TESIS DOCTORAL

2019

# A new model to enable a remote connection with hardware devices using Javascript

Jacobo Sáenz Valiente

PROGRAMA DE DOCTORADO EN INGENIERÍA DE SISTEMAS Y CONTROL

Prof. Sebastián Dormido Bencomo (UNED)

Prof. Luis de la Torre Cubillo (UNED)

Prof. Jesús Chacón Sombría (UCM)

**Department** Informática y Automática

**Faculty** E.T.S. de Ingeniería Informática

**Dissertation Title**

**Author** Jacobo Sáenz Valiente

**Academic Degree** M.Sc. Systems and Control Engineering
Univ. Nacional de Educación a Distancia

**Advisors** Ph.D. Sebastián Dormido Bencomo
Ph.D. Luis de la Torre Cubillo
Ph.D Jesús Chacón Sombría

**To *Durin*, the one who will wake again from sleep**

# Acknowledgments

Writing these lines I realize how fast the time has past. It is incredible the amount of good experiences lived during this years and I want to thank all these people who have helped and supported me during this journey.

Foremost, thanks to my supervisors, who guided me and shared their knowledge with me. Thanks for your advices, patience and expertise. I'd like to express my gratitude to Sebastián Dormido for the possibility to work with the research group and for receiving me with open arms. Thanks to Luis de la Torre, who helped me to immerse myself in the research and to forget my initial fears about speaking in English. Thanks to Jesus Chacón, for his friendship and for being there to share his work and ideas; we have many things to build and many topics of martial arts to discuss about.

I want to say thanks to Eva Besada, who taught me in advanced control and gave me the possibility to collaborate in some projects when I was just a student.

Thanks to Francisco Esquembre for the guided tours to the ins and outs of Easy Java Javascript simulations, he advised me and provided good ideas to work with.

I'd also like to express my gratitude to Juan Pedro Sánchez and Jaime Arturo de la Torre, they have been an incredible source of knowledge.

I would thank the whole research team of the Department of Informática and Automática, for their presence and help, specially to Rubén, Ernesto Aranda, Dictino, Miguel Angel, María, Daniel, David and Victorino. Thanks to Jose Sanchez for his help on obtaining a scholarship to continue with my PhD and for his support during these years. Thanks to Raquel, she knows that I forget many things and she have been always there to remember me those things. Thanks to Ernesto Fabregas, he guided me to understand how to walk the way and to focus in problems one by one. Thanks to Pilar and María José, they have known always how to solve our problems.

I'd express my gratitude to Christophe Salzmann, Denis Gillet and his research group, thanks to invite me to stay in your university and to share you knowledge about virtual and remote labs. My stay in Lausanne was a pleasure thanks to you.

I want to thank the Paulo Abreu and Teresa Restivo from *Universidade do Porto*,

they invite me to their research group to work with pneumatics. They share their knowledge about making virtual and remote labs with me. Thanks to Daniel, he help me so much in Porto and it was amazing.

I'd like to thank my friends, those from my childhood and from university. A special thank to Mario, Elvira, Alba, Helena and Lucas. Thanks Mario to support me in the bad moments and for all the good times together. Thanks to Andrea, you made me a better person. Thanks to Teresa and Vicente, you encouraged me to be a "*mad*" scientist.

Thanks to my family, you have been always there helping me in the stressing moments, thanks to Salome, Patricia, Triana, Jose, Pancha and Luna. Thanks to my mother, Salome, to put my feet on the ground during my whole life and during this journey, for making me the person who I am. I could have never learned the multiplication table without you. Thanks to my big sister, Patricia, for taking care of us and to show us how to make things with our hands. Thank to my little sister, for being Tri, and for knowing what I am thinking, finish my sentences and to help me to forget the problems with gladiators afternoons.

All these people have supported me in many ways, and if I have forgotten somebody, thanks to you also =).

# Contents

*Part I PRELIMINARIES* 1

*1. State of Art* . . . 3
1.1 Online Resources in Education . . . 3
1.1.1 Web-Based Courses . . . 4
1.1.2 Web Platforms for Online Courses . . . 5
1.2 VRLs in Web-Based Courses . . . 7
1.2.1 Virtual labs . . . 10
1.2.2 Remote labs . . . 11
1.3 Architectures and approaches . . . 11
1.3.1 Hardware-to-network . . . 12
1.3.2 Network-to-device . . . 15
1.3.3 Programming languages . . . 19

*2. Easy Java Javascript Simulations (EjsS)* . . . 21
2.1 EjsS: The tool . . . 21
2.2 Applications . . . 23
2.2.1 Enabled versions . . . 23
2.2.2 Java-enabled version and Javascript-enabled version . . . 24
2.3 Architecture of a VRL inside UNILabs . . . 24

*Part II Development* 27

*3. Problem Discussion* . . . 29
3.1 Controller Location . . . 29
3.2 Model and view dependency . . . 32
3.3 Computational cost . . . 35

*4. An Architecture for VRLs* . . . 39
4.1 Decoupling model and view . . . 39
4.2 Communications . . . 41
4.2.1 Technologies . . . 42
4.3 Client-Server Structure . . . 45
4.3.1 Client Side . . . 46
4.3.2 Server Side . . . 47
4.4 Programming Languages . . . 48
4.4.1 Java . . . 48
4.4.2 Javascript . . . 50

*5. EjsS Implementation* . . . 53
5.1 Easy Java & Javascript . . . 53
5.2 View side . . . 56
5.2.1 HTML+Javascript . . . 56
5.2.2 Initialization process . . . 59
5.2.3 Auto-generation process . . . 61
5.2.4 Update process . . . 71
5.3 Model side . . . 72
5.3.1 EjsS Model . . . 72
5.3.2 Non-EjsS Models . . . 79
5.4 Timeline in a VRL session . . . 80
5.5 EjsS Editor . . . 85
5.5.1 SDS element . . . 85

*6. Advantages and limitations of implementation* . . . 91
6.1 Advantages . . . 91
6.1.1 Multiuser sessions . . . 91
6.1.2 Multiview sessions . . . 92
6.1.3 Reutilization . . . 92
6.2 Limitations . . . 93
6.2.1 Methods or dynamic calculus . . . 93
6.2.2 Access to resources . . . 93
6.2.3 View elements association . . . 93
6.2.4 Communication speed and delays . . . 94

*7. New Elements and additional features* . . . 97
7.1 Performance and parameterization . . . 97
7.1.1 Minimum information exchange . . . 97

7.1.2 Reconfigurable elements . . . 98
7.1.3 Quality of Service . . . 100
7.2 J&JS as elements . . . 100
7.2.1 Java SDS and Javascript SDS Elements . . . 100

*Part III Developed Labs* 103

*8. Virtual Labs* . . . 105
8.1 Virtual labs developed in Javascript . . . 106
8.1.1 Two coupled electric drives . . . 106
8.1.2 The vibrating wires . . . 113
8.2 Virtual labs developed in Java & Javascript . . . 119
8.2.1 The servo motor lab . . . 119
8.2.2 Planar parallel robots lab . . . 122
8.2.3 Heatflow lab . . . 127

*9. Remote Labs* . . . 133
9.1 Remote labs developed in Javascript . . . 133
9.1.1 Two coupled electric drives . . . 134
9.1.2 The vibrating wires . . . 138
9.2 Remote labs developed in Java & Javascript . . . 144
9.2.1 The servo motor lab . . . 144
9.2.2 Heatflow lab . . . 147

*10. Conclusions and Future Work* . . . 153
10.1 Conclusions . . . 153
10.1.1 Software results . . . 153
10.1.2 Developed VRLs results . . . 154
10.2 Future Work . . . 155

*Appendix* 157

*A. Coupled Electric Drives Mathematical Model* . . . 159

*B. Vibrating Strings Mathematical Model* . . . 163

*C. Servo Motor Mathematical Model* . . . 167

*D. Document: How to change a lab from Java to Java&Javascript* . . . . . 171
D.1 First steps . . . . . 171
D.1.1 Locate your files . . . . . 171
D.1.2 Open your file . . . . . 172
D.1.3 Save with new extension . . . . . 172
D.2 Prevent problems and create the GUI . . . . . 172
D.2.1 Reduce errors . . . . . 174
D.2.2 Create you GUI . . . . . 175
D.2.3 Get ready to use it . . . . . 176

*E. Full Metadata* . . . . . 177

# Abbreviations

*API* Application Programming Interface. 34, 35

*CRUD* Create, Read, Update, and Delete. 18

*CSDB* Server, Client and DataBase. 13, 14

*CSS* Cascading Style Sheets. 56

*DBaaS* Database as a service. 45

*DC* Direct Current. 106

*DOF* Degree Of Freedom. 123

*EjsS* Easy Java JavaScript Simulations. 21–26, 55, 56, 58, 62, 64, 66, 68, 71–74, 78, 79, 81, 83, 85–88, 91–93, 123, 140, 154

*GUI* Graphical User Interface. 7, 22, 29, 34, 43, 44, 46, 55–58, 61–63, 68, 71, 75–77, 79–84, 92, 93, 105, 108, 109, 116, 118, 120, 125, 129, 133, 136, 140, 141, 146, 149

*HTTP* Hypertext Transfer Protocol. 15, 16, 18, 19, 73, 82

*HTTPS* Hypertext Transfer Protocol Secure. 15, 16

*IaaS* Infrastructure as a Service. 45

*IoT* Internet of Things. 36

*JSON* JavaScript Object Notation. 17, 18, 42–45, 55, 56, 63–65, 67, 70, 75, 77, 79, 83, 87, 93, 134, 135

*LaaS* Lab as a Service. 45

*LMS* Learning Management System. 4–6

*MIMO* Multiple Input Multiple Output. 106

*MOOC* Massive Open Online Courses. 4

*NCS* Networked Control Systems. 30, 31, 46

*NFV* Network Function Virtualization. 37

*RDF* Resource Description Framework. 17

*REST* Representational State Transfer. 18, 19

*RL* Remote Lab. 6, 137

*RPC* Remote Procedure Call. 18, 19, 134, 140

*SaaS* Sensing as a Service. 45

*SDN* Software-Defined Network. 37

*SOAP* Simple Object Access Protocol. 18, 19

*STEM* Science, Technology, Engineering and Mathematics. 4, 5, 7

*TCP* Transmission Control Protocol. 15, 16, 31, 42

*UDP* User Datagram Protocol. 31

*UNILabs* University Network of Interactive Labs. 6, 24–26

*UPS* Uninterruptible Power Supply. 145, 148

*VL* Virtual Lab. 10, 11, 26, 109, 111, 137

*VRL* Virtual and/or Remote Labs. 5–7, 11, 12, 15, 18–22, 25, 26, 29, 30, 34, 40–44, 46, 55, 79, 91, 94, 125, 141

*YAML* YAML Ain't Markup Language. 17

# Part I

# PRELIMINARIES

# 1. State of Art

Internet hosts millions of sites and it is home of other resources such as animations, video, simulations or laboratories which are ready to use or consult at any moment. This vast ocean of resources is changing how learning takes place and, in this sense, schools and universities are also changing the concept of education. These changes on teaching methods affect also how the laboratory practices and the experimental procedures are taught. This research work focuses in how the experimental laboratory practices are done through the Internet.

## 1.1. Online Resources in Education

Nowadays, online resources for education have become common in most courses, both in distance education and in classroom-based courses [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]. Those resources can illustrate concepts, explain theoretical notions or be a complement to a speech. As it is known, *a picture is worth a thousand words*, but, when pictures are not enough we can take advantage of the Internet. In this regard, distance education universities, as UNED, found in Internet a breeding ground for developing their applications and courses, taking advantage of the advances in communication and computation technology to make the learning process easier.

The hands-on learning, where the student must be in place, is a proven teaching method, but widespread of new devices and the great interest of the students in remote learning is awakening more and more ways of teaching. On the one hand, online resources are becoming common in student life-cycle, then is easy to add more capabilities to these resources as interactivity or dynamics. On the other hand, the learning scheme is broadening, and in some cases defining full online pedagogical scenarios. Those scenarios are not anymore complementary materials to use in, or outside, the classroom, but full and self-contained courses with links to other resources and in some cases, they even provide experimentation capabilities. Next subsections are dedicated to these web-based or online courses.

### 1.1.1. Web-Based Courses

Technology has become an important part of our life and a main component of most activities, therefore, it is also true on web-based courses, that can be found today all over the Internet in many different shapes. Online courses have not changed only how the users access to materials, this technology also changed how the students plan their studies during the week [12]. It has influenced also the age-range of students [1] and the economic status of students [13]. Regarding this, every year there are new students coming to university/school to take classes, and many of them have grown up with the Internet. Some courses now include the use of Learning Management System (LMS), where the students become familiar with online education. Some universities have widen their niches developing Massive Open Online Courses (MOOC), where students can acquire their skills at distance. *Educational social media* networks have also influenced the way of education and sharing is perceived, and those sites conform a good engagement for new students, due to their familiarity with social networks. The different types of Web-based courses are not well established, and therefore, the presented types in last lines are not unique neither independent and sometimes can be found in any combination [14, 15].

Online and classroom education have a wide variety of subjects available and the areas of knowledge are not restricted by the online education nature. Although, the work, exercises, practices and processes are not equally deployed inside a webpage for an engineering or literature subject. As this document cannot cover the broad spectrum of education, we will focus in Science, Technology, Engineering and Mathematics (STEM) courses, where the experimental laboratory practices are very common.

A key element inside STEM subjects is the need to learn theoretical concepts which are applied in practice shaped like procedures, examples and practices. A common approach to teach those real-world applications is to use: exercises, tests and laboratories. Laboratories and simulations are the main differentiating element with non-STEM areas, and conventionally are on-site group classes. As these experiences cannot be done without the lab equipment, it is posed a problem to the remote education in its early days. Nowadays, remote experimentation is possible and common on remote education paradigm. Online accessible labs appeared to enhance the performance of remote education, allowing the student to pass their laboratory practices by means of simulations and remote laboratories. On this matter, remote experimentation plays an important role inside STEM area and web-based courses.

---

[1] https://www.insidehighered.com/news/2014/06/06/one-semester-students-satisfied-unfinished-georgia-tech-online-degree-program

Web-based courses have reached a high level of acceptance inside the student community, therefore, many education institutions have launched their courses online. As a consequence, online courses platforms have recently appeared as websites to deploy, share and maintain courses from different sources: universities, schools and other education institutions. Next section will describe this platforms and will introduce some of them.

### 1.1.2. Web Platforms for Online Courses

Online courses platforms have been appearing and growing from 1960 [16] to nowadays, adding new courses everyday and motivating student to widen their knowledge. These platforms are designed in different ways depending on the number of students expected and how resources are shared. Because it is impossible to name all of the platforms which are offering online courses, we will name some web-based courses inside the STEM area and with Virtual and/or Remote Labs (VRL) available.

**Graasp** is a website which contains lots of spaces to share material resources, teaching experiences. It allows teachers to create and organize their online resources, like VRLs. Inquiry learning spaces[2] can also be found in this platform, as full learning sequences about a specific subject. Therefore, students or teachers can find in Graasp apps, labs, materials and full pedagogical frameworks.

Massive learning is now present in the form of different websites. MOOCs inside these sites contain videos to teach in a equivalent way as classical classrooms using independent teaching modules and sometimes accessible remote labs or links to them [3]. Some of those sites are **Coursera** [4] , **EDX** [5] and **MiriadaX** [6].

**The Go-Lab Ecosystem**[7] *targets science teachers from primary and secondary schools and aims to help them enrich their teaching practices with innovative teaching approaches and supportive technical tools* [8]. Go-Lab is divided in two parts, one dedicated to sharing VRLs from different institutions and the other one to develop, maintain and share inquiry learning spaces.

LMSs are server applications that offer many tools related with online distance education. Some plugins and tools help to deploy courses or materials, add docu-

---

[2] http://www.ceebl.manchester.ac.uk/ebl/
[3] https://moocs.epfl.ch
[4] https://www.coursera.org
[5] https://www.edx.org/es
[6] https://miriadax.net/home
[7] https://www.golabz.eu
[8] http://support.golabz.eu/about

mentation, configure of maintenance procedures, tracking the usage and other functionalities. Therefore, LMSs are mainly designed to help in deployment of online resources or full Web-based courses. Moodle[9] is a LMS which is used by many universities and undergraduate courses. Those courses are inside a website where enrolled students and teachers can navigate, read, download or interact with the resources embedded there. Not all the courses include online VRLs in their subjects, however, the text will briefly introduce a couple of them. The first one, the ***Laboratório de Instrumentação para Medição*** **(LIM)** [10], that is a Moodle based site, developed by the Porto University, which contains 8 VRLs inside the mechanical engineering area. The second one, University Network of Interactive Labs (UNILabs), all the examples of chapters 8 and 9 are deployed in this website, next subsection will talk about this Moodle based Website.

### UNILabs platform

UNILabs is the University Network of Interactive Labs. It was born on September 2013 from a previous project, UNEDLabs [17], which originated from another project [18]. This project was created inside the *"Informática y Automática"* department, at *Escuela Técnica Superior de Ingeniería Informática*, Madrid, Spain, that has been working in the area of VRLs experimentation since nineties.

It is a network formed by a large number of universities that share their VRLs in form of online laboratories which cover several topics: Physics, control theory, electronics, etc. On the one hand, it is frequent that each existing Remote Lab (RL) is based on different hardware equipments. On the other hand, it is also usual to find VRLs applications in UNILabs that were built and deployed using similar technologies. In this regard, all of the labs inside UNILabs share the same three free and open source applications: Moodle (the LMS that supports the web content of the open course in which the VRLs are), Easy Java Simulations (for building the VRLs applications), and middleware software created in the department (RIP, NodeJSONRPCElement, LabviewElement, etc).

UNILabs already offers fifteen courses from eight different universities with around 30 VRLs and contains many control education online laboratories that are complementary to university courses and also an *Open course* that can be visited to see some online labs, without needing a student or teacher account.

---

[9] https://moodle.org/?lang=es
[10] https://limserver.fe.up.pt

## 1.2. VRLs in Web-Based Courses

VRLs are included in some online courses and are unvaluable digital resources in Engineering education [19, 20, 21, 22]. Lab sessions play a critical role in the STEM area, where the student needs to learn the practical behavior of complex theoretical subjects, practice laboratory skills and learn protocols that can be applied in their future careers. To fulfill that needs, web-based scientific courses must include remote experimentation or simulations that provide that knowledge and laboratory capabilities. Once a laboratory objective is defined by the teaching team (like learning about the system itself or about the procedure to obtain data), lab practices may follow a common scheme. Without regard to the nature of the lab, remote or simulation, lab practices of a web-based course usually follow a common structure:

1. Book in advance to access the lab, following the schedule rules.
2. Access the lab activity.
3. Interact with the lab in order to obtain the data, learn the procedure or both.
4. Use the information gathered from the lab. This may imply analyze the data in a qualitative or quantitative manner and/or complete additional tasks to widen the knowledge about the lab system.
5. Make a laboratory report to consolidate the acquired skills and knowledge.

Consequently, if the purpose of the VRLs is to be used as teaching tool in the same way as real hands-on lab, it is the duty of teacher and developers to enhance their system to be satisfactory learning tools. In a certain way, a VRL replicates the hands-on lab, and in the same way, experimentation is usually related to *touch and see* real systems, in the sense that touching lab elements produces changes which must be perceived by the student. To reproduce the feeling of a real interaction with equipment is not an easy task and "the psychology of presence may be as important as technology" [23].

Building a Graphical User Interface (GUI) to give the feeling of being physically in a laboratory it not immediate, but can be associated to how the data is obtained and presented to the user. Therefore, when, how and how much data needs to be exchanged is a key factor. The following example illustrates, step by step, the process of building up this *consciousness* of being working with a lab:

- Think about a student going through a lab practice with only a notebook and a sheet of paper with previously generated data. But, is not the best possible

laboratory, as long as there are no interaction, no dynamic change and is textual, with no graphical elements.

