VISIR',
8       }
9   })
```

This way, any request to the experiment *physics-1* of the category *Physics* will be renamed as a request to *visir* of the category *VISIR* in the remote system. In Figure 4.13, the VISIR system in ISEP is called *visir-isep*, which is not the name used in the remote system.

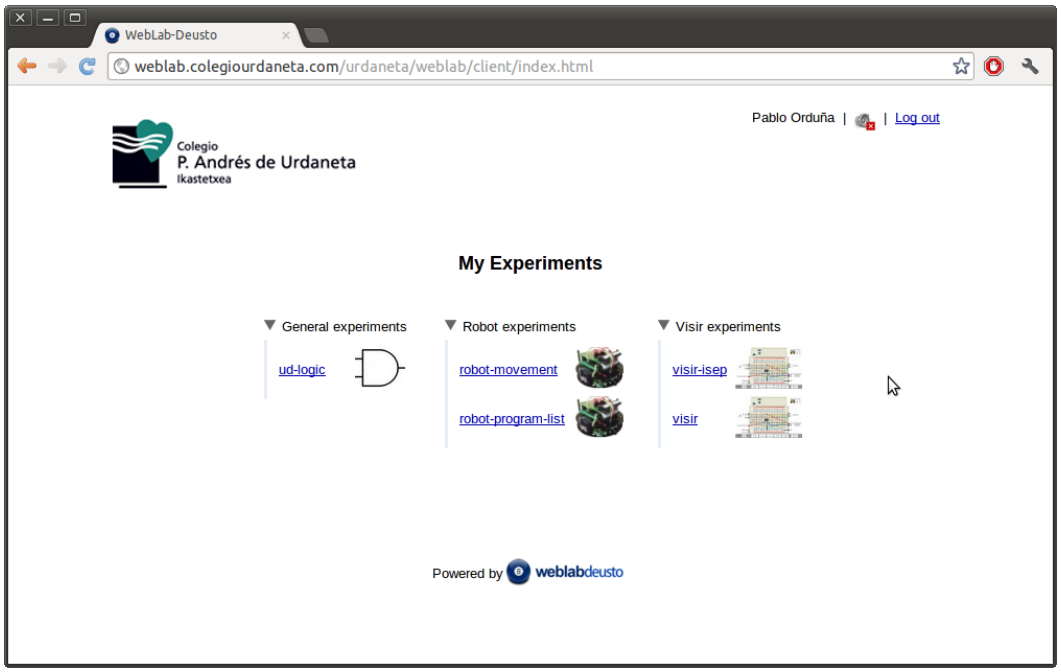


Figure 4.13: List of experiments in the Colegio Urdaneta secondary school

In the client side, whenever a student reserves an experiment and the experiment is located in a foreign entity, it must perform requests directly to the remote entity. However, this is limited by web browsers to avoid cross-site scripting attacks (XSS). Therefore, it was required to open a popup window with the remote system whenever a federation was being used. As seen in Figure 4.14, a request to open a new window is shown to the student, who will see the window shown in Figure 4.15. It is possible to see how the logo of the remote system is shown at the bottom of the web page. For instance, in Figure 4.16, the logo of ISEP can be seen, since the resources being used are in ISEP.

The transitive federation is seamless for the user, as well as the federated load balance. In the screenshots shown, the Colegio Urdaneta secondary school does not have configured any contact with ISEP. Therefore, the relationship is this one:

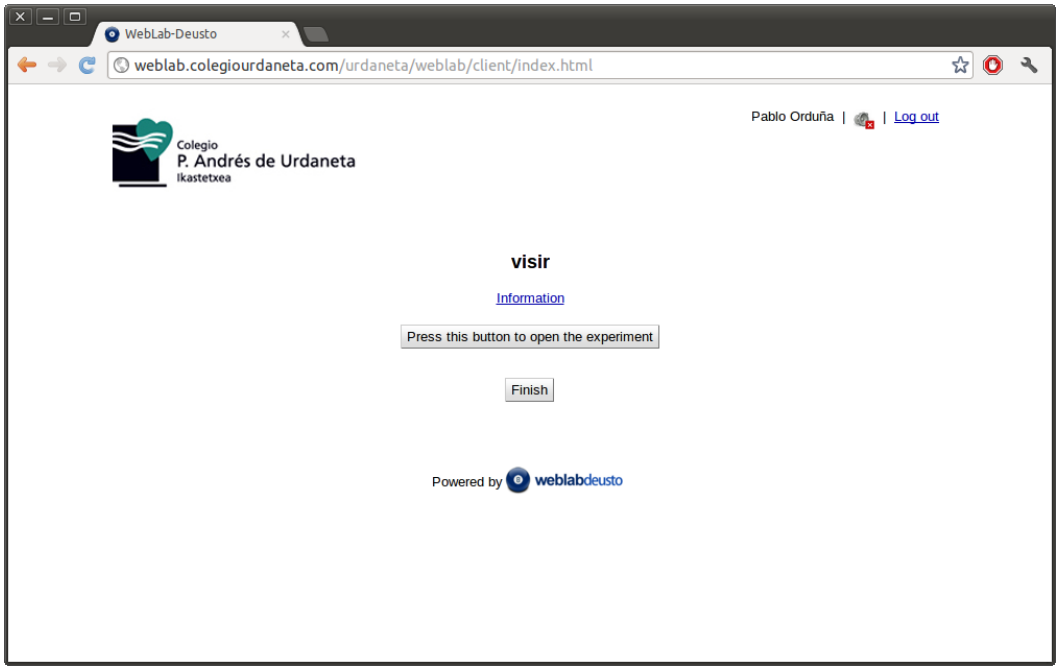


Figure 4.14: WebLab-Deusto in Colegio Urdaneta: reserved experiment in remote system

$$E_{UD} \xleftarrow{VISIR_{ISEP}} E_{ISEP}$$
$$E_{CU} \xleftarrow{VISIR_{ISEP}} E_{UD}$$

However, the final client is not aware of how this process is handled. It does not either know if the requested experiment will be local or federated, allowing the consumer system to find a proper resource among different local and federated copies.

4.3 Applications of the Federation Model

The federation model is focused on sharing laboratories among different copies of WebLab-Deusto. However, during the design it was also aimed to be useful in other situations. This section describes how the federation model can be applied on different applications of related fields. It also details how other models may not achieve this.

The section is divided in three subsections:

- Integration of remote laboratories on Learning Management Systems through federation model → Section 4.3.1.

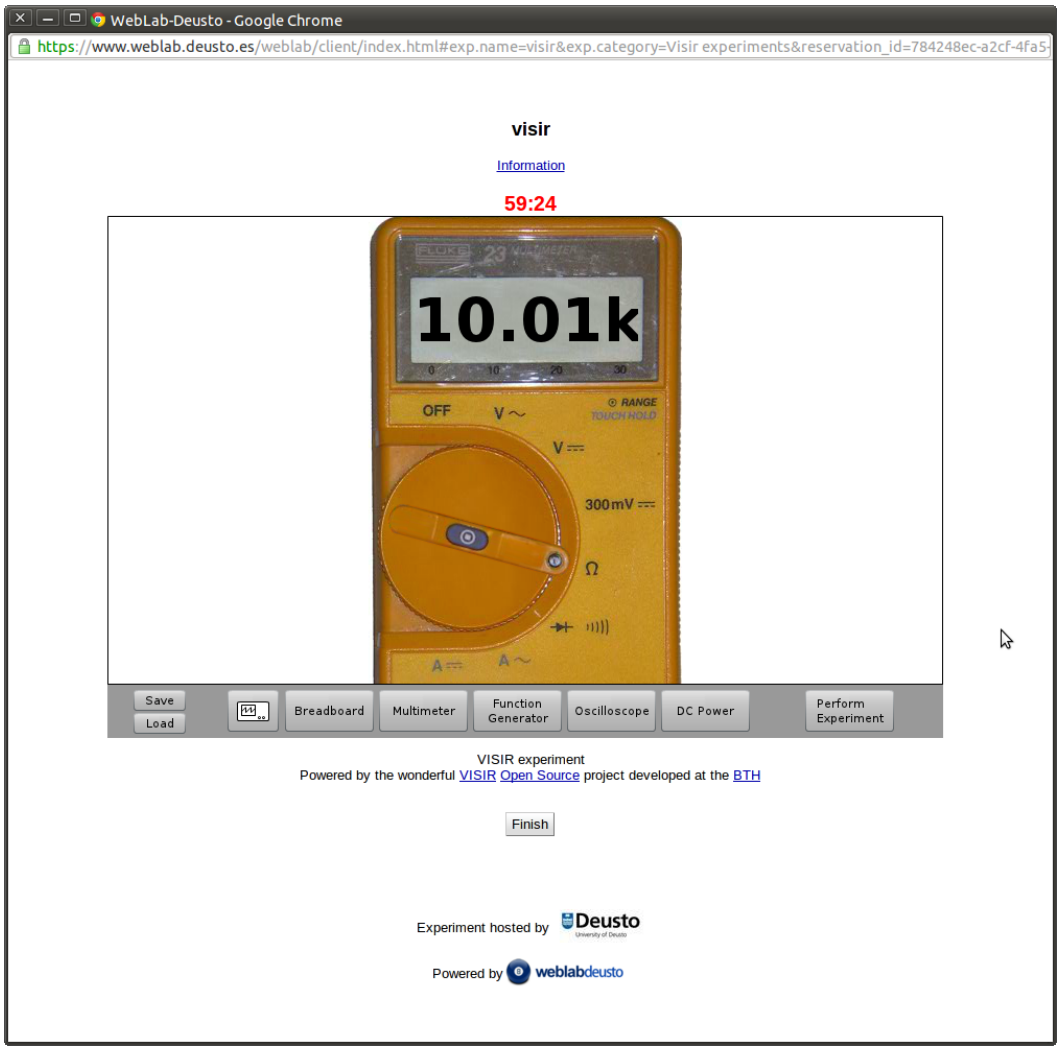


Figure 4.15: VISIR experiment reserved in the University of Deusto

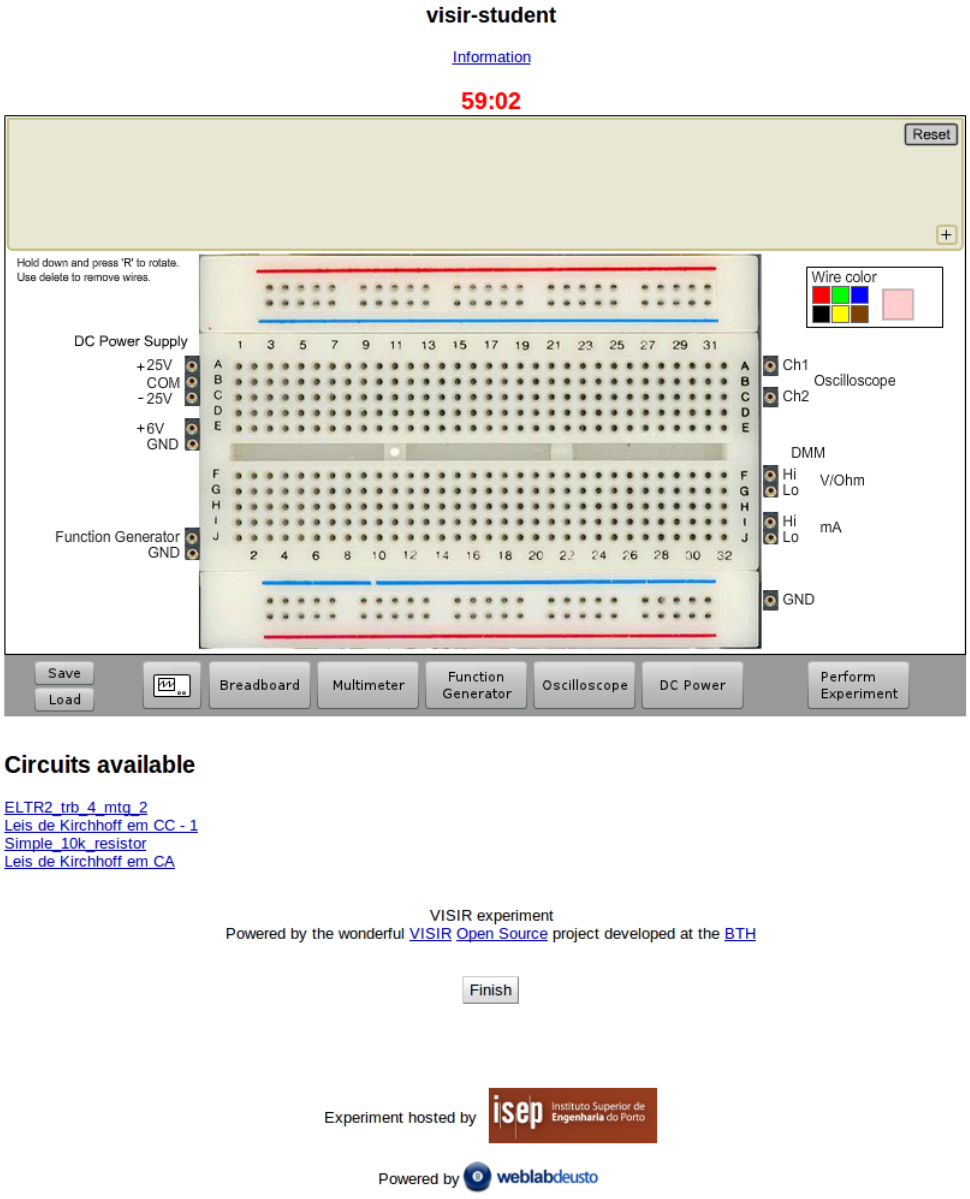


Figure 4.16: VISIR experiment reserved in ISEP

- Interoperability among different remote laboratory management systems → Section 4.3.2.
- Deploying resource limited remote laboratories outside university campus → Section 4.3.3.

4.3.1 Learning Management Systems and Content Management Systems

The aim of this section is to model and evaluate how the proposed federation model for remote laboratories can be used to support third party consumer systems such as LMSs, CMSs or e-learning tools. Since the discussed federation model is focused on simple rules (priorities, access) for third-party agents consuming laboratories -regardless they are remote laboratories or other kind of software- it becomes reusable for the integration of other e-learning tools. With this approach, the remote laboratory will still establish particular rules for each system and for each external institution. Furthermore, the provider university can establish high level rules (e.g., University A grants 10,000 accesses on Laboratory 1 to University B) and the consumer university can decide to let their users consume it through different technologies (e.g., different LMSs or remote laboratories) without notifying the provider.

Proper integration of remote laboratories on LMSs is an active field of research. The relevance of this field is that, as detailed in (Sancristobal et al., 2010; Sancristobal, 2010), there are several services duplicated between remote laboratories and learning management systems. The administration and user experience would increase if they were merged. Both systems usually support user authentication, authorization, group management, administrative tools, user tracking, and even scheduling. Some integration approaches suggest to delegate all these services to the LMS, but some of these services will still be at least shared, such as scheduling (especially when federation systems arise) or user tracking (since some interactions with the remote laboratory might occur outside the scope of the web browser).

In order to integrate remote laboratories and LMSs, (Gomes and Bogosyan, 2009) discusses the usage of SCORM and (Richter et al., 2011a) implements an architecture around it. This technology is designed to be supported by different LMSs and indeed multiple LMSs have implemented different versions. However, since it is a client-side technology and therefore it cannot contain any server code, it does not support a secure way to exchange credentials, ensure reservations or return results to the LMS.

Other approach is to develop an ad-hoc plug-in to include a particular remote laboratory on a LMS (Ferreira et al., 2005). This approach is common in the literature, and sometimes it is implemented by just copying or exchanging the users

between both systems. Within the field of integrating remote laboratories on electronic tools appears the integration of remote laboratories on CMSs. In (Abdulwahed and Nagy, 2010), the remote laboratory relies on Joomla to perform all the administrative tasks. This approach is interesting since it does not duplicates all the tasks referred in (Sancristobal et al., 2010). However, it is an example of an ad-hoc integration which does not support the integration of other remote laboratories neither the integration on other CMSs.

Most integrations among remote laboratories and LMSs or CMSs are ad hoc integrations of one particular LMS/CMS to a particular remote laboratory. The target of these integrations is to delegate the tasks for authentication, authorization, user tracking and scheduling to the LMS/CMS. This delegation is the same delegation described in Section 4.2 as the target of federation models: the consumer remote laboratory will manage the authentication, authorization and user tracking, while the provider remove laboratory will be in charge of enabling the interaction of the user with the experiment.

4.3.1.1 Integration LMS/CMSs Using a Federation Model

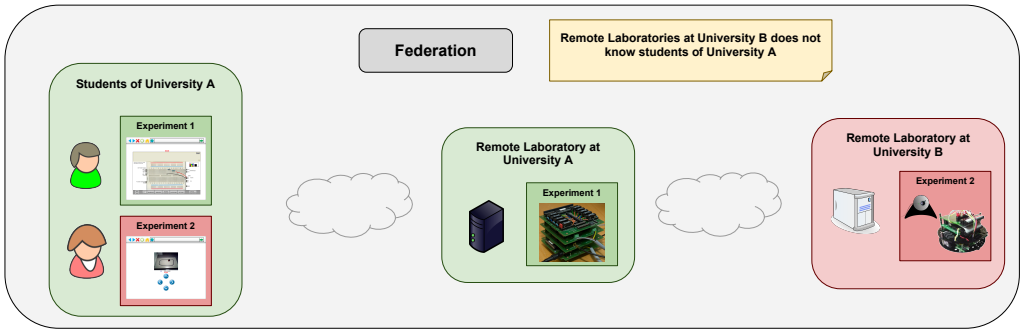


Figure 4.17: Federation of remote labs. Students registered in the federation

The key concept introduced in this section is that the remote laboratory integration on LMS/CMSs is essentially equivalent to the integration of two federated remote laboratories. Thus, a LMS/CMS could be considered an external consumer of the federation protocol of a remote laboratory management system. As described in Figure 4.18 and compared to Figure 4.17, the LMS/CMS acts as a federated node, even if it is in the same university as the remote laboratory. This way, the student in the figure will access through the LMS/CMS to the remote laboratory.

The relevance of this concept is that it simplifies notably the integration of both systems. Under this approach, the LMS would not need to exchange user information, or use standards such as OAuth or Single Sign On protocols to effectively

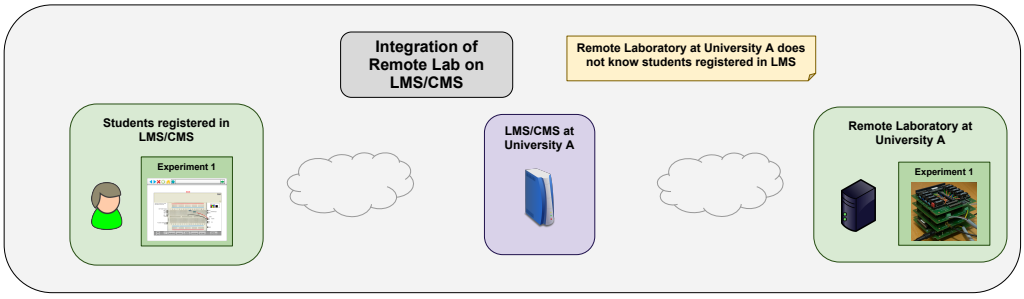


Figure 4.18: A LMS/CMS can act as a federated node consuming the federation protocol

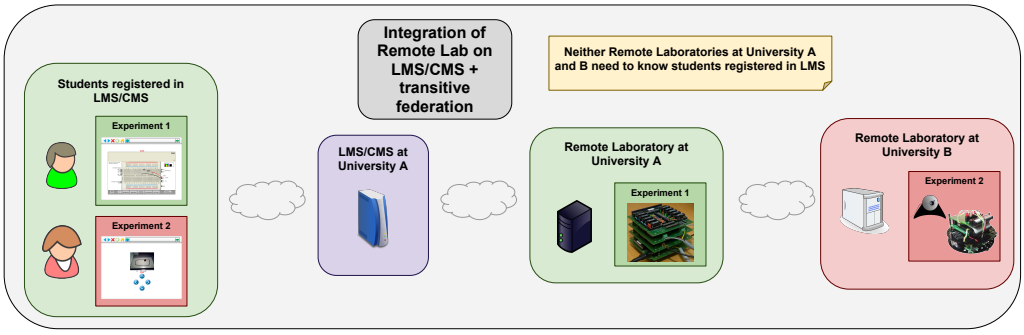


Figure 4.19: If the federation is transitive, the LMS/CMS will still be able to consume remote laboratories in other universities

perform reservations in the remote system. If the protocol is simple enough, different systems can adopt it, since the interface only needs to provide three actions:

- Listing of possible experiments (so administrators of the LMS/CMS can choose who has access to what).
- Reserving the remote experiment.
- Usage retrieval (to be stored in the LMS/CMS database).

It is worth mentioning that if the federation model does not support transitivity, the LMS will not be able to consume remote laboratories on other universities through the institutional remote laboratory. This means that in the example of Figure 4.18, students registered in the LMS will be able to use experiments located in University A. But, if no transitivity is supported, and if the remote laboratory of University A was federated with one of University B, students registered in the LMS would not be able to use them. The only way to achieve this would be registering also the LMS in University B. However, given the transitivity property of the federation model supported in WebLab-Deusto, Figure 4.19 becomes possible without expliciting all the contracts and subcontracts.

For the sake of clarity and evaluation, two case studies are presented. They describe sample integrations in a LMS (Moodle) and a CMS (Joomla) of the remote laboratory WebLab-Deusto.

4.3.1.2 **Case Study: Moodle**

A Moodle plug-in using the federation protocol has been developed. In the Moodle system, different remote laboratory management systems can be registered. The connection data will include the URL of the remote system, and a single username and password, identifying the university. Three roles are defined:

- *Administrators:* Administrators can associate courses to experiments. As shown in Figure 4.20, they can define that the course "Electronics IV, Electronics Engineering can access the VISIR experiment in University A." In order to do this, the plug-in use the *list_experiments* method detailed above.
- *Teachers:* Teachers of courses with associated experiments are automatically allowed to add accesses to laboratories as Moodle activities to the course. In the menu shown in Figure 4.21, in the second step they will choose among the associated laboratories. They can add, remove or hide the activity as any other Moodle activity.

- *Students:* Students of courses with associated experiments can automatically use the experiments (see Figure 4.22), once teachers have created the activities. Whenever a student requests an access, the plug-in will perform the request in the name of the student, using the university credentials.

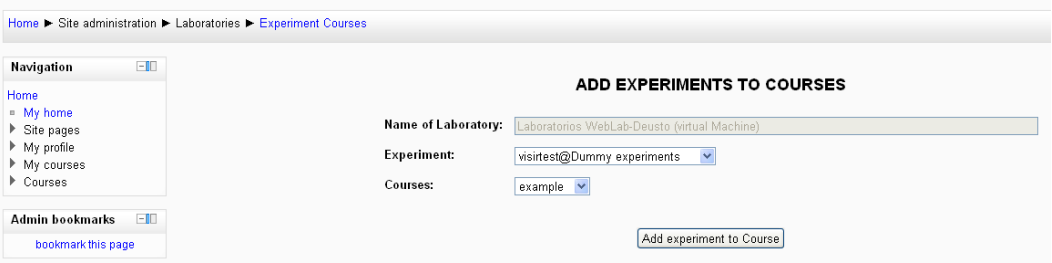


Figure 4.20: Admin granting permissions of an experiment on a course

Figure 4.21: Teacher adding a remote lab as an activity for a course

Only Moodle administrators can define which courses have access to each external remote lab. The management becomes simple: every teacher added to a course through the Moodle administrative tools can automatically manage the associated laboratories. Every student enrolled in a course can use the laboratory. Remote laboratories do not need to exchange any user information (credentials, full name, etc.) with the LMS.

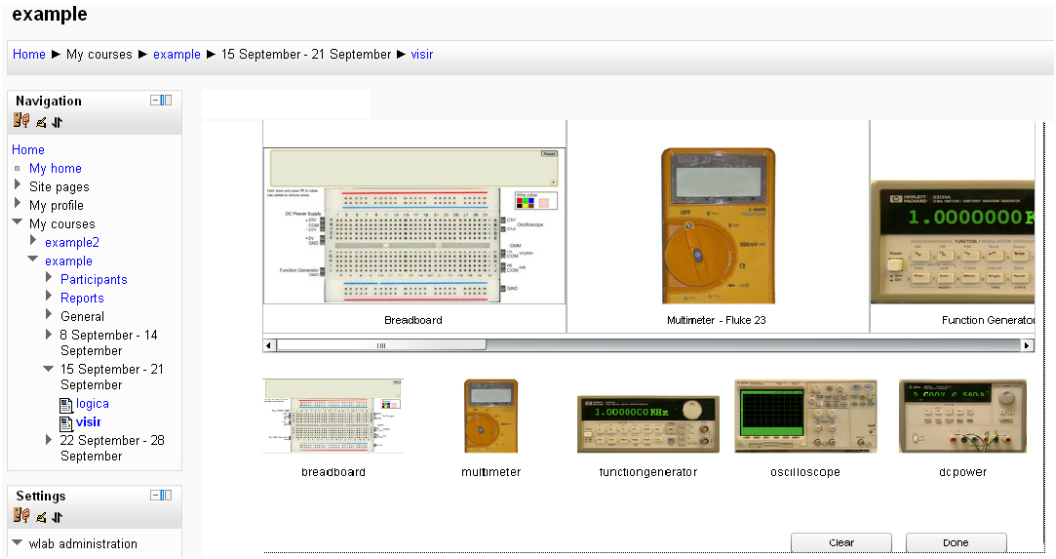


Figure 4.22: Student running an experiment

4.3.1.3 Case Study: Joomla

A plug-in for Joomla using the federation protocol has been developed (live demo[1]). The schema used for this integration was simpler, with only two roles involved:

- *Administrators:* they will manage which groups have access to what remote experiments. This is done through the standard Joomla's administration site (see Figure 4.23, Figure 4.24 and Figure 4.25).