- One step forward, the notebook and the paper sheet may be replaced by a mobile device, where the screen data is changing with the time, this lab is still not so interactive but **dynamic**, leaving to the user the data analysis.

- The data is already dynamic but it is still textual. A chart to visualize the data enhances the student experience with the lab, as now is also a **graphical** environment.

- Adding an animation or, even better, a webcam to monitor the real object takes the lab to the next level, making a conceptual link between both, the interface and the real system. However, up to this point, the data flow is unidirectional, as it is flowing from the system to the user.

- Adding fields, buttons, sliders or other modifiable elements improves the sensation of interactivity and free will. Now, the lab is actually a remotely operated system and fits perfectly with the definition of laboratory.

| | Local | | Remote | | |
|---|---|---|---|---|---|
| Real | Hands-on | Data-sets | Real Time | Near Real Time | Batch |
| Non-Real | Math Simulation | | Data-sets | | |

**Table 1.1** Different data sources to be used in a online lab, it specifies if the data source is real or non-real equipment

Laboratory users, whether they are students, researchers or any other role, can find over the Internet different ways of experimentation, from a monitoring tool to interactive user interface. Therefore, the way they carry out their experiences can vary greatly from one to another, but in every case a lot of data is usually generated and gathered. How all these data are obtained may affect the global structure of the lab, the goal of the experiments or the pedagogical framework. Therefore it is important to discuss data sources. Table 1.1 contains a summary of all these data sources:

- **Data from real equipment**: Information provided by real machines, sensors, actuators and so on. How is it acquired imposes three categories:

- *Hands-on*: This is the classical experimental situation, where the students physically interact with the real equipment. The acquisition is made in-situ and the data gathered depends-on the user interaction.
- *Data-sets*: The user is not actually performing the experiment, but rather is working on data previously gathered from another lab experience. This approach is common when the process is complex to reproduce or when the objective of the lab practice is not to learn the process but to extract information from the data.
- *Remote*: Remotely accessible laboratories have become so common and the widespread of this kind of experimentation have driven the launch of different approaches:
  - *Real time*: It is known as hard real time, and considers that if some data is not acquired following the schedule it can cause the system failure. These systems involve solid timing constraints and allows the viewer to observe data that is being acquired at the same time from real equipment.
  - *Near-Real time*: It is the same as before, but with smoother timing constraints, the schedule allows infrequent losses or delays. Near-Real time systems assume some delays which are almost harmless to the system or user experience. Non-time-critical systems are grouped together inside this classification.
  - *Batch lab*: This approach is a time delayed data acquisition from an experience. Batch experiments are usually preferred when the time to complete a experience or the processing is so long that an interactive session is not convenient or even not feasible. This approach is also used in situations where a new experience can be run regardless of the experiments history.

- **Data not gathered from real equipment**: This type of data includes virtual labs or simulations, where the data does not proceed from experiment, but from computer simulations. Therefore, the evolution of the system can be obtained from:
  - *Mathematical simulation*: The labs lies on a mathematical core and the status of the system is being calculated every $\delta t$ time step.
  - *Simulated data-sets*: Data is not calculated but obtained from a previously computed data-set which contains all the possible changes of the simulated system.

The previous classification is not unique and must be taken as a guide to discuss possible architectures and configurations which are representative, [24, 25]. We are interested in virtual and remote labs in all its forms, but this work will make special emphasis in the remotely operated ones.

Next two subsections will present definitions of virtual and remote labs using the information provided along the chapter.

### 1.2.1. Virtual labs

Virtual Lab (VL) are designed to do experimentation without the real system providing simulated data to the user. It usually contains animations or dynamic schemes to provide a graphical representation of the system status. Frequently a simulated lab imply less restrictions than the real one and also has more configurable parameters because VLs are not built-in a fixed set-up. Using as example the coupled drives system, from chapters 8 and 9, the lab has been developed in both versions, virtual and remote. On the one hand, the remote version is fixed and contains many limitations to prevent the rupture of the elastic band. On the other hand, virtual lab allows to change the distance between motors, the elasticity of the band and the speed, leaving to the student total freedom to explore the behavior of the system in different situations. Therefore, to be highly configurable is one of the strengths of VLs and usually have sufficient configuration freedom to explore different behaviors.

As said in previous sections, a virtual lab can be based on a simulation of a real lab or a theoretical simulation. Both are valuable resources to explain concepts without the expense of building and maintaining a real system. In the former case, the VL provides a safe framework to explain extensively concepts and procedures and can be useful to prepare students to use the real one. From a educational point of view both labs are valuable resources which can be used together in a step-by-step approach:

1. Students learn about the theory of the system and lab procedures in a previous part of the course or studying all the documentation available.

2. They access the virtual lab to understand the theoretical concepts in a wide testing ground where they can explore fully the opportunities of the system. They will also learn lab procedures, which may be similar to those to be performed in the real version.

3. Finally, when students pass the VL, they can access the remote one, where the knowledge acquired can be useful to fully exploiting the time and resources of the lab session.

From the data point of view, virtual labs do not need to access to hardware or hard-disk and it is easier to embed one inside a web-page. Nevertheless, the consumption of computational resources in VLs can be too high, due to mathematical simulation, the graphical elements or both. Therefore, developers must consider the requirements needed to be run in low profile devices (smart-phones, tablets, etc).

### 1.2.2. Remote labs

Remote labs (RLs) are designed to do experimentation over the real systems, but from distance, using applications which provide a user interface to replicate the experience of handling the equipment on-site. As RLs are remotely operated systems they need interactive elements to control the status of the lab, in this regard, the user interface must allow the user to edit and see the behavior of the lab. To make a strong link between the interface and the real system, RLs usually contain webcam images and graphical charts to visualize the data from the lab. As the data provided is real (has been provided from a working system), cam images and data representation gives the needed feedback to change the status of the lab and correlate actions and re-actions. Laboratory practices on real systems are more restricted than VRs, in this sense, the intervals to modify parameters tend to be narrower and it is very common to find noisy signals, delays and less points of data. For this reason, the presented step-by-step approach may be useful to provide the students with a theoretical and virtual knowledge before dealing with the real system.

The data gathering method depend on the system, the architecture, developers' design and pedagogical target, but, in any case, the interface will need to access the data from real devices. The data acquisition implies a direct or indirect connection to the hardware, which is the cornerstone of this work. Considering the previous classifications and definitions, it is not an overstatement to say that the number of solutions and approaches to make VRLs is enormous and, as seen in previous section, there are also many platforms where the labs can be deployed. Next section will present different architectures that are being used to develop VRLs.

## 1.3. Architectures and approaches

Accessing hardware remotely from an user interface can not be defined as an single issue, because it groups different problems together. In order to analyze the situation inside the area of remote experimentation, problems can be divided in four major classes:

- **Accessing the hardware**: Obtain/transfer data from/to any system implies using sensing and actuating devices. Without any additional element, the control of the lab usually is restricted to on-site activities, as looking at sensor values and changing knobs to interact with the lab.

- **Remote access to the system**: Remotization is the process to make a face-to-face lab remote. Although, it usually implies adding more actuators and sensors, to control the system. Additionally, the system will need connectivity capabilities to allow the remote access.

- **Control from the interface**: Using an application to control a laboratory implies finding a design that fixes with the educational and procedural goals established by the developers and teachers.

- **Concurrency and load management**: When the system is accessible from the network and many users want to access, the developers can face problems with the amount of users and scheduling.

Developers, researchers and teachers have been dealing with these problems from the early nineties [26] revealing many ways to achieve the objective, but, there is not actually a best option, because each solution has to face different problems. The emergence of the Internet of Things (IoT) has also driven the development of libraries and tools, getting closer to sensor-actuator-device networks, which is directly related to virtual and remote experimentation.

The section is divided into three subsections, but, there is no full equivalence between sections and problems, as some of these problems may be coupled. The subsections are: hardware-to-network, network-to-device and architectural styles.

First one, problems related to the remote *access to hardware* and *systems*. The second subsection will describe common approaches to data exchange and communications. Finally, the third one is about full solutions which in some cases groups both, hardware-to-network and network-to-device.

### 1.3.1. Hardware-to-network

Data acquisition from hardware is a wide area of knowledge, and it is beyond of the scope of this work to discuss in depth the technologies available to obtain it. Although, as the hardware must have a computational layer and server capabilities, we can define the global situation in a *bird's-eye view*.

Nowadays, when any system is studied to be transformed into a VRL it usually contains multiple sensors and actuators. Those can be used to acquire data and/or

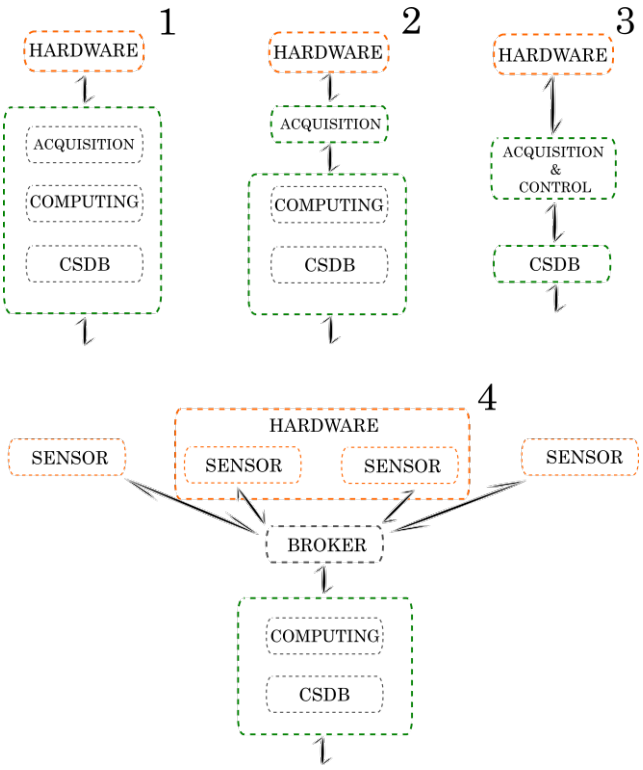



**Figure. 1.1** Hardware to network schemes

change the system status. As long as the data provided by sensors is needed in the user interface, the readings must be stored, processed and sent. Just as with the actuators, if a user needs to send a message containing the new values to actuators, it is also needed a system to acquire the message, process it and apply the changes into the lab. Therefore, the needs of this kind of labs push to choose a hardware-software combination.

Figure 1.1 illustrates different schemes that have been shown to be successful in many works. Next classification is done dividing the tasks by entities, which can be real or virtual devices. The communication between devices can be wired or not, and can use any available machine-to-machine or computer bus protocol (I2C, parallel, serial, CoAP, MQTT, etc). To simplify, we will group similar structures, including *Server, Client and DataBase* (CSDB) as one entity, but is also common to find those in different devices. CSDB will provide connectivity characteristics like services to attend client petitions or to connect with other servers (for example, databases).

- **Hardware-[Acquisition-Data Processor-CSDB]**: Two different entities, the

| | 2Devices (Opt.1) | 3Devices (Opt.2) | 3Devices (Opt.3) | 3Devices (Opt.4) |
|---|---|---|---|---|
| Adv | Less devices and faster acquisition | Faster acquisition, as is a dedicated device | Acquisition and control together add insignificant delays | Multi-source acquisition. |
| Disadv | Resource costly, as one device must do everything | Additional delays due to communication between devices | Additional communication tasks | Additional delays due to communication between devices |

**Table 1.2** Advantages and disadvantages of the hardware to network schemes.

system and a device in charge of acquisition, data processing and send/receive messages (Number 1 of Figure 1.1).

- **Hardware-Acquisition-[Data Processor-CSDB]**: Three different entities, system, acquisition board or device which is dedicated to one system, and a device on charge of processing and send/receive messages (Number 2 of Figure 1.1).

- **Hardware-Acquisition/Data Processor-CSDB**: Same as the last one, but the acquisition and data processing is made by one device (Number 3 of Figure 1.1).

- **Hardware-Broker-[Data Processor-CSDB]**: Three different entities, system, acquisition board or device which is listening to many sensors and actuators from one or multiple systems, and a device on charge of processing and send/receive messages (Number 4 of Figure 1.1).

As the data flow pushes new data to the CSDB entity, it is ready to be sent using the network and the Internet. Therefore, CSDB defines how the data is sent to the user, and it is important if the target is a student, database or monitoring systems. Table 1.2 contains *pros and cons* of each scheme regarding their use in remote experimentation.

Therefore, architecture, protocol and data format are key elements when considering the communications. First, we consider some common protocols, then the data format and finally some full solutions (architectural styles).

### 1.3.2. Network-to-device

Virtual and remote experimentation through Internet is a very specific area inside the computer networks science, therefore just a small set of protocols has been used to create laboratories. The following paragraphs present briefly four well known protocols as a basis for the discussion of more complex *solutions* that will be discussed over the architectural styles part.

#### Protocols

In the computer networks area, a protocol is as set of rules to define and to restrict the ways in which one system communicates with another. Some well known protocols which have been used in VRLs development are being presented within Table 1.3 which contains *pros and cons* of each one.

**Raw Transmission Control Protocol (TCP)** or TCP streams are considered one of the "most popular protocol in the development of web-based applications", [27] and some works have based their real time systems in this protocol [28]. *It is a host-to-host protocol in packet-switched computer networks*[11]. However, a major drawback of using a raw TCP stream is that the developers must code from scratch other capabilities, as the detection of connection failures or security layers.

**HTTP/HTTPS**: Hypertext Transfer Protocol/Secure is also, one of the "most popular protocol in the development of web-based applications", [27]. *The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems*[12]. It was originally designed for communication between web browsers and web servers, but it is currently being used in almost any *surfing the web* applications. Therefore, this protocol is being used in some steps to make the lab accessible from a web browser. HTTP follows a client-server structure, a client opens a connection to make a request, then waits until it receives a response. It is a stateless protocol, meaning that the server does not keep any data (state) between two requests. Some solutions (long polling, push, comet) use HTTP to obtain data from the server, but, as these are not immediate, this protocol has been used with some limitations in remote experimentation [27]. Nowadays, Server-Sent Events (SSE) allow to enable the client to receive messages from the server, making SSE also a good option to implement communications in remote lab experimentation, being the trend nowadays in some applications which need this feature in their applications. The widespread of HTTPS to obtain secure communications has incremented the difficulty to connect *dumb* devices to the In-

[11] https://tools.ietf.org/html/rfc793
[12] https://tools.ietf.org/html/rfc2616

| | Raw TCP | HTTP | Websocket |
|---|---|---|---|
| Adv. | Less delays in real-time apps. | Widespread and ready to use | Widespread and full-duplex communications |
| Disadv. | Higher level capabilities must be coded by hand | Is not a full-duplex communication and needs SSE to obtain streaming updates from server | Needs to upgrade from HTTP to Websocket which may cause problems with some network elements, as proxies |

**Table 1.3** Advantages and disadvantages of the different protocols.

ternet (which only support HTTP) and has no additional security layer.

**Websocket**[13]: WebSockets provide full-duplex communication channels over HTTP/ HTTPS, although, they are not the same. It uses one TCP connection for the server-client and client-server traffic, using any available port (for example: 80, 443 or 8081). This protocol enables a data exchange between the client and a server that can be done without a previous request from the client. Due to the good users' reception, major browsers support the Websocket protocol and are widespread in data exchange application. Nevertheless, Websockets implies some pros and cons regarding their use in communications [29, 30, 27, 28].

### Data formats

Since data transmission is a central issue in remote experimentation, data format, the way information is encoded, plays an important role in communications. As those formats may be either proprietary or free and may be either unpublished or open, this thesis only considers those formats which are open and free, because they provide better integration into other systems and are more extensible. Most of the data transmitted in remote experimentation is encoded as numerical or string-like data, and in a lesser extent binary. Usually this data has a tree structure and is self-contained, therefore, it is needed a solution to enclose multiple types of data together, transmitted and used on the client. Table 1.4 which contains *pros and cons* of each one.

Systems communicate to exchange information, but, as said before, the information or data comprises many types, some of those are complex structures, as files

[13] https://tools.ietf.org/html/rfc6455

or objects (for example, from an object oriented programming language). Sending complex elements which need to be used in both sides implies a methodology to translate and reconstruct these elements. These processes are called: *serialization* and *deserialization*, and are used in many communication system due to their flexibility to be used in most systems. Serialization formats are widely used to send data. Next lines contains four main data formats that have been selected because they are the basis of a big number of other data formats.

**XML**, Extensible Markup Language (XML) defines a set of rules for encoding documents. It is both human-readable, machine-readable and has strong support via Unicode for different human languages. The language is widely used for sending data from devices or machines to web services. Some of the labs included at the UNILabs Web are developed using this format [31] to exchange data between both sides.

**JavaScript Object Notation (JSON)**, is a language-independent data format and was derived from JavaScript. JSON is a human-readable text format consisting of pairs attribute/value and serializable data types. It is commonly used for asynchronous browser–server communication and is included in major browsers.

**YAML Ain't Markup Language (YAML)** is based in many other formats and languages (XML, JSON, HTML) is not binary, is human readable and is available for multiple programming languages [14]. This format is now being used in metadata, log evaluation, model-based information and storage [32, 33].

**Resource Description Framework (RDF)** is being used widely on Semantic-web and Big Data tools [34, 35]. Although, their knowledge-models for the data exchange in remote experimentation could be like using sledgehammers to crack nuts if the remote system has just a few sensors and actuators, but could be useful for large remote monitoring frameworks [36] .

| | XML | JSON | YAML | RDF |
|---|---|---|---|---|
| Adv. | Human readable and widespread | Flexible, human readable and widespread | Based in other formats and languages | Perfect for Big Data and knowledge models |
| Disadv. | The syntax is verbose and strict | No support of namespaces | Not included by default in all browsers | Can be tricky and complex to be used in small systems |

**Table 1.4** Advantages and disadvantages of the different data-formats.

[14] http://yaml.org/spec/1.2/spec.html

### Architectural styles

Architectural styles are set of principles and patterns to solve common problems that arise in the development of an application. Architectural styles provide abstract frameworks to develop robust solutions and enhance the design reuse. Since communication between systems is essential in VRLs, the architecture of the solution should consider the exchange of information with sensors and actuators, whether they are or not inside a laboratory system. In the following it is presented some common architectural styles that have been used in the remote experimentation field. Table 1.5 contains *pros and cons* of each one.

**Representational State Transfer (REST)**: It is used to design networked applications using just HTTP to call between machines. The concept relies on resources addressable through a URL using a stateless, client-server, cacheable communications protocol. REST defines constraints to be used when creating web services [37] and in networked applications. Requests made to a resource's URI will need a response with a payload formatted in either HTML, XML, JSON, or some other format. The response can confirm if some alteration has been made to the stored resource. REST applications use HTTP requests to create, read, update, and delete (CRUD) data. REST is commonly used along with HTTP, but, there are Internet examples where some users have been also implemented REST with Websocket instead of HTTP[15].

**Remote Procedure Call (RPC)**: It is usually used with distributed systems, making possible calling procedures in the local machine or on a remote machine. Therefore, if a computer program calls a procedure on a remote machine it is coded as a normal or local subroutine, making the process to communicate with the other system transparent for the user. This location transparency is useful to handle remote values, methods or services as local ones [38]. RPC is commonly used along with HTTP, but, can be also implemented with other protocols. RPC is also used for exchanging information in UNILabs VRLs [31].

**Simple Object Access Protocol (SOAP)**: SOAP is used for web-services, and uses XML information for the message format and relies on HTTP or SMTP. SOAP allows clients to invoke web-services and receive messages independent of the operating system. SOAP defines three part, one to set the message structure (envelope), the encoding rules and conventions to structure procedure calls and responses. Then, calling to a web service and integrating the response information into the client application can be done directly as all the information about the structure of the data is already in the message. SOAP has been used to obtain data from remote devices [39], and is still in use [40], but in some cases is being replaced smoothly

[15]https://hackernoon.com/rest-over-Websockets-instead-of-http-81b0f0632295

by REST [41, 42, 43].

| | REST | RPC | SOAP |
|---|---|---|---|
| Adv. | Very flexible to create web services even regarding the protocol | Remote calls are transparent for the developer | Rules and conventions make the message self-contained and has a built-in retry system |
| Disadv. | Has no built-in retry system | The client needs to know the method name and parameters, this couples client and server | It uses only HTTP and SMTP protocols and message format is heavier than the other two |

**Table 1.5** Identify synchronous and asynchronous messages using their origin and cause

### 1.3.3. Programming languages

Last sections have presented the different parts to consider when developing a lab from scratch. Programming languages are the last pillar needed to establish a starting basis, and at this step we will make a difference between the user interface and system software. On the one hand, the user interface should use a web-browser-friendly programming language to embed the resulting application inside a web-page or similar, because the general trend of last years is to move the application to web-browsers allowing the access from many different systems and through the Internet. On the other hand, the server side interacts with the real hardware and/or with the devices involved in the data transfer. Therefore the programming language at the server side must be a hardware-interaction-friendly.

In the early nineties, the idea of remote experimentation was considered a futuristic approach [26]. Few years later the first remote lab, was developed as a proof of concept, was developed [44]. In the same year, the first remote lab using Java technology inside a web browser was created [45], thus beginning a path to develop remote and virtual labs using the World Wide Web [46, 47, 48, 49]. Java-based VRLs were the state of the art, but two decades later, after the development of many remote labs, Java has been restricted to be used in the server side and not anymore in the client side. This was consequence of Java vulnerabilities found when using applets to run code inside a web browser, in addition, the restrictions for running non digitally signed Java applets made even worse the common dissemination of VRL based on this technology.

Nowadays, the development of VRLs can be done using one or more programming languages, some are used to interface with hardware, as LABView and others can be used on the browser at the client side, as Javascript. Other languages have been used in this field, some well known as: the C family, MATLAB, PHP, SQL. Other have become widespread recently, like Python, which is general-purpose, interpreted, multi-paradigm programming language and can be used in remote experimentations [50, 51].

Next chapter contains a description of the main tool used to develop the examples in chapters 9 and 8, Easy Java Javascript Simulations, and how it evolved along with developers until today.

# 2. Easy Java Javascript Simulations (EjsS)

EjsS is going to be a major issue all along this document, therefore this chapter is dedicated to introduce the tool and its features. At the end of the chapter, the reader will have a global vision of the topic and will be able to focus in the main research contained in later chapters.

## 2.1. EjsS: The tool

EjsS is a open-source tool developed in Java that assists non-expert programmers in the creation of dynamic simulations. The tool was originally designed to be used for learning purposes, and relies on a model-controller-view architecture to ease the design process of the virtual or remote lab [1]. A user can neglect the implementation details and focus on the science and knowledge behind the idea, writing equations, coding the behavior and/or the visual interface. The motivation of this approach is to enlighten the creation process to teachers. They may or may not have advanced knowledge of programming, but that are proficient in their topics and can benefit from the incorporation of digital tools into their courses.

As a tool to develop labs, EjsS is intended to be intuitive and easy to handle. To make it possible EjsS contains a main editor that allows the user to create an application defining the model and the view as separated entities. In this regard, the main window is divided in a tab-like structure where we can distinguish the following components: *Description, Model and View* (see Figure 2.1):

1. **Description**: It is a minimal text editor or a HTML-enabled window to write information of the application. It is shown to the user at the beginning of an experience.

[1] for simplicity, we will call them VRLs or applications from here on

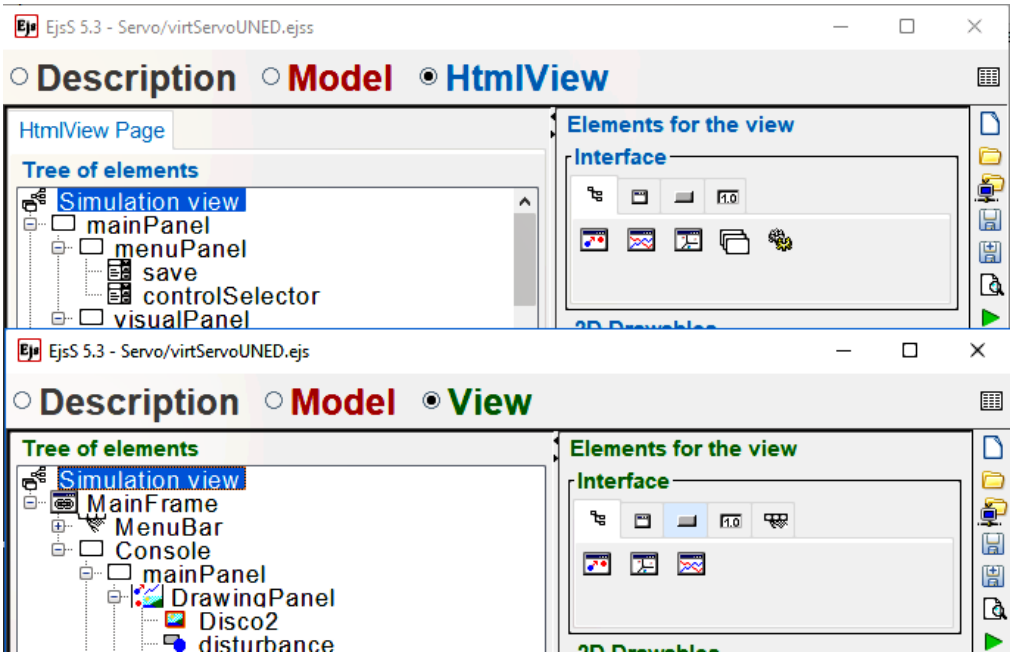



**Figure. 2.1** Both editors, the Java (a) and the Javascript (b) enabled versions

2. **Model**: It describes the process under study in terms of 1) variables, which hold the different possible states of the system, and 2) relationships among these variables, expressed by computer algorithms. This tab contains also sub-tabs *variables, initialization, evolution, fixed relations, custom and elements*. In fact, the model panel is the core of the application and its behavior depends on the parameters and code defined inside each sub-tab.

3. **View**[2]: It can be defined from scratch or using some templates, included within the editor. The tool offers two panels to define the view:

   - Left panel to create a tree structure for the view.
   - Right panel with the components to be added.

   The creator can define the properties of the final application, its interactivity or how the view is capable of reacting to some user actions. The view can be as complex as desired, but some of these reactions do not need to be coded by the user, because they are already precoded inside the core of EjsS. For example, a VRL may show a graphical representation (either realistic or schematic) of the states of the process, a web-cam or both.

[2]From here on, the term ”view” will be used to refer to GUI.

The tool contains also packaging capabilities to be used when the app is finished. Thus, the developer of the lab can deploy it as a standalone app or embed it in a web-page.

## 2.2. Applications

Since the first launch of EjsS to nowadays some research groups have developed their simulations, labs and apps [52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62]. The tool has also evolved in the hands of the group of Francisco Esquembre [63, 64], the creator of the tool, and other developers from UNED and other universities. In this regard, from the first version, many other features have been included to enhance the tool itself.

As is a tool in continuous development, enhancing EjsS is possible in many ways, from changing the core code to adding capabilities in a modular way. Each one of these modules is called *model-element*. These elements add features not included in the main tool, such as a file chooser, tools for accessing hardware or to connect with external software [31, 65].

### 2.2.1. Enabled versions

As said before, EjsS can be used by those who has minimal knowledge about programming. Inside the tool, Java or Javascript are the available programming languages, but each one uses a different editor. Each one of these editors is called an *enabled version*. Although, at the beginning, EjsS was only able to create Java applets, in recent versions (5.0 or newer) Javascript is also an option. Figure 2.1 explore the differences and similarities between both versions.

Similarities may be summarized as follows. In addition to the graphical likeness, the main process of interacting with the editor and building the model or the view remain the same in both versions. To build the model, using any of the versions, the user can select each sub-tab to define:

- *Variables* that can be used in any part of the application.
- *Initialization code*, to be accessed on the startup of the application.
- *Differential equations or code* for each step in the evolution during the application life-time.
- *Functions defined by the user* to be accessed from anywhere inside the app.

- *Model elements* to add new features to the main model code.

The creation of the view in EjsS relies on a drag and drop philosophy, where users can choose the components of the interface. These are inserted inside a hierarchical tree structure. Each view components has also a configuration menu, in which the individual parameters can be tuned in order to obtain the behavior and the look desired by the developer.

As some key words are being using in the text and some are quite similar, from here in advance the text will follow next two conventions:

- *View components* or just *components* refer to each independent particle of the view obtained from the panel on the view tab.

- *Model elements* or just *elements* refer to the additional features that can be added to the model using the matching sub-tab.

### 2.2.2. Java-enabled version and Javascript-enabled version

Top part of Figure 2.1 shows the Java editor. Since it is using Java as development language the final application can not be used anymore inside web browsers, but is still useful as standalone application. The main difference between the editor is related to the language, but *model* tab contains a *model elements* sub-tab, where the elements are not all the same as in the Javascript version.

Bottom part of Figure 2.1 shows the Javascript editor, which uses Javascript as development language. As far as the application is made using these two languages the final application can be embedded in web pages and also as standalone application. The main window is also divided in the same tab-like structure where we can distinguish *Description, model and HTML-View*.

## 2.3. Architecture of a VRL inside UNILabs

The creation and configuration of UNED labs, integrated with UNILabs, follow design patterns that simplifies the lifecycle of development, maintenance and update. The main topology followed in the labs is shown in Figure 2.2.

- Moodle provides the web interface and support for the booking system. Several plugins have been added to the UNILabs website, to provide extra capabilities. All these plugins were created by UNED and have been accepted and published by Moodle:

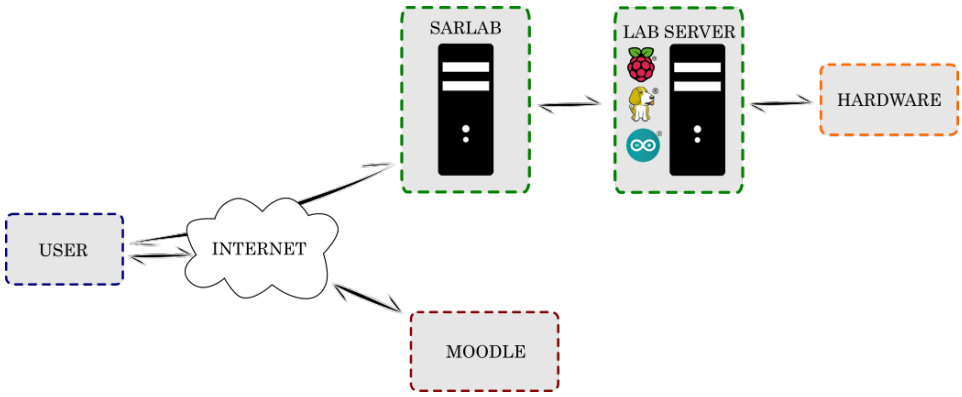



**Figure. 2.2** VRLs topology inside UNILabs

  - Add Java applets or Javascript applications using the *EjsApp* plugin[3].
  - *EjsAppBooking* plugin adds a booking system to schedule connections to the applications and labs [4].
  - The *remlab manager block* used along with the *Ejsapp* plugins allows to manage the remote labs in Moodle courses. [5]
  - Gathering user interactions, make collaborative sessions and other features are also available in the Github repository[6].

- SARLAB (Sistema de Acceso a Recursos de LABoratorio) solves the problem of having remote laboratories with public IP directions. It hides laboratory computers behind a private network, preventing the access for unauthorized users. It also manages the access of external users (clients) to laboratory computers and offers control over power supplies. It is a free system which requires additional software to be used within the VRLs, but it is an optional parameter inside the *remlab_manager* plugin.

- Lab Server is the device or combinations of devices on charge of the hardware to networks capabilities described in chapter 1. Those devices may be PCs or single board computers, such as raspberry pi [7] or beagleboard [8]. Other devices, like development boards, as arduino [9], can be used to perform some

[3]https://github.com/UNEDLabs/moodle-mod_ejsapp
[4]https://github.com/UNEDLabs/moodle-mod_ejsappbooking
[5]https://github.com/UNEDLabs/moodle-block_remlab_manager
[6]https://github.com/UNEDLabs
[7]https://www.raspberrypi.org
[8]https://beagleboard.org/bone
[9]https://www.arduino.cc

lab server tasks.

- Hardware: The real system, which contains one or more sensors and/or actuators. The lab may be provided by private companies, which implies additional hardware, as acquisition cards, and also additional software, like interfacing programs or drivers to be used in LabVIEW, Matlab, Python, etc.

The chapter has presented EjsS as a flexible tool to create VRLs and the architecture used in the *Informática y Automática* department of UNED. A common architecture when making courses with remote laboratories gives the developer a known environment to deploy and test new functionalities when making VRLs. For example, with the given architecture, if the lab is virtual, it is simple to add the EjsS packaged file inside a UNILabs course. The packaged file contains the needed files to run the lab. Inside the package it is included at least a .jar file (in the Java version) or a XHTML file (in the Javascript version). Then, when the user wants to practice with the VL, the XHTML file is served from Moodle, ready to use. When using RLs, the architecture is quite different. However, if the Lab Server and Hardware are ready to use, the process is similar, as the Moodle plugins and EjsS model elements are in charge of connection through SARLAB and other interfacing tasks.

This chapter finishes the introduction part, that defines the main concepts in the development of VRLs courses which will be revisited in next chapters.

# Part II

# DEVELOPMENT

# 3. Problem Discussion

Some of the most usual issues when developing VRLs have been presented. But, in addition, developers may face problems regarding the deployment of VRLs. As said before, some authoring tools are prepared to build VRLs and get them packaged and ready to be deployed, as EjsS. Even in this easier situation when these VRLs are remote and accessible through web-courses, other limitations and issues must be solved. The deployment can impose its own limitations due to the architecture of the application and/or the device which is running the lab.

Decisions must be made regarding the way data is acquired from hardware, which protocols are being used and how is the application deployed. But, even an excellent selection of these can result in a bad user experience due to some other factors. In this sense, a collection of structural problems have been identified. These problems can be grouped inside three major categories:

- *Locate the controller at the client*: This architecture leaves the control tasks to the GUI, which can change the number of useful control strategies.
- *The model and view dependency*: The strong coupling between model and view can be restrictive, for example with code, model or view reutilization.
- *The computational cost*: The calculations needed to run these applications can be excessive for some devices.

## 3.1. Controller Location

Some configurations when developing VRLs may imply building together the GUI and the software in charge of controlling the system. This software is called controller, and can perform two tasks. The first one, concerns the automation of the laboratory system. For example, a servo to control one dial of a machine. This is a control task which is translating interactions from the user. The second one, is related with the lab experience itself. The best example are control engineering labs,

where the goal of the lab is usually to study, modify and implement a controller. The two tasks are then coupled, as both commonly use actuators and sensors available to control some variables. Both controllers can face complications when are embedded inside networked VRLs.

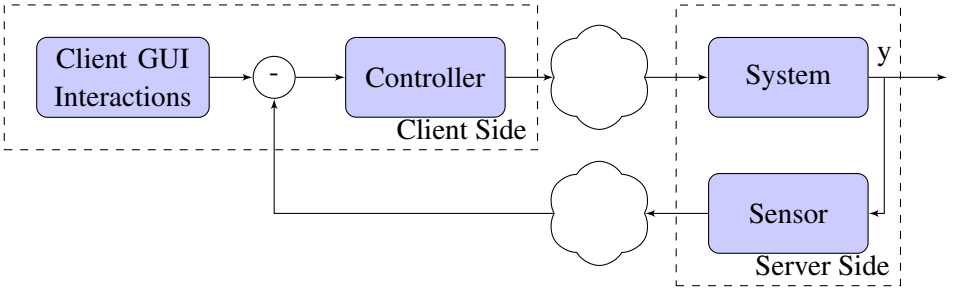



**Figure. 3.1** Diagram of the control problem with the controller at server side

Figure 3.1 shows two separated parts: the client and server side. The first, contains the controller and the user interface. The second, contains the system and the sensors. On this subject, the signal to control the system is computed in the client side and must travel across the network to reach the real system. The limitations that appear when studying these systems are complex and have been faced together in different areas. Networked Control Systems (NCS) are an intersection between control theory and communication theories, and it have been studied intensively [66]. The VRL development constraints are numerous as result of controlling the system in a networked environment [67, 68, 69, 70], as the one shown in figure 3.1. Commonly, these issues are divided in three categories:

- **Timing:** This category is wide, and it groups together three sub-categories: communication delays, sampling intervals and bandwidth limitations.
  - **Delays:** Delayed systems are studied in depth in control engineering without involving NCSs. The study of delays in networked systems became important ”as information about the network state can only be observed or relayed to controllers after a time delay, and the effect of a local control action can be felt throughout the network after substantial delay” [68].
  - **Bandwidth limitations:** The amount of bits that can be sent over the network is limited. In this sense, data from applications with high bandwidth consumption can be affected or even be impossible to be transmitted.

  - **Sampling intervals:** Sensor readings, status messages and other data exchange in NCSs can be scheduled precisely in a VRL. Although, ”the transmission intervals will be very likely to be time varying. in fact, the time between two successive transmission/sampling can be very uncertain and significantly large.” [69].

- **Channel reliability:** Failures or congestion over unreliable channels may imply packet losses, errors and disordered arrival. Then, reliability depends on which transport protocol is used to exchange data. There are many protocols, but, regarding Internet, only two are considered: UDP or TCP.

  - **User Datagram Protocol (UDP)** gives a connectionless service, which handles each packet independently of others. As it is an unreliable channel, it allows packet losses and out-of-order packet arrive.
  - **TCP** protocol is based in point-to-point connections which ensures in-order delivery and reliable communication channels.

- **Data quantization and architecture:** How signals are transformed from analog to digital ”is a common phenomenon in all digital control systems and thus has been a classical topic in conventional digital control theory even before the popularity of NCSs” [69].

There are different approaches to solve or minimize problems regarding NCSs. VRLs developers can take advantage of some of this efforts to neglect or reduce some of these restrictions.

- Using a reliable protocol to perform the communication can reduce losses and errors in the data exchange. All protocols considered in Chapter 1 rely on TCP channels. Most of VRLs are developed using TCP protocols, on this point, as UDP is not used, the reliability of the communication channel is ensured.

- Dividing data in small pieces, the quantization, has an effect on the stability of controlled systems. It has been studied in depth regarding the size of each packet, concluding that if each packet can contain a big enough number of bits, the quantization effects can be neglected [66].

- NCS have studied multiple control strategies. Some control strategies reduce the data exchange and thus the delays and obtain better sampling characteristics [67].

Changing the control strategy, choosing a reliable communication protocol and controlling the size and rate of messages can be used to reduce the issues regarding the NCS systems. Although, the depth of the problematic imply that each NCS needs further research and no general conclusions can be assumed.

## 3.2. Model and view dependency

Interactive online applications require great efforts in the form of programming, designing and testing. In parallel, the design of a laboratory to enhance the science learning outcomes implies that multiple areas must share its resources and efforts. When a remote system is used as an online lab, both processes have to merge. This join creates a complex but great breeding ground to develop many lab experiences.