- *Users:* there is a link where every user can see what experiments can be run. This way, users who are members of groups with associated experiments will see those experiments and they will be able to start using any of them (see Figure 4.26).

As in the Moodle case, users and administrators of Joomla use the standard tools provided by the framework. In this case, authentication and group membership is managed with the standard tools. The plug-in only needs to define which groups can access which laboratories, as well as provide a small component to start the reservations. The remote laboratory does not need to exchange credentials or register students.

[1] http://www.weblab.deusto.es/weblab_joomla/

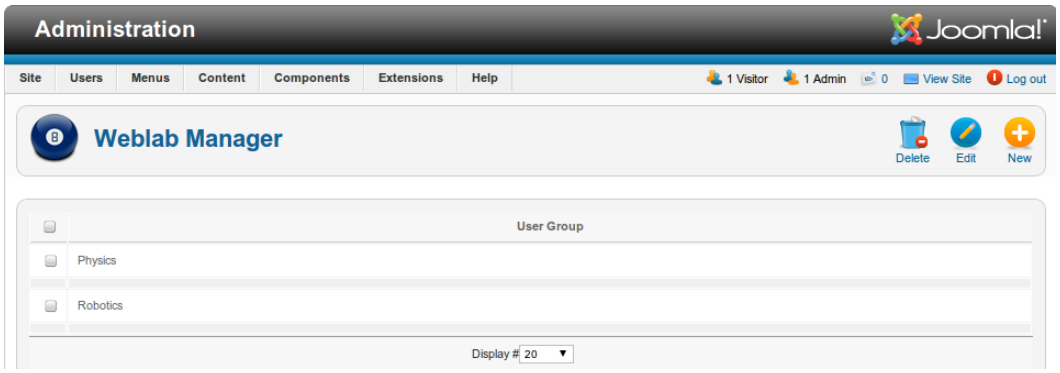


Figure 4.23: Available Joomla user groups to be edited

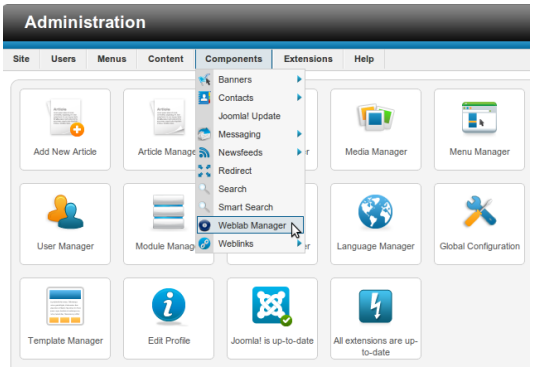


Figure 4.24: WebLab-Deusto manager in the Joomla Administration panel

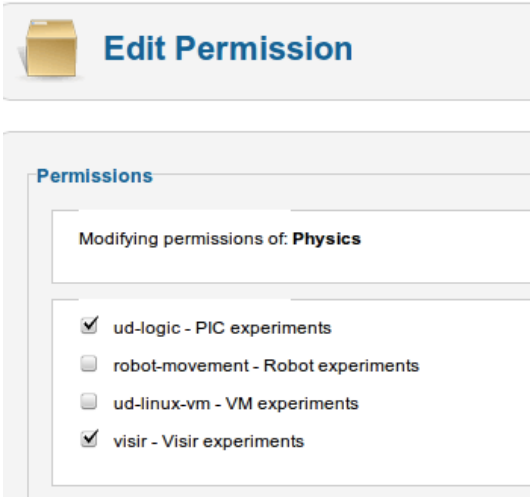


Figure 4.25: Permissions editor for each group

Figure 4.26: List of experiments available for current user

4.3.2 Interoperability

As detailed in Chapter 2, there are different Remote Laboratory Management Systems (RLMS) that support the development and maintenance of remote laboratories.

The approaches taken by these systems are different, and even key features of some of them are not supported on the rest. This is common given the wide background differences in remote laboratories in terms of technologies (Gravier et al., 2008) and approaches to create the laboratories. In order to build an ecology of remote laboratories (Harward et al., 2008a), not only a software infrastructure is required, but also a deep understanding of the student audiences. Since each system has been influenced by different student audiences, building bridges between two systems, when feasible, make it possible for each system to consume laboratories designed for other audience. This concept, described in Figure 4.27, will be denoted as *heterogeneous federation*, where two different systems will interoperate in a federated way.

In this line, (Yeung et al., 2010) proposed the LabConnector application protocol interface (API) as a bridge between iLabs and Labshare Sahara focused at protocol level, evaluating it with an iLab laboratory located in the University of Queensland being consumed by Labshare Sahara. While the bridge itself might have technical difficulties in becoming adopted by other systems, it represented a clear step forward in the interoperability of remote laboratory management systems.

In this section, a bridge to consume WebLab-Deusto laboratories by the iLab Shared Architecture (Harward et al., 2008b), as well as an experimental bridge to consume iLab batch laboratories from WebLab-Deusto is presented. While this solution does not pretend to be directly extrapolated to other remote laboratory management systems, it focuses on drawing the potential advantages of federating other laboratories for both remote laboratory management systems rather than at protocol issues. In order to achieve a global solution, an interface defined by the

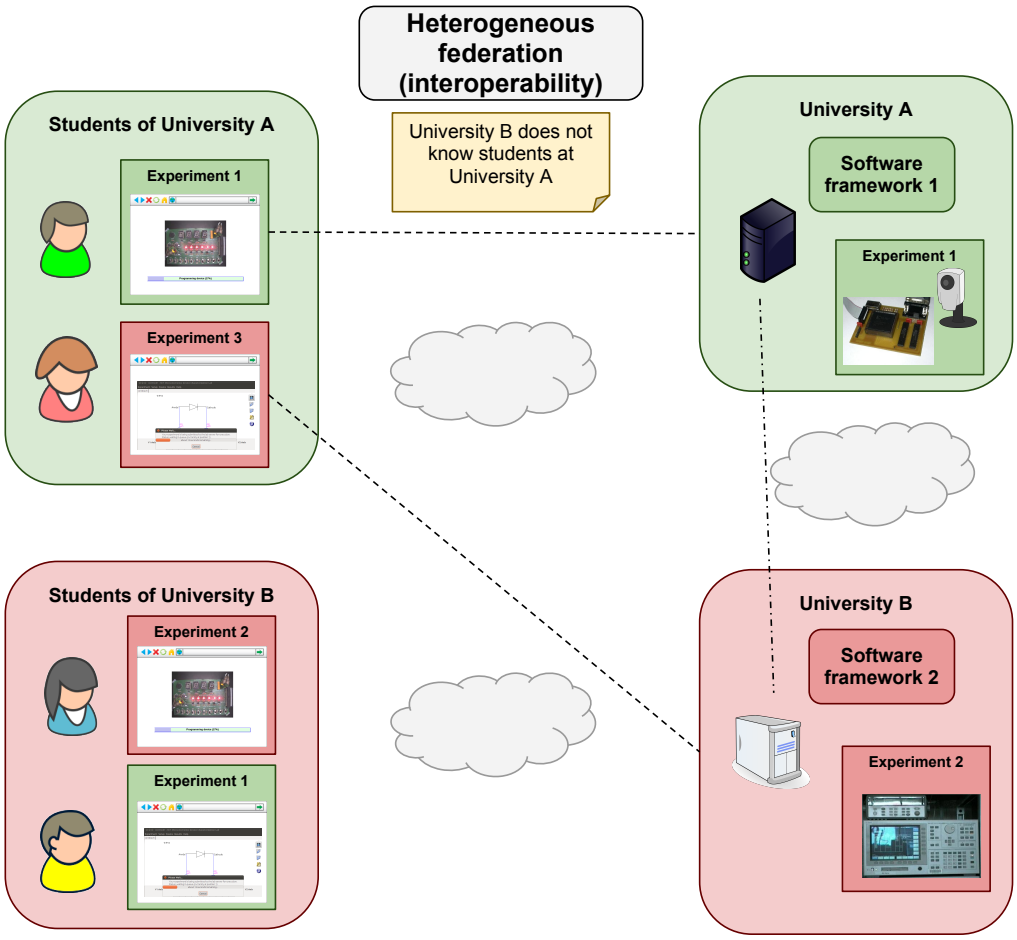


Figure 4.27: Heterogeneous federation

Global Online Laboratory Consortium (GOLC[1]) or the IEEE P1876[2] (which is a new initiative aiming to standardize online laboratories) would be required. The relevance of this bridge is that it shows how using the proposed federation model it is possible to support and be supported by other remote laboratories.

4.3.2.1 Case Study: WebLab-Deusto Running in MIT iLabs (ISA)

WebLab-Deusto supports batch and interactive (managed and unmanaged) laboratories, but all the laboratories used by students nowadays are interactive. Therefore, the target is to use WebLab-Deusto interactive laboratories in iLabs, regardless of whether they are managed or not. However, the iLab Shared Architecture (ISA) only supports scheduling through booking in the case of interactive laboratories, while WebLab-Deusto nowadays only supports queuing.

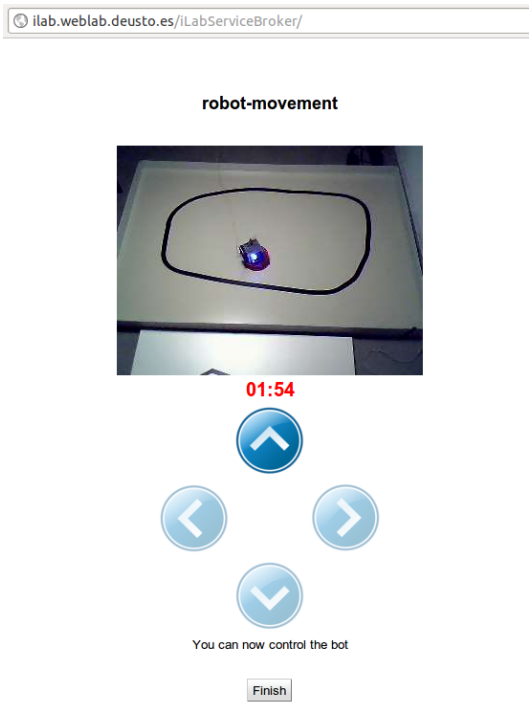


Figure 4.28: WebLab-Deusto robot laboratory running through an iLab system

In order to handle this issue, the no scheduling option was selected in the ISA for the integration. Using this option, the ISA relies completely on the Lab Server, sending all the users that attempt to use the laboratory to the Lab Server. With

[1] http://www.online-lab.org/
[2] http://standards.ieee.org/email/2012_09_cfp_P1876wg_web.html

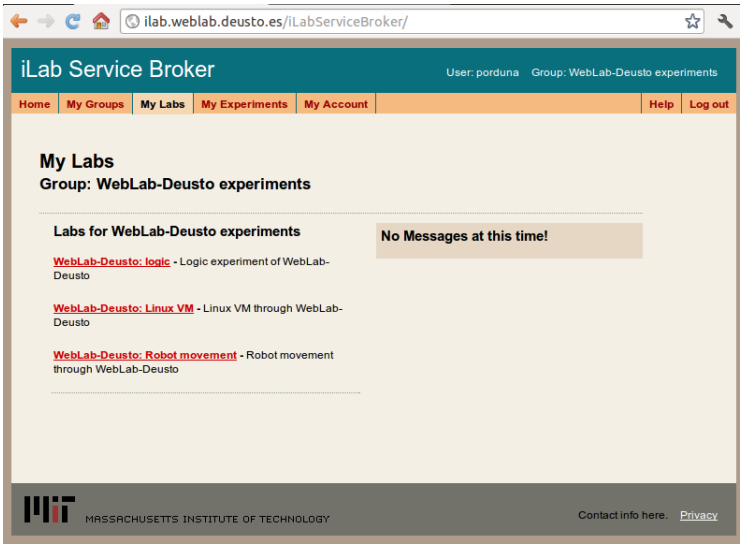


Figure 4.29: WebLab-Deusto laboratories listed in iLab

this scheme, it was possible to develop a special type of Lab Server, which relied on WebLab-Deusto through web services. Therefore, every time a student tries to use the laboratory, the Lab Server will interact with WebLab-Deusto. In the configuration file of each Lab Server it is possible to define not only the credentials required and the desired laboratory type, but also advanced arguments such as the priority required or the time assigned in the remote system.

For instance, an iLab system could have three copies of this "glue" Lab Server, two of them configured to use a robotics laboratory, and another to use a FPGA laboratory. At ISA level, they are three different laboratories, even if they all use the same WebLab-Deusto system that manages three real laboratories. The two robotics laboratories could even be the same at WebLab-Deusto but split at ISA to manage different priorities. This way, if students of 2nd grade are assigned to a robotics laboratory and students of 4th grade are assigned to the other, the iLab administrator can choose which group has a higher priority over time, even if this priority is managed in the queues in the remote WebLab-Deusto system.

Being WebLab-Deusto laboratories presented as regular iLab laboratories, the authentication and authorization can be managed through the iLab tools (see Figure 4.28 and Figure 4.29). Furthermore, if the particular host institution desires to share the laboratories with other universities, it is possible using the ISA.

4.3.2.2 Case Study: iLab Batch Laboratories Running on WebLab-Deusto

An experimental bridge of batch iLab laboratories into WebLab-Deusto has been implemented. This way, it becomes possible to consume iLab laboratories from WebLab-Deusto instances. Given that it has been implemented as a plug-in in the core of the scheduling system of WebLab-Deusto, all the messages are automatically stored so educators can track the usage performed by students. The authentication and authorization is managed by WebLab-Deusto, so it can rely on systems supported by WebLab-Deusto. For instance, WebLab-Deusto supports creating accounts and being authenticated through OAuth 2.0 with Facebook, so as seen in Figure 4.30, a student with permissions to use an iLab laboratory can run it from facebook.

So as to process the requests sent by iLab Lab Clients, a translator of a subset of the possible requests to their corresponding request type in WebLab-Deusto was developed. Once they have become WebLab-Deusto requests, WebLab-Deusto uses its pluggable scheduling system to handle the requests with a new scheduling plug-in for the iLab integration. This plug-in will convert the requests again and forward them to an external iLab Lab Server, therefore acting WebLab-Deusto as a Service Broker.

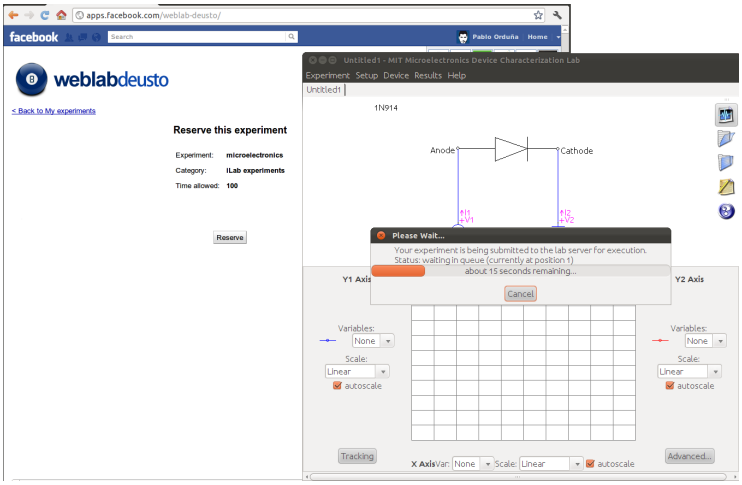


Figure 4.30: MIT Microelectronics WebLab running through WebLab-Deusto in Facebook

Finally, since this bridge has been implemented as a plug-in also allows WebLab-Deusto to use other plug-ins of WebLab-Deusto, so inter-institutional chains can be built. As described in Figure 4.31, students can access WebLab-Deusto in their institution (Institution A in the example), and through the plug-in resolver they can use an iLab plug-in contacting an iLab Lab Server in the same

institution. Furthermore, through the federation plug-in, it is possible to connect WebLab-Deusto with other WebLab-Deusto in the Institution B, which has also set up the iLab plug-in with yet another institution (Institution C). More complex chains, even supporting distributed load balance, could be built with this approach.

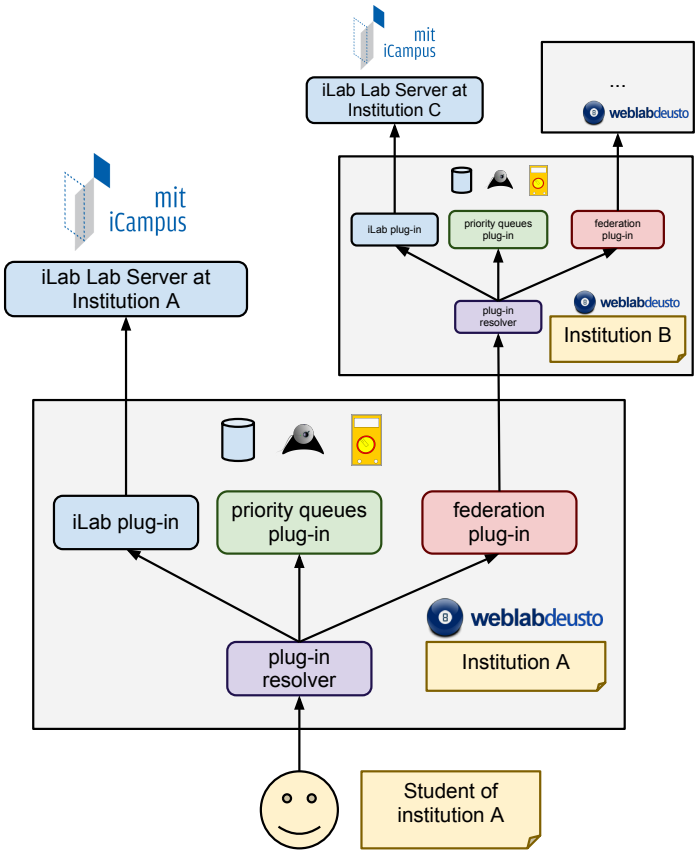


Figure 4.31: Architectural overview of the iLab on WebLab-Deusto integration

However, the system is experimental since only a subset of request types has been implemented. In particular, only those request types required to validate, submit, wait and retrieve the results. Those request types required to store user information in their session have not been implemented. For instance, in the iLab Shared Architecture it is possible that a student stores the results retrieved for a future use. Since WebLab-Deusto lacks of this interesting feature, it could not be implemented in the bridge.

4.3.2.3 **Benefits of Interoperability**

This section has described two bridges between WebLab-Deusto and the iLab Shared Architecture. While this work might not be directly extrapolated to other systems, it draws the potential advantages, drawbacks and the motivation for exploring this area. These bridges show that these solutions can automatically federate laboratories that might not fit in one of the two systems or that would require a notable amount of work.

In both bridges, the major advantage is that students already used to one system can consume laboratories of the other using the solution they know and for which they already have credentials. Students and educators of one institution who are using one system with the laboratories developed on it for different classes can start using laboratories developed in the other, which will appear in the same menus.

In the case of WebLab-Deusto being consumed by the iLab Shared Architecture, it enables the use of WebLab-Deusto using the iLab federation model. This means that if an iLab Service Broker has a number of WebLab-Deusto laboratories included, it can share these to other iLab Service Broker. For instance, the Service Broker at the University of Deusto can share WebLab-Deusto laboratories through the iLab system to the iLab-Europe Service Broker[1], so users there will automatically be able to consume those laboratories.

Additionally, the iLab Shared Architecture benefits from three features provided by WebLab-Deusto:

- Consuming other federation model, so users of an iLab Service Broker bound to a WebLab-Deusto instance will be able to use laboratories of other WebLab-Deusto instance if the two WebLab-Deusto instances are federated.
- Supporting queue based interactive laboratories.
- Load balancing of laboratories among different copies –i.e., if there are two copies of one WebLab-Deusto laboratory, it will manage the queues so the load of users will be balanced in a transparent way for the iLab system–.

4.3.3 **Ubiquitous Laboratories**

Transitive federation enables administrators to deploy the laboratories in different locations. One laboratory can be deployed outside the university "hosting" it, and it still can provide it to its students. This was previously explained in Section 4.2.3.1. However, the potential of this feature is more widely extended here.

WebLab-Deusto has been developed using Python, which is available in a wide number of operating systems and architectures. The memory consumption of the

[1] http://www.ilab-europe.net/

application is considerably small. It does not rely on MySQL (being able to work with the SQLite engine, which is used in Android and Nokia phones, as well as in several web browsers) neither on Redis (being able to use also SQLite for coordination at the cost of efficiency). This way, WebLab-Deusto has been deployed in embedded devices such as the IGEPv2[1], which is an ARM device which has 512 MB RAM. WebLab-Deusto running for 1 month on it was consuming less than 14 MB.

This makes it possible to run WebLab-Deusto in low cost devices that can easily act as servers, and deploy them in other locations. As a proof of concept, a fishtank laboratory was created. In this laboratory, students can feed the fish, turn on and off the lights and even control a small radiofrequence submarine (when it is charged). The whole laboratory (including all the layers) was deployed in an IGEPv2 device. Users of WebLab-Deusto can use this laboratory using the federation protocol. Given the transitive property of the federation model integrated in WebLab-Deusto, it is possible to share this laboratory with other institutions such as secondary schools (see Figure 4.32). The most important part here is that this low cost solution could be located somewhere else.

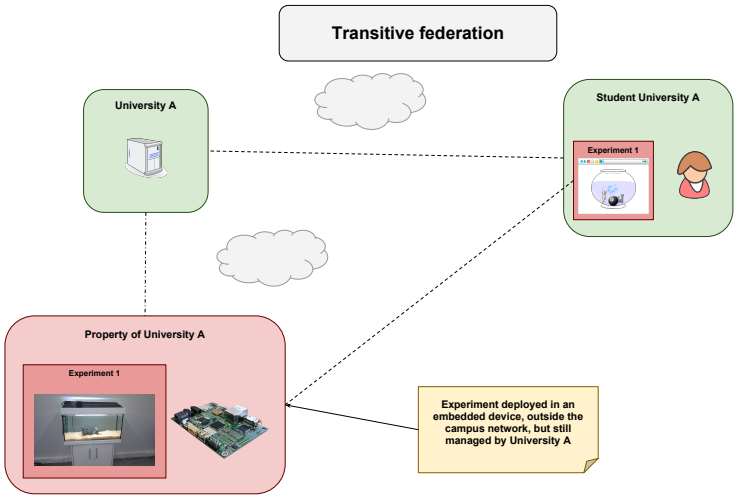


Figure 4.32: Fishtank laboratory deployed outside the University but transitively being accessed by its students

In Section 4.2.3.1, a scenario where a farm school could share a laboratory to the local university and this one could even share it with other university abroad that could also reshare it to a local secondary school. This type of situations is

[1] http://igep.es/

especially affordable when, as just described, the local secondary school can use a low-cost server to serve the laboratory, and still is able to serve a big amount of institutions abroad through a federation chain.

4.4 Qualitative Evaluation

A qualitative evaluation to summarize the features explained before is presented in this section. Chapter 5 evaluates both WebLab-Deusto and the implementation of the federation model presented in this chapter, from a quantitative perspective.

The federation model presented in this chapter supports transitivity and federated load balance for all the WebLab-Deusto laboratories (batch or interactive). Since it relies on the existing scheduling mechanisms of WebLab-Deusto, it only supports queueing (and not booking). In Table 4.6, this is explained in more detail, comparing it with the federation models available in the literature.

First, WebLab-Deusto, MIT iLabs and Labshare Sahara supports federation, as explained in Section 2.4.4. All of them support interactive laboratories, but WebLab-Deusto and Labshare Sahara support queueing (more appropriate for short sessions), and MIT iLabs and Labshare Sahara support calendar-based booking.

Second, WebLab-Deusto and MIT iLabs support batch laboratories, and both support them through queues. No laboratory supports batch laboratories through booking, since it is not the case in most laboratories in the literature. It might only make sense if students want the laboratory to happen at a certain moment in the future, regardless they are online or not, such as if it depends on external conditions such as weather, time of the day in a telescope or so.

From the scheduling point of view, at the time of this writing only Labshare Sahara is able to support both queuing and booking at the same time. This policy can be overridden in this system, enabling only booking or only queueing.

Finally, only WebLab-Deusto supports both transitive federation and federated load balance, which are the key features of this dissertation.

4.5 Conclusions

This chapter described the federation model of WebLab-Deusto. The system is implemented as a scheduling plug-in in the overall system. This is so since, as detailed in Section 4.1, conceptually the federation requirements are the same as the scheduling mechanism requirements. Both concepts enable to grant the exclusivity on a set of resources to a set of users.