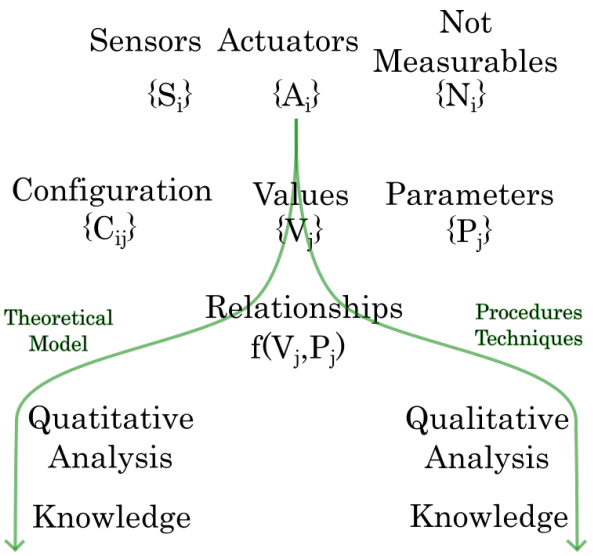



**Figure. 3.2** Two paths to describe the process of acquiring knowledge regarding their qualitative or quantitative nature

Figure 3.2 describe the process of acquiring knowledge from a system with VRLs. Remote labs are commonly automated systems to control and read a set of variables and/or parameters. Let's suppose that every value, configuration or parameter can be read or controlled online. In this situation, the number of available modifications (and states possible) increases greatly, so it does the number of tasks that can be performed. Lets call *the teachers lab* to the lab which can control everything in a particular lab. Thus, all the conceivable VRLs which can be made with with one specific system must be contained in *the teachers lab*. This concept can be

applied to the knowledge that can be extracted from the VRL. Then, the ideal concept of *the teachers lab* may push somebody to think that labs must be as similar as possible to it. Nevertheless, VRLs for education are commonly simple and well designed to help the students in the learning process. These VRLs usually contain only those fields which are needed in the lab experience. Educators who want to offer many simple labs, instead of *the teachers lab*, may need to create many different *student labs* with the same system.

One online lab can be used to perform a finite set of lab tasks. Each task is used to obtain small pieces of knowledge. Figure 3.3 shows a graphical representation of how the GUI can produce the extraction of knowledge from the system. For example, it is possible to suit the lab to be used by three different groups of students with different knowledge level by using the same set of variables. But, any additional task or change in the accessible variables may imply modifications or even the implementation of a new GUI. It means that there is a high dependency between the user interface and the system side. Thus, any of the lab experiences that can be developed will be tightly coupled with the online GUI.

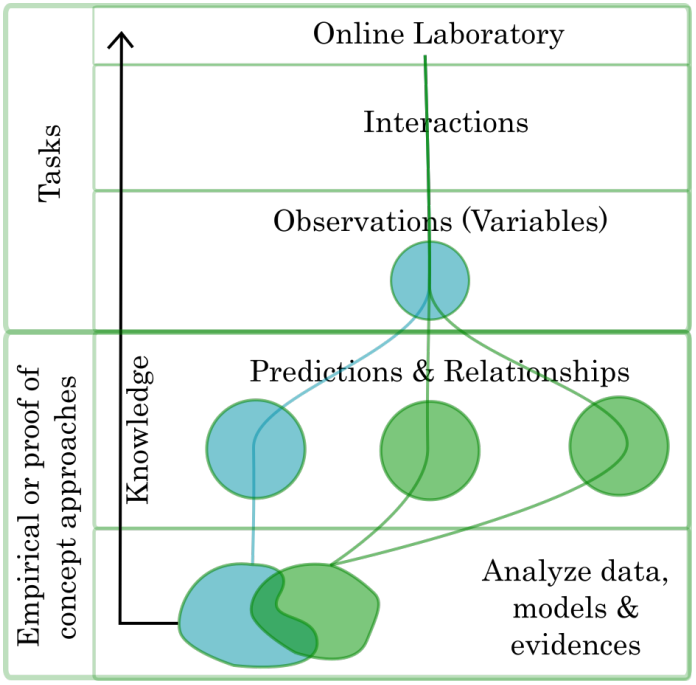



**Figure. 3.3** Extracting knowledge from a online laboratory. Accessible variables define a finite set of lab tasks and the knowledge that can be obtained through their observation and manipulation.

Last paragraphs show that the coupling is not just a superficial problem but rather an architectural one. Other problems may arise when developing VRLs, which are divided into three categories:

- The design of the experiments depends on the number of tasks that can be performed and the knowledge that can be gained from each experiment, then new ideas from teaching may imply new applications.

- Usually, VRLs development needs a close work between the controller side, the model, and the GUI. Work with both sides together is almost mandatory in the first steps of the development. When the real system is placed in its final location, the development of the GUI commonly becomes difficult if there is a tight coupling between both sides. The difficulty is as higher as the number of changes to be done in the model. For example, changing the name of a variable, sensor or actuator may be done twice, in the GUI and in the model. This problems may end in reducing the number of changes in the lab or in making heavier applications to cover all the possible future desires of teachers. Both consequences limit the flexibility when developing VRLs, and increase the effort of building new labs.

- The GUI is the main front-end for students, and represents the lab for them. As said before, the experiment design influences the GUI design, and both are usually designed considering the final user knowledge and experience. As a consequence, a GUI may perfectly suit the needs of a group of students, but, it may not be appropriate when is used by other students. Therefore, to some point at least, the success when learning from lab experiences gets entangled by the design of the GUI.

Regardless of the final purpose of the web application, developers have faced these problems and some solutions have been applied to reduce the effects of coupling between the model and the view:

- *Modular programming techniques* allow developers to simplify the coding of VRLs. These techniques define a module with a functionality and required data to be run. This level of decomposition leads to simpler solutions when coding VRLs, and can be applied to reduce the coupling problems. In this context, modular programming and reusable code allow to reduce the work needed to develop the view and the controller of a system.

- An *Application Programming Interface (API)* defines a collection of procedures to communicate with components. An API can define how to use variables, remote calls, processes, data objects or structures, etc. APIs show the

accessible and useful elements for the developers, abstracting the coding below. A well defined APIs can suit most of the needs of developers of applications. As result, the efforts are focused on designing and developing a good API for a specific system.

- Developing single-task labs or modules, where each one will illustrate one concepts. The *single responsibility principle* states that "*A class should have only one reason to change*", where "*a responsibility is a family of functions that serves one particular actor*" [71]. In the case of VRLs, depending on the final goal, these modules may focus on one of two: (i) explain a simple concept or (ii) complete a big task which is divided in a set of simpler tasks. Regarding the second option, the lab can be adapted to the level of knowledge of the student by adding or removing labs from this learning path.

Considering the VRL development area, the solutions presented reduce or counteract the effects of coupling. This is achieved by designing the software to be more flexible and reusable in many other laboratory systems.

## 3.3. Computational cost

The rise of smart devices in the market and society presents lots of changes in people's life. Smartphones have become, in most cases, the main device to browse the Internet, consume multimedia content, mailing or exchanging messages, etc. The increase in their importance lies on the shift in user preferences from computers as desktops or laptops, to smart devices. The usability of these devices has been studied, as well as their usage in cloud environments [72, 73, 74], concluding they can be useful for the main user. Remote or virtual web-based labs have been also designed and created to be run inside smartphones, as in [75, 76, 77, 78]. The existence of so many potential target devices are a great opportunity to reach more users, but also an important increase in the effort of development due to:

- VRL developers may consider the possibility to make their labs accessible from smart devices, supporting as many operating systems and web browsers as possible. The support should cover near to one hundred percent of users, as mobile devices are a must-have product nowadays.

- On the one side, there are powerful devices that can be compared with some computers. These devices can run, without problems, the calculations of quite

complex virtual labs. On the other side, many users have old or low-end devices, that sometimes can not run simulations fluently. In the worst case, applications suffer a crash or the device gets overheated. To reach more potential target users, the solution must provide a low computational cost alternative.

- Because of the wide variety of devices in the market, the compatibility with smart devices is sometimes not well defined. In combination with available web-browsers and programming languages the compatibility becomes even fuzzier.

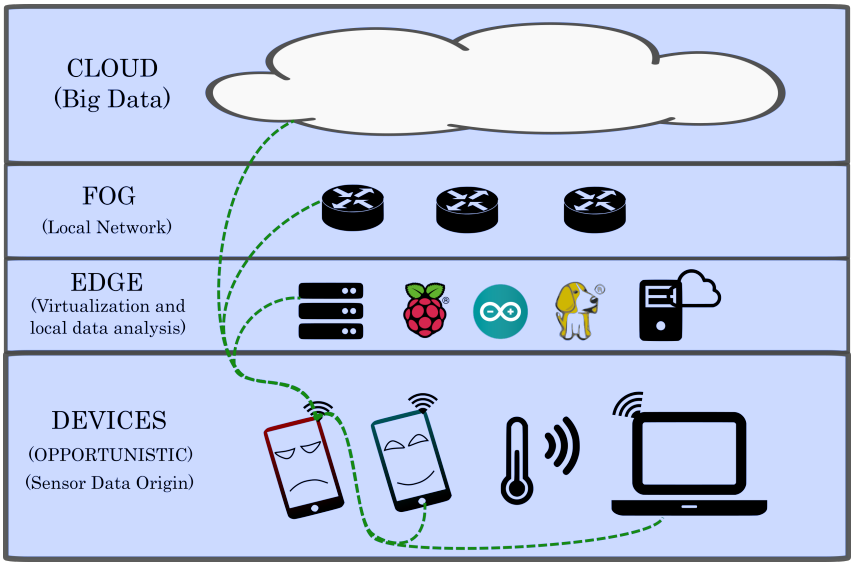



**Figure. 3.4** Possible offloading solutions, different approaches used in mobile devices.

Available solutions include many different approaches, and it is not in the scope of this thesis to describe all of them. However, most of them suffer the same limitation: the computational resources. The lack of resources in mobile devices suppose a challenge to developers. Many researches focus their work in offloading the mobile devices [79, 80, 81, 82, 83, 84, 85, 86, 87, 88]. These studies widened the mobile device definition including sometimes: smart-phones, tablets, laptops, robots and IoT elements. Resource-contrained devices can take advantage of different strategies [79]. Figure 3.4 shows how these strategies are divided in different categories depending on how and where the computations are done:

- *Cloud computing*: It is focused on the idea of user as data consumer. In this regard, mobile devices delegate the data storage and the data computations

to the cloud. "Putting all the computing tasks on the cloud has been proved to be an efficient way for data processing since the computing power on the cloud outclasses the capability of the things at the edge" [89]. Data production inside user devices is now a reality, but the bandwidth may result in a bottleneck. Thus, cloud computing becomes less powerful as the user needs to send more data from its device.

- *Fog computing*: Originally, it was designed to work with IoT devices as it provides interconnectivity between services or devices with different technologies. "To provide a self-organizing network infrastructure for IoT services, it incorporates network virtualization and traffic engineering aspects from Network Function Virtualization (NFV) and Software-Defined Network (SDN)" [89]. Due to the time response and the flexibility of the fog computing, it appears as a good offloading solution.

- *Edge computing*: The data is also processed at the edge of the network. Then, the processing delays are smaller than in cloud approaches. This proximity between data source and computing makes it suitable for IoT. It appeared as a complement to IoT and cloud computing, where the origin of the data are terminal devices. Sometimes it is mixed with fog computing and some studies consider fog as a subset of edge computing [89].

- *Others*: Many other strategies are capable to offload devices by delegating the computation in other elements of the network.

  - *Superfluid cloud* [90]: This solution creates multiple virtualization platforms where the deployment of these elements is not restricted to one location: the edge, the access network or the core. The increase in the number of devices and virtual machines enhances the capabilities of the cloud and its flexibility.

  - E*dge-centric computing*: "It is a novel paradigm that moves the locus of control of Cloud Computing applications and services to the edges of the network. The nodes defined are deployed across data centers and nano data centers and collaborate with each other in a peer-to-peer fashion" [91].

  - *Opportunistic*: Most of the information delivered to users comes from the cloud and content service providers. Sometimes obtain some of this data from other close and connected devices appears as a good solution to decrease the computing, delays and network resources consumption.

The opportunistic solution was originally conceived to offload the mobile network traffic [92], but it has been considered to offload smart devices [93].

The paradigms presented reduce greatly the computational effort in mobile devices. Nevertheless, considering the VRL development area, computing and data creation is done at client and server sides. As each lab is different and the available architectures are many, the usability of these solutions is not generalized for VRLs.

This chapter has presented many problems that arise at the development process of VRLs. The placement of the controller is a complex problematic which combines control and communications. The coupling between view and model is a development restriction, increasing the efforts of creating, updating and maintaining labs. Computational cost can result in a limitation for the user even in powerful devices. Next chapter contains a solution proposal, which is the main contribution of this research work.

# 4. An Architecture for VRLs

A good selection of structure, protocol and data format chosen from Chapter 1 can improve the potential of the VRLs. From last chapter, we have seen that some decisions on the design of the architecture involve complex issues, that must be solved. Some of these problems regard a higher design/development effort, a decrease in performance, the user experience and/or the system controllability. When grouping the decisions to be made, the number of available subsets is enormous.

The goal is to design an excellent architecture to be used in all VRLs. To achieve the objective, it is imperative to choose a compatible subset of options and to apply mechanisms to overcome the impact of last chapter problems. Next sections show a solution proposal which groups together different approaches. Additional issues, that appear along with the architecture, are also solved across the chapter.

## 4.1. Decoupling model and view

This section discusses main aspects of the solution for all the problems presented in last chapters. On the one side, many VRLs, as presented in Chapter 1 consider a design paradigm where the model and the view are two indivisible concepts. That coupling worked well in the past but, the set of issues presented suggests that decoupling the model and the view may provide benefits. On the other side, Chapter 1 has described many different ways to create VRLs. Both sides together provide the perfect breeding ground to decouple model and view inside a functional architecture.

Decoupling the model and the view is not solely confined to the separation of both elements. Model and view have different tasks to perform. The nature of each one defines a different set of tasks. The set of tasks of view or model can be implemented in many programming languages. However, to achieve a full decoupling, the underlying language should not imply an immovable link between both parts.

To summarize, the solution provides the possibility to create two different entities: the model, which runs on the server side accepting connections of local and

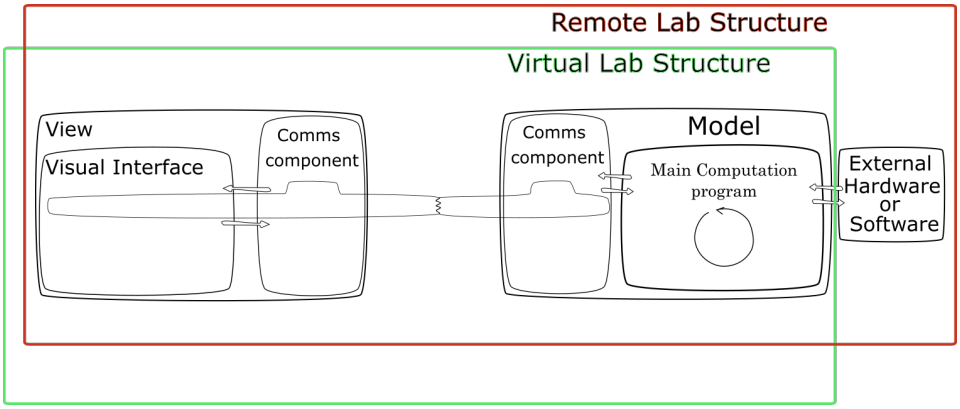



**Figure. 4.1** Architecture of a VRL made using Java and Javascript enabled version

remote views, and the view, which should be able to obtain data from connected models and show an interactive view to the user. The communication protocol provides means to exchange information between the model and the views, without considering which programming language has been used to develop the model.

Figure 4.1 shows the overall picture of the system and contains the view, model and communications. It also illustrates the two types of labs, virtual and remote, from the perspective of decoupling:

- **Virtual labs**: The structure comprises the *GUI* and the *model* as separated entities.
  - The *model* of a VL plays the role of the computation software. A basic model contains a communications component and a main computation core. All the calculus are made in this side and usually is not connected to external software.
  - The *view* of a VL is the *GUI* and contains a communications component and the visual elements of the interface.
- **Remote labs**: The structure comprises the *GUI*, the *model* and the *external hardware* as separated entities.
  - The *model* of a RL makes similar task as VL regarding the computation software but it also includes hardware connection abilities. A basic model contains a communications component and a main computation core, but the core can connect to external hardware, software or both.
  - The *view* of a VL is the GUI and contains the same structure as in the case of a VL.

Model and view independence is the first step to create VRLs with better characteristics. A decoupled architecture fixes the tasks to be performed by each part and reduces the effect of problems presented in sections 3.1, 3.2 and 3.3.

- C*ontroller at client*: Deploying the view in a different device to the model can lead the solution to the problems inside NCSs. Nevertheless, the key factor is the task division, which defines where the controller must be placed.

- *Model-View dependence*: Dividing the tasks of model and view and deploying each one in a different device can produce a decoupling between both. However, it is not totally ensured, as it also depends on the particular coding of view and model.

- *Computational cost*: The computational load in the user device is reduced by decoupling both parts, as the model can be in control of heavy calculus.

The decoupling may reduce the problems presented in last chapter, but, the communications, specific tasks and detailed structures are not yet fully defined.

## 4.2. Communications

Communications are a central issue in the proposed framework. In many cases, the difference between VRLs relates to how communications are handled. Then, the protocol, format and design may suppose a barrier and is useful to define standards to access to hardware or to other connected devices.

Actual applications are made following many different designs, as seen in Chapter 1. A communication standard must include a solution where the data format and protocols are common between applications. It also must be adopted by the community and readily available to any user. These three conditions set a restriction over the privative and commercial formats and protocols, which are not readily available for everyone. In this scenario, the set of candidate technologies is smaller and the choice of format and protocol should be made considering:

- *Usability*, the easier to use, the better to migrate from other IDEs or tools.

- *Availability*, the protocol should be easy to implement at least in most common programming languages.

- *Use of standard technologies* and combined with careful design of the software must contribute to reduce maintenance.

In this regard, communications are implemented with the WebSocket protocol, and using the JavaScript Object Notation (JSON) data format. Both have been presented briefly in Chapter 1 and fulfill the requirements of usability, availability and maintenance.

### 4.2.1. Technologies

Communication technologies set the available features that can be developed inside the applications and how these are deployed online. Choosing WebSocket protocol and JSON data format the solution present the following characteristics and advantages:

```
"vectorList":[
  {
    "origin": {
      "xCoordinate" : 1.0,
      "yCoordinate" : 0.0
    },
    "end":{
      "xCoordinate" : 3.0,
      "yCoordinate" : -4.0
    },
    "lenght": 5,
    "degAngle": 306.8
  },
  {...}}
]
```

**Figure. 4.2** A vector list defined using a JSON structure

- WebSocket: Using Websocket protocol in web applications has become a widespread solution in the last years and it is supported by most browsers. It uses one TCP which provides full-duplex communication and uses any available port. All these characteristics make WebSocket a good choice to develop VRLs.

- JSON is a human-readable data format used for asynchronous browser–server communication. Though it was derived from Javascript, it is considered language-independent and many programming languages provide means to parse and generate JSON. The format uses name/value pairs (see Figure 4.2), which can represent from simple data types to complex structures.

Remote experimentation may become even more important in next years, and therefore, other protocols and formats should fit in the restrictions imposed. As the field of VRLs is in constant update and upgrade, new technologies will also appear eventually, fitting or not with the conditions presented before. However,

nowadays, converging into a standard for implementing VRLs has been discussed by different research groups. Communications architecture tries to get closer to these standardization efforts and fulfills all the conditions presented.

RLs development implies data exchange, at least between the user and the real system. Moreover, many web applications need to exchange some information with databases, external software, devices, local hardware or even between different sub-modules inside the same application. Defining the communications upon the same basis, a set of common technologies, is fundamental to establish a standard and makes the code reusable. This unification in communications is not complete yet, but, considering all the advantages, establishing the use of the Websocket protocol and the JSON data format is one step forward on the process of generalizing the communications between:

- Models: As the model is an individual entity, it can run without interactions from outside. Eventually, a client may want to connect with the model to carry out a experimental practice. When the interactions and requests begin, the model is on charge of parsing, processing and sending back all the information required. At the same time, the model is listening to answer to other messages from external software, devices or hardware connected.

- Views: A different individual entity that contains a visual representation and capabilities to connect with the model. The laboratory data exchange will be possible if the model is available or any other element needs to establish a communication with the GUI.

- Hardware may be present or not, depending on the nature of the lab. The complexity of the element can go from a single sensor or actuator to complex systems as the engine of a car. The origin of the data is also important, as it can be local, like in built-in GPS of mobile devices, or remote, like the status of a solar panel. If the messages and/or the protocol used to exchange this information are common, the number of technologies needed decreases, and so the complexity.

- At present, smart devices are actively involved in daily tasks, as smart phones or tablets. Technology has worked in favour of development of the IoT, thus, systems like Rapsberry Pi, Arduino or Beaglebone can be used to create smart devices connected to the network. It is essential to share the communications structure when using such devices, in order to make any solution to grow along with the new IoT technologies.

**Message format and smart device specification**

In the suggested solution, one VRL is divided in two parts: the view and the model. Both parts, are already equipped with the capabilities to exchange data using JSON. The messages include data that express the interaction between the model and the view in a way. The way this exchange is done, can vary from one developer to another. Two options are proposed as valid and compatible:

- **Mixed format**: The data is sent in plain text, but, not all of the information is inside a JSON structure. The message contains two parts: the header and the information. The last one can be JSON-parsed and is human readable. To understand the header, both parts in the communication process must know the meaning. It is possible that this option leads the solution to a low level coupling, which is commonly not desired, as the solution looks for the independence between both.

- **Pure JSON format**: One goal of the JSON format is to provide an easy way to understand the information inside a message. The JSON structure can contain detailed data about the lab and its functionalities. The JSON by itself is not enough to assist the generalization process, neither the decoupled solution. To satisfy both requisites, an excellent option is to use the **Smart Device Specification (SDS)**. The SDS was developed to decouple server and client in remote lab implementations, and to obtain a well-defined representation of the server, including services and functionalities [94]. Therefore, this format allows the views to obtain details from the remote or virtual system, and also to interact with it using JSON messages. SDS transmits a bigger amount of data, compared to common data exchange in VRLs. In this sense, this additional information allows to virtually replicate the system, with its components, and to know the status of sensors, actuator, etc. As this format provides wider and more standardized communications, it is a better option.

Considering both structures, and in order to join the efforts to make a standard, the communications should use the SDS. The latter provides a useful self-documented interface, so the client can build the user interface without any additional information. Therefore, the GUI can be developed to use the laboratory without information about hardware and internal work of the server. Additionally, the information of the model is reusable and can be maintained in many other systems just modifying (adding or deleting) the list of sensors and actuators. Also, there are considerable benefits to exchange data and re-configure real equipment using a

standard. A full description of the Smart Device Specification for remote labs can be obtained from [94] and the complete specification is published on GitHub [1].

The client can easily obtain a description of the lab and information about the status of the equipment by using the available services. This information is given in JSON notation and can contain data from sensors, actuators or additional features included by the developer of the lab. Therefore, the possible interaction options in a lab depend on the services defined in the server, such as, send values to actuators, obtain data from sensors, modify the configuration of internal hardware, obtain information about the status of sensors or actuators, etc.

The decoupled laboratory architecture results in a model that commonly makes no assumptions about the GUI. The model then offers a set of services which are accessible using the protocol and format described in a metadata file. It is common to find remote labs which also assume this approach, a Lab as a Service (LaaS) where functionalities are made as independent modules [95, 96, 97, 98, 99, 100].

The smart device model uses a close approach to the LaaS, describing the lab as a single entity. LaaS is seen through a set of services and aims to solve the common challenges in VRLs developing and implementation. Both approaches obtain benefits from the interoperability with other heterogeneous systems, modular design and development and the steps forward standardization.

Researchers have been focusing in the implementation of utilities as services LaaS [101]. Some of these models includes Sensing as a Service (SaaS) [102], Infrastructure as a Service (IaaS) [103], Database as a service (DBaaS) [104], SaaS (Sensing as a Service) [105]. This way, the same services can be adapted to the needs of its developers or users, widening the usefulness of the software, infrastructure, database, sensors or the VRLs. Thus, fixed GUIs are commonly avoided, as they confines consumers to a certain Web technology. Developers can customize their application using their own needs to create a learning environment.

## 4.3. Client-Server Structure

Proposed solution defines two different sides: 1) the *server*, which contains the real system and runs the model of the lab, and 2) the *client*, which runs an interactive interface to the user. The communication protocol provides means to exchange information between the model and the views, without considering which programming language has been used to develop the model.

This task division allows to move a step forward, as the model and the controller are at the server side. Then, as the server side controls the real system and do the

[1] https://github.com/go-lab/smart-device-metadata/raw/master/smart-device-specification/

computations, the VRLs can be used as non-networked control system, reducing its difficulties. The proposed solution is flexible, and is suitable to be configured as a NCS, as long as developers want to. Figure 4.3 shows the diagram for the control system, where the controller and system are now at the same side.

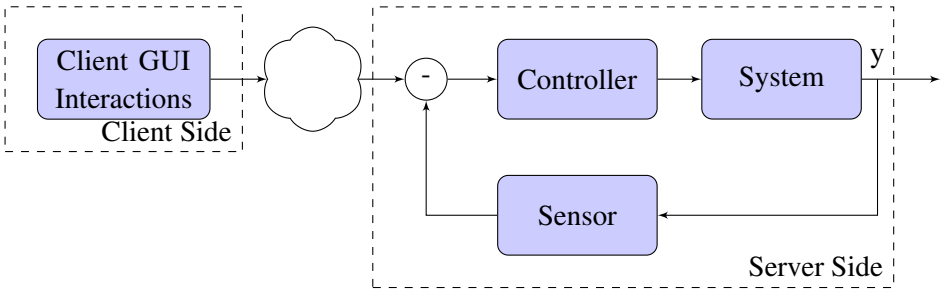



**Figure. 4.3** Diagram of the controller at server side. The server side computes the control signal, actuates in the system and read the sensors.

The minimal set of tasks that must be carried out by each side is not clearly defined from last sections. Client and server sides have to be designed to be independent one from the other.

### 4.3.1. Client Side

The GUI is a key element when talking about VRLs, as it is the front-end of the real system or simulation. A GUI has to perform different tasks: synchronizing, drawing and plotting activities inside the view. In this regard, the model-view pair must maintain the information flow between both sides, when each one is in a different machine. The user side is divided in main tasks and subtasks:

- **Visual interfacing:** GUI, which contains also the graphical representations and interactive elements:
  - *Graphical handlers*: All the visual elements in a GUI may contain available methods to modify or edit them.
  - *Updating data subtask*: If an event produces a change which affect the visual appearance of the GUI, it must be updated.
  - *Subtask to wait for and handle user interactions*: loops which wait for a particular event are called listeners, and must be defined for each interactive element of the GUI.

- **Communication:** The Websocket protocol implies at least three subtasks to control the communications:
  - *Definition*: A client must be defined using an IP direction and port where the GUI can find a working server.
  - *Configuration*: The parameters of the communication need to be editable, at least the IP and port mentioned.
  - *Listeners*: Incoming messages need to be handled to process the data.
- **Message processing**: Messages and internal processing of the data must be considered as a separated task, to allow developers to introduce new changes without affecting all the architecture. Thus, at least two sub-task must be done:
  - *Formating task*: The data is sent inside a message structure. This structure is known by model and view. Therefore, tasks to read or write messages must consider how the methods must be used. Which is the same as how the API defined inside the SDS.
  - *Wrapping or interfacing task*: To reduce the additional data inside one SDS message, this task asks for the data needed by the GUI. It makes a translation work, in order to interface between GUI and format task, preparing the data to be used by the view. The wrapper reduces also the amount of methods accessible to the developer.

### 4.3.2. Server Side

The model carry out the data processing and interacting with the real system. The model will send updates periodically and needs to know how the user is interacting with the view at every moment. Some basic tasks of a model inside a VRL can be covered by adding the characteristics listed below:

- *Metadata Services*: The view will need the metadata to inform about the internal structures of the lab. If the metadata is available it must be obtained using a metadata service. Minimal requirements include three metadata services to answer request regarding:
  - The API and data-models.
  - The sensors set.
  - The actuator set.

- *Data Services*: The view and model need to exchange the data from the lab by sending/receiving messages. Thus, the model must be able to receive and identify the different types of message. Minimal requirements includes four data services, in charge of receiving the messages and then:
  - Check, parse and apply the requested changes in actuators.
  - Check, parse and read data from sensors.
  - Check, parse and configure sensors.
  - Check, parse and configure actuators.

## 4.4. Programming Languages

As the decoupling is meant to be language independent, the solution has to study some considerations regarding the language. As the VRL is usually designed to be embedded in a web-page, the programming language plays a key role, mainly at the client side. These considerations shows restrictions that have been imposed during the last years and how they have been countered.

### 4.4.1. Java

In early 2015, web browsers started dropping Java support due to many vulnerabilities and security problems [106, 107, 108, 109, 110]. As a consequence, browser embedded Java-based VRLs were sentenced to death. Figure 4.4 contains a histogram including all Java vulnerabilities reported in NVD (National Vulnerability Database [2]) during time divided by its severity and most known restrictions involving applets and NPAPI plugins[3].

The graph in Figure 4.4 shows that 2013 was the year with more high and medium impact vulnerabilities detected. As consequence, solutions and patches to minimize Java issues appeared in time, adding restriction to run applets and decreasing their usability:

- April 2013: Applets should be signed by a trusted certificate. More or less at the same time web browsers announce the end of support for NPAPI-based plugins.

---

[2] nvd.nist.gov/vuln/data-feeds

[3] NPAPI-based plugins allows the browser to handle content types that are not natively handled, some of these are: Java Runtime Environment, Flash Player or Adobe Shockwave Player

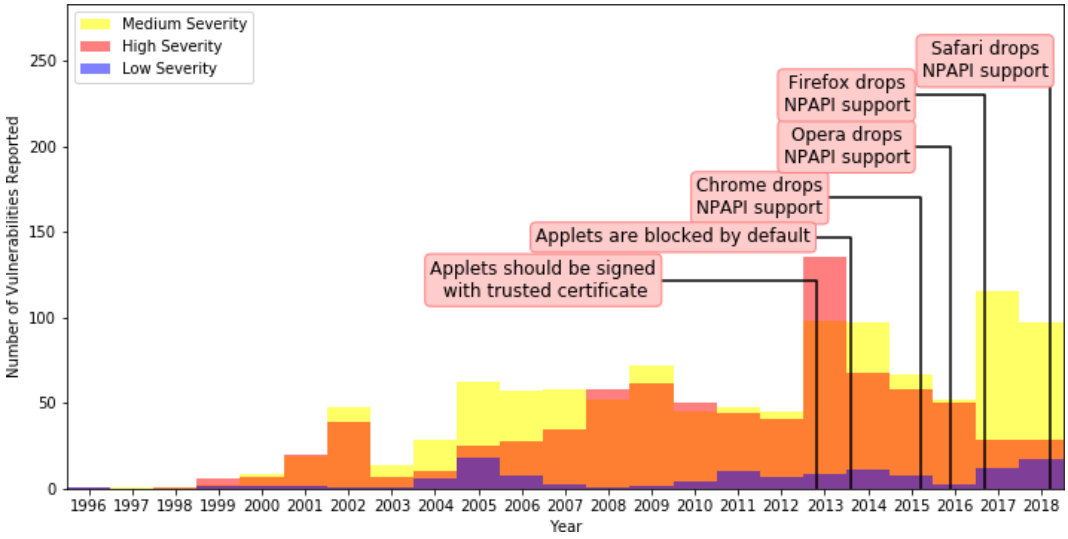



**Figure. 4.4** Java vulnerabilities reported during time divided by its severity plotted from the data of the NVD (National Vulnerability Database). The graph includes the most known restrictions involving applets and NPAPI plugins

- February 2014: Applets are blocked by default, system prompts a message to run or not the Java app. Java options can be configured to allow running not signed applets by reducing the security level, but informing the user about security risks. Additionally, Java update informs that this option for non-signed applets will be deleted in next updates.
- September 2015: Google Chrome browser drops NPAPI support in its last update.
- May 2016: Opera browser drops NPAPI support in its last update.
- March 2017: Mozilla Firefox browser drops NPAPI support in its last update.
- September 2018: Safari browser drops NPAPI support in its last update.

On the one side, users have devised strategies to solve these limitations in the use of Java applications embedded in web browser. The simplest solutions lie on using extensions or browsers that still allows the NPAPI-based plugins. Java started also other strategy, Java Web Start or JNLP, to run Java applications using a web browser by running it inside the user machine. JNLP allows to ”launch applications simply by clicking on a Web page link. If the application is not present on local computer, Java Web Start automatically downloads all necessary files. It then caches the files

on the computer so the application is always ready to be relaunched anytime you want"[4]. On the other side, Javascript labs shows up as a fresh alternative, based on standard and modern technologies that opens up new possibilities.

### 4.4.2. Javascript

Javascript shows like the best choice to obtain flexible and usable VRLs inside web environment. On this subject, web accessible labs that use HTML+Javascript commonly are compatible with smart devices. However, changing the programming language to develop VRLs is not an easy task. There are some advantages and drawbacks when using Javascript:

- **Access to local data**: Accessing hardware or the users' file system using only Javascript. The restrictions imposed to Javascript, related to accessing local resources, gave the user a security layer. But, the developers may need to implement this functionality in the GUI if the lab need to access it.
- **Use of third-party libraries**: Due to the nature of Javascript or because Javascript is younger than other languages, some libraries are not yet developed or will never be. Some problems have to be solved using new libraries, slowing down for a short time the development of new laboratories.
- **Multi-device experience**: Chapter 3 shows that smart devices must be able to run VRLs. Both problems regarding the use of mobile devices are solved when combining the solution proposal and Javascript:
  - The labs are accessible from mobile devices, no matter the operating system the user just needs a browser that supports JSON and Websocket.
  - The computational cost is reduced as the solution leaves the heavier computations to the model, which is done in the server side. As the model is the main producer of data, the client just receives the data that needs to be plotted, drawn or shown. This functionality works as an edge computing solution, as a terminal device is on charge of doing the calculations for the user.

In summary, this chapter has presented a solution that comprises an architecture, protocol, format, task division and programming language for the development of VRLs. The proposed architecture moves forward to the standardization and solves actual problems that appears in the design and development steps. The theoretical

[4]Obtained from : https://docs.oracle.com/javase/8/docs/technotes/guides/javaws/

solution has been implemented to be used inside the EjsS tool. This implementation is described in next chapter materializing some key elements shown along the last chapters.

# 5. EjsS Implementation

Last chapters presented the proposed solution as well as a glimpse about the implementation in the EJS tool. Replacing Java-based VRLs with HTML and Javascript enabled applications is a good approach to, at least, solve the Java problem. Still, Java is considered excellent for not web-embedded tasks. The TIOBE index[1] shows the percentage of hits that search engines returns when searching about programming languages. Consulting TIOBE Java index[2] shows that is an extremely active programming language, as nowadays it is in the first place of the index. Actually, all the information that can be obtained from the Internet suggests that Java is being used massively and that it is evolving to counteract the restrictions as web-embedded applications.

A trade-off implementation of the solution is to combine both languages, Java and Javascript, using the best of each one. Javascript can be used on the client, where it is naturally supported inside web browsers. Java can be used on the server side, taking advantage of their computational power and the use of well known libraries. This approach also allows the reutilization of previously developed Java labs by deploying the model in a separated device, the server. This chapter addresses the architecture and detailed characteristics of this implementation.

## 5.1. Easy Java & Javascript

The **Java & Javascript (J&Js)** solution is a logical evolution of the tool. Java, as inherited programming language is still an excellent choice to implement the functionality on the server side, (i.e. hardware access). Using Java at the server also reduces the computational load by taking care of the simulation or image processing. On the client-side, a web application (based on HTML and JavaScript) is a better option to provide visual feedback and interaction to the user. In this regard,

[1] https://www.tiobe.com/tiobe-index
[2] https://www.tiobe.com/tiobe-index/java/

to obtain a complete decoupling, the architecture also allows both parts, the Java model and the HTML-Js view, to exist as individual entities.

Unfortunately, there are some problems associated with major changes in software on continuous development, as EjsS is. Some drawbacks associated with the migration from one language to another have been presented before. In the EjsS implementation some of these drawbacks are also present:

- **Programming language**. The context of use of EjsS greatly reduces the common implications of changing the programming language to two considerations:
  - **Re-Build pre-existent labs**. The migration from Java to Javascript implies the need to rebuild the labs or even redesign them from scratch, considering the restrictions imposed by Javascript. There are two main concerns that must be addressed:
    - **Access to local data**: Developers have different approaches to cope with this restriction, and there are github repositories, public *Do It Yourself* (DIY) web pages that contains examples of different ways to access the file-system or hardware using Javascript, but this is not always in the best practices when developing JS and HTML laboratories. Javascript is used in mostly all web-sites to allow the interactivity in web-pages and all web-browsers contains a Javascript engine. In this sense, JavaScript code inside a web-page should be restricted to web-related actions [111]. The Javascript sandboxing and same-origin policy [3] are two constraints which are included in browsers to minimize non-web actions like accessing the hard-drive. Therefore, problems arise in research areas like IoT or navigation apps, where connecting with hardware is not just a good practice but is fundamental to be functional. However, the GUI of VRLs do not usually require access to the user file-system or hardware, but some data exchange. The exchange of data is a key element at experimental courses. On this matter, architectures for online laboratories, presented in Chapter 1 provide solutions to handle data exchange, file upload, storage and download using different approaches.
    - **Use of third-party libraries**: Java has been used as programming language to develop applications since around 1995[4], therefore,

[3] https://tools.ietf.org/html/rfc6454

[4] http://www.oracle.com/technetwork/java/javase/overview/javahistory-index-198355.html

many libraries have virtually grown with the programming language, and also many of them have died during Java lifetime. Many Javascript libraries have not been and will not be developed. This lack of libraries implies, that some labs must be re-built due to the nature of Javascript or because Javascript is younger than Java. In any case, some problems have to be solved using new libraries, slowing down for a short time the development of new laboratories.

- **Multi-device access**: The change in the programming language should consider the compatibility with mobile devices. The solution presented is compatible with new devices and also with those which have less computational resources.