As explained in Section 4.2, the major novelty of this federation model is that it introduces transitivity and federated load balance. Transitivity refers to the fact that when one entity (e.g., University A) shares a particular laboratory with another

Table 4.6: Summary of the federation model

	WebLab-Deusto	MIT iLabs	Labshare Sahara
Federation	✓	✓	✓
Queueing on interactive	✓	✗	✓
Booking on interactive	✗	✓	✓
Queueing on batch	✓	✓	✗
Booking on batch	✗	✗	✗
Mixed booking and queuing	✗	✗	✓
Local load balancing	✓	✗	✓
Transitive federation	✓	✗	✗
Federated load balance	✓	✗	✗

entity (e.g., University B), then this entity can re-share the laboratory with a third entity (e.g., University C, or Secondary School A). This simple concept enables a higher distribution of laboratories, crossing cultural or language boundaries, and reduces the constraints in the sharing of laboratories. While there is a certain consensus in the remote laboratories community that states that providers are willing to share their laboratories and that they are frequently underused, this feature had not been directly supported. The second feature that is supported in this federation model is federated load balance. If two or more entities have an exact copy of a laboratory, with this federation model they can share the queue of users among them in an automatic way. VISIR is a clear example of how different institutions have bought the same copy of the same laboratory and could benefit from this property. Low cost solutions might also benefit from this.

Altogether, both properties enable to scale up the potential number of supported concurrent remote laboratory users to numbers not achieved before in the literature, keeping it sustainable. The measurements of this approach are detailed in Chapter 5. Additionally, these results open multiple research lines, some of them explored in this dissertation. Examples of these are explained in Section 4.3.

In particular, Section 4.3.1 explores how the integration of remote laboratories in Learning Management Systems (LMS) and Content Management Systems (CMS) can be achieved though federation, especially if the remote laboratory supports transitivity. Federation, and a pluggable scheduling system, also enables a higher degree of interoperability, as explained in Section 4.3.2 with the iLab Shared Architecture. Finally, the federation model can be used to share and re-share laboratories located in multiple remote locations, as detailed in Section 4.3.3.

To sum up, this chapter explains the theoretical core of this dissertation. A qualitative evaluation is presented in Section 4.4, comparing it with the literature

studied in Chapter 2. The next chapter studies a quantitative evaluation of the system.

The Federation has gone too far this time.

Queen Amidala

CHAPTER

5

Evaluation

THIS chapter is focused on demonstrating to what extent the federation model proposed in this dissertation scales, and at the cost of what amount of time. Therefore this chapter takes the implementation described in chapter Chapter 3 and the federation model described in chapter Chapter 4, and measures how the system behaves under different conditions with different loads of users.

In order to do this, a student simulator has been developed, which is described in Section 5.1. Given that different deployments will fit better in different systems, Section 5.2 shows the drawbacks and advantages of each configuration in a non federated environment. Once this is analyzed, the configuration with the best results is selected and in Section 5.3, the different types of federations are compared with different load of users. Finally, Section 5.4 shows how the system behaves in constrained devices.

This chapter does not contain a qualitative analysis of how this federation model compared with other federation models. This has been previously addressed in Section 4.4.

5.1 Scenario Overview

This chapter has been measured with the simulator called WebLab-Bot (Section 5.1.2). This system basically runs a particular WebLab-Deusto configuration, simulates a set of users, measuring the time of each operation, and then it stops it and generates the figures presented in this chapter. The Bot repeats this

procedure for different sets of users to see the overall trend. The reference model behind the simulation is explained in Section 5.1.1.

The assumption behind the WebLab-Bot is that measuring the response time of the system to the client requests under different situations, it is possible to measure the stress of the system and what is going to be the behavior perceived by students in the analyzed situations. The validation of this assumption is detailed in Section 5.1.3.

5.1.1 Reference Model

When students use WebLab-Deusto, they're using their web browser to load a web application. This means that their web browser downloads a set of HTML, CSS and JavaScript files and executes them. In conventional web applications, users download the HTML code of the current view, with a set of links to other parts of the web application. When they click on these links, the whole new view is downloaded and processed, while the previous one is unloaded. For example, in a typical newspaper web page, users download the whole HTML of the main page, and whenever they click on a section or a new item, they will download the whole HTML code of that particular new item (including the same headers and similar items). A set of caching techniques are usually deployed to make this more efficient.

As opposed to this approach, WebLab-Deusto uses AJAX extensively. When the user goes to the WebLab-Deusto web page, the WebLab-Deusto application is downloaded in the web browser –as long as it was not previously cached–, and it starts running. From this point, there are two types of requests: those calling the server performing requests using a web service, and those downloading other static parts of the client. For instance, when the students attempt to log in, the client is using a public web service of WebLab-Deusto with an underlying HTTP call. If the user is successfully authenticated, the web application will download the static HTML code that is required for the next screen, and will use other web services to retrieve the laboratories that this particular student can access. When student selects a laboratory, the client downloads the static HTML code of that laboratory. Finally, when the user clicks on "Reserve", the client will start performing other HTTP calls to interact with the server, manage the queue and finally access to the real equipment.

To sum up, the WebLab-Deusto server does not generate any presentation code (e.g., HTML or CSS). Students download an AJAX application which automatically downloads the missing HTML code as requested, and which calls a public API through web services. Given that most of the client is cached, the main stress is generated when students are interacting with the system using these web services.

The typical cycle is: students log in, list the available experiments, and reserve one. Then students are retrieving the status of the reservation until it is assigned.

Once assigned, students send commands and files as they use the laboratory. These commands and files depend completely on the logic of the particular laboratory. Finally, they finish the reservation and log out. A public web service for each of these tasks is accessed by the client.

5.1.2 WebLab-Bot

WebLab-Bot simulates different sets of concurrent students. Each simulated student will do what a student typically does in WebLab-Deusto in a generic laboratory. As shown in Figure 5.1 and explained above, students first log in and list the available laboratories, then request a laboratory reservation and wait until the server states that a laboratory has been assigned to this user, retrieving the position in the queue from time to time (typically between 1 and 10 seconds, depending on the position retrieved). Once a laboratory is assigned, the student submits a command and a file to the laboratory, obtaining a response. Finally, the student finishes the reservation (so other student can be assigned) and logs out the system.

Each of these operations is performed through a set of web services based commands, using JSON over HTTP. The real web client used by real students calls the same exact operations using this HTTP interface, so basically WebLab-Bot simulates the whole user web browser by calling these methods, and measures the time required by each of these methods individually.

5.1.3 Assumption Validation

The main differences between the simulator and a real environment:

- As later explained in Section 5.1.4, real students as synchronously coordinated as the simulator. It is very difficult that 150 students click on the reserve button at nearly the same time. Therefore, they do not add the same amount of stress to the server as the simulator. This is however intended so as to be able to measure the system behavior under such stress.

- The client is completely different, and it may add higher or lower latency. In the real environment, the client is a web browser using the AJAX web application, while in the WebLab-Bot the client is a simulator (which is implemented as a Python application). Therefore, depending on the web browser it could be faster or slower than the real client. In the measurements presented in Table 5.1, the popular Google Chrome web browser is compared to the WebLab-Bot system in a single execution, not being especially slower or faster.

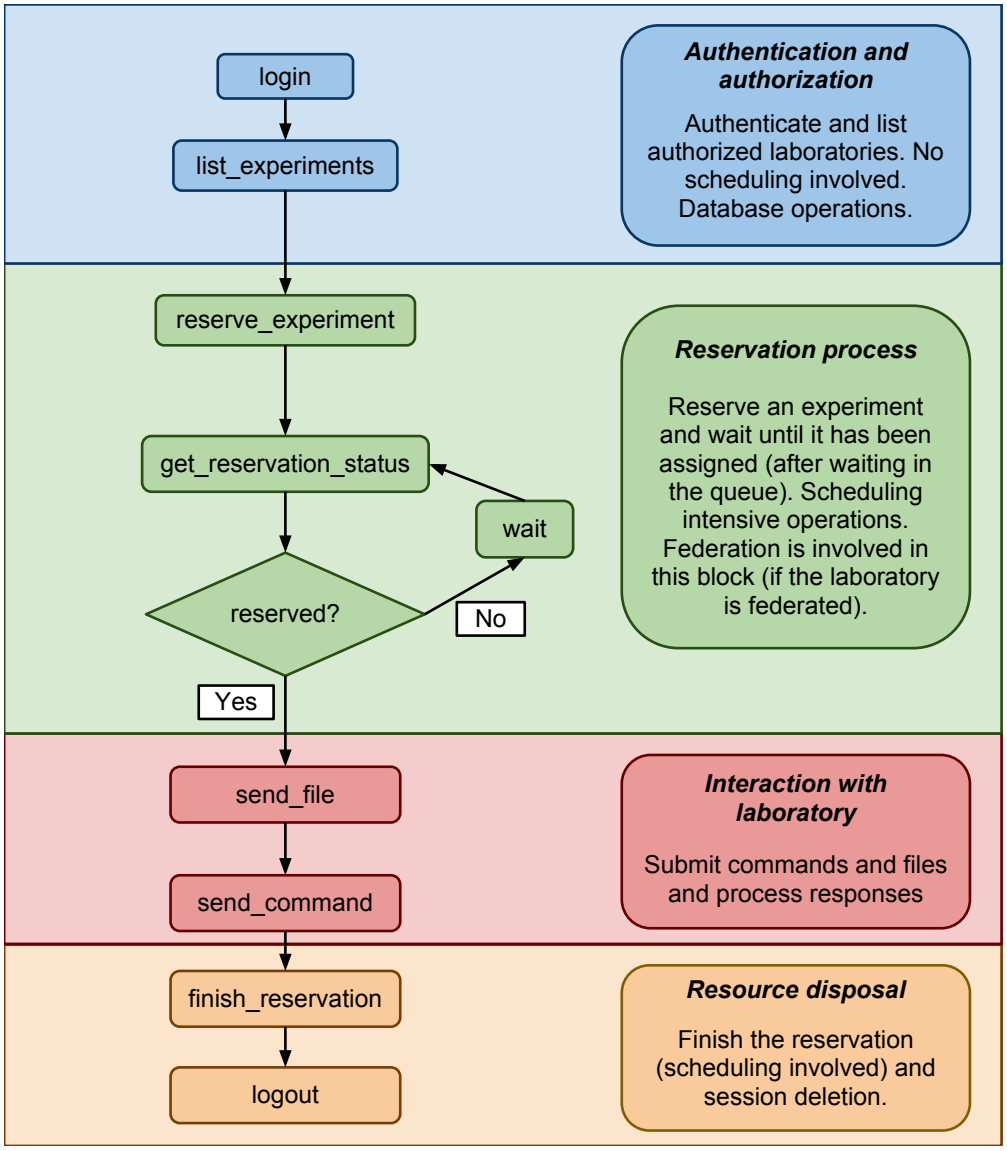


Figure 5.1: WebLab-Bot simulated user life cycle. Real users submit more commands and files, but the bot does not stress this situation.

- In most of the executions (all except for the described in Section 5.3.5 and Section 5.4), the Bot will be run in the same machine of the system under test. The first objective is being able to restart it after each iteration, and therefore having a completely new environment each time. The second objective is avoiding that temporary network issues may affect the measurements. However, this has an impact on the computing measurements of the machine, since some time slots are dedicated to the Bot itself. That said, this impact is not relevant since the Bot, during each execution, does not use the disk neither any external resource (such as the database or Redis) except for the WebLab-Deusto deployment under test, and most of the time is not computing anything but waiting for the system to reply the requests submitted.

- Given that the system under test is in the same machine as the WebLab-Bot, all the connections are local, and therefore, much faster than using a real environment. In Section 5.3.5, a direct Gigabit Ethernet connection is used, and in Section 5.4, the device used has a 10/100 Ethernet card, and a direct cable is used to the machine where the WebLab-Bot is deployed. This avoids that network issues may affect the measurements, but it must be taken into account that, depending on the network, the measurements would be different.

For these reasons, WebLab-Bot is not measuring what the student's client will exactly see, since this depends on a series of external variables (network, web browser, operating system), but how the server will behave. If in a particular situation studied in this chapter, the server takes 1.5 seconds, a real student in the exact same situation (which, once again, is very difficult that randomly students are so coordinated) would not perceive the same value, but a slightly different -probably higher due to the network- one.

In Table 5.2, each of these methods is analyzed, presenting if the operation performed in the server uses the regular database, the scheduling system and if federation may be involved. As it will be later seen with the results, depending on the database backend, certain operations will take longer, and especially depending on the scheduling system backend, both the methods related with the scheduling system and the methods related to the federation system (which is indeed implemented as a scheduling plug-in in the scheduling system) will present different results. Therefore, whenever a method is slow, this table makes it possible to classify what external system (e.g., the federation layers, the scheduling system or the database) is potentially acting as a bottleneck.

Table 5.1: Measurements of a single, random execution with the WebLab-Bot client and the Google Chrome (version 23.0.1271.97 under Ubuntu 12.04) web browser, using a single local node with SQLite. This is not intended to provide an accurate comparison (since it will depend on the web browser and the operating system) but to show that the WebLab-Bot client is not very different to the real client

Method	Google Chrome	WebLab-Bot
`login`	**30 ms**	40 ms
`list_experiments`	**61 ms**	62 ms
`reserve_experiment`	1220 ms	**933 ms**
`get_reservation_status`	258 ms	**218 ms**
`send_command`	23 ms	**22 ms**
`send_file`	**29 ms**	66 ms
`finish_reservation`	**812 ms**	911 ms
`logout`	68 ms	**33 ms**

Table 5.2: Methods used by the student simulator and impact on the database, scheduling system and federation system

Method name	Federation involved*	Scheduling involved	Database involved
`login`	No	No	**Yes**
`list_experiments`	No	No	**Yes**
`reserve_experiment`	**Yes**	**Yes**	**Yes**
`get_reservation_status`	**Yes**	**Yes**	No
`send_command`	No	No	No**
`send_file`	No	No	No**
`finish_reservation`	No***	**Yes**	No**
`logout`	No	No	No

* Only if working on a federated environment

**Information to be stored in a database is temporarily stored in a memory queue. A single thread takes sets of this queue and stores the available information in the database in fewer commits. Therefore, these methods have an impact on the database while they cant not be directly measured by the latency of each method, but by the general time required by the whole system.

*** This method is called in the final system directly, so it only involved local scheduling.

5.1.4 Notes on Concurrency and Stress

In this chapter, a variable number of *concurrent students* will be simulated (in most cases, from 1 to 150 students, in steps of 5). As it was already addressed in Section 2.2.2.1 and shown in Figure 2.15, the number of concurrent students is very different to the number of enrolled students. The shorter the laboratory session time is, the less concurrent users there will be. Also, the more copies of the laboratory there are, the quicker they will advance and the shorter the queue will be.

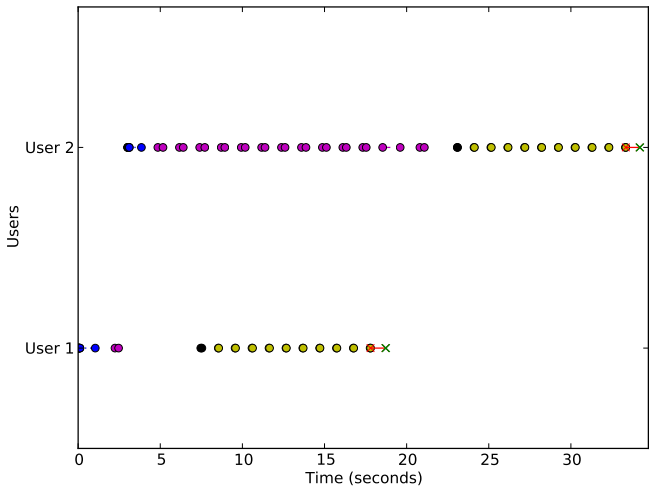


Figure 5.2: Simple concurrent use case (not considered in this chapter). Blue points represent the reservation process, the purple ones the requests for reservation status and the yellow ones the commands submitted to the laboratory. As there are big delays between the requests of one user and the other, each method finishes very quickly

However, in that section *concurrent users* were referred as those students who attempt to use the laboratory while other student is using it. For instance, if one laboratory session takes 2 minutes, and while one student is using it, other attempts to use it, then they would have been considered 2 concurrent students. This is represented in Figure 5.2: User 1 starts using the system, and User 2 comes 3 seconds later. Since User 1 already logged in, listed the available laboratories and reserved the laboratory in those 3 seconds, User 2 will do it while the User 1 is already interacting with the laboratory. Once User 1 finishes the reservation (around second 18), User 2 starts using it. Since almost no request of both users is actually concurrent and all the resources are available to each request instead of being shared, the time per request is very small. However, they would usually be called *concurrent*

students, since both have their web browser open using the system and they are performing calls to it.

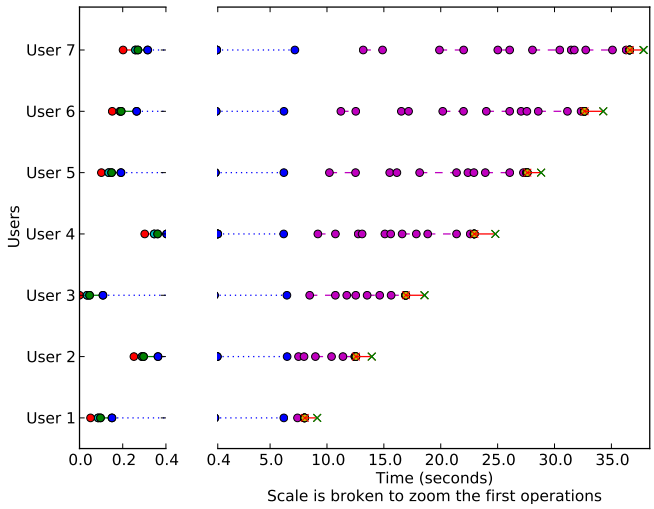


Figure 5.3: Concurrent use case as considered in this chapter. Blue lines represent the reservation process, purple lines the requests for reservation status. Reservation time is much longer since all the requests are concurrent on a single monothread SQLite database.

In this chapter, though, *concurrent students* will be referred as a far more stressful concept, which is the number of concurrent students that log in and perform a reservation in a very small window of time (see Figure 5.3). This stressful situation does not resemble reality: it is really difficult that 150 students will click on the *reserve* button in less than one minute. In order to mitigate this effect, the simulator submits one student every 0.05 seconds. In the end, 150 students will take 7.5 seconds to be submitted, which is still a smaller window than a regular one. If this number of seconds was increased, the level of stress to the system would decrease, and therefore, the seconds that operations need. For instance, if one operation requires 0.1 second, and 150 students submit it, the load for the server is going to keep increasing. If students submitted it every 0.2 seconds, the server would have enough time to process it before the next student came and therefore the figure would represent a horizontal line. This is shown in Figure 5.4, where simulated students are introduced every 5 seconds, and therefore each one finishes before the next one starts and the time of each request is independent of the number of students.

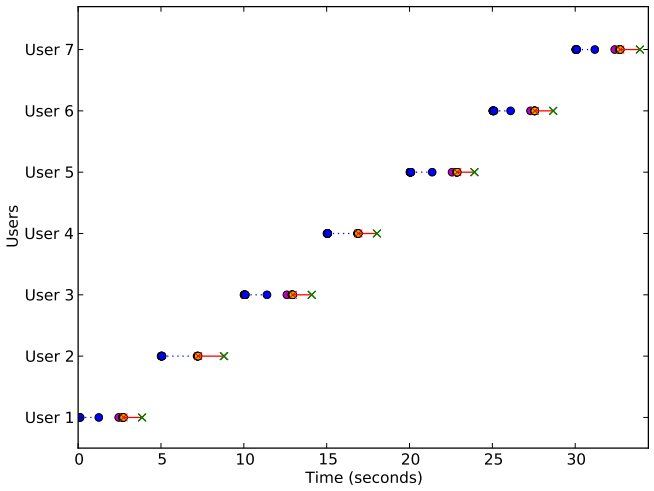


Figure 5.4: Sequential case (not considered in this chapter). Blue lines are the reservation process. As in Figure 5.2, even having 150 students in this way does not stress the system (since each previous student has already finished before the next one enters).

5.1.5 Comparison Framework

The reason for using such stressful situations is that it helps to create a comparison framework on top of which different configuration values can be measured, using the real system. This lets us to measure the *trends* that could be defined based on the theory. It is clear that using two different backends for the scheduling system will have different results, and it may be clear which one is faster. But this comparison framework let us define how faster under stressful situations.

Other approaches, such as taking from the database real user sessions (at the time of this writing, there are 576,414 commands by real students stored in the production database since it February 2009, as well as commands stored in log files since 2005, and it is possible to simulate when exactly they have interacted) and reproducing them has been discarded precisely due to the lack of stress that this produces on the system.

To measure this stress, the most relevant commands must be selected. In order to analyze the scheduling system (and the federation system), the relevant methods are `reserve_experiment` and `get_reservation_status`, attending to Table 5.2. However, it must be taken into account that while the former is called by all the concurrent users at the same time, the latter is being called periodically and therefore with time it is called less frequently and therefore it will take less time to process, decreasing the mean and increasing the deviation. Additionally,

to measure the latency in the interaction, both `send_command` and `send_file` would be equally relevant. However, given than the former is used more often in production, it has been selected.

Regarding the `reserve_experiment` command, other design options –such as queuing the reservation requests internally and returning a response such as "*submitted, not processed yet*"– have been discarded to properly measure the system, while they should be considered in highly concurrent situations.

In order to take valid measurements, the simulator runs each configuration with each number of simulated students several times, called *iterations*. In this chapter, 5 iterations have been used. This causes that simulating a particular configuration may take between 8 and 16 hours. In the graphs, three lines will be drawn: a red line (on the top) representing the maximum time taken (in the 5 iterations), a blue line (on the bottom) representing the minimum time taken (again, in the 5 iterations), and the green line (in the medium) showing the mean time. The standard deviation is represented by a green bar crossing the mean, which goes from $\mu - \sigma$ to $\mu + \sigma$. This is shown in Figure 5.5.

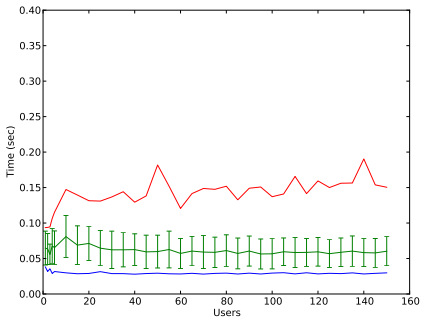


Figure 5.5: WebLab-Bot sample measurement. The green line (on the middle) represents the mean, while the vertical lines crossing it represent the standard deviation. The maximum and minimum values are represented by the red (on the top) and the blue (on the bottom) lines.

During this chapter, the scale of the graph changes from comparison to comparison, but it is the same within a comparison. For instance, when comparing two fast operations in a server, they will have the same scale (e.g., from zero to 1 second), and when comparing two constrained devices, it will also be the same (e.g., from zero to 20 seconds), but it will not be the same between both comparisons.

Finally, this chapter has not compared WebLab-Deusto with other Remote Laboratory Management System. The reason for this is not the lack of a common comparison framework, but the impossibility of developing such framework. The

iLab Shared Architecture uses calendar based booking in the interactive laboratories, so it is simply not possible to use the same flow presented here. Additionally, this chapter is focused on measuring the transitivity property and the federated load balance, which are features not present in other systems. For a comparison with other systems, refer to the qualitative comparison described in Section 4.4.

5.2 Non-federated Environments

This section focuses on the evaluation of WebLab-Deusto with a single provider. Therefore, there is no federation in this section, but it is still useful to for selecting the proper configuration and for being able to make comparisons in Section 5.3. First, those aspects which will be analyzed are presented, then each particular aspect is presented and finally a global summary is provided.

5.2.1 Aspects to Analyze in Non-federated Environments

It is important to show that there are different aspects which have an impact on the overall performance:

- Protocol used (Section 5.2.2). WebLab-Deusto supports a RPC system based on JSON, as well as XML-RPC and SOAP. JSON is the one used by the regular client, while the XML-RPC protocol has been used from Second Life in the past and SOAP is used by third party systems. The advantage of the JSON version is that it is faster than the others, especially in the client side, where web browsers come with a quick JSON native parser.

- Database engine (Section 5.2.3). WebLab-Deusto can use different relational database engines. MySQL and SQLite are the favorite ones, and they have an impact on the performance and restrictions.

- Scheduling backend (Section 5.2.4). WebLab-Deusto supports two main scheduling backend: first, a solution implemented on Redis, which is a fast key-value store. Second, an SQL based implementation, which will be able to be used in SQLite (with a single process) or in MySQL (which supports different processes). Depending on the implementation used, the amount of time differs considerably, and not all the solutions are available for all the situations.

- Session management (Section 5.2.5). WebLab-Deusto can store the user sessions in a relational database (which is the slowest solution, but it supports

fault tolerance), in a Redis database (which is faster than the relational database, but it requires Redis) and in memory (which is the fastest solution, but it does not support fault tolerance).