As said in previous sections, EjsS manages two parts of a VRL, the view and the model. Both parts, are already equipped with the capabilities to exchange data using JSON. EjsS is able to manage these interactions by identifying the *method* which is going to be used. Each method will trigger a different process in the model or the view. The method name is included in the format of the message, and the implemented solution supports two structures:

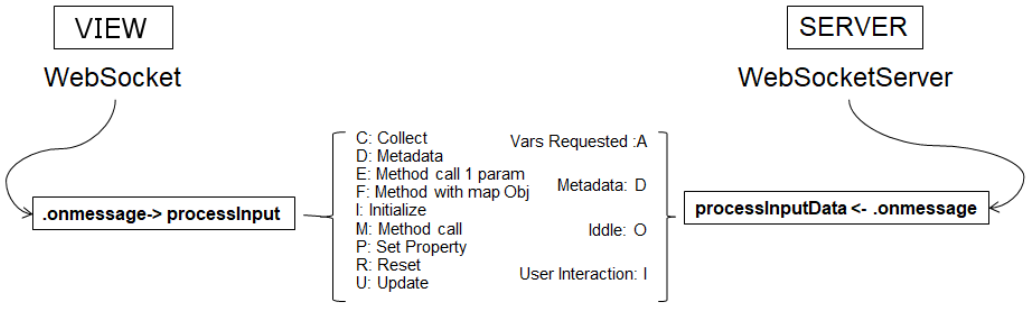



**Figure. 5.1** All headers of the EjsS oriented structure in a single message

- **EjsS oriented structure**: Is a mixed format, composed by a header and a JSON message. Figure 5.1 shows the headers supported by both the view and the model. EjsS oriented structure is meant to be minimal, exchanging just the minimum information needed, relying on the coding of the GUI and the model to perform the tasks and to retrieve the additional information from itself. One goal of the JSON format is to provide an easy way to understand the information inside a message. As these headers difficult the debugging process for the final user of the editor, in practice, they are not being used at

present. There are a limited set of situations when the format is used. First, to support model made by prior versions of the tool. Second, when the system is running simple examples in the local machine.

- **Smart Device Specification (SDS)**: SDS has been presented before, and it is the best option, as it provides wider and more standardized communications. The implementation of J&JS enabled version uses the SDS and it contains additional features to write, read and fulfill SDS like data structures. Another key benefit of this choice, is that the architecture is similar to other EjsS elements, such as RIP element[5]. Some model elements of EjsS have been using the JSON format for a long time using structures close to SDS format [6].

Therefore, the J&Js version should be able to create views to communicate with systems written in other programming languages just considering a few rules regarding the communication. Consequently, some problems related with changing the programming language (see Chapter 3) will not be an issue anymore in future similar situations.

## 5.2. View side

The main aspects of building the GUI are already solved when using EjsS. Since the tool is in charge of initializing, synchronizing, drawing and plotting the elements inside the view. Then, decoupling the model-view pair must maintain the information flow between both sides, even when each one is in a different machine.

### 5.2.1. HTML+Javascript

A closer look to web pages designs shows that making an interactive, user-friendly and flexible GUI is not an easy task. The GUI for a VRL must contain interactive elements, some computation capabilities, dynamic data and it is always nice to have a good looking interface. The implemented solution facilitates the work of developers, by giving tools to combine HTML, Javascript and Cascading Style Sheets (CSS), Figure 5.2.

- **HTML** is in charge of the structural building of the view. It uses a hierarchical structure, defining parent and child elements inside a tree.

---

[5]https://github.com/UNEDLabs/rip-spec
[6]https://github.com/UNEDLabs/rip-python-server

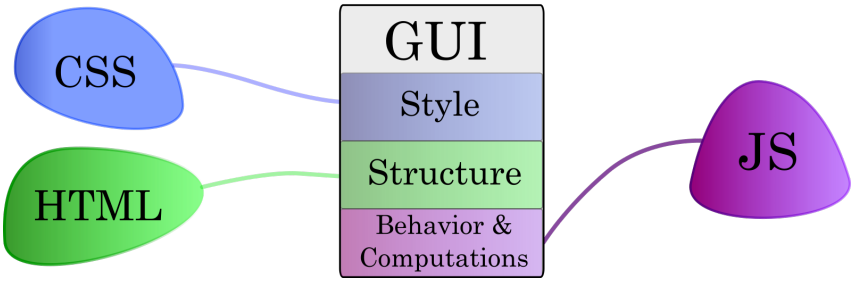



**Figure. 5.2** A GUI of a virtual laboratory

- **Javascript** carries out the behavioral part of the GUI. From special actions, when interacting with the view to computations, Javascript is used to code any additional capability.
- **CSS** is a stylesheet language which guides browsers to change HTML elements' properties (the color, font, layout, etc.)

Developers create their GUIs and can control some interactive functionalities or style parameters just editing fields. A user interface created by using the J&Js enabled version of EjsS will generate all the code needed to build the web-page. The webpage will contain generated code in Javascript, HTML elements and CSS even if the developer has a minimal knowledge about all these technologies.

Figure 5.3 shows a GUI made in EjsS using: HTML, Javascript and CSS. The Javascript version of EjsS helps in the development of GUI made with this three languages. The same view can be also created in the Java&Javascript editor, but the user can not differentiate between both. In the first one, model and view both use the same programming language and the communication is done in a straight way, as both are running in the same machine and browser. In the second one, model and view are defined in different programming languages and the communication processes are made in a transparent way for the final users as well as for the developers. The solution implementation adds the needed Javascript code to control the communication, update and interaction.

The user side is divided in parts and components. Each part supports main tasks: visual interfacing, communication and message processing. One main task is divided in a set of subtasks, some of these relies on components. This section describes how this components work together inside the GUI. Figure 5.4 shows the client structure:

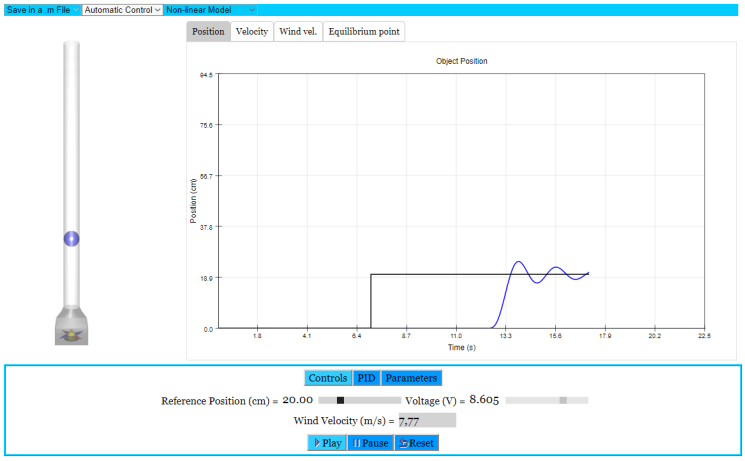



**Figure. 5.3** A GUI of a virtual laboratory

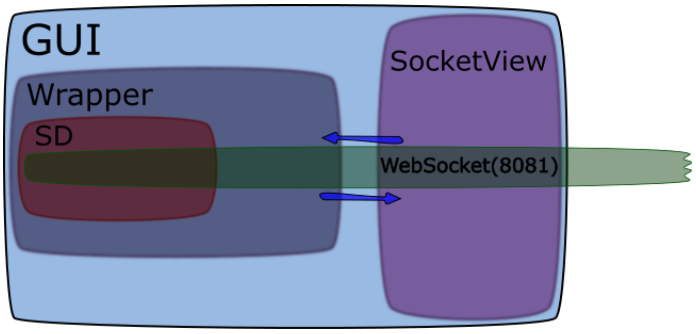



**Figure. 5.4** Client side internal structure

- A general container, the GUI, which contains also the graphical representations and interactive elements.

- The Socket View component is defined inside the global container, this socket view in in charge of the communications: definition, configuration and listeners related to the Websocket protocol.

- The wrapper is defined to surround the format element, is an interface between both, and prepares the data to be used by the EjsS view.

New capabilities have been added in the initialization, auto-generation and the update processes. Figure 5.5 show these features in a flowchart, further lines give a description of each process and detailed information.

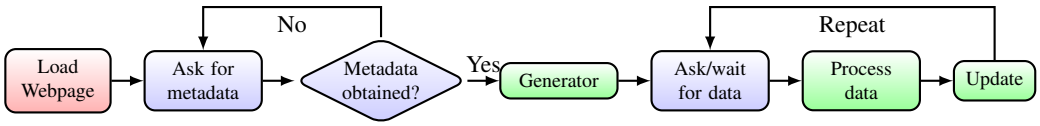



**Figure. 5.5** General flow chart of the initialization of the VRL

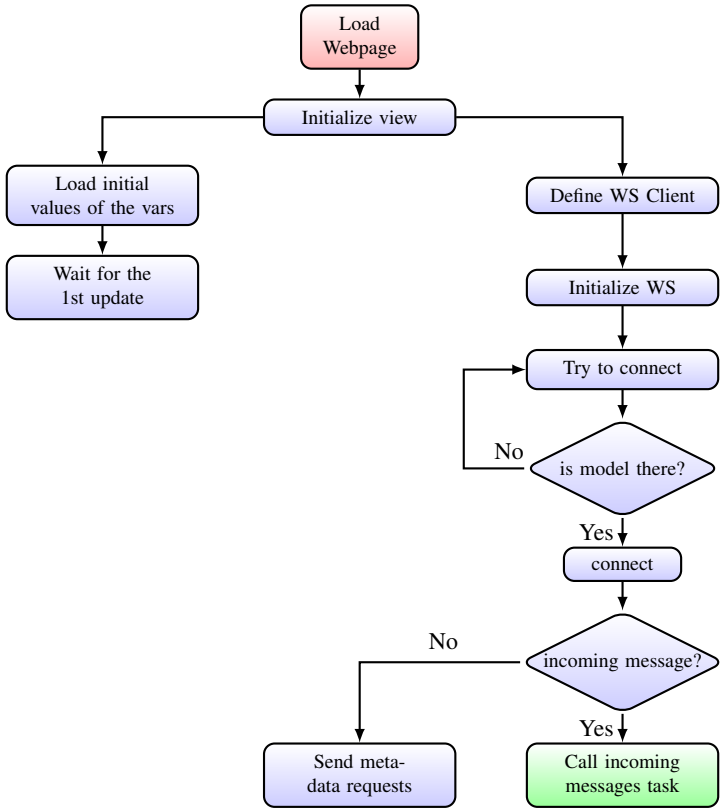



**Figure. 5.6** Initialization flow chart of the GUI

### 5.2.2. Initialization process

The initialization process is called as soon as the web-page is loaded. The Figure 5.6 shows how the initialization is made inside the GUI.

1. On the load of the webpage the view creates a basic, but not yet functional view structure. To be functional, the GUI needs to obtain the metadata which defines the lab and the first update message.

2. The Socketview component tries to connect with the server side by sending

requests to open a Websocket with the server side.

The flow chart shows that the first step of the initialization ends with the visual representation waiting for the first update and the client waiting for a metadata answer from the server. The time sequence diagram of Figure 5.7 shows how the threads are created and when the processes are called. If the message contains metadata, the work flow goes directly to the next step, which is in charge of auto-generating the needed view structure. To do that, the view contains a method which is called *Generator*.

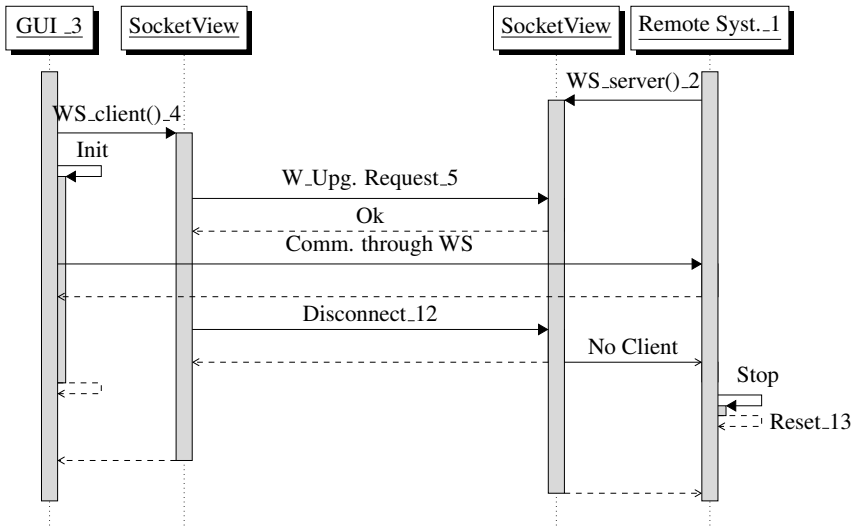



**Figure. 5.7** A sequence diagram of the initialization of the client

When the first message arrives the client, the view is ready to process it by using the *incoming message* task which is described in Figure 5.8. The flow chart shows two branches, the left one will carry out the non-metadata messages, while the right will process the metadata messages.

At the initialization, the right branch is the first to be used, as the first expected messages are metadata. When the Websocket is established, the connection acknowledge from the server usually contains no payload, then, the Socketview will ask for the metadata of the laboratory. This metadata contains information about the lab and defines the API to interact with it. It also contains information about the type of the data to be send/received and also all the sensors and actuators accessible remotely. As said in Chapter 3, by default, the format used follows the Smart Device Specification.

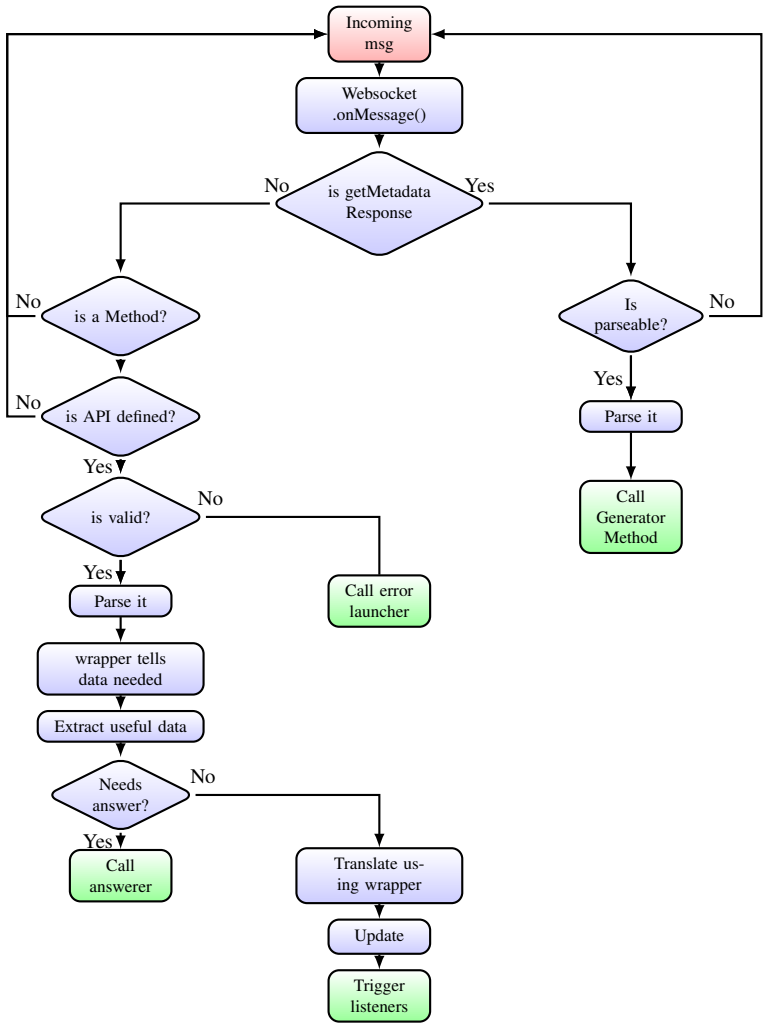



**Figure. 5.8** A flow chart of the behavior with a message from server

### 5.2.3. Auto-generation process

Generator method will parse, classify and redirect the messages using a workflow which is described in Figure 5.9. The first type of message expected is the API definition, using this message the GUI will create, in an autonomous way, methods to support the message building and verification. Building methods include templates for any data-model needed. The next subsections provide further details on that matter.

After the handshake, the client is ready to *talk* with the remote lab's model.

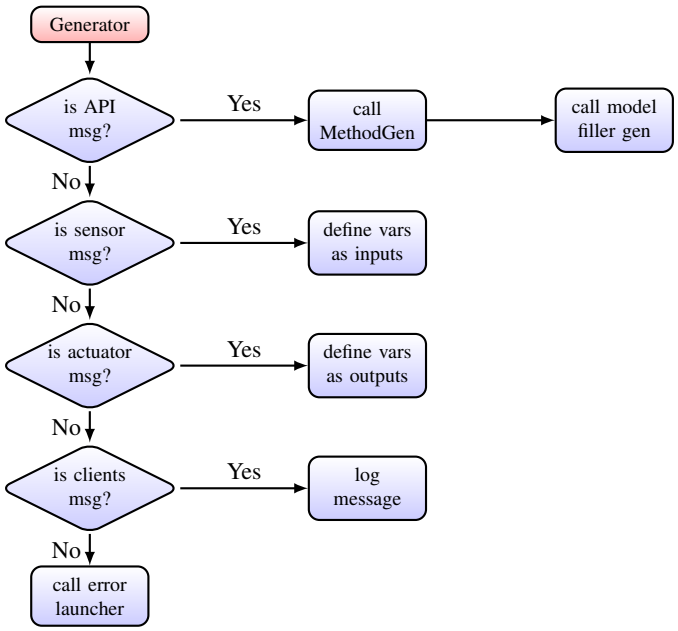



**Figure. 5.9** A flow chart describes the generator process

If the metadata is known, the format on the conversation is also known. Thus, the client side knows how to send and process the messages, but the GUI is still unaware of the sensors, actuators or variables that can be exchanged. Therefore, the next step is to ask for the list of sensor and actuators. SDS defines three types of elements: sensors (readable), actuators (writable) and variables (both). EjsS defines four accessibility modifiers: input, output, public and protected. The mapping is shown in Figure 5.10.

| Type | EjsS | SDS |
|---|---|---|
| Readable | Inputs | Sensor |
| Writable | Outputs | Actuator |
| Mixed | Public | Variables |
| | Protected (Local GUI vars) | |

**Figure. 5.10** SDS defines two types of elements: sensors (readable) and actuators (writable). EjsS defines four accessibility modifiers: input, output, public and protected

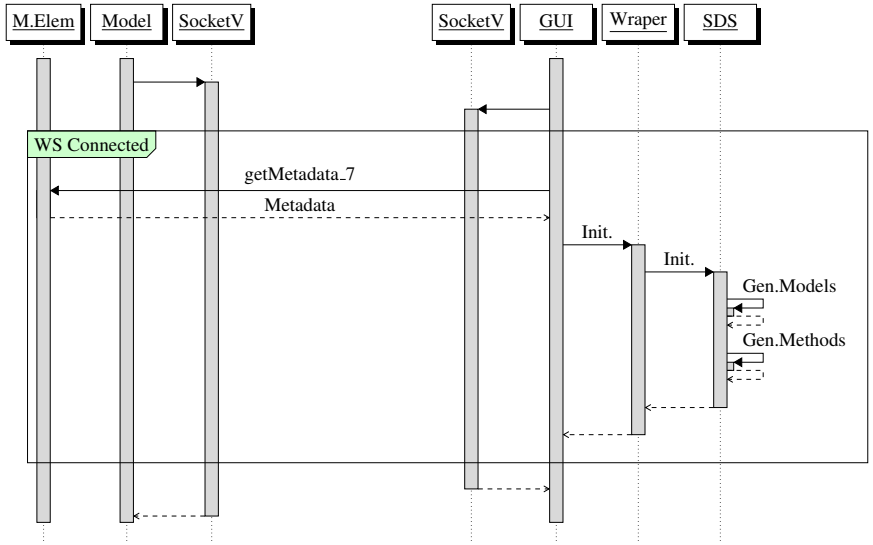



**Figure. 5.11** A sequence diagram of the initialization of the client (inside the WS connected box)

When all the exchangeable items are defined the system will be fully prepared to perform theirs tasks as user interface. Figure 5.11 shows a sequence diagram which describes the metadata request and code generation. At the time of writing this document these messages are just logged inside the tool, but can be used in future works inside multiuser sessions.

```
    "apis": [
    {
      "protocol": "websocket",
      "produces": [
        "application/json"
      ],
      "operations": [
      {
        "method": "Send",
        "nickname": "getSensorData",
        "summary": "Get data from the sensor with the given sensor identifier",
        "type": "SensorDataResponse",
        "parameters": [
        {
          "name": "message",
          "description": "Returns captured values",
          "required": true,
          "type": "SensorDataRequest",
          "paramType": "message",
          "allowMultiple": false
    }],{...}]}]
```

**Figure. 5.12** First API function defined using SDS in a JSON structure

### SDS and format

The tool allows using different formats to process the data. In most cases, the default data format (the Smart Device Specification) fulfils the laboratory needs. In this respect, EjsS has coded functions to create the main methods to interact with the SDS component, and the wrapper to interface with our tool.