- Number of copies of experiments (Section 5.2.6). The more copies of laboratories are available, the more latency will be added. This is due to two reasons: the first one is that the system is more stressed when more devices are being used, since more scheduling resources are required. The second one is that students in the queue will contact more often the closest they are to be assigned an experiment. If there are 100 users waiting in a queue, the first 5 positions will be requesting the reservation information every few seconds, while the latest will be requesting this information less often. If there are 50 copies of the experiment, more students will be closer to use the experiment and therefore more stress will be placed.

- Number of processes (Section 5.2.7). WebLab-Deusto is written in Python. Python has some limitations on thread management due to an internal lock called Global Interpreter Lock (GIL). While this discussion is outside the scope of this document, the summary is that in Python, 4 different threads will not run in different processors even if the machine where is running has them, since Python internally locks different threads when they are being executed. This is not the case whenever a thread is waiting for something (which, in WebLab-Deusto, is a considerable amount of the time), such as waiting for a network socket or a database. However, if 4 processes are run, the performance will likely increase considerably.

All these aspects are described in detail in the following subsections, focusing on trends. All these tests have been implemented on a Dell poweredge R410 Intel Xeon CPU E5606 with 4 core processors at 2.13GHz, 8 GB RAM, Ubuntu 12.04 and Python 2.7.3.

5.2.2 Protocols

Back in the beginning of WebLab-Deusto 3.0 (2007), it only supported SOAP as an external protocol. The web client generated SOAP requests and processed SOAP responses. In 2009, to support Second Life, the XML-RPC protocol was also supported, providing the same public API. At the same time, a custom, simple protocol based on HTTP and JSON was implemented. JSON (JavaScript Object Notation) is a serialization format based on JavaScript which is natively implemented in web browsers, being much faster to process than XML, and which has also been implemented with native libraries in other languages (such as Python). Therefore, the

WebLab-Deusto client dropped support for SOAP and implemented the JSON over HTTP version.

As seen on Table 5.3 and summarized on Table 5.4, the three implementations are essentially similar in performance terms, but the fact that the web client parses natively JSON makes it potentially faster in execution and in any case faster to implement (the Google Web Toolkit framework, on top of which WebLab-Deusto is implemented, supports it natively), so this is the protocol used in all the rest of the chapter.

5.2.3 Database Engine

WebLab-Deusto relies on sqlalchemy[1], which is an Object-Relational Mapping (ORM). This means that it permits the programmer to write code in Python as if it was accessing regular objects and internally it converts all the requests to SQL code, using extensions of each database engine. This way, applications developed on top of sqlalchemy are independent of the database engine, supporting in theory those database engines available in sqlalchemy. At the time of this writing, this includes Drizzle, Firebird, Informix, MaxDB, Microsoft Access, Microsoft SQL Server, MySQL, Oracle, PostgreSQL, SQLite, Sybase. However, WebLab-Deusto has only been tested with MySQL and SQLite, and these two engines are the only ones covered here.

In order to provide some background, SQLite is an open source cross-platform and lightweight database engine which stores the entire database in a single file (or in memory, if requested). It is very fast and light enough to be suitable for mobile phones and desktop applications. Indeed, it is distributed in Android phones (and it was available for Symbian phones), has been integrated in different web browsers such as Google Chrome, Firefox or Opera, and it is distributed by default with Python. On the other hand, it is mono-thread (so there cannot be concurrent requests) and it does not support multiple processes using it. Table 5.5 shows the performance of WebLab-Deusto with 1 to 150 students using SQLite for both the database and the scheduling, and using a single process and a single experiment.

MySQL, on the other hand, is a robust and popular open source database engine. It is cross-platform, but it requires an installation process, and it provides several security mechanisms common in database engines. There are two main Python libraries that enable the access to MySQL. The oldest one and still more popular is MySQLdb (sometimes referred as mysql-python). It is developed using the MySQL development library and therefore it requires being compiled for each platform (such as Windows 64 bits with Python 2.7 and libmysqlclient18), which becomes a problem for being maintained. The other one is PyMySQL, which is

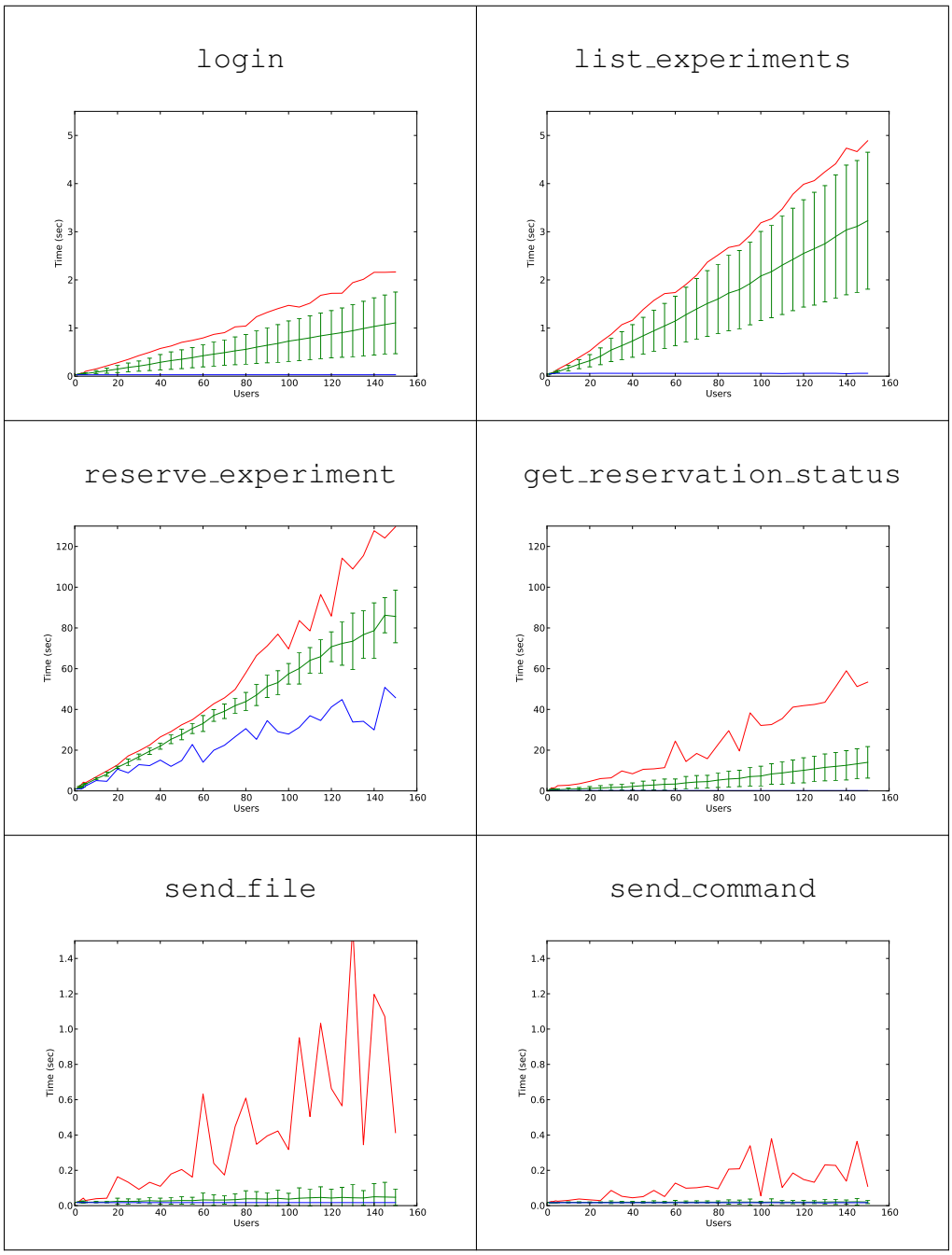
[1] http://www.sqlalchemy.org

Table 5.3: Different protocols, 4 processes with 1 copy of the laboratory using Redis for scheduling and MySQLdb. On the left column, the `reserve_method`; on the right column, the `send_command` method

Table 5.4: Numerical summary of Table 5.3

Protocols				
Users	Method	Protocol	μ (ms.)	σ (ms.)
1	send command	JSON	20	4
		XML-RPC	18	3
		SOAP	23	0
	reserve	JSON	64	23
		XML-RPC	63	15
		SOAP	67	17
60	send command	JSON	33	19
		XML-RPC	33	24
		SOAP	37	23
	reserve	JSON	57	21
		XML-RPC	58	22
		SOAP	77	26
150	send command	JSON	77	68
		XML-RPC	81	57
		SOAP	88	62
	reserve	JSON	60	20
		XML-RPC	62	27
		SOAP	90	41

Table 5.5: Results with SQLite for scheduling and database with 1 core and 1 experiment. The vertical scale has been changed given the big difference among methods types (interaction methods are not affected by the database, reservation methods are affected by the scheduling database, `login` and `list_experiment` methods are affected by the regular database)



written in pure Python, so it is very easy to distribute, and it works even on Python 3. However, the performance of this library is quite poor when compared with MySQLdb. Table 5.6 shows the performance using MySQLdb with 1 process. As it can be appreciated (the scale is the same as in Figure 5.5), it is slightly slower than SQLite. However, as opposed to SQLite, the load of users can be balanced among different processes that still connect to the same database engine.

In production, it is recommended to use MySQL rather than SQLite (since it supports using more than one process) and MySQLdb rather than PyMySQL. But for development scenarios, SQLite is fine (and it does not require to install MySQL), and in those platforms where it is difficult to find or compile MySQLdb, it is fine to use PyMySQL. Indeed, several processes running PyMySQL will be scale better than one process running SQLite. In order to compare this fact, Table 5.7 shows the three engines with one and 5 processes in the same scale and the `reserve_experiment` method, which is the one that suffers for the stress of the higher amount of students.

A subset of these measurements is presented numerically in Table 5.8

5.2.4 Scheduling Backend

As it is appreciated in the tables presented in the previous section, the impact on the latency of the interaction methods (such as *send_command*) does not increase considerably with higher loads of users as the reservation process.

For this reason, WebLab-Deusto provides two scheduling backends:

1. Based on Redis[1]. Redis is an open source key-value store which stores everything in memory and persists it into disk asynchronously after some time. However, in WebLab-Deusto, this persistence is disabled since it is not useful and it may affect performance.

2. Based on sqlalchemy. Once again, sqlalchemy can be mapped to SQLite or to MySQL. The measurements shown in Table 5.7 are all using sqlalchemy with the different engines.

So as to compare both backends, Table 5.9 (summarized in Table 5.10 compares the behavior of the main scheduling operations using Redis and using MySQLdb, with 1 and 5 processes. As it can be seen, Redis offers a higher throughput, especially when run in multiple processes.

The main problem with Redis is that while it has received patches by Microsoft to be supported in Microsoft Windows, at the time of this writing it is still only

[1] http://redis.io/

Table 5.6: Results with MySQL for scheduling and database with 1 core and 1 experiment

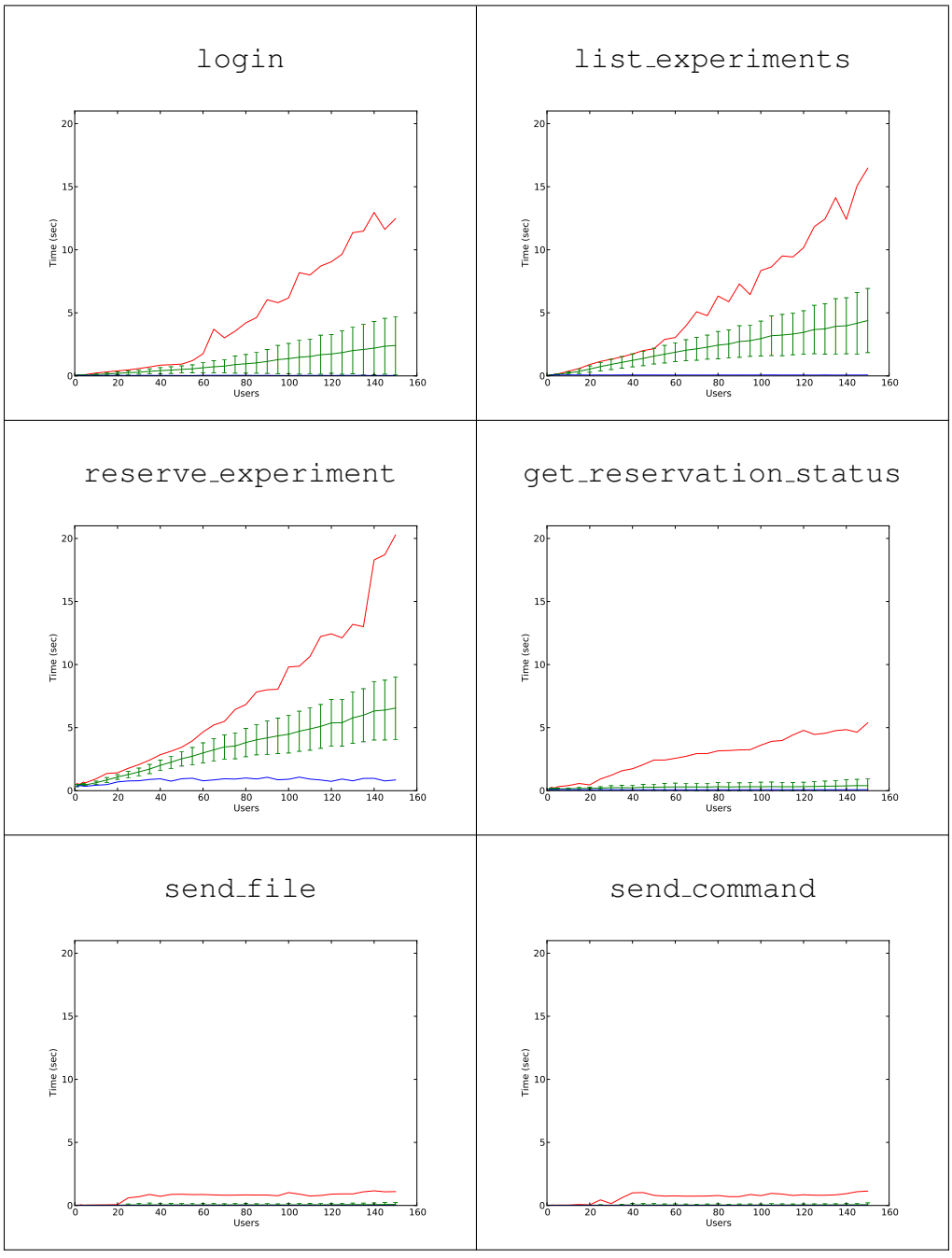


Table 5.7: Results of `reserve_experiment` with different database engines

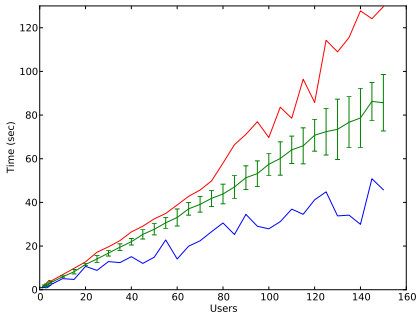
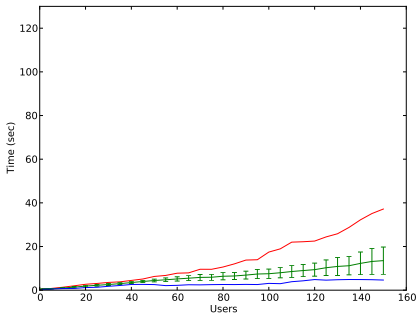
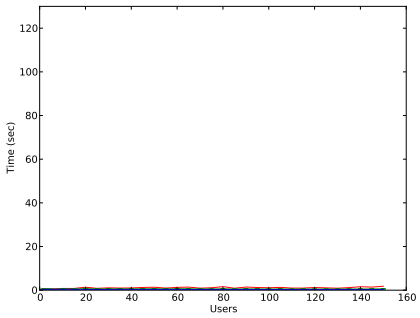
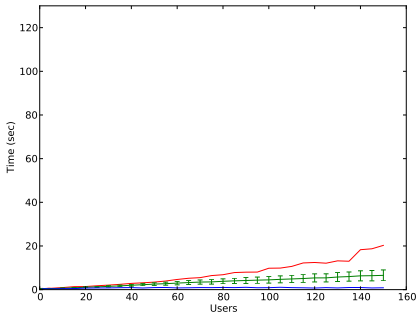
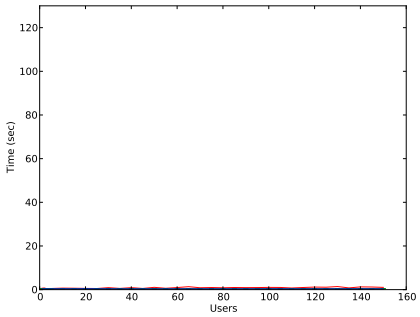
SQLite - 1 core	(SQLite is monothread)
	
PyMySQL - 1 core	PyMySQL - 5 core
	
MySQLdb - 1 core	MySQLdb - 5 core
	

Table 5.8: Numerical summary of Table 5.5, Table 5.6 and Table 5.7

Database engines					
Users	Method	Core Servers	Database and scheduling	μ (ms.)	σ (ms.)
1	send command	1	SQLite	17	1
			PyMySQL	18	1
			MySQLdb	17	0
		5	PyMySQL	19	4
			MySQLdb	20	4
	reserve	1	SQLite	933	40
			PyMySQL	465	126
			MySQLdb	402	88
		5	PyMySQL	442	55
			MySQLdb	458	135
60	send command	1	SQLite	19	9
			PyMySQL	50	152
			MySQLdb	36	90
		5	PyMySQL	20	12
			MySQLdb	19	5
	reserve	1	SQLite	33,088	3,908
			PyMySQL	5,152	976
			MySQLdb	2,992	801
		5	PyMySQL	651	177
			MySQLdb	412	96
150	send command	1	SQLite	20	8
			PyMySQL	350	486
			MySQLdb	61	146
		5	PyMySQL	23	14
			MySQLdb	21	8
	reserve	1	SQLite	85,652	12,923
			PyMySQL	13,533	6,225
			MySQLdb	6,547	2,476
		5	PyMySQL	589	208
			MySQLdb	457	114

Table 5.9: `reserve_experiment` method using Redis and MySQL with 1 and 5 processes

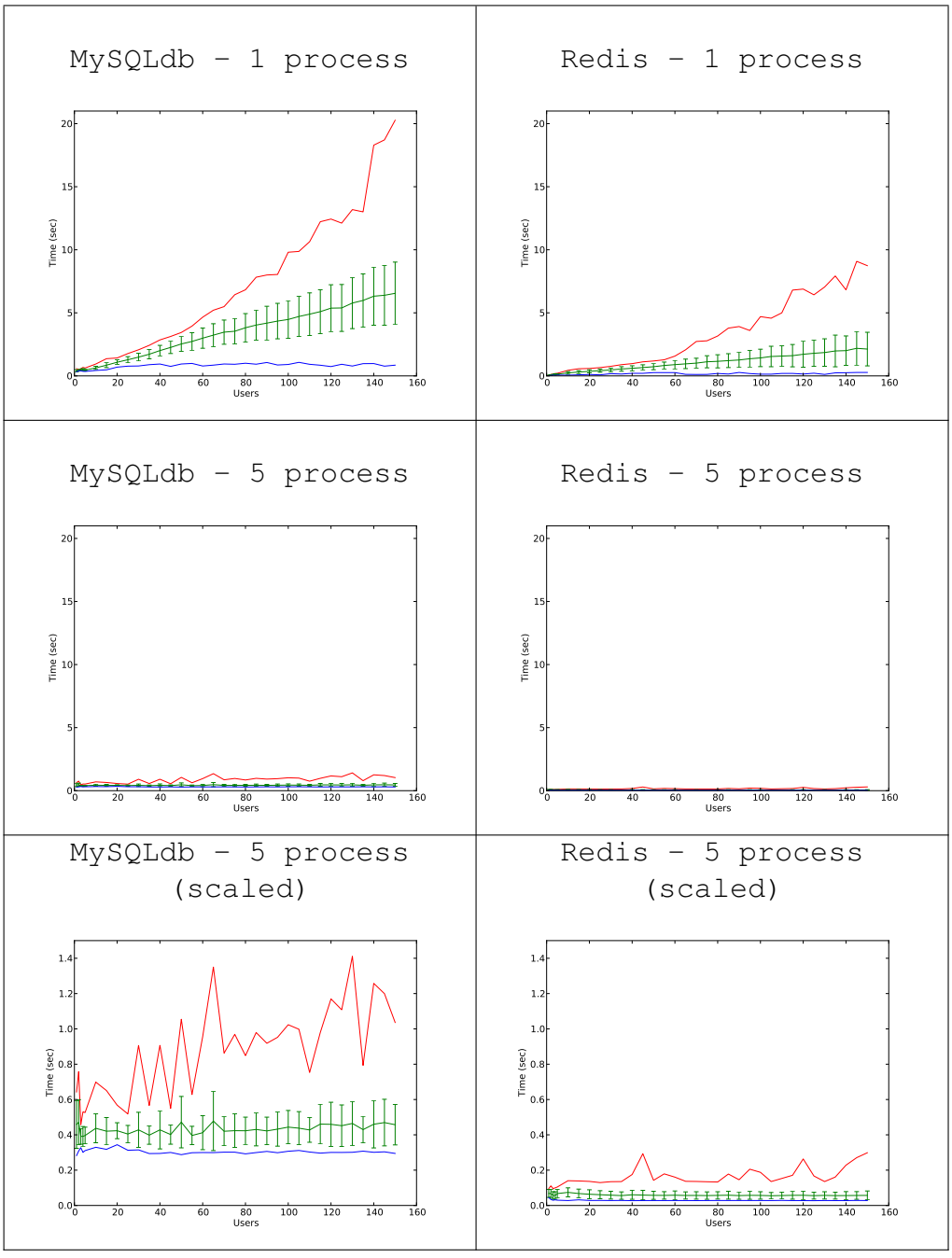


Table 5.10: Numerical summary of Table 5.9

Scheduling backends				
Users	Processes	Scheduling	μ (ms.)	σ (ms.)
1	1	MySQLdb	402	88
		Redis	44	11
	5	MySQLdb	458	135
		Redis	68	22
60	1	MySQLdb	2,992	801
		Redis	893	333
	5	MySQLdb	412	96
		Redis	58	24
150	1	MySQLdb	6,547	2,476
		Redis	2,126	1,332
	5	MySQLdb	457	114
		Redis	56	25

fully supported under UNIX systems. Therefore, while in UNIX the Redis implementation is highly recommended, in Microsoft Windows the SQL implementation of the scheduling backend is suggested.

5.2.5 Session Management

WebLab-Deusto services are stateful, since it needs to access the reservation status at any point. Sessions can be stored locally in the memory of each process or they can be stored in a database (Fowler, 2003). Therefore, three different backends were implemented:

- Memory storage → sessions are stored locally in the same process.
- Redis storage → sessions can be stored in a Redis database.
- sqlalchemy → sessions are stored in the database, in MySQL or SQLite.

The first backend (Memory storage) is the fastest, but it requires that all the requests coming from the same user go to the same process. If a request goes to other process, the session will not be there, so the request would fail. There are mainly two approaches to implement this:

- Using a *sticky* session identifier: the WebLab-Deusto server can establish a cookie to the client in the first request (in the log in process). From this

point, the client will send the request and the load balancer will check if it has the cookie. If it does, it will forward the request to the proper process. Otherwise, it will send to the less stressed process.

- Configuring the load balancer to check the client's IP address, apply some hashing algorithm and submit it to the proper process.

The problems with the first approach come when the client does not support cookies. This happens quite often in web services, or when being consumed by a Learning Management System or even by other WebLab-Deusto system with certain libraries. The main problem with the second approach is that in federated environments, if students are in a class they will typically go to the same process, given that they would have the same IP address. That is the main reason for adapting WebLab-Deusto to the first approach, while it still works with the second one.

The other two engines (Redis and sqlalchemy) are significantly slower, but they grant fault tolerance, since one process can be even rebooted safely, and the requests will go to other process which is aware of the session information.

5.2.6 Number of Copies of Experiments

WebLab-Deusto supports load balancing among different copies of the same experiment. For instance, it is possible to have one experiment identified by one name and internally have 2, 5 or 80 implementations of it, and redirect each user to an available implementation.

However, this has a negative impact on performance for several reasons. The management of 80 copies concurrently requires more processing, while having them on a queue is easier. Also, students will wait longer before requesting the position in the queue if their position is longer, and when there are many copies, more often they will call. Additionally, the stress generated is higher, and that is reflected in all the methods.

In order to prove this, different measurements have been taken with a single process using Redis for scheduling and MySQLdb as database, and different number of copies of the laboratory (1, 10, 20, 40, 60, 80). Table 5.11 shows the reservation process, while Table 5.12 shows the interaction with the equipment. Both tables are summarized in Table 5.13. As described, the higher the number of copies of laboratories is, the higher the latency is provided. However, as it is presented in the next section, this effect can be partially mitigated by adding more processes.

In any case, it must be remarked that while this feature adds certain latency, it permits that several students can be simultaneously using different copies of the same remote laboratory, therefore attending more users at the same time, increasing

Table 5.11: Results of the `reserve_experiment` method with MySQLdb, Redis, a single core and different numbers of copies

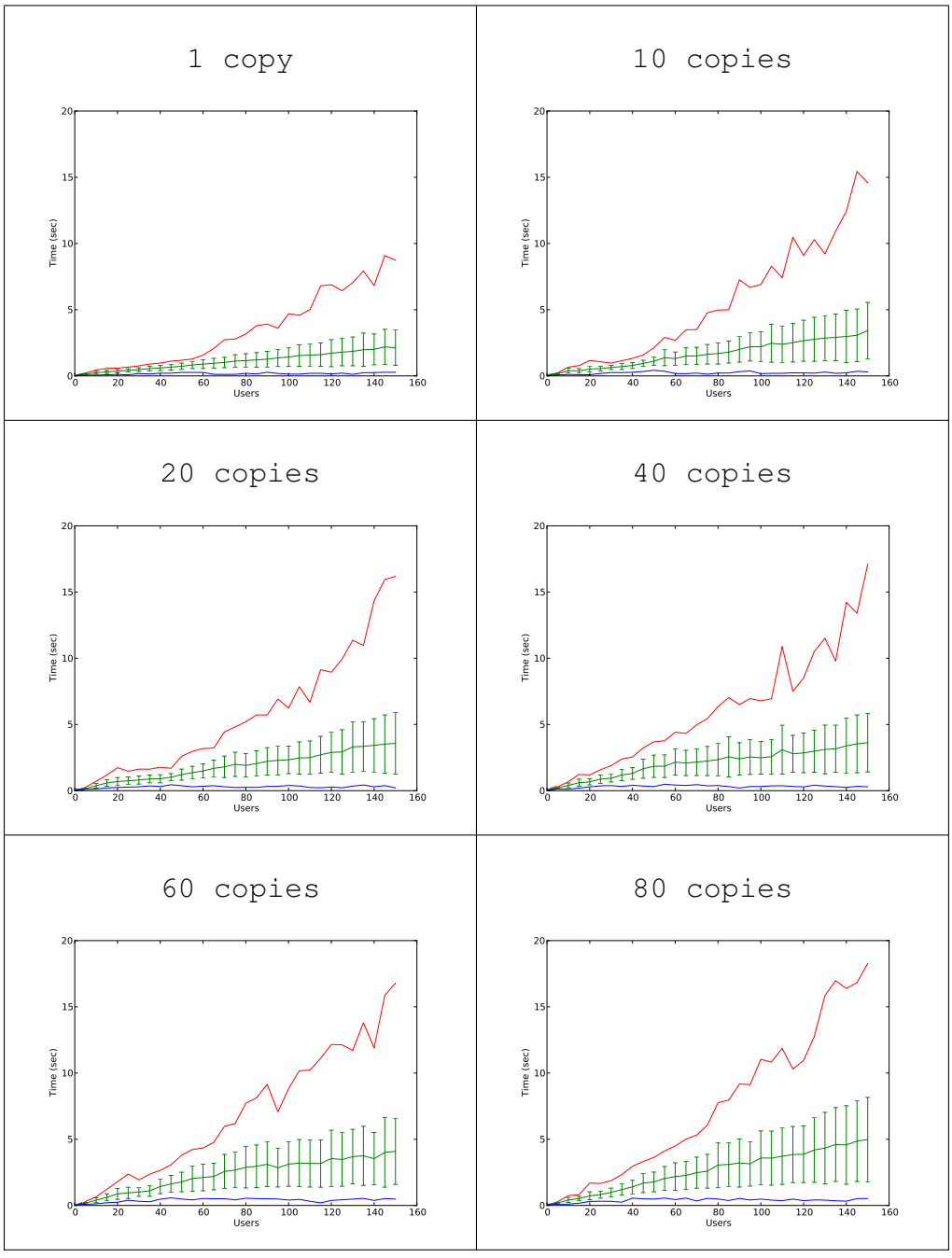


Table 5.12: Results of the `send_command` method with MySQLdb, Redis, a single core and different numbers of copies

Table 5.13: Numerical summary of Table 5.11 and Table 5.12

Number of copies of the laboratory				
Users	Method	Number of copies	μ (ms.)	σ (ms.)
1	send command	1	17	0
		10	20	3
		20	19	3
		40	18	1
		80	24	14
	reserve	1	44	11
		10	49	4
		20	81	26
		40	74	23
		80	95	2
60	send command	1	62	103
		10	353	334
		20	677	483
		40	1,156	255
		80	1,236	279
	reserve	1	893	333
		10	1,307	497
		20	1,489	534
		40	2,156	986
		80	2,181	1,034
150	send command	1	251	272
		10	1,197	422
		20	1,225	699
		40	1,772	1,035
		80	2,983	1,167
	reserve	1	2,126	1,332
		10	3,423	2,119
		20	3,572	2,315
		40	3,627	2,111
		80	4,977	3,181

the throughput and reducing the queue of users. So while the more copies added, the more stress is added to the system, the user experience and the potential of the remote laboratory are enhanced.

5.2.7 Number of Processes

As previously stated, Python has a Global Interpreter Lock (GIL) that makes the threading model not work as could be expected when coming from other programming languages. Internally in Python, when a thread is being run, it executes a number of instructions which can be configured and then it swaps the context and other thread is executed. Whenever there is an IO (input/output) operation, or some extension developed in C is called, the context will also be swapped. The problem is that during the execution of the pure Python not-IO operations, the GIL is locked, so no other operation can be executed in other processor.

Given the amount of IO in WebLab-Deusto (IO includes not only the requests from the client and the requests to the particular laboratory, but also the database and all the access to the Redis server), certain concurrence will often occur even with a single process. However, in order to take advantage of the nowadays common dual, quad or more core processors, it can be managed using multiple copies of Python processes instead of relying on the Python threading model.

For this reason, WebLab-Deusto has been designed so it can scale and multiple independent processes can be executed not only in different machines, but also in the same machine to mitigate this effect. This way, having 4 processes running WebLab-Deusto in a quad core machine will increase the throughput.

In Table 5.14, the `reserve_experiment` method is compared with different amount of core processes (1, 2, 3, 4, 5 and 20) and 80 copies of laboratories (since the performance improvement is not visible with 1 copy). As it is expected, increasing the number of core processes will enhance the performance (4 is much better than 1), but the enhancement does not increase with higher number of core processes. Table 5.15, shows the same situation with the `send_command` method, but with much lower values. Finally, Table 5.16 summarizes both tables.

5.2.8 Summary of the Non-federated Environments Evaluation

In order to evaluate WebLab-Deusto without federation, a set of aspects must be taken into account (Section 5.2.1). Three different protocols can be used by the client (Section 5.2.2), and JSON over HTTP is the recommended. Regarding the database, while SQLite works fine and comes by default with the Python distribution, installing MySQL is recommended (in particular, with the MySQLdb plug-in), since it provides better results and supports multiple processes Section 5.2.3. Two scheduling backends are supported: one based on SQL (which can rely on

Table 5.14: Results of the `reserve_experiment` method with MySQLdb, Redis, 80 copies of the laboratory and different number of processes

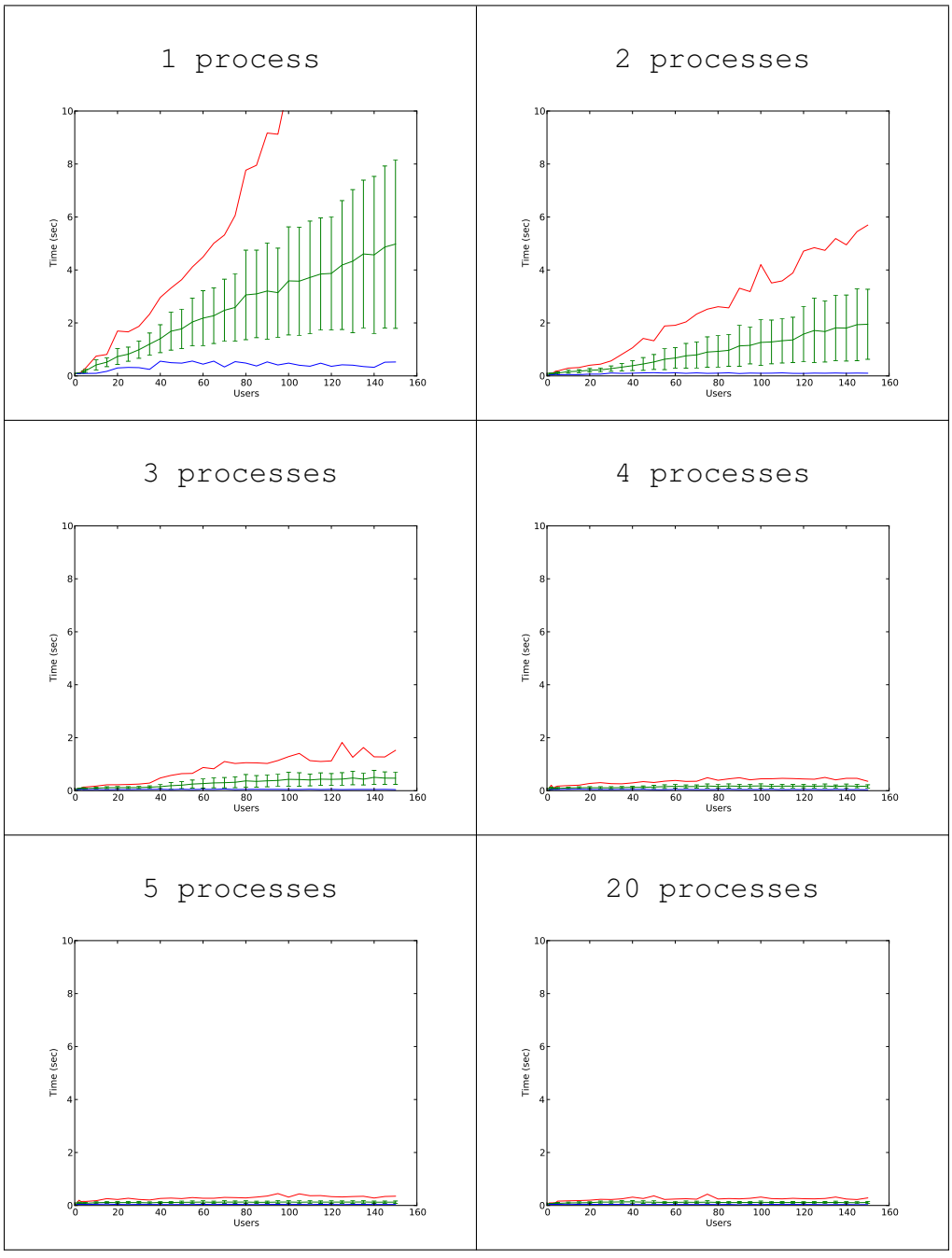


Table 5.15: Results of the `send_command` method with MySQLdb, Redis, 80 copies of the laboratory and different number of processes

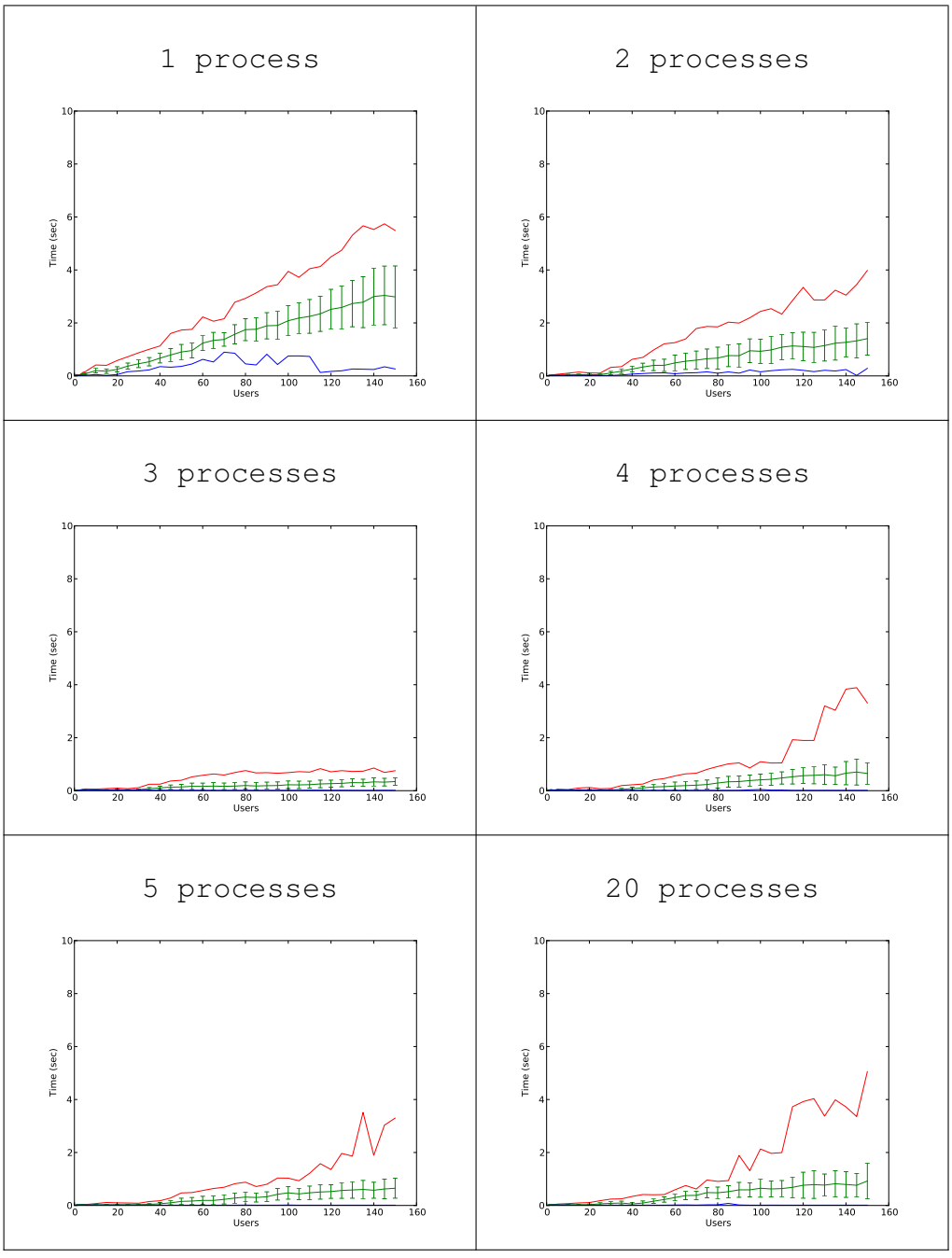


Table 5.16: Numerical summary of Table 5.14

Number of core processes				
Users	Method	Number of processes	μ (ms.)	σ (ms.)
1	send command	1	24	14
		2	17	0
		4	17	0
		5	19	2
		20	19	2
	reserve	1	95	2
		2	75	21
		4	64	17
		5	66	26
		20	66	16
60	send command	1	1,236	279
		2	490	295
		4	175	150
		5	191	155
		20	298	122
	reserve	1	2,181	1,034
		2	675	388
		4	163	69
		5	128	48
		20	116	41
150	send command	1	2,983	1,167
		2	1,407	619
		4	650	405
		5	650	375
		20	923	676
	reserve	1	4,977	3,181
		2	1,949	1,322
		4	163	60
		5	124	44
		20	112	39

MySQL or SQLite) and one based on Redis. The former is cross-platform, while the latter works is only supported in UNIX environments, where it performs much better than the SQL solution (Section 5.2.4). The session can be stored in memory, a database or Redis; and the memory version works better, so it is recommended (Section 5.2.5). The number of processes in different processes is essential. As explained in Section 5.2.7, having at least one process per processor is recommended.

Finally, it must be taken into account that the number of copies of laboratories decreases the performance of the system, since more users are concurrently using them (Section 5.2.6). However, this impact is not really big when the rest of the studied options are properly customized: as seen on Table 5.16, even with 80 copies of the laboratory the reservation system takes around 112 milliseconds with 150 concurrent students.

Therefore, according to the simulations, the best configuration that can be used is: JSON over HTTP for the protocol, MySQLdb for the database, sessions stored in memory, Redis for the scheduling backend, and as many processes as processors available.

5.3 Federated Environment

This section details the behavior of the implementation of the proposed federation model under different configurations. As opposed to the previous section, where all the students were connecting to a single provider that had the laboratories, here different independent but federated WebLab-Deusto instances will be running. Students will use laboratories that are in any of those instances.

As in the previous section, the measurements presented here were performed on a Dell poweredge R410 Intel Xeon CPU E5606 with 4 core processors at 2.13GHz, 8GB RAM, Ubuntu 12.04 and Python 2.7.3. However, in Section 5.3.5, a different configuration will be used, which is detailed in that section.

Taking into account the results of the previous section, all the WebLab-Deusto instances deployed were using 4 processes, Redis for the scheduling system, MySQLdb to access to the database and the sessions were stored in memory.

5.3.1 Aspects to Analyze

In the single node evaluation Section 5.2, for each aspect to be compared, it was possible to compare it with more than one solution. For instance, it was possible to compare the different database systems one each other, or the different protocols used one each other. However, in the federated environment, this situation is not really relevant. For instance, in order to measure the transitive federation Section 5.3.3, it is going to be compared with the basic federation Section 5.3.2,

which is compared with a not federated environment. Additionally, for each situation explained, it is going to be taken into account what happens if the provider has a single copy of the laboratory or several copies. Therefore, each federation feature is going to explain in its particular section what is going to be compared.

The federation features to analyze are the following:

- The behavior of the federation model with two WebLab-Deusto instances is shown in Section 5.3.2, where students are connecting to one and the laboratories are located in the other.
- The transitivity property of the federation model is explored in Section 5.3.3, by locating different number of laboratories in different servers, transitively shared.
- The federated load balance property of the federation model is studied in Section 5.3.4, measuring times with different amounts of load balance in different servers.
- The different possible combinations of transitivity and load balancing are shown in Section 5.3.5 to distribute higher loads of users among distributed copies of the same equipment.

Finally, this section is going to use the terms defined in Section 4.2.1, where an entity E_α sharing a local resource Γ_α with a priority ρ and a time t to a set of students (from 1 to 150) of its institution Σ_α^{1-150} is represented by:

$$\Sigma_\alpha^{1-150} \xleftarrow{\Gamma_\alpha,(\rho,t)} E_\alpha$$

5.3.2 Basic Federation

This section measures federating two entities E_α and E_β, where different number of students of E_α use resources located in E_β. E_α is assumed to have no requested resource (it may have other resources, but not the one requested during the measurements).

Two situations are compared: first, in Table 5.17, E_β has a single resource, and therefore it is compared with a non-federated environment where a single entity has a single device, under the same conditions (using MySQLdb as database, 4 processes, Redis for scheduling system and memory for session management). As it can be appreciated, in the `reserve_experiment` method, the time required is slightly higher than in the not federated version. This time is however stable, fixed under 1 second in the worst case scenario (and a mean between 0.2 and 0.6

Table 5.17: Basic federation with a single copy of the laboratory, comparison of federated and non-federated environments. Scales are different per method

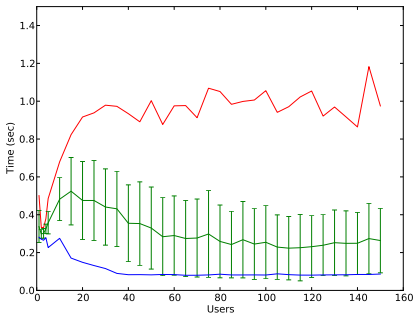
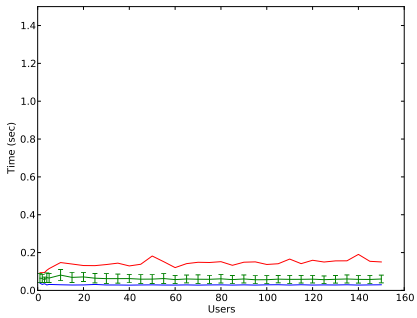
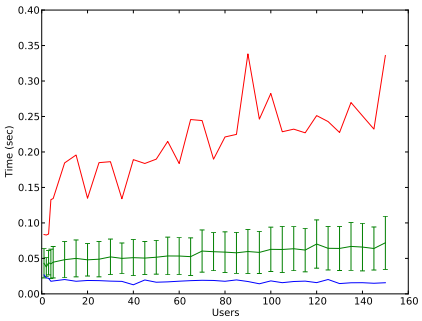
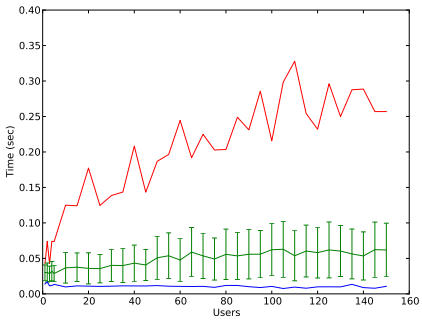
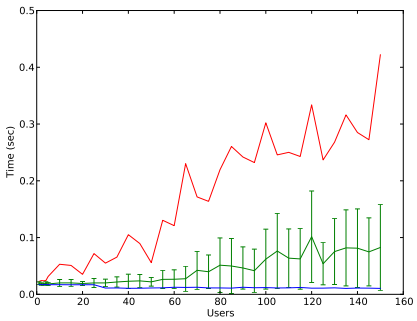
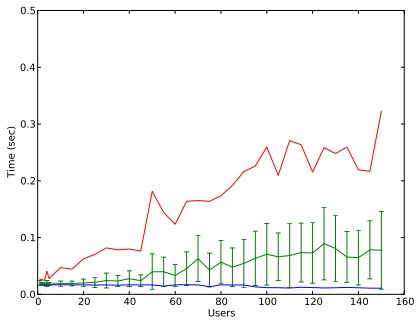
Federated	Not federated
reserve_experiment 	reserve_experiment 
get_reservation_status 	get_reservation_status 
send_command 	send_command 

Table 5.18: Basic federation with 80 copies of the laboratory, comparison of federated and non-federated environments. Scales are different per method

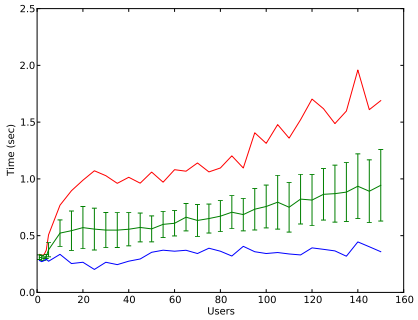
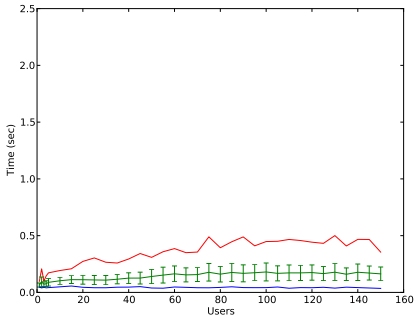
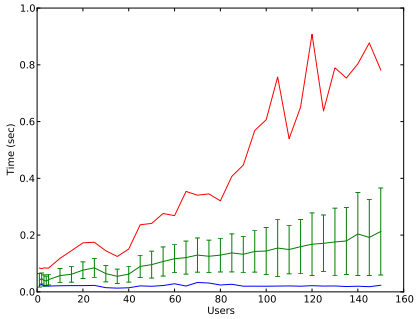
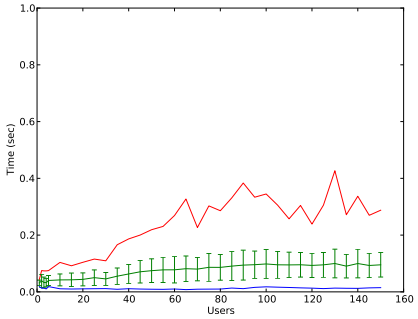
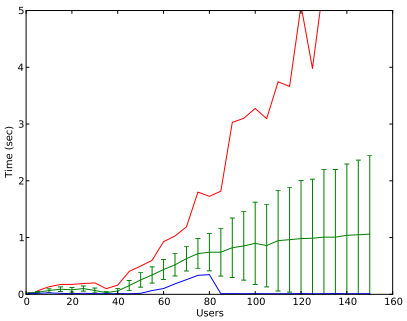
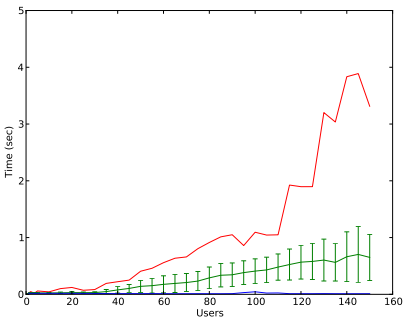
Federated	Not federated
reserve_experiment 	reserve_experiment 
get_reservation_status 	get_reservation_status 
send_command 	send_command 

Table 5.19: Numerical summary of Table 5.17 and Table 5.18

Basic federation compared with no federation					
Users	Method	Number of copies	Federated	μ (ms.)	σ (ms.)
1	send command	1	✓	19	2
			✗	17	0
		80	✓	18	2
			✗	17	0
	reserve	1	✓	337	84
			✗	54	10
		80	✓	310	20
			✗	64	17
60	send command	1	✓	26	16
			✗	38	27
		80	✓	436	176
			✗	175	150
	reserve	1	✓	290	209
			✗	64	24
		80	✓	608	111
			✗	163	69
150	send command	1	✓	82	75
			✗	85	68
		80	✓	1,061	1,379
			✗	650	405
	reserve	1	✓	263	170
			✗	58	18
		80	✓	943	314
			✗	163	60

seconds). The mean decreases as more students come in since most of those students will be in a queue automatically, which requires less resources. The other two methods, however, are not particularly affected in the federated version.