Figure 5.9 shows respective calls to methods: *Generate Methods* and *Generate Model Filling*. Both methods use the SDS to easily create in/out capabilities inside the main SDS structure, leaving to the wrapper a small set of methods to send/receive/check data.

***Generate Methods***: this method uses the definition of APIs inside the first metadata message. Each API is defined by a JSON structure like the one shown on Figure 5.12. The JSON metadata contains the *nickname* to call the method, the *type* of data produced and the parameters needed to use the method. The parameters are defined following a common shape: *name*, whether is *required* in the structure or not and the *type* of data. Using this information, the *method generator*, makes a *just-required* version of the method which is shown in Figure 5.12. As API definition contains also optional parameters, the *method generator* will also create a full method, containing required and optional parameters as inputs. The flowchart of Figure 5.15 describes how the process is carried out.

When all the APIs are processed, the methods are accessible from the SDS structure and also from the wrapper, to perform all the operations needed to send a coherent message to the model. The method generated by the tool is created following a specific definition of the API operation, but this is not fixed. Methods and models are created dynamically after the metadata arrives. Then, if the API definition inside the metadata changes (between two accesses to the web-page) the auto-generated methods will change also. If the API defined in the metadata is changed, the SDS structure is created following the new specification, and usually the developer has no need to modify the wrapper or the GUI.

***Generate Model Filling*** uses the definition of models and types of data inside the first metadata message. Each model is defined by a JSON structure like the one shown on Figure 5.16. A model contains the information and structure to define requests, sensors, actuator, clients or any other element desired by the developers. The model contains a unique *id* to identify it, a set of optional *properties* that can be added to the structure and *required* properties, needed to make a coherent model. On the one side, required properties must be included to send a structured and comprehensible message to the model. For example, if you need the value of a sensor, you need to specify which sensor, if there is more than one. On the other side, optional properties include additional information or requests which are needed eventually.

```
function getSensorData(method,sensorId,configuration,accessRole) {
  if( method == undefined)  {
    console.log('Required parameter, method : must be an input');
    errorLauncher('Required parameter');
    return null;
  }
  if( sensorId == undefined)  {
    console.log('Required parameter, sensorId : must be an input');
    errorLauncher('Required parameter');
    return null;
  }
  this.websocket.send(\acs{JSON}.stringify({
    'method' : method,
    'sensorId' : sensorId,
    'configuration' : configuration,
    'accessRole' : accessRole}));}
```

**Figure. 5.13** First autogenerated method

For example, obtain the value of a sensor and also, configure it to be read every 50 milliseconds. Using this information, the autogenerator takes into account all the required parameters and makes a method to fulfill models. Figure 5.17 shows a flowchart representing the full process.

```
{
  "method": "getSensorData",
  "sensorId": "Position"
}
```

**Figure. 5.14** The simplest message to be sent using the getSensorData API definition of Figurefig:autogenMethod

To explain this process, we compare Figure 5.13, which contains the API defined in JSON and Figure 5.16, which contains the data-model and Figure 5.13, with the generated method. The API definition contain severals fields which are needed to the example: *method, nickname, parameters* and return *type*. Figure 5.14 will help to understand the minimum information needed to be sent.

- *Method* defines how to use the API operation: In the example the value is *send*, which mean that the user must send a message tto the server.
- *Nickname* is the name to be used to call the method, i.e the name of the method at the model side (see Figure 5.14.). The nickname field, Figure 5.12, contains *getSensorData* which corresponds to the name of the Javascript method.
- *Parameters*: The parameter JSON object contains six fields, in this example, we will consider just two: the *required* and the *type* fields. The type is

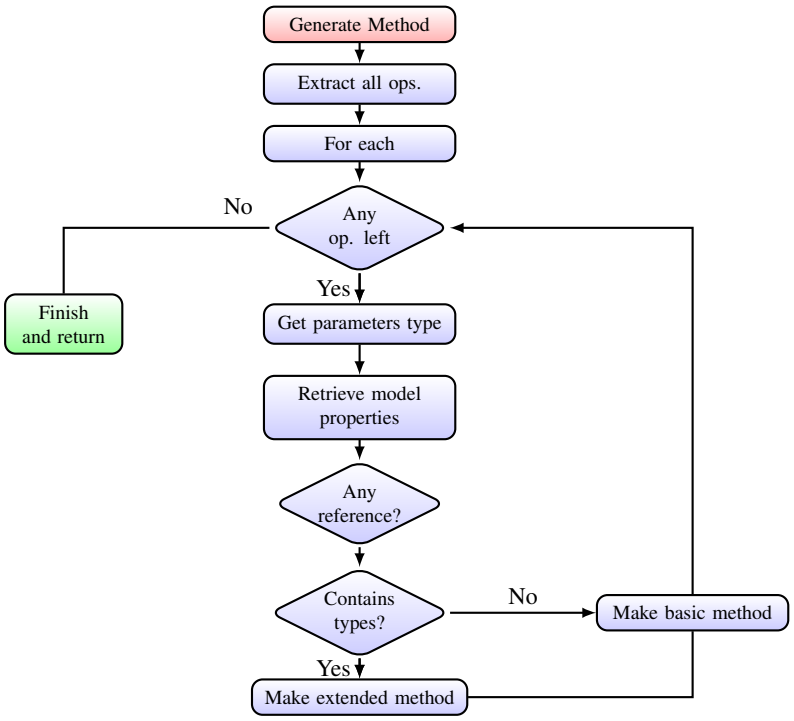



**Figure. 5.15** A flow chart of the method generator

*SensorDataRequest*, which is a data-model, Figure 5.16. It shows that the structure needs a *method* and a *sensorId* values to be built. Which are the first two input parameters in the autogenerated method (Figure 5.13). These two parameters are required, therefore, are also inside conditional structures to verify that they are defined. Last two parameters are optional: *configuration* and *access-role*, both will need their own models to be built (Appendix E), but are not included in the example.

- Return *type* has the value: *SensorDataResponse*. This name also references to a data-model, which is shown in the Appendix E.

Both capabilities, the autogeneration of methods and models, assist developers to reduce the coding effort, following the foundations of EjsS and enhancing the flexibility of the tool with respect to a fixed definition of the APIs and Models. Therefore, when the API is changed in the server side, the client side can adapt itself. Eventually, a change may imply a major modification in the data structure. In those cases, the structure surrounding the SDS component (the wrapper) must be changed too.

```
    "SensorDataRequest": {
      "id": "SensorDataRequest",
      "required": ["method", "sensorId"],
      "properties": {
      "method": {
        "type": "string",
        "description": "The method should be equal to the nickname of one of the provided services."
      },
      "sensorId": {
        "type": "string"
      },
      "configuration": {
        "type": "array",
        "items": {
          "$ref": "ConfigurationItem"
        }
      },
      "accessRole": {
      "type": "string",
      "description": "This field contains one of the roles defined in the concurrency roles list. If
          accessRole is not defined, the controller role is assumed."
    }}},
```

**Figure. 5.16** First autogenerated models

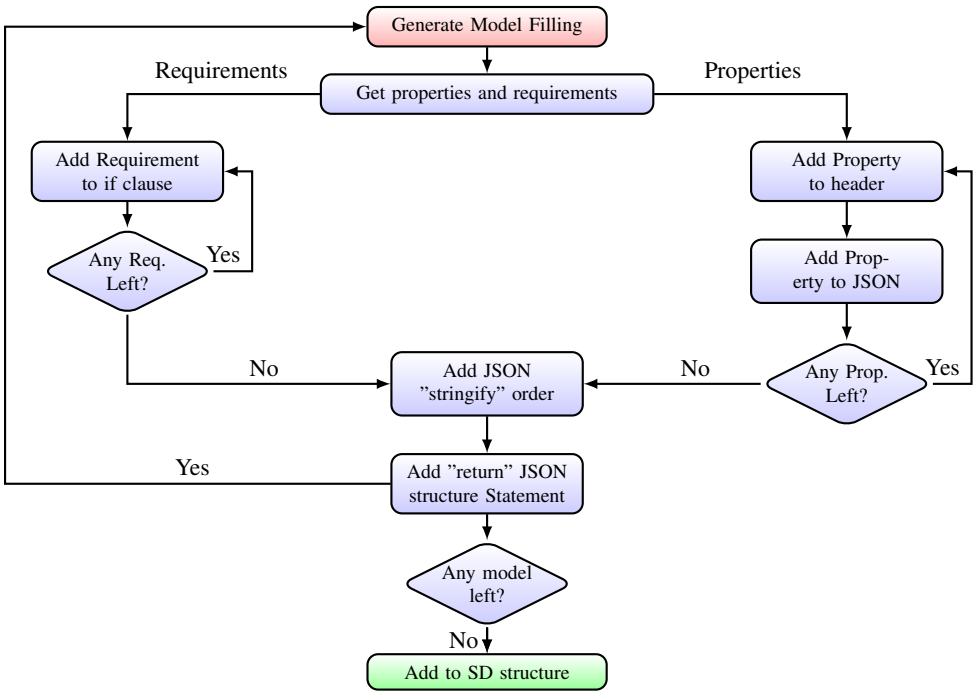



**Figure. 5.17** A flowchart representing the model filling method

### SDS Wrapper

"A wrapper is a software design pattern that allows the interface of an existing class to be used as another interface" [112]. The SDS wrapper is an interfacing software to allow the communication between both structures, SDS and GUI. The wrapper adapts the implementation of the SDS structure to provide the client with an interface that complies with the API of the lab. Those methods are not fixed and the wrapper is defined as a configurable element inside the J&JS enabled version. Therefore, the interface is configurable and the developer can edit the main code or even re-write it from scratch, but always maintaining the interface skeleton: methods from/to the SDS and from/to GUI. The wrapper solution provides developers with more flexibility to cope with different scenarios.

As seen in Figure 5.4, the wrapper is a structure which surrounds the SDS element, and will adapt, route and translate the inbound/outbound messages. On the one hand, when the lab's developer creates the model and the view inside the EjsS editor, the default wrapper is commonly enough to maintain an adequate communication flow. On the other hand, connecting to non-EjsS models or labs may need some minimum changes to adapt the interfacing. This shall be accomplished by one of the following three options:

- Modifying the main code of the wrapper, which is accessible to the user, adapting the methods to cover the needs of the actual model or the view.

- Filling a template that is also supplied by the editor. It contains the skeleton of the wrapper with the methods required to communicate with the *view* and the SDS element. Figure 5.18 shows the template from the editor, it contains empty and coded methods, which correspond to optional and interfacing methods.

- Writing the wrapper from scratch, if none of the previous options fulfills the needs of the developer. Though, the developer has absolute flexibility with this approach, it is usually the less desirable option, because it implies a reimplementation of the functionality. Writing the wrapper is also the most uncommon option, because the APIs are defined in a similar way inside the metadata and the EjsS view needs are fixed.

As Figure 5.4 shows, all the components of the GUI share a unique Websocket element. In this sense, visual elements, SDS and wrapper can use the WebSocket to send messages. Sharing the Websocket connection is not needed, as long as the SDS component contains methods to handle the functionalities available. Nevertheless,

```
Wrapper.prototype = {
  methods: {
    connect:  'connect',    // Open a new connection
    disconnect: 'disconnect',   // Close the connection
    get:    'get',      // Get some variables from the server
    set:    'set',      // Set some variables in the server
    callAction: 'callAction', // Calls a method of the model
    extract:  "extract",    // Get/check useful info from a message
    eval:     'eval',     // Send code to evaluate
    open:     'open',     // Send an order to open files or similar
    step:     'step',     // If the evolution in the server is done by steps, make just one
  },
  post: function(method, params) {//calling some method with no prebuild method, just sends a
      message
    if(method == undefined) this.websocket.send(\acs{JSON}.stringify(params));
    else          this.smartdevice[method](params);
  },
  sync: function(callback) {
    if(callback != undefined)   callback(response);
  },
  connect: function(callback) {
    this.smartdevice = new _smartDevice(fullData);
    this.smartdevice.loadApis();
    this.smartdevice.loadModels();
    this.smartdevice.generateMethodCall();
  },
  get: function(vars, eachStep, callback) {
    if(eachStep === undefined || eachStep == false) {
      this.smartdevice.getSensorData("getSensorData",vars);
    }else{
      this.smartdevice.getSensorData("getSensorData",vars,[this.smartdevice.fillConfigurationItem(
          "updateFrequency",2.0)]);
    }
  },
  set: function(vars, values, auth, callback) {
    if (vars.length == 1) this.smartdevice.sendActuatorData(auth, "sendActuatorData", "modelVars",
        [vars], [values]);
    else    this.smartdevice.sendActuatorData(auth, "sendActuatorData", "modelVars", vars, values)
        ;
  },
  extract: function(strMsg, callback){
    var jsonMsg = \acs{JSON}.parse(strMsg);
    var vars = {}, extracted = {};
    if(jsonMsg.hasOwnProperty("method")){
      var method = jsonMsg.method;
      if(method == "getSensorData"){
        vars = ["sensorId", "responseData"];
      //(...) all the types needed
      }else return undefined;
    }
    extracted = this.smartdevice.extract(vars, jsonMsg);
    return this.toEjssReadable(method,extracted);
  },
  toEjssReadable: function(method,dataExtracted){
    var nameValue = {};
    if(method == "getSensorData"){
      var valueNames = dataExtracted.responseData.valueNames;
      var dataValues = dataExtracted.responseData.data;
      for(var i = 0; i<valueNames.length;i++)   nameValue[valueNames[i]] = dataValues[i];
      return nameValue;
    }else if (method == "sendActuatorData"){
      var valueNames = dataExtracted.payload.valueNames;
      var dataValues = dataExtracted.payload.data;
      for(var i = 0; i<valueNames.length;i++)   nameValue[valueNames[i]] = dataValues[i];
      return nameValue;
    }else{
      //(...) all the types needed
    }else{
      console.log("The method : " + method + " is not handled in this wrapper. \n" +
      "Will be ignored.");
      return undefined;
    }
    return dataExtracted;
}}
```

**Figure. 5.18** Wrapper template given to the developers from the editor

to enhance the default wrapper by providing a higher level of flexibility, the Websocket is accessible by using the *post* method. Calling the *post* method will send parameters inside a JSON structure without using the API specifications, providing a method to debug errors and to do some testing in development time. The tool becomes more usable because it is adapted to developers with different programming knowledge. For example, lets define an API as simple as possible for a system with just one sensor and one actuator:

- To change an actuator just send the raw value embedded in JSON structure. The system contains only one actuator, then, its name or ID is useless, and the structure will be like

$$\{\text{"}method\text{"} : \text{"}sendActuatorData\text{"}, \text{"}value\text{"} : 0.0\}$$

- To obtain a value, send a *read* message. Using one sensor implies that a reading can be done just calling the method like:

$$\{\text{"}method\text{"} : \text{"}getSensorData\text{"}\}$$

In this situation, the API definition is simple but enough to interface with the hardware, if there is no need to add other capabilities, developers may use one of these two solutions:

- After the initialization, generated methods and models can achieve both actions using the methods *set* and *get*, from the wrapper. Both processes will imply that the SDS fills model templates, therefore, are simpler for the developer.

- The developer can pass over the SDS component and send the messages from the wrapper, because SDS component has no need to check the messages or extract the information from the message, because its simplicity.

Choosing the solution is up to the developer needs, however, ignoring the SDS element is a solution suited for testing simple JSON structures or to be used in the course of modifications of APIs or the metadata.

Wrapper component has been presented like the main configurable part of the view from the developer point of view in the client side. The presented architecture needs another piece to define a complete lab.

### 5.2.4. Update process

The end of the initialization process ensures that the GUI is ready to be used. Models, methods, variables, sensors and actuators are generated to represent and exchange the data. Also, the user interface is prepared to process user interactions or to be updated when new data arrives.

A view defined using the EjsS editor will contain many individual elements, each one can be added, edited or deleted from a tree structure to configure the GUI. The configuration of these elements is commonly made using links between properties of the DOM element from the HTML view to variables, constant or even methods, defined locally. In this regard, those links are used also in *J&JS* enabled version, and works in a similar way, following the next update process:

- If the user interacts with the GUI: The variable will be updated locally, changing the immediate view aspects related to it, then the new value of the variable is sent to the model. The consequences of the change will be processed in the model, and finally, an update will be sent to all the available views connected.

- The model will send updates to the available views periodically, depending on the evolution time rate defined periodically (depending on the evolution time rate), event triggered or on request. Also, events can be defined, and if any is triggered a new update will be sent to the connected clients.

The messages carrying update information are sent from the model and are processed using the same flowchart presented in Figure 5.8. This flowchart shows how the left branch obtain the type of the message and then triggers and update of the view or call a method to send back an answer.

As said before, the complexity level of the laboratory's view is related to the goals to be achieved. The number of capabilities, and the complexity of the view is usually proportional to the number of sensors and actuator that are used. From the client to the model side, the view send a message when an interaction is detected. Each variable linked to a sensor or an actuator is connected to a listener, which triggers an action when detects the change. Updates and message events continously arrive to the client from the model. Building remote labs implies a trade-off between the volume of messages and the quality of service are defined in the lab application, and this will be on hands of the developers. Next section will describe this piece: the server side details and the model features.

## 5.3. Model side

The model is in charge of processing the data and interacting with the real system at the server side. On the one hand, the model is running its code every step of time. If one or more views are connected, the model will send updates periodically. On the other hand, the model needs to know how the user is interacting with the view at any moment and in some cases to send a specific change which only involves modifying one visual element (a color, adding point to a graph, pop-ups, etc). These features are in the hands of the model, and how it does the processing depends on the type of model which is working at the remote lab. EjsS supports two types of models, as both are capable to perform a coherent communication with views created using the tool.

### 5.3.1. EjsS Model

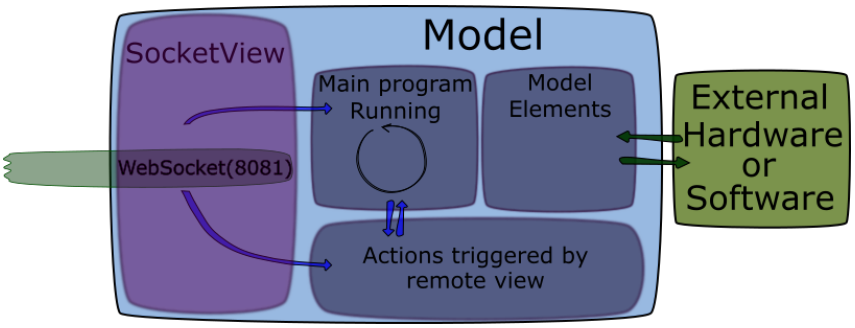



**Figure. 5.19** Server side internal structure

Model-view pairs created with Java & Javascript enabled version have built-in communication capabilities to transmit data between them. It is also possible to establish a communication with a preexistent EjsS model and just create the view.