The second situation is where E_β has 80 copies of the same resource. This situation is compared in Table 5.18 with a non-federated environment where a single entity has also 80 copies of the same laboratory under the same conditions. As in the previous example, the reserve experiment method adds more latency to than in the not federated version, almost achieving a mean of one second with 150 concurrent students. Once again, the status retrieval does not take much longer, but the interaction through commands is more affected since the overall stress is considerably higher.

The data of both situations is summarized in Table 5.19.

5.3.3 Transitive Federation

This section measures federating more than two entities transitively. This means that one entity E_α is used by a variable number of students (from 1 to 150), and this one uses the federation protocol to ask E_β, which will transitively contact E_γ, which will have the resources. Finally, students will be redirected to E_γ. This way, while some latency is added to those methods involved in the reservation process, the interaction should still be fast.

In order to measure this, two different situations are compared. In both, 3 entities are involved, having a single copy of the laboratory in the first case and 80 copies of the laboratory in the second case.

The first case is detailed in Table 5.20. On it, E_γ is sharing Γ_γ to E_β, which is resharing it to E_α. WebLab-Bot is simulating a set of students Σ_α^{1-150}.

$$E_\beta \xleftarrow{\Gamma_\gamma} E_\gamma \wedge E_\alpha \xleftarrow{\Gamma_\gamma} E_\beta$$
$$\Downarrow$$
$$\Sigma_\alpha^{1-150} \xleftarrow{\Gamma_\gamma} E_\alpha$$

As it can be appreciated, the latency is increased in those methods involved in the reservation processing by 1-3 seconds in the worst case scenario (and 1-2 seconds attending to the mean). The impact on interaction (measured by the `send_command` method) is still not considerable, since the client directly contacts with the final entity. There are some peaks, but this happens because the first assigned users submit commands while the system stress is still very high. If they

Table 5.20: Results of 1 laboratory (single copy) shared through three entities transitively compared with the same situation in 2 entities. Scales are different per method

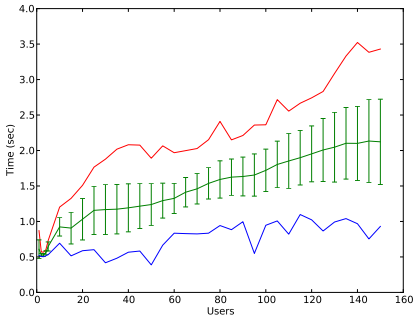
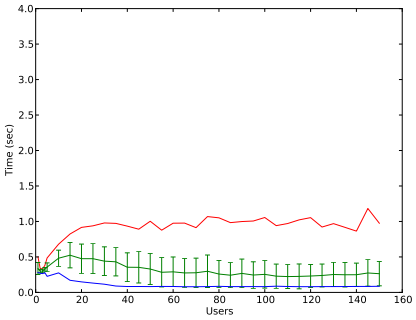
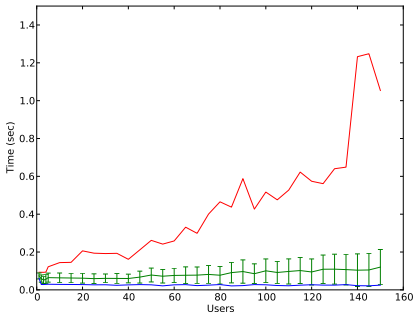
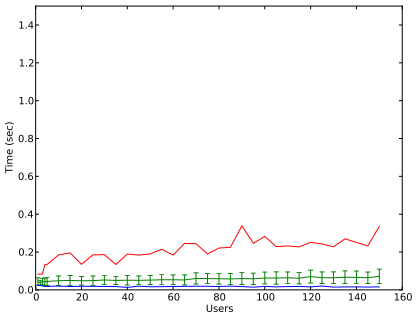
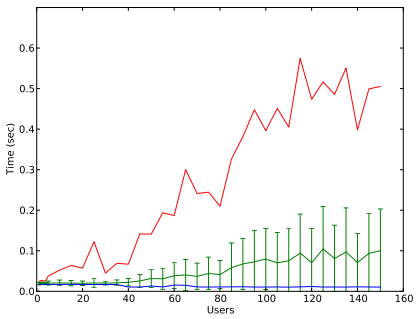
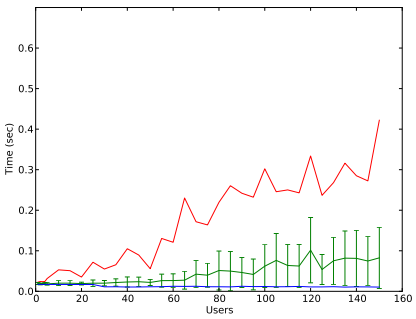
Federated - 3 entities	**Federated - 2 entities**
reserve_experiment 	reserve_experiment 
get_reservation_status 	get_reservation_status 
send_command 	send_command 

Table 5.21: Results of 1 laboratory (80 copies) shared through three entities transitively compared with the same situation in 2 entities. Scales are different per method

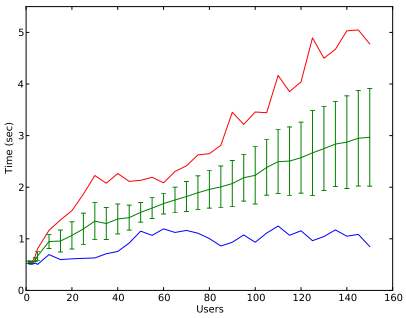
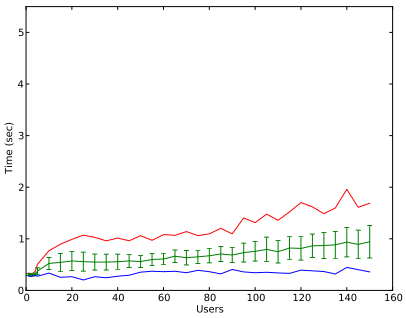
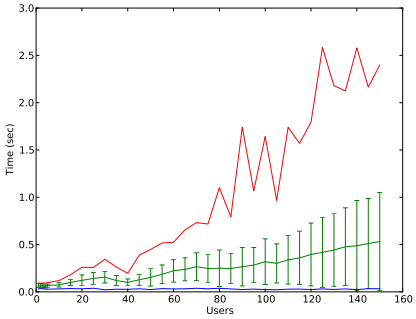
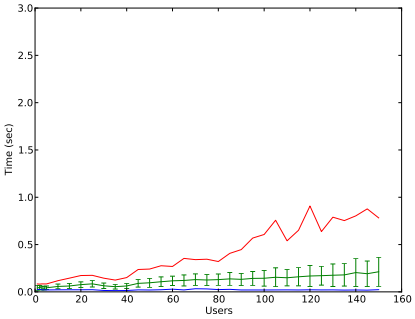
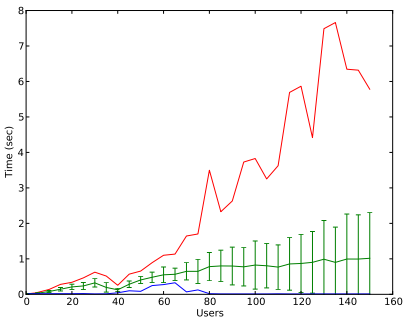
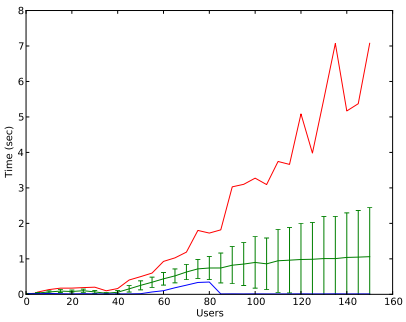
Federated - 3 entities	**Federated - 2 entities**
`reserve_experiment` 	`reserve_experiment` 
`get_reservation_status` 	`get_reservation_status` 
`send_command` 	`send_command` 

Table 5.22: Numerical summary of Table 5.20 and Table 5.21

Transitive federation: 2 federated nodes versus 3 federated nodes					
Users	Method	Number of copies	Federated nodes	μ (ms.)	σ (ms.)
1	send command	1	2	19	2
			3	18	2
		80	2	18	2
			3	18	1
	reserve	1	2	337	84
			3	607	132
		80	2	310	20
			3	546	220
60	send command	1	2	26	16
			3	38	32
		80	2	436	176
			3	549	220
	reserve	1	2	290	209
			3	1,325	212
		80	2	608	111
			3	1,682	201
150	send command	1	2	82	75
			3	100	102
		80	2	1,061	1,379
			3	1,050	1,284
	reserve	1	2	263	170
			3	2,122	600
		80	2	943	314
			3	2,966	946

waited a couple of seconds after getting reserved but before submitting commands, the time would be lower.

The second case (same setting, but E_γ is sharing 80 copies of Γ_γ, and E_β is re-sharing them all) is detailed in Table 5.21. As in the previous case, a slight latency of 1-3 seconds is added attending to the maximum time (but 1-2 seconds attending to the mean), especially in the reservation method. The interaction with the laboratory is maintained also very similar.

In both cases, the tests were run in a single computer. This way, the two or three entities were competing for the system resources (CPU, disk...). Therefore, the slight latency added is a product of at least two things: first, the fact that all the entities are competing for the same system resources, and second, that adding other entity implies deteriorating the latency since the requests have to pass through yet another server. However, given the small size of this latency, it is clear that it has a minor impact, which is visible in the data acquired, and summarized in Table 5.22.

5.3.4 Federated Load Balancing

This section analyzes the impact of distributing the load of users among distributed copies of the laboratories. Two different cases have been analyzed in this category.

In the first case, two entities (E_β and E_γ) are sharing two copies of the laboratory (Γ_β by E_β and Γ_γ by E_γ) to a third entity E_α, which is sharing these two copies as well as a local copy Γ_α to students Σ_α^{1-150}. This way, students have access to the three copies distributed in the three entities:

$$E_\alpha \xleftarrow{\Gamma_\beta} E_\beta \wedge E_\alpha \xleftarrow{\Gamma_\gamma} E_\gamma$$
$$\Downarrow$$
$$\Sigma_\alpha^{1-150} \xleftarrow{\Gamma_\alpha,\Gamma_\beta,\Gamma_\gamma} E_\alpha$$

This case is compared in Table 5.23 with the simple federation case detailed in Section 5.3.2. On it, there were only two entities sharing a single laboratory. As it can be appreciated, the maximum value in the reservation process increases by 3 seconds, as well as the mean. However, the main difference is that in the basic federation sample, the consumer entity will just forward the request to the provider entity, being a single queuing scheduler involved. In the load balance example, there are three queuing schedulers involved, since E_α has its own one and it must contact E_β and E_γ. This affects both scheduling methods (reserve and retrieve status), adding more latency to the responses. However, this does not affect

Table 5.23: Load balance 3 copies in 3 institutions, compared with basic federation. Scales are different per method

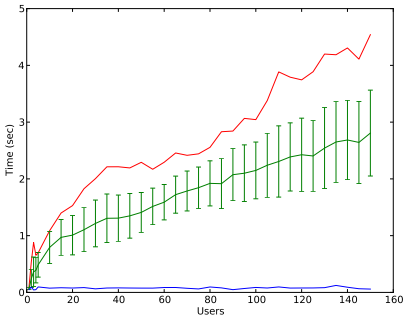
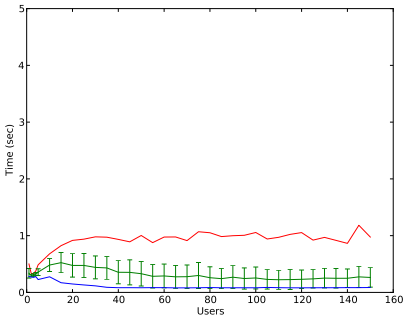
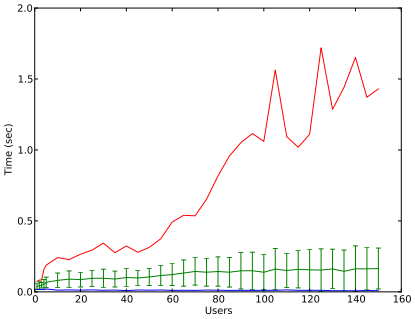
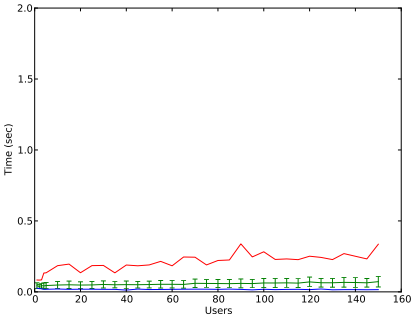
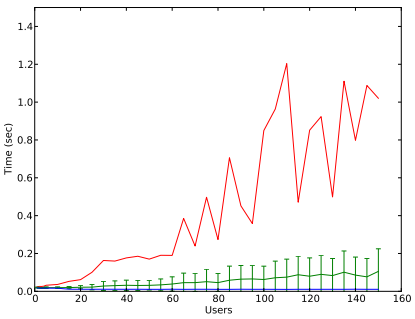
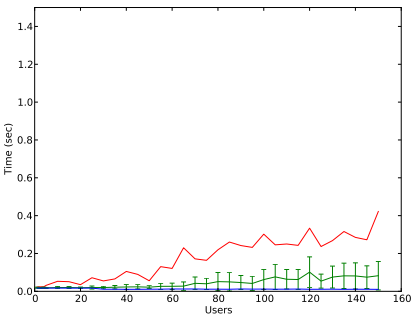
Federated - federated load balance	**Federated - no load balance**
`reserve_experiment` 	`reserve_experiment` 
`get_reservation_status` 	`get_reservation_status` 
`send_command` 	`send_command` 

Table 5.24: Load balance 240 copies in 3 institutions, compared with basic federation with 80 devices. Scales are different per method

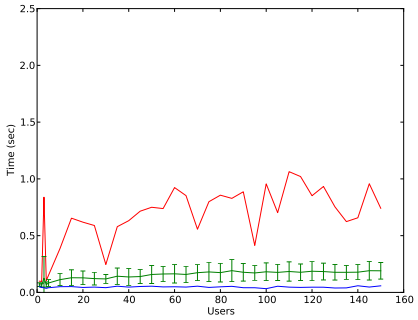
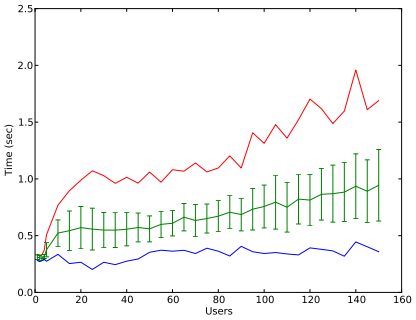
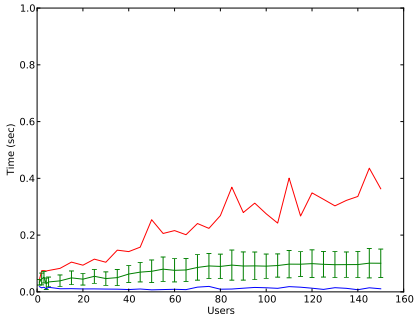
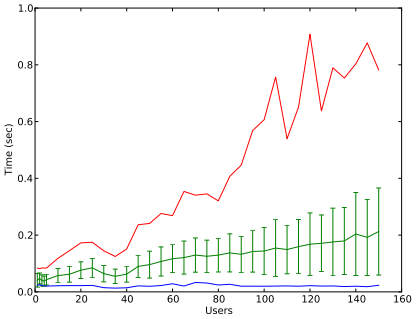
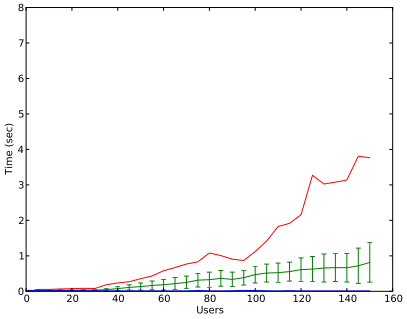
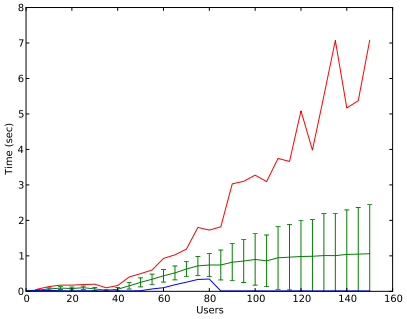
Federated - federated load balance	**Federated - no load balance**
reserve_experiment 	reserve_experiment 
get_reservation_status 	get_reservation_status 
send_command 	send_command 

Table 5.25: Numerical summary of Table 5.23 and Table 5.24

Load balance federation compared with basic federation					
Users	Method	Number of copies	Load balance	μ (ms.)	σ (ms.)
1	send command	1	✗	19	2
			✓	19	2
		80	✗	18	2
			✓	19	3
	reserve	1	✗	337	84
			✓	64	17
		80	✗	310	20
			✓	68	16
60	send command	1	✗	26	16
			✓	38	37
		80	✗	436	176
			✓	185	151
	reserve	1	✗	290	209
			✓	1,589	311
		80	✗	608	111
			✓	163	80
150	send command	1	✗	82	75
			✓	105	119
		80	✗	1,061	1,379
			✓	815	555
	reserve	1	✗	263	170
			✓	2,807	756
		80	✗	943	314
			✓	190	73

the interaction method, keeping the mean and standard deviation in similar levels (except for those situations where they submit commands while the load of stress is still high).

In the second case, the same three entities are sharing 240 copies of the laboratory (80 copies each). The entities E_β and E_γ are sharing 80 copies each to the third entity E_α, which also has 80 copies. This way, there are 240 copies of the laboratory. In Table 5.24, this situation is compared to the basic federation one with 80 copies of a laboratory detailed in Section 5.3.2. Given that there are always more copies of laboratories than students, in the reservation process the minimum value is very stable and the standard deviation is very low. The reservation retrieval and the interaction methods are not negatively affected when compared with the no load balance version: while it might seem that it actually improves, the scale is low, so for example in the reservation retrieval the improvement is of a tenth of second.

Both tables are summarized in Table 5.25.

5.3.5 Transitively Federated Load Balancing

This section analyzes the behavior of the system with larger number of students, laboratories and federated environments.

All the scenarios presented in this section use both federated load balance and transitivity. They all have 6 entities deployed, in two different copies of Dell poweredge R410 Intel Xeon CPU E5606 with 4 core processors at 2.13GHz, 8GB RAM, Ubuntu 12.04 and Python 2.7.3 (3 entities in each server). One MySQL and Redis process has been launched per entity. The client is launched from a third server, a Dell poweredge R310 with 4 core processors at 2.40GHz, 8GB RAM, Ubuntu 12.04 and Python 2.7.3. Each entity has 60 copies of the same laboratory, so in total there are 360 copies of the laboratory.

Two topologies have been deployed: star configuration and ring configuration. In the *ring configuration*, described in Figure 5.6, entities form a ring where each entity shared its laboratories with the previous entity in the ring. If a student of Entity A requests a laboratory located in Entity C, Entity A will request it to Entity B and Entity B will request it to Entity C, so transitively it will access it. In the *star configuration*, described in figure Figure 5.7, every entity knows the rest of the entities, and they are contacted once the local entity does not have more resources.

Creating a long chain using the ring configuration increases considerably the chances of a system failure. In this case, if the RLMS of any of the particular links (Entity B, Entity C, Entity D or Entity E) fail, Entity A will not be able to connect to the Entity F system. Debugging this system will also be difficult, and while the latency during the experiment does not increase (since users are redirected to the final system directly), the latency during the reservation process may increase. This way, whenever students of Entity A tests the system and is willing to use it in its

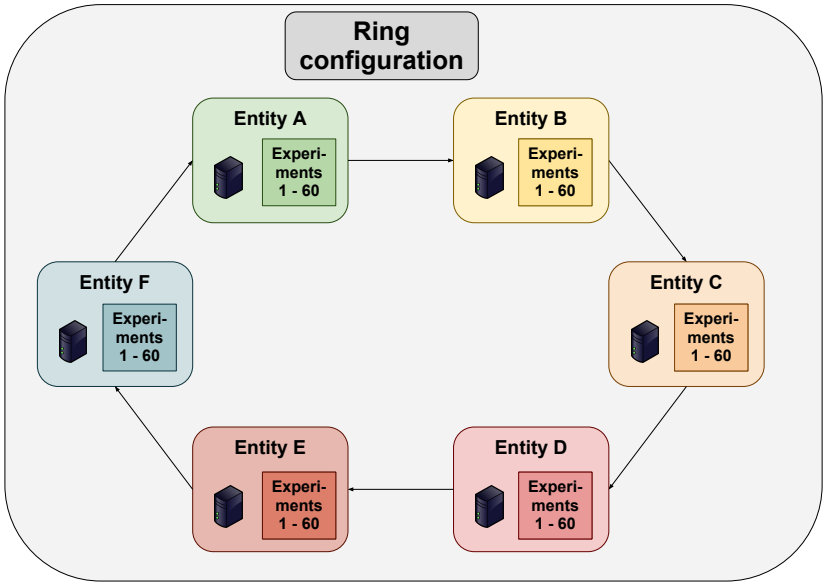


Figure 5.6: Ring configuration: every entity has access to each other through a chain of relationships

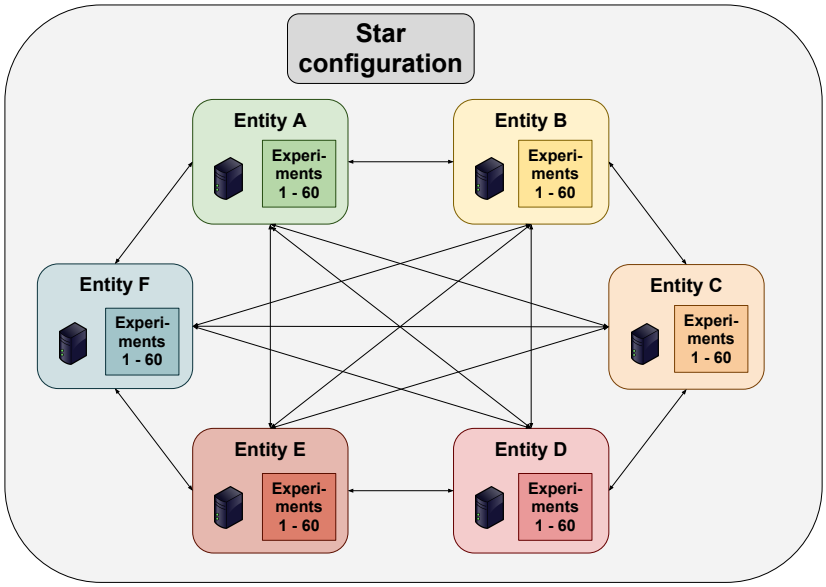


Figure 5.7: Star configuration: every entity knows each other

curricula, it would be desirable that they contact directly Entity F and get registered it there (using the star configuration).

However, while this situation might seem technically ideal, adding a few layers of indirection is helpful in certain situations. First, for each layer, there is an attached responsibility chain. Entity A may not know or trust Entity C, or Entity D. Indeed, in the ring configuration, Entity C is being responsible for what Entity D does. Associated costs or priorities in queues also work in the same way: Entity A may state that Entity B has 10,000 accesses or a certain priority, and Entity B may split those among its students and only let Entity C to access 2,000 times. Furthermore, there are cultural barriers (such as languages, especially when secondary schools are involved) that would be at certain level avoided through these layers. It might be easier for a secondary school to use a laboratory that is also used by a local university (even if it is not hosted there) than contacting somebody else in other language in other time zone.

Additionally, from a technical point of view, it is not clear that one configuration is much better than the other. First, once reserved, the interaction is always direct with the final system, so it is not affected. The reservation process, however, is more complex in these scenarios. The star configuration is more stressful, since at some point all nodes could be communicating each other. That said, the reservation status retrieval is simpler in the star configuration, since each node talks directly with the others instead of forwarding a message through different layers. This way, the reservation status retrieval can be quicker in the star configuration.

So as to measure both topologies, two tests have been done:

1. From 1 to 200 students using a single entity. Unlike the previous measurements, where 150 students were selected, 200 students are being analyzed to find jumps in the trends when more nodes are involved. Given that there are 60 copies of the laboratory per node, the jumps are expected when crossing the 60, 120 and 180 students.

2. From 1 to 420 students in total, requesting to all the entities at the same time. For instance, 420 students means 70 students requesting laboratories to each entity. This number is used since a lower one (e.g., 360) would not actually use federation, since each student would be using only the local resources. In this case, it is important to remark, as stated in Section 5.1.4, that this means that all these students from different universities are clicking the reserve button at nearly the same time, which, once again, is very unlikely but useful for stressing the system and checking the behavior.

The first setup is analyzed in Table 5.26 and summarized in Table 5.27. In this case, up to 200 students access a single entity. It shows the typical situation where 200 students of a single entity use it. If 1,200 students of the six entities (200

Table 5.26: Up to 200 students accessing a single entity, with laboratories in 6 entities

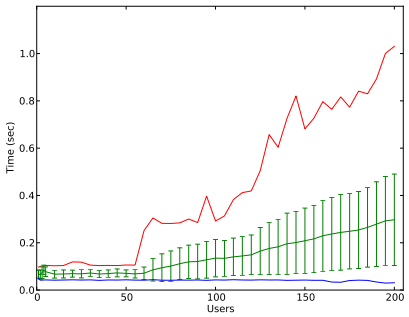
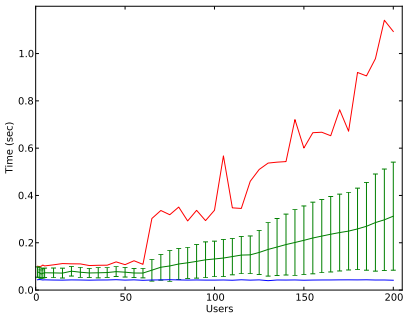
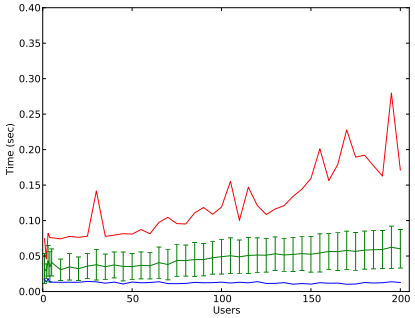
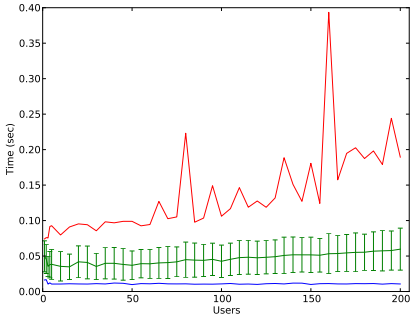
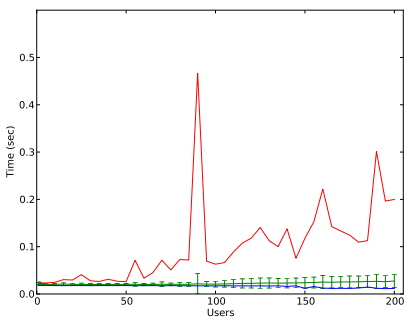
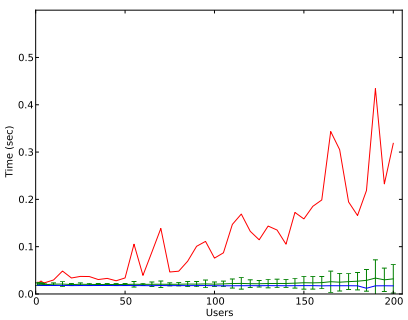
Ring configuration	Star configuration
reserve_experiment 	reserve_experiment 
get_reservation_status 	get_reservation_status 
send_command 	send_command 

Table 5.27: Numerical summary of Table 5.26

Transitive federation with federated load balance accessing a single entity				
Users	Method	Configuration	μ (ms.)	σ (ms.)
1	send command	Ring	21	2
		Star	20	2
	reserve	Ring	67	16
		Star	76	21
60	send command	Ring	19	2
		Star	20	2
	reserve	Ring	71	26
		Star	71	20
120	send command	Ring	22	9
		Star	21	8
	reserve	Ring	148	83
		Star	148	79
180	send command	Ring	25	12
		Star	27	18
	reserve	Ring	253	162
		Star	258	172
200	send command	Ring	27	14
		Star	31	29
	reserve	Ring	297	193
		Star	312	228

Table 5.28: Up to 420 students accessing 6 entities concurrently, with laboratories in 6 entities

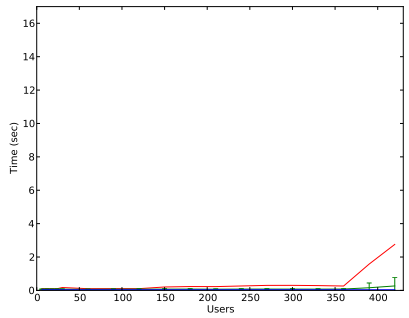
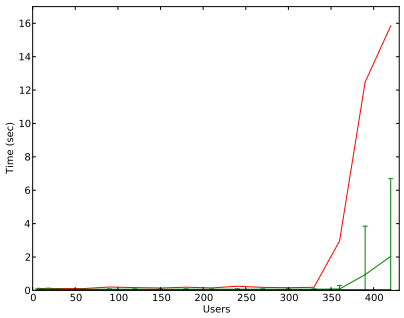
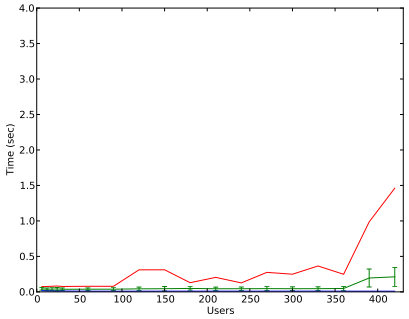
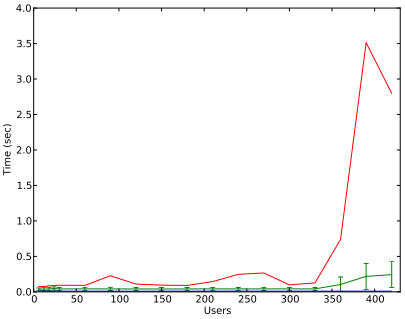
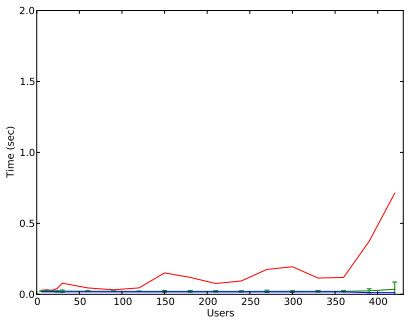
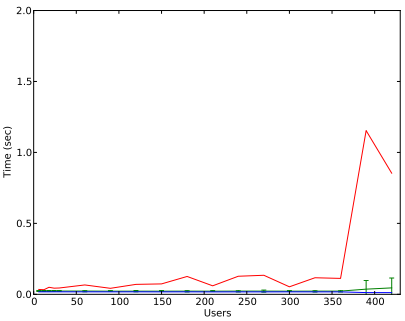
Ring configuration	Star configuration
reserve_experiment 	reserve_experiment 
get_reservation_status 	get_reservation_status 
send_command 	send_command 

Table 5.29: Numerical summary of Table 5.28

Transitive federation with federated load balance accessing six entities				
Users	Method	Configuration	μ (ms.)	σ (ms.)
6	send command	Ring	21	3
		Star	23	3
	reserve	Ring	67	20
		Star	70	23
300	send command	Ring	21	7
		Star	22	3
	reserve	Ring	79	25
		Star	72	19
360	send command	Ring	21	4
		Star	22	6
	reserve	Ring	83	22
		Star	86	198
390	send command	Ring	24	14
		Star	36	60
	reserve	Ring	159	285
		Star	942	2,907
420	send command	Ring	35	50
		Star	45	69
	reserve	Ring	267	495
		Star	2,039	4,652

per entity) were going to use the laboratories and they are not using it at the very same time, the situation would be the one represented here (and once again, the measurements are taken if the 200 students of the same entity pressed the reserve button at the very same time).

For the first 60 students, the time to process reservations increases only slightly, since it accesses local resources. However, from 60 to 120, the ring configuration only contacts one more entity, as well as for 120 to 180 to a third entity, and finally 180 to 200 to a fourth entity. As explained in the previous section, in these points the overhead increases more, changing the trend slightly in those points. The behavior in the star configuration is not very different in this case.

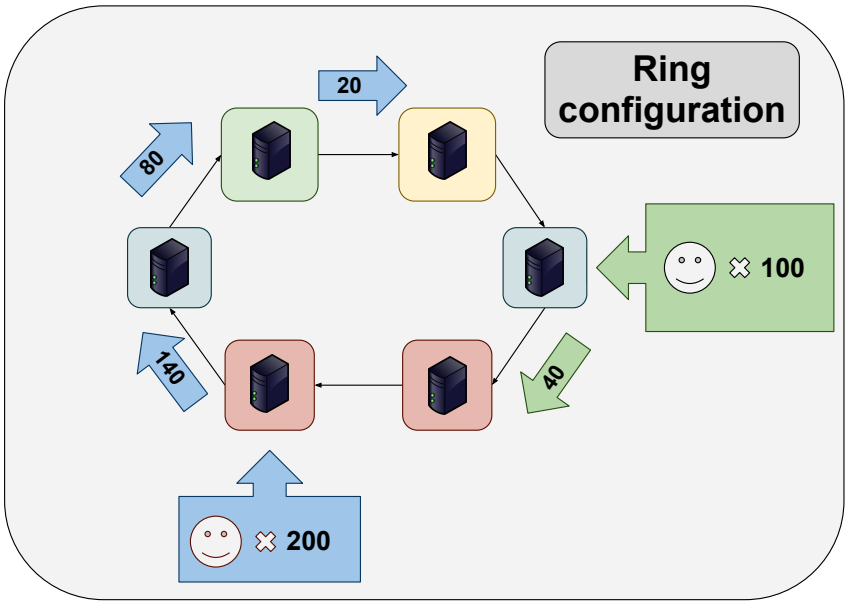


Figure 5.8: Distribution of students in the ring configuration

Additionally, as shown in Figure 5.8, if 200 students of a single entity start using one resource, the reservations will be propagated through the ring. The entity itself will use 60 local resources for 60 students, while it will need to pass 140 students to the second entity. This second entity will grant access to 60 students, passing 80 of them to the third entity. The third entity will grant access to 60 students, passing only 20 of them to the fourth entity. Therefore, the fifth and sixth entities are not affected at all by this process. Furthermore, if in the fifth entity 100 students start requesting accesses, 60 of them will be processed by this entity while 40 will be reserved in the sixth entity. Once again, this process does not affect the other 4 entities in the ring. This way, both situations (200 students in the first entity and 100 students in the fifth entity) could happen at the very same time

without affecting each other. However, in the case of the Star configuration, all the entities are propagating the requests to all the other entities. Therefore, all the nodes (including fifth and sixth) will handle the requests from the first situation. This way, the second situation will be affect by the first one if it occurred at the same time.

This is indeed clearer in the second case, which is studied in Table 5.28 and summarized in Table 5.29. On it, up to 420 students access concurrently to the 6 entities. Until 360 students, the values tend to be low, since they are managed locally. However, when more students come in, all the entities forward them to all the rest. In the ring scenario, this is done slowly from one entity to the other working on the ring, and the values are kept low. In the star scenario, the values are multiplied, especially in the reservation process. While the ring configuration seems to work better, this happens in the stressful situation of the WebLab-Bot and, as explained before, in case of failure in one of the nodes the federation affects other federated environments. Finally, in these measurements a local network is used, so the network latency is small. If the network latency was higher, the effect would be multiplied 5 times in the ring configuration (since there are 6 nodes, and messages cross the whole ring), while it would be multiplied only twice in the star configuration (since the interaction is direct).

In summary, while the adequate configuration depends on the contracts among the different entities (and barriers such as languages), from a technical point of view, both have advantages and disadvantages, so it depends on the particular scenario. However, WebLab-Bot can be used scenario to find the proper configuration for each situation.

5.3.6 Summary of the Federated Environment

Supporting federation adds an overhead in those methods involved in scheduling, as shown in Section 5.3.2. However, as shown in that section, this impact can be perceived especially in the `reserve_experiment` method, since the difference in the interaction methods is lower. This means that the overhead is only paid in the reservation process.

In Section 5.3.3, transitive federation adds more latency. Adding more layers will have a network latency impact not measured here (as explained in Section 5.1).

The federated load balance is measured in Section 5.3.4. On it, the load balance behaves better since the first 80 students in each configuration use the local resources. For instance, if 150 students come in, the federated systems will only manage 70 students, so the stress is even lower.

Finally, Section 5.3.5 shows both properties working together with higher number of concurrent students. While, as explained in Section 5.1.4, these numbers are not realistic (420 students pressing the “reserve” button at the very same time), it

shows how the ring configuration performs better than the star configuration in the most complex situation, while it also provides other drawbacks such as not supporting the failure of federated links. Overall, it would be better to use the ring configuration, although there are contractual barriers that may make this fail (e.g., if one entity shares only 500 accesses to another, this one could only share a subset of this one).

5.4 Low Cost Environments

In Section 4.3.3, it was described how the transitivity property of the federation model makes it possible to deploy WebLab-Deusto in low cost servers, and that these servers could be deployed somewhere else and re-shared by certain institutions. Indeed, it stated that the fishtank laboratory had been deployed on a IGEPv2[1] device, which is a low cost device that has 512 MB RAM and an ARMv7 processor.

In order to provide empirical data on the behavior of WebLab-Deusto in such systems, it has been measured in a Raspberry Pi[2]. Raspberry Pi is a popular low cost device which only has 256 MB RAM (shared with the GPU) and an ARMv6-compatible processor (see Figure 5.9). The cost of the model used (Model B) is $35.00 at the time of this writing.

Figure 5.9: Low cost Raspberry Pi

Given the constraints of the system, MySQL was not installed, and instead the SQLite database was used. As explained in Section 3.2.1, WebLab-Deusto supports that all the servers are deployed in a single process, and the communication will be performed directly (without any network latency). While measuring the times, it was observed that this single process consumes between the 10 and the 20 percent of the memory available in the system, so it was possible to use the Redis

[1] http://www.igep.es
[2] http://www.raspberrypi.org/

database as scheduling backend. Just in case it started swapping memory with disk, it was also tested with SQLite for managing the scheduling.

The results are detailed in Table 5.30, normalizing the scale for each method. As it can be appreciated, in the reservation process the performance when using Redis is almost doubled when compared to the SQLite version. This could be surprising given the constraints of memory of the system, but during the measurement, there were always 8-12 free megabytes and no swapping between disk and memory occurred. On the other hand, the impact on the interaction with the laboratory and the log in process is not really affected by the choice.

While the mean waiting time for 20 concurrent students is still considerably high, it demonstrates that it can be deployed in these conditions. If better performance is required, the spectrum of low cost devices is wide enough to choose a more powerful one. And once again, it must be taken into account that this data is generated when 20 concurrent simulated students log in and reserve a laboratory in a period of 1 second. If this period is increased (since it is very unlikely in the real world that students will click on the reserve button in such a short time span), the stress for the system is decreased and therefore the measurements are improved and more acceptable. For example, as appreciated in the graph, for one user the reservation process takes around 5 seconds. If 20 students log in and reserve the laboratory perfectly distributed in a period of 100 seconds (1 minute, 40 seconds), the reserve method would take around 5 seconds for each user, given that the stress for the system would be the same for each user.

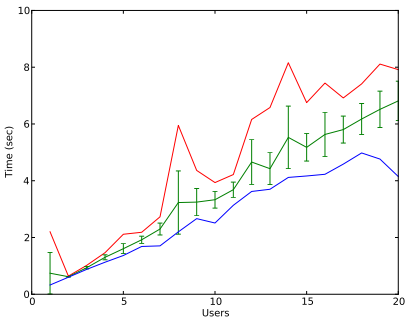
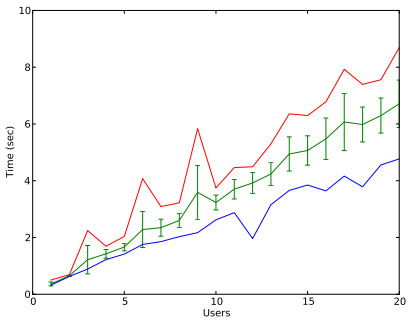
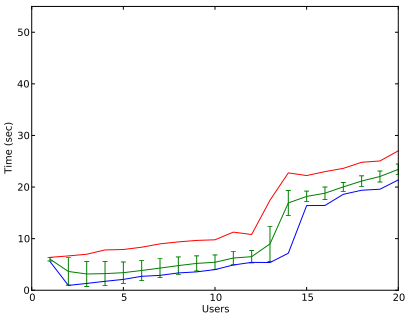
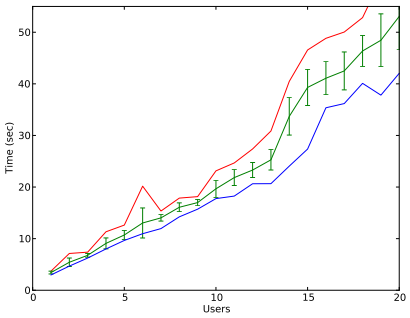
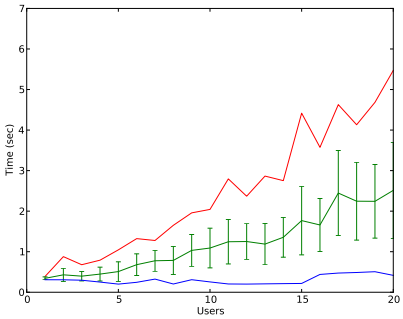
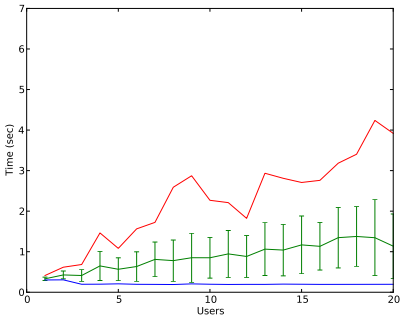
5.5 Conclusions

This chapter offers a quantitative evaluation of WebLab-Deusto (explained in Chapter 3) and of the implementation of the federation model (explained in Chapter 4), developed on top of it. It defines which settings are more adequate for different situations.

The first part, Section 5.2, evaluates WebLab-Deusto itself, described in Chapter 3, showing which features affect performance and measuring how much. It shows how the best configuration is using JSON over HTTP for the protocol, MySQLdb for the database, storing the sessions in memory, Redis for the scheduling backend, and as many processes as processors available.

The second part, Section 5.3, evaluates the federation protocol itself, built on top of WebLab-Deusto, and measured under highly stressful settings. It shows how the federation protocol adds latency (especially on the reservation process), while the other features (transitive federation and load balance) do not have a considerable bigger impact. Altogether, it shows how the ring configuration performs better than the star configuration, since the nodes stress less each other under stressful

Table 5.30: Results with raspberry using Redis and SQLite for scheduling. Scales are normalized per method

login - (Redis) 	login - (SQLite) 
reserve_experiment (Redis) 	reserve_experiment (SQLite) 
send_command (Redis) 	send_command (SQLite) 

situations.

Finally, Section 5.4 shows how this federation protocol and this implementation can run in constrained devices such as the Raspberry Pi. While this is not recommended with a high number of users, it validates the situations explained in Section 4.3.3 and it shows that the system does not have important memory requirements.

In summary, the main contribution of a federation model is that it shares resources, enhancing the service for students, who have access to a wider range of laboratories. The qualitative improvement presented in Chapter 4 adds a measurable quantitative performance cost analyzed in this chapter. In order to measure it, the federation protocol has been tested even with 420 simulated students clicking at the very same time on the *reserve* button in 6 different entities, and even in that fictitious and highly stressful situation the system has worked and the latency in the interaction was kept under a tenth of second in the most stressing configuration.

Now this is not the end. It is not even the beginning of the end. But it is, perhaps, the end of the beginning.

Winston Churchill

CHAPTER

6

Conclusions

In the introduction of this dissertation, the target problem was described and contextualized. The main target problem is defining a new federation model for sharing laboratories, which supports re-sharing of laboratory accesses (defined as *transitive federation*, Section 4.2.3.1) and scaling up in number of students by distributing them among different copies of laboratories spread in different institutions (defined as *federated load balance*, Section 4.2.3.2). Both concepts are defined in Section 4.2.2. Together, new situations will emerge in the landscape of remote laboratories, such as hundreds of students of different institutions using concurrently distributed laboratories.

This federation model has been implemented on top of WebLab-Deusto, the open source remote laboratory management system of the University of Deusto. The author of this dissertation has led the design and development of this software since 2004. WebLab-Deusto (described in Chapter 3) uses a plug-in system for the scheduling algorithms, and the federation model has been implemented as a plug-in. This makes it possible to benefit from other features of WebLab-Deusto, such as the local priority queue which supports multi-resource local load balance (Section 3.4), as well as the integration of other systems, enabling interoperability (Section 4.3.2). WebLab-Deusto is additionally a mature system (developed since October 2004, in production with students since February 2005, Section 3.1) which provides tools and different approaches to develop new laboratories (Section 3.5). All these experiments will automatically be exported using the federation model, as well as other features: authentication with multiple backends (LDAP, OpenID, OAuth 2.0, databases) or scalability strategies (Section 3.3).

The performance of WebLab-Deusto has been evaluated in a number of situations in Section 5.2, taking into account the particular features and deployment schemes of WebLab-Deusto in different settings (e.g., different backend technology –Redis, MySQL or SQLite–; different number of servers deployed; or different session managers) in a regular server (an Intel Xeon CPU E5606 with 4 core processors at 2.13GHz, 8 GB RAM, Ubuntu 12.04 and Python 2.7.3). To do this, a student simulator has been developed, which emulates a group of concurrent students using the real system with different configurations. With the obtained data, it has been possible to measure and evaluate the implementation of the federation model on WebLab-Deusto (Section 5.3), first using a single server to test the particular properties and then using two servers to test complex situations (Section 5.3.5). As it has been measured, even with 420 concurrent students reserving at the same time 360 copies of a laboratory in 6 WebLab-Deusto deployments split in two servers, using a ring configuration, the mean time of retrieving reservations and of interacting was under one tenth of second, which makes it suitable for production environments.

Indeed, as presented in Section 1.2, one of the foci of this dissertation was exploring the suitability of scaling up remote laboratories to support Massive Open Online Courses (MOOCs). With the results presented in this dissertation, it becomes technically possible to support large numbers of concurrent students in certain circumstances. A clear example is VISIR(Gustavsson et al., 2007), which supports up to 60 concurrent students, and there are already 7 copies already deployed and other 3 are in process of being deployed. If the federation model presented in this dissertation is used in all these copies, up to 600 students can be concurrently accessing VISIR and the rest would be in a queue. Section 5.3.5 evaluates this type of situation. 600 slots are very few compared with the hundreds of thousands that certain MOOCs are achieving, but it is important to clarify that not every MOOC get this amount of students in every course, nor at the very same time, neither in every assignment of the course, since an important percentage of students drops the course in the beginning. And the number of slots required for a particular number of students in a type of course has not been defined, so 600 slots may well support a few thousand students enrolled, depending on the policies and load modeling applied. If traditional laboratories sometimes define different shifts for class groups, this could well be applied in this field too.

Furthermore, other type of complex situations discussed (Section 4.3.3), such as deploying the whole remote laboratory management system in low cost devices (and not only the laboratory itself) have been targeted by running WebLab-Deusto in a Raspberry Pi (Section 5.4) with interesting results (even supporting 20 concurrent students with a reservation time of around 25 seconds and a latency in the interaction of between 2 and 3 seconds). This shows that while WebLab-Deusto supports typical server configurations, it is also suitable for small devices while

keeping the federation model properties.

Additionally, the federation model has demonstrated to be effective for other tasks, such as the integration of remote laboratories in Learning Management Systems -LMS- and Content Management Systems -CMS- (Section 4.3.1). Indeed, the use of federation as the core for integrating RLMSs in LMSs is the kernel of the design of the lms4labs initiative which I lead for integrating remote laboratories in LMSs, on which WebLab-Deusto, FCEIA (National University of Rosario, Argentine) are already working and efforts by MIT CECI staff are being placed to support iLabs (among other tasks, such as the support of LTI and big improvements in administration layers of lms4labs components). Additionally UNED staff is also working on this initiative to support .LRN and Moodle 1.9.

As detailed on the introduction (Section 1.2) the interest on the federation of remote laboratories is increasing during the last few years. Indeed, the interest on this particular federation model has also increased during the last year for its potential on the STEM education. In addition to the institutions that are already using WebLab-Deusto through its federation protocol, I have been awarded in November 2012 by the Massachusetts Institute of Technology as one of the 10 top Spanish innovators under 35 years in 2012 (TR35 2012) for this work. Secondly, the Singularity University (NASA Ames, Mountain View) chose this work for one of the three talks related to Education during its Global Grand Challenges week, where I presented these results in June 2012.

The remainder of the chapter is structured as follows: Section 6.1 draws the limitations of the presented work, Section 6.2 shows the future work and Section 6.3 outlines the final remarks.

6.1 Limitations

As discussed in Chapter 4, the presented federation model and its implementation has a number of limitations, summarized here.

Firstly, its design has focused on queues rather than on calendar based booking. In particular, it has used the queuing model of WebLab-Deusto (for instance, it does not provide any estimation of the remaining time to users in the queue, which is something that the iLabs Shared Architecture does). The algorithm itself could easily be extended using the same propagation algorithm among the different entities, so the particularities of the scheduling system have not been considered necessary for the scope of this research, while it would be desirable that in the near future it was supported.

Secondly, while the use of federation as the basis of interoperability bridges has been tested both with the iLab Shared Architecture and with Learning and Content Management Systems (LMS/CMS), this research has not implemented more in-

teroperability layers with other remote laboratory management systems, especially with Labshare Sahara. While this would be an interesting feature, defining an interoperability interface for all the RLMSs is outside the scope of this dissertation.