EjsS models are divided into different structures, as the views. These structures contains components, each one in charge of one or more tasks. All the components are shown in Figure 5.19, representing its most important components and the relationship between them and with external entities. Figure 5.19 shows a general container, each EjsS model contains:

- A SocketView component that is in charge of the communications. It will establish communication with its counterpart in the view side. Therefore, is in charge of the communications: definition, configuration and listeners related to the WebSocket protocol.

- The structure also contains the *main program* or *evolution* of the system as a separated component. The evolution carry out with the processes programmed by the developer and a set of minimal model tasks. On this subject, the evolution is at least in charge of:

  - Control the internal time evolution. It allows to update the view periodically and to solve evolution equations.
  - Running the code of the model elements, like those to interact with hardware.

- Model elements, as said in previous chapters, may connect with hardware, external software or other devices, that the remote lab could need. Model elements also add functionalities, like the wrapper/SDS editing and translation

- The last component is responsible of actions and custom methods. Those user-actions can be triggered from the view and are defined, by the developer, inside the *custom* tab of the EjsS editor (Figure 5.29). Additionally, model and view internal methods are callable remotely. For example, *play* and *pause*, which are intrinsic methods to the model, may be triggered from the view when the user clicks in a button.

First launch of the laboratory begins at the server side, where the model runs. At the initialization, the model executes a set of tasks that has to be completed before being able to exchange data with the view. The model must make sure that all components have a valid state, run user-defined initialization code and configure the communications. Of course, not all the tasks are needed while the user is not connected. Therefore, the model will initialize the variables and the Websocket server. After initializing, the server listens for incoming connections. If a model is run from the terminal, the Websocket server will prompt a message if there are no clients connected for a long time.

When the user opens the webpage at the client side, the Websocket client is initialized. At this point, the client sends a handshake, (an HTTP upgrade of communications to the model). If the model is listening, the handshake from the server will accept the upgrade, establishing a communication. When the socket is created the model will run the initialization code, and then listen for the first message. Figure 5.20 shows how the model process incoming messages. Regarding this, the first message must be a petition for all the metadata: the APIs definition, the sensor set and the actuators set. All this information is processed during the package of the application and would be sent without delay to the client.

At this point, the model has the media to send and receive messages to and from the client. Although, the model has no clue about the data needed by the client. Client can make a request including the variables, sensors and actuators needed to be sent in the updates from model. If the model receives no request from the client, it will send all the variables (regarding or not to sensors and actuators) defined as *public, input* or *output* (see Figure 5.10).

Figure 5.21 contains the flowchart of all the processes needed to process this message. Periodic updates are not necessarily needed for every item. Only those elements containing an *updaterate* field are stored in the periodic update list and are sent with the synchronous updates. The elements without the *updaterate* field are sent just once, in the first update message.

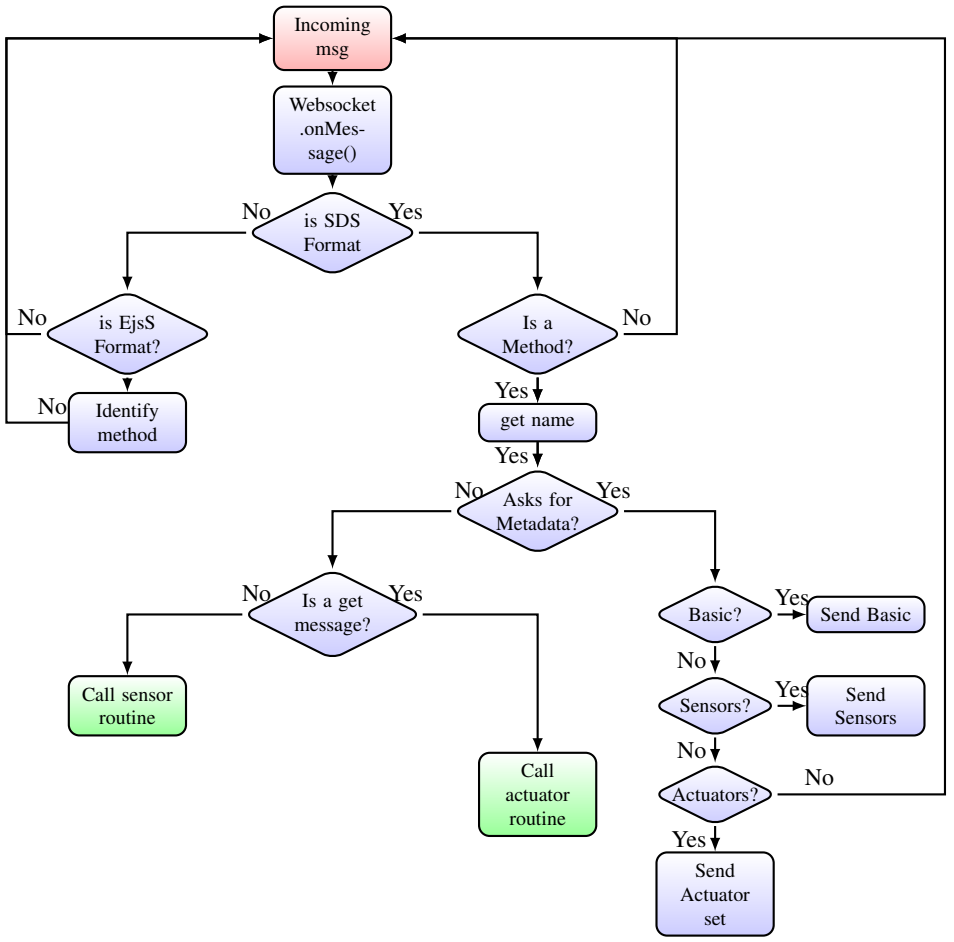



**Figure. 5.20** A flowchart representing how the model processes an incoming message

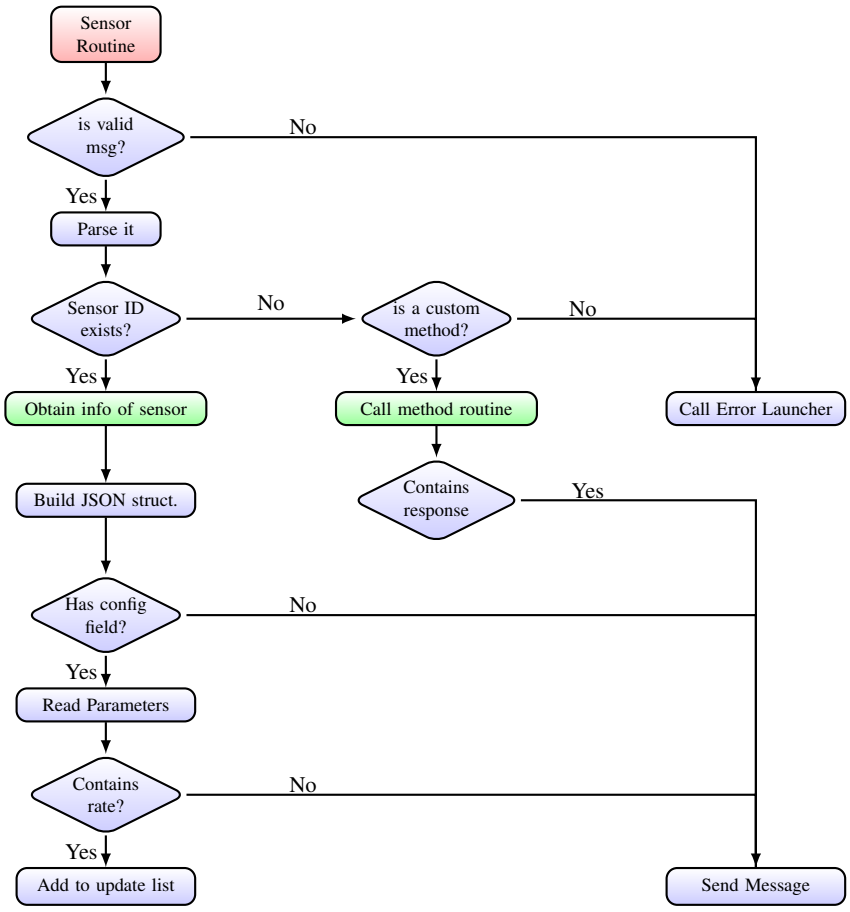



**Figure. 5.21** A flow chart representing the getSensorData routines inside the model

The next steps depend on the initialization code, commonly one of the following: an update from the model or a interaction from the user. If the evolution is already started, the model sends the first update message, with all the values of the sensors and actuators used in the GUI. If the evolution is waiting for an action, the next message incoming to the model can be an interaction from the GUI (for example, a click on a button). Figure 5.22 contains a sequence diagram to illustrate how the first update is obtained as result of a request message sent from the view.

During the normal life cycle of the lab there will be many messages from/to the model. Synchronous messages are sent at fixed periods of time, related to the step-time of the evolution. Asynchronous messages are not sent in regular intervals and can be triggered at any time. The causes can be many, from actions (which are calls

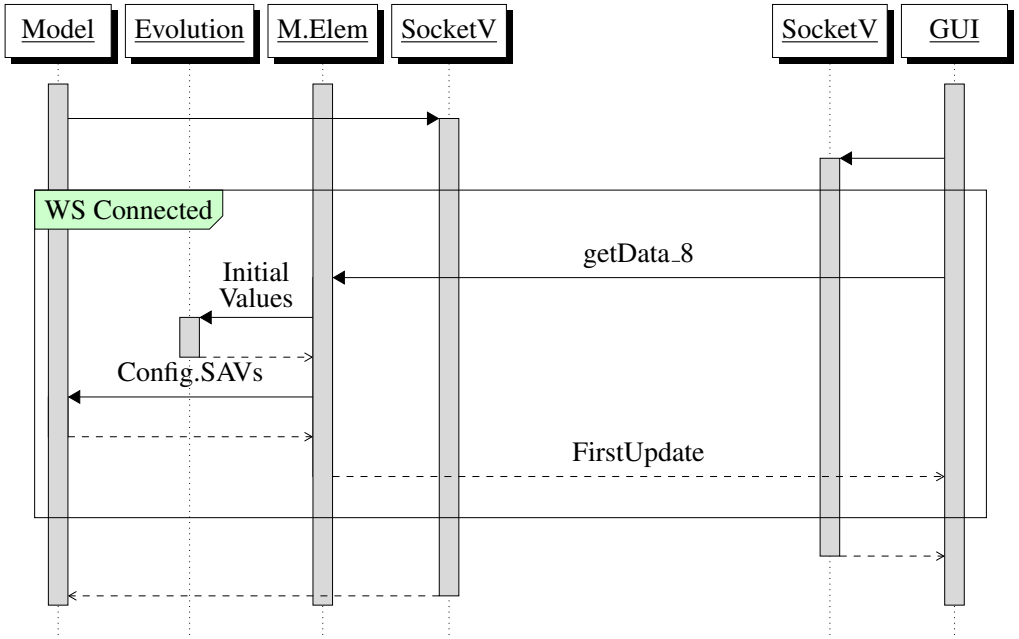



**Figure. 5.22** A sequence diagram of the SAVs[7] request and first update from model side (inside the WS connected box)

| Cause / Origin | Evolution Step | Action | Interaction | Model Event |
|---|---|---|---|---|
| Model or Java view | Sync | Sync/Async | ASync | Async |
| HTML+Js View | None | Async | Async | None |

**Table 5.1** Identify synchronous and asynchronous messages using their origin and cause

to model, custom or view methods) to periodic messages. On this subject, Table 5.1 presents a mapping between triggering causes and the synchrony of messages.

| Defined in | Model (Core) | View (Core) | Model (User) | View (User) |
|---|---|---|---|---|
| Target Prefix | _ | _view | _customMethod | _htmlMethod |
| Prog. lenguage | J (Model) | JS (View) | J (Model) | J (Model) |

**Table 5.2** Classification of actions by its target, programming language and their location (where are defined)

The structure of all the messages is defined inside the APIs, and, if there is

no addition to the main metadata file, the messages will be processed as shown in Figure 5.20. Incoming messages from the client are restricted to a finite set and are divided into three groups: Reading, writing and action call.

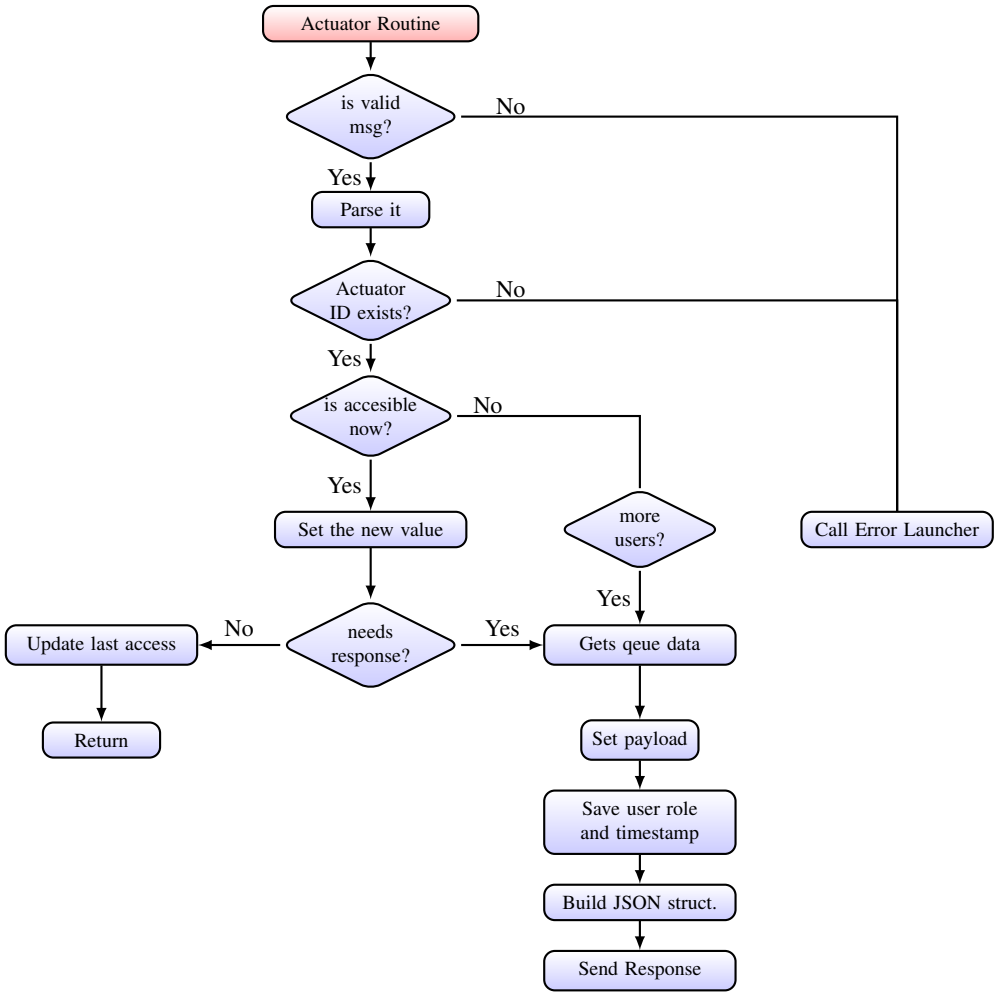



**Figure. 5.23** A flow chart representing the sendActuatorData routines inside the model

On the one hand, **Reading** is commonly a non-disruptive method, which only implies the return of data from the model to the GUI. The model does not need to be updated after the process (except for registering of variables in the synchronour update list). On the other hand, reading always implies an answer, then, after a variable or sensor has been located and the value has been obtained, the only task that is needed is to package it into a JSON message understandable to he GUI. Figure 5.21 contains the full process.

**Writing** means a change in actuators or in *input* variables, therefore, implies a different processing in the model with regard to the reading. Commonly, writings provoke an update in the model after the change, however an answer is not always needed. Figure 5.23 contains the full process, including some controls to verify that the lab is working like a multiuser session. On this point, if the change of the value is not possible, the system will return a message with the error or status. Actuators and sensors are physical elements and commonly are linked to model variables. Although, the model can define other variables to support additional functionalities. Figures 5.21 and 5.23 shows that the process make no-difference between variables or real sensor/actuators in the first coarse processing.

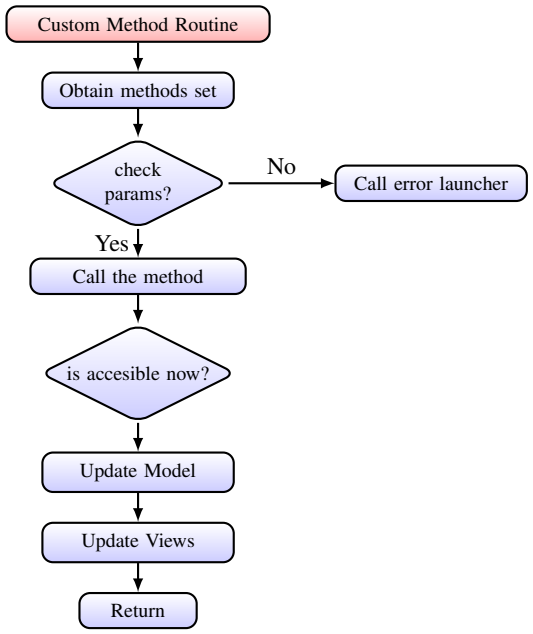



**Figure. 5.24** A flow chart of the custom method routine

Writing and reading values are intrinsic to input and output information to the system. Variables are entities which are by default read and write. Although, other processes and messages can trigger a change in *variables, sensors* and *actuators*. Actions are defined as methods to be called, therefore, any accessible element can be changed as result of the call.

**Actions** are defined in EjsS as calls to model, custom or view methods. Model and view methods are included by default inside the core of EjsS and custom methods are added by the developer. Messages to handle the actions are needed to maintain the normal work of EjsS Java & Javascript enabled version. Model and view

decoupling produces the splitting of some methods, related to being written in Java or Javascript. Actions are further identified by its target, which is specified by a prefix in the name of the sensor. Table 5.2 classifies the action by its target and their location (where are defined).

Prefixes allow the model to distinguish between actions. Excepting the *core* functionalities, the process is: message parsing, identification and calling method routines. Figure 5.24 shows the *custom methods routine* as a flow-chart. As methods are user-defined, the model needs to verify that the call request match with it. Non-custom methods are called without additional processes, because they can not be changed by the developer.

### 5.3.2. Non-EjsS Models

EjsS models are the most common situation when using the proposed tool, but a benefit of using the SDS is that it is possible to connect to Non-EjsS models too. Connecting to Non-EjsS models is an advantage when talking about sharing VRLs in education and research areas. In relation to this, commonly a GUI developer, who makes a interface for a remote system, needs to know many details about the system and how it works. The SDS provides that information in a standard and human-readable structure, grouping together: API's, models, descriptions for lab, sensors, actuator and so on.

In addition to the information about the lab, the developer will need to include some features in the model to meet the needs of the view. Basic needs of the view can be covered in the model by adding the characteristics listed below:

- *Metadata Services*: The view needs the metadata to provide tools to the user and to build many of the internal structures of the GUI; a precise description of the lab is needed. If the metadata is available and can be obtained from the server due to a metadata service, the editor of EjsS will handle the situation and will make easier the build of the view. Minimal requirements includes three metadata services.

  - To answer any *getMetadata* call with the API and data-models. The information is stored in a JSON file and sent on first connection or as a response to the call.
  - A service to answer to *getSensorMetadata* call with the sensors set. It is also a good practice to include inside the JSON structure information regarding the nature of the elements, the type of data obtained and appropriate names.

  - A service to answer to *getActuatorMetadata* calls with the actuator set, including all the additional information about them.

- *Data Services*: The view and model need to exchange