Thirdly, the federation model does not provide any primitive for collaboration. As explained in Section 3.5.1.1, WebLab-Deusto supports that experiment developers create collaborative experiments, where students can concurrently access the same laboratory and work together. This does not affect basic federation or transitive federation, but it does affect federated load balance, where some students might be redirected to copies of laboratories in other universities. Defining the primitives for properly managing this in the federation scheduler would be challenging and an interesting new line of research.

6.2 Future work

As explained in the previous section, there are new lines of work which can be extracted from the core of this contribution into new research projects: adopting more scheduling algorithms in the core of the federation model would be interesting (and not only calendar based booking, but other schemes (Lowe and Orou, 2012)), as well as adding the primitives required for supporting collaboration in distributed environments (Schulz and Long, 2012) and supporting a wider interoperability with other RLMSs.

Without modifying the model, it is also possible, at implementation level, to devise heuristics that make a better use of the federation model, reducing the load of work. In this line, it is also possible to implement some of the possibilities discussed in Chapter 4 such as canceling reservations in other systems when one is close to be assigned, caching the responses during some time, or prioritizing the external entities. These features can be implemented in a consumer system without requiring the rest to upgrade, since the protocol is not modified.

Aiming modular standards which cover the features presented in this dissertation is also an important challenge for the future. Both the Global Online Laboratory Consortium (GOLC[1]) and the IEEE P1876[2] will play a critical role in this task.

However, new lines of related but not directly derived work can be opened based on the results presented in this dissertation. First, a cloud-like system that enables institutions (e.g., secondary schools) to create RLMS deployments in the cloud, which uses the federation protocol to talk with a set of RLMSs with real equipment deployed becomes possible. This could increase the number of instructors of different institutions creating their own remote laboratory deployment for

[1] http://www.online-lab.org/

[2] http://standards.ieee.org/email/2012_09_cfp_P1876wg_web.html

free, without requiring them complex contracts, and directly benefiting from the complex situations that can be built with this protocol. Secondly, and most interestingly, this federation model could be tested in Massive Open Online Courses (MOOC). Before this, a deep study on the load balance of these courses would be required, and only some particular courses could be achieved. But this integration could be used as an accelerator for remote laboratory developers to standardize the particular labs that are currently developing.

6.3 Final Remarks

Remote laboratories have been out there for more than 20 years. My efforts in this dissertation and during the 8 years that I have been working on WebLab-Deusto aim to take a humble step forward in sharing them, to improve the potential ecosystem of remote laboratories and the toolkits that instructors have for teaching students with real equipment. Interesting results have been presented, and new lines of work on top of these have been defined, which may disrupt the future of online STEM education.

Bibliography

Abdulwahed, M. and Nagy, Z. (2010). Developing the trilab, a triple access mode (hands-on, virtual, remote) laboratory, of a process control rig using labview and joomla. *Computer Applications in Engineering Education*.

Aktan, B., Bohus, C., Crowl, L., and Shor, M. (1996). Distance learning applied to control engineering laboratories. *Education, IEEE Transactions on*, 39(3):320–326.

Alves, G., Marques, M., Viegas, C., Lobo, C., Barral, R., Couto, R., Jacob, F., Ramos, C., Vilão, G., Covita, D., et al. (2011). Using visir in a large undergraduate course: Preliminary assessment results. In *Global Engineering Education Conference (EDUCON), 2011 IEEE*, pages 1125–1132. IEEE.

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

Arodzero, A. (1998). World wide student laboratory. Technical report.

Atkinson, I., du Boulay, D., Chee, C., Chiu, K., King, T., McMullen, D., Quilici, R., Sim, N., Turner, P., and Wyatt, M. (2006). Cima based remote instrument and data access: An extension into the australian e-science environment. In *e-Science and Grid Computing, 2006. e-Science'06. Second IEEE International Conference on*, pages 125–125. IEEE.

Bagnasco, A., Poggi, A., and Scapolla, A. (2006). A grid-based architecture for the composition and the execution of remote interactive measurements. In *e-Science and Grid Computing, 2006. e-Science'06. Second IEEE International Conference on*, pages 122–122. IEEE.

Bauer, P., Fedák, V., and Grijpink-van den Biggelaar, K. (2008). Application of elearning in electrical engineering. In *6th Int. Conference on Emerging eLearning Technologies and Applications*, pages 7–15.

Bogdanov, E., Limpens, F., Li, N., El Helou, S., Salzmann, C., and Gillet, D. (2012a). A social media platform in higher education. In *Global Engineering Education Conference (EDUCON), 2012 IEEE*, pages 1–8. IEEE.

Bogdanov, E., Ullrich, C., Isaksson, E., Palmer, M., and Gillet, D. (2012b). From lms to ple: A step forward through opensocial apps in moodle. *Advances in Web-Based Learning-ICWL 2012*, pages 69–78.

Bright, C., Lindsay, E., Lowe, D., Murray, S., and Liu, D. (2008). Factors that impact learning outcomes in remote laboratories. In *Proceedings of World Conference on Educational Multimedia Hypermedia and Telecommunications 2008*, pages 6251–6258. AACE: Association for Advancement of Computing in Education.

Callaghan, M., McCusker, K., Losada, J., Harkin, J., and Wilson, S. (2009). Teaching engineering education using virtual worlds and virtual learning environments. In *Advances in Computing, Control, & Telecommunication Technologies, 2009. ACT'09. International Conference on*, pages 295–299. IEEE.

Carisa, B., Burain, A., Molly H, S., and Lawrence, C. (1995). Running control engineering experiments over the internet.

Cedazo, R., Sanchez, F., Sebastian, J., Martínez, A., Pinazo, A., Barros, B., and Read, T. (2006). Ciclope chemical: a remote laboratory to control a spectrograph. *Advances in Control Education–ACE*, 6.

Chen, X., Song, G., and Zhang, Y. (2010). Virtual and remote laboratory development: A review. *Proceedings of Earth and Space*, pages 3843–3852.

Coble, A., Smallbone, A., Bhave, A., Watson, R., Braumann, A., and Kraft, M. (2010). Delivering authentic experiences for engineering students and professionals through e-labs. In *Education Engineering (EDUCON), 2010 IEEE*, pages 1085–1090. IEEE.

Corter, J., Esche, S., Chassapis, C., Ma, J., and Nickerson, J. (2011). Process and learning outcomes from remotely-operated, simulated, and hands-on student laboratories. *Computers & Education*.

Deaky, B., Zutin, D., and Bailey, P. (2012). A detailed view of the first android client application for the ilab shared architecture. In *Global Engineering Education Conference (EDUCON), 2012 IEEE*, pages 1–6. IEEE.

Del Alamo, J., Brooks, L., McLean, C., Hardison, J., Mishuris, G., Chang, V., and Hui, L. (2002). The mit microelectronics weblab: A web-enabled remote

laboratory for microelectronic device characterization. In *World Congress on Networked Learning in a Global Environment, Berlin (Germany)*.

del Alamo, J., Hardison, J., Mishuris, G., Brooks, L., McLean, C., Chan, V., and Hui, L. (2002). Educational experiments with an online microelectronics characterization laboratory. In *Proceedings Int. Conf. Eng. Educ.*

Diponio, M., Lowe, D., and de la Villefromoya, M. (2012). Supporting local access to collections of distributed remote laboratories. In *Proceedings of the Australasian Association for Engineering Education Conference*.

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

Ferreira, J., Cardoso, A., et al. (2005). A moodle extension to book online labs. *International Journal of Online Engineering*, 1(2).

Foster, I. and Kesselman, C. (2004). *The grid: blueprint for a new computing infrastructure*. Morgan Kaufmann.

Fowler, M. (2003). *Patterns of enterprise application architecture*. Addison-Wesley Professional.

Froyd, J., Wankat, P., and Smith, K. (2012). Five major shifts in 100 years of engineering education. *Proceedings of the IEEE*, 100(13):1344–1360.

Garbi Zutin, D. (2011). A visir lab server for the ilab shared architecture. *International Journal of Online Engineering (iJOE)*, 7(S1):pp–14.

Garcia-Zubia, J. and Alves, G. (2012). *Using Remote Labs in Education: Two Little Ducks in Remote Experimentation*. Universidad de Deusto.

Garcia-Zubia, J., Angulo, I., Hernandez, U., Castro, M., Sancristobal, E., Orduña, P., Irurzun, J., and de Garibay, J. (2010a). Easily integrable platform for the deployment of a remote laboratory for microcontrollers. In *Education Engineering (EDUCON), 2010 IEEE*, pages 327–334. IEEE.

Garcia-Zubia, J., Irurzun, J., Angulo, I., Hernández, U., Castro, M., Sancristobal, E., Orduña, P., and Ruiz-de Garibay, J. (2010b). Secondlab: A remote laboratory under second life. In *Education Engineering (EDUCON), 2010 IEEE*, pages 351–356. IEEE.

Garcia-Zubia, J., Irurzun, J., Angulo, I., Orduña, P., Ruiz-de Garibay, J., Hernández, U., and Castro, M. (2010c). Developing a second-life-based remote

lab over the weblab-deusto architecture. In *Proceedings of the Remote Engineering and Virtual Instrumentation Conference (REV2010), Stockholm, Sweden*, pages 171–176.

Garcia-Zubia, J., López-de Ipiña, D., Hernández, U., Orduña, P., and Trueba, I. (2007a). An approach for weblabs analysis. *International Journal of Online Engineering*, 3(2).

Garcia-Zubia, J., López-de Ipiña, D., Hernández, U., Orduña, P., and Trueba, I. (2007b). Weblab-gpib at the university of deusto. In *Proceedings of the REV 2007 Conference*, pages 25–27.

Garcia-Zubia, J., López-de Ipiña, D., and Orduña, P. (2005). Towards a canonical software architecture for multi-device weblabs. In *Industrial Electronics Society, 2005. IECON 2005. 31st Annual Conference of IEEE*, pages 6–pp. IEEE.

Garcia-Zubia, J., Orduña, P., Angulo, I., Hernandez, U., Dziabenko, O., López-Ipiña, D., and Rodriguez-Gil, L. (2011). Application and user perceptions of using the weblab-deusto-pld in technical education. In *Global Online Laboratory Consortium Remote Laboratories Workshop (GOLC), 2011 First*, pages 1–6. IEEE.

Garcia-Zubia, J., Orduña, P., Angulo, I., Irurzun, J., and Hernández, U. (2008). Towards a distributed architecture for remote laboratories. *International Journal of Online Engineering (iJOE)*, 4(0):pp–11.

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

Garrett, J. et al. (2005). Ajax: A new approach to web applications.

Gillet, D. and Bogdanov, E. (2012). Personal learning environments and embedded contextual spaces as aggregator of cloud resources.

Gillet, D., El Helou, S., Yu, C., and Salzmann, C. (2008). Turning web 2.0 social software into versatile collaborative learning solutions. In *Advances in Computer-Human Interaction, 2008 First International Conference on*, pages 170–176. IEEE.

Gillet, D., Latchman, H., Salzmann, C., and Crisalle, O. (2001). Hands-on laboratory experiments in flexible and distance learning. *Journal of Engineering Education*, 90(2):187–191.

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

Gravier, C., Fayolle, J., Bayard, B., Ates, M., and Lardon, J. (2008). State of the art about remote laboratories paradigms-foundations of ongoing mutations. *iJOE*, 4(1).

Gustavsson, I., Nilsson, K., Zackrisson, J., Garcia-Zubia, J., Hernandez-Jayo, U., Nafalski, A., Nedic, Z., Gol, O., Machotka, J., Pettersson, M., et al. (2009). On objectives of instructional laboratories, individual assessment, and use of collaborative remote laboratories. *Learning Technologies, IEEE Transactions on*, 2(4):263–274.

Gustavsson, I., Zackrisson, J., Håkansson, L., Claesson, I., and Lagö, T. (2007). The visir project–an open source software initiative for distributed online laboratories. In *Proceedings of the REV 2007 Conference, Porto, Portugal*.

Hardison, J., DeLong, K., Bailey, P., and Harward, V. (2008). Deploying interactive remote labs using the ilab shared architecture. In *Frontiers in Education Conference, 2008. FIE 2008. 38th Annual*, pages S2A–1. IEEE.

Harward, V., del Alamo, J., et al. (2008a). Building an ecology of online labs. In *Proceeding of the International Conference on Interactive Collaborative Learning - ICL2008*.

Harward, V., del Alamo, J., Lerman, S., Bailey, P., Carpenter, J., DeLong, K., Felknor, C., Hardison, J., Harrison, B., Jabbour, I., Long, P., Mao, T., Naamani, L., Northridge, J., Schulz, M., Talavera, D., Varadharajan, C., Wang, S., Yehia, K., Zbib, R., and Zych, D. (2008b). The ilab shared architecture: A web services infrastructure to build communities of internet accessible laboratories. *Proceedings of the IEEE*, 96(6):931–950.

Henry, J. (1996). Running laboratory experiments via the world wide web. In *ASEE Annual Conference*.

Hernández-Jayo, U. (2012). Metodología de control independiente de instrumentos y experimentos para su despliegue en laboratorios remotos. *PhD dissertation*.

Hernández-Jayo, U. and Garcia-Zubia, J. (2012). Control methodology independent of the experiments to be deployed in remote labs of analog electronic. In *Global Engineering Education Conference (EDUCON), 2012 IEEE*, pages 1–6. IEEE.

Hernandez-Jayo, U., Garcia-Zubia, J., Angulo, I., López-de Ipiña, D., Orduña, P., Irurzun, J., and Dziabenko, O. (2010). Lxi technologies for remote labs: An extension of the visir project. *International Journal of Online Engineering (iJOE)*, 6(5):pp–25.

Jona, K. and Vondracek, M. (2013). A remote radioactivity experiment. *The Physics Teacher*, 51:25–27.

Kotulski, T. and Murray, S. (2010). The national engineering laboratory survey. *Labshare Project. December*.

Langmann, R. (2004). E-learning & doing-a remote lab network for the education on the engineering. *Proceedings of the TELEC, Santiago de Cuba*.

Langmann, R. (2005). Web-based remote control by liveconnect. *International Journal of Online Engineering (iJOE)*, 1(1).

Li, N., El Helou, S., and Gillet, D. (2012). Using social media for collaborative learning in higher education: A case study. In *ACHI 2012, The Fifth International Conference on Advances in Computer-Human Interactions*, pages 285–290.

López-de Ipiña, D., Garcia-Zubia, J., and Orduña, P. (2006). Remote control of web 2.0-enabled laboratories from mobile devices. In *e-Science and Grid Computing, 2006. e-Science'06. Second IEEE International Conference on*, pages 123–123. IEEE.

Lowe, D., Conlon, S., Murray, S., Weber, L., de la Villefromoy, M., Lindsay, E., Nafalski, A., Tang, T., and Nageswaran, W. (2011). Labshare: Towards cross-institutional. *Internet Accessible Remote Laboratories: Scalable E-Learning Tools for Engineering and Science Disciplines*, page 453.

Lowe, D., de la VilJefromoy, M., Jona, K., and Yeoh, L. (2012a). Remote Laboratories: Uncovering the True Costs. In *Remote Engineering Virtual Instrumentation 2012, REV 2012*.

Lowe, D., Machet, T., and Kostulski, T. (2012b). Uts remote labs, labshare, and the sahara architecture. *Using Remote Labs in Education: Two Little Ducks in Remote Experimentation*, page 403.

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

Lowe, D. and Orou, N. (2012). Interdependence of booking and queuing in remote laboratory scheduling . In *Remote Engineering Virtual Instrumentation 2012, REV 2012*.

Lowe, D. and Yeung, H. (2011). Remote laboratory systems interoperability standard: Interface definitions. RFC of the Technical Committee of the Global Online Laboratory Consortium.

Lyalina, Y., Makarov, O., and Langmann, R. Smart labs for remote experimentation in communication technologies. In *Proceedings of the REV 2011 Conference, Brasov, Romania*.

Machotka, J., Nedic, Z., Nafalski, A., and Göl, Ö. (2010). Collaboration in the remote laboratory netlab. In *1st WIETE Annual Conference on Engineering and Technology Education*, pages 22–25.

Maiti, A. (2010a). Netlab: An online laboratory management system. In *Education Engineering (EDUCON), 2010 IEEE*, pages 1351–1358. IEEE.

Maiti, A. (2010b). Time scheduling schemes in online laboratory management systems. *International Journal of Online Engineering (iJOE)*, 6(4):pp–44.

Maiti, A. (2011). A hybrid algorithm for time scheduling in remotely triggered online laboratories. In *Global Engineering Education Conference (EDUCON), 2011 IEEE*, pages 921–926. IEEE.

Marcelino, R., Silva, J., Alves, G., and Shaeffer, L. (2010). Extended immersive learning environment: A hybrid remote/virtual laboratory. *International Journal of Online Engineering (iJOE)*, 6(5):pp–46.

Mateos, V., Gallardo, A., Richter, T., Bellido, L., Debicki, P., and Villagrá, D. (2011). Lila booking system: Architecture and conceptual model of a rig booking system for on-line laboratories. *International Journal of Online Engineering (iJOE)*, 7(4):pp–26.

Muros-Cobos, J. and Holgado-Terriza, J. (2012). RVLab: A server-side framework to build remote and virtual laboratories. In *Remote Engineering Virtual Instrumentation 2012, REV 2012*.

Nedic, Z. and Machotka, J. (2007). Remote laboratory netlab for effective teaching of 1st year engineering students. *International Journal of Online Engineering (iJOE)*, 3(3).

Nedic, Z., Machotka, J., and Nafalski, A. (2003). Remote laboratories versus virtual and real laboratories. In *Frontiers in Education, 2003. FIE 2003. 33rd Annual*, volume 1, pages T3E–1. IEEE.

Nedic, Z., Machotka, J., and Nafalski, A. (2008). Remote laboratory netlab for effective interaction with real equipment over the internet. In *Human System Interactions, 2008 Conference on*, pages 846–851. IEEE.

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

Orduña, P. and Garcia-Zubia, J. (2011). Scheduling schemes among internet laboratories ecosystems.

Orduña, P., Garcia-Zubia, J., Irurzun, J., Sancristobal, E., Martín, S., Castro, M., López-de Ipiña, D., Hernández, U., Angulo, I., and González, J. (2009). Designing experiment agnostic remote laboratories. *Remote Engineering and Virtual Instrumentation. Bridgeport, CT, USA*.

Orduña, P., Garcia-Zubia, J., López-de Ipiña, D., and Irurzun, J. (2011a). Accessing remote laboratories from mobile devices. *Open Source Mobile Learning: Mobile Linux Applications*, page 233.

Orduña, P., Garcia-Zubia, J., Rodriguez-Gil, L., Irurzun, J., López-de Ipiña, D., and Gazzola, F. (2012a). Using labview remote panel in remote laboratories: Advantages and disadvantages. In *Global Engineering Education Conference (EDUCON), 2012 IEEE*, pages 1–7. IEEE.

Orduña, P., Irurzun, J., Rodriguez-Gil, L., Garcia-Zubia, J., Gazzola, F., and López-de Ipiña, D. (2011b). Adding new features to new and existing remote experiments through their integration in weblab-deusto. *International Journal of Online Engineering (iJOE)*, 7(S2):pp–33.

Orduña, P., Rodriguez-Gil, L., Angulo, I., Dziabenko, O., López-de Ipiña, D., and Garcia-Zubia, J. (2012b). Exploring students collaboration in remote laboratory infrastructures . In *Remote Engineering Virtual Instrumentation 2012, REV 2012*.

Payne, L. and Schulz, M. (2013). JAVA Implementation of the Batched iLab Shared Architecture . In *Remote Engineering Virtual Instrumentation 2013, REV 2013*.

Ponta, D., Scapolla, A., and Buschiazzo, P. (1861). Survey of remote laboratories using service oriented architectures. *International Journal of Online Engineering (iJOE). ISSN*, 2121.

Qiao, Y., Liu, G., Zheng, G., and Hu, W. (2010). Ncslab: A web-based global-scale control laboratory with rich interactive features. *Industrial Electronics, IEEE Transactions on*, 57(10):3253–3265.

Raman, R., Nedungadi, P., et al. (2011). Integrating collaboration and accessibility for deploying virtual labs using vlcap.

Richter, T., Boehringer, D., and Jeschke, S. (2011a). Lila: A european project on networked experiments. *Automation, Communication and Cybernetics in Science and Engineering 2009/2010*, pages 307–317.

Richter, T., Tetour, Y., and Boehringer, D. (2011b). Library of labs-a european project on the dissemination of remote experiments and virtual laboratories. In *Multimedia (ISM), 2011 IEEE International Symposium on*, pages 543–548. IEEE.

Rochadel, W., da Silva, S., da Silva, J., Luz, T., and Alves, G. (2012). Utilization of remote experimentation in mobile devices for education. In *Global Engineering Education Conference (EDUCON), 2012 IEEE*, pages 1–6. IEEE.

Safaric, R., Truntič, M., Hercog, D., and Pačnik, G. (2005). Control and robotics remote laboratory for engineering education. *International Journal of Online Engineering (iJOE)*, 1(1).

Salzmann, C., Latchman, H., Gillet, D., and Crisalle, O. (1998). Requirements for real time laboratory experimentation over the internet. In *International Conference on Engineering Education, ICEE. Rio de Janeiro, Brazil*, pages 17–20.

Sancristobal, E. (2010). Metodología, estructura y desarrollo de interfaces intermedias para la conexión de laboratorios remotos y virtuales a plataformas educativas. *PhD dissertation*.

Sancristobal, E., Castro, M., Harward, J., et al. (2010). Integration view of web labs and learning management systems. In *Education Engineering (EDUCON), 2010 IEEE*, pages 1409–1417. IEEE.

Sarukkalige, R., Lindsay, E., and Anwar, A. (2010). Laboratory demonstrators' perceptions of the remote laboratory implementation of a fluid mechanics laboratory.

Scheucher, T., Bailey, P., Gütl, C., and Harward, V. (2009). Collaborative virtual 3d environment for internet-accessible physics experiments. In *Proceedings of the International Conference of Remote Engineering and Virtual Instrumentaion*.

Schulz, M. and Long, P. (2012). Possible futures for remote laboratories. In Azad, A., Auer, M., and Harward, V., editors, *Internet Accessible Remote Laboratories: Scalable E-Learning Tools for Engineering and Science Disciplines*, pages 493–510. IGI Global.

Sotomayor Basilio, B. (2011). *Provisioning computational resources using virtual machines and leases*. PhD thesis, University of Chicago.

Sousa, N., Alves, G., and Gericota, M. (2010). An integrated reusable remote laboratory to complement electronics teaching. *Learning Technologies, IEEE Transactions on*, 3(3):265–271.

Tawfik, M., Martin, S., Gil, R., Losada, P., Pesquera, A., Sancristobal, E., Diaz, G., Peire, J., and Castro, M. (2011a). Visir installation and start-up guide.

Tawfik, M., Sancristobal, E., Martin, S., Gil, C., Losada, P., Diaz, G., and Castro, M. (2011b). Remote laboratories for electrical & electronic subjects in new engineering grades. In *Promotion and Innovation with New Technologies in Engineering Education (FINTDI), 2011*, pages 1–6. IEEE.

Tawfik, M., Sancristobal, E., Martín, S., Gil, C., Pesquera, A., Losada, P., Díaz, G., Peire, J., Castro, M., Garcia-Zubia, J., et al. (2011c). Visir deployment in undergraduate engineering practices. In *Global Online Laboratory Consortium Remote Laboratories Workshop (GOLC), 2011 First*, pages 1–7. IEEE.

Tawfik, M., Sancristobal, E., Martin, S., Gil, C., Pesquera, A., Losada, P., Diaz, G., Peire, J., Castro, M., Garcia-Zubia, J., et al. (2012). Visir: Experiences and challenges. *International Journal of Online Engineering (iJOE)*, 8(1):pp–25.

Torres, F., Candelas, F., Puente, S., Pomares, J., Gil, P., and Ortiz, F. (2006). Experiences with virtual environment and remote laboratory for teaching and learning robotics at the university of alicante. *International Journal of Engineering Education*, 22(4):766–776.

Wuttke, H. (2006). A mixed-reality environment for digital control systems. *International Journal of Online Engineering (iJOE)*, 2(2).

Yeung, H., Lowe, D., and Murray, S. (2010). Interoperability of remote laboratories systems. *International Journal of Online Engineering (iJOE)*, 6(5):pp–71.

Zackrisson, J. and Svahnberg, C. (2008). Openlabs security laboratory-the online security experiment platform. *International Journal of Online Engineering*, 4(special issue: REV2008):63–68.

Zutin, D. and Auer, M. (2011). Work in progress—integrating educational online lab platforms around the ilab shared architecture. In *Frontiers in Education Conference (FIE), 2011*, pages F1G–1. IEEE.

Zutin, D., Auer, M., Maier, C., and Niederstatter, M. (2010). Lab2go—a repository to locate educational online laboratories. In *Education Engineering (EDUCON), 2010 IEEE*, pages 1741–1746. IEEE.

This dissertation was finished writing in Bilbao on April 8^{th}, 2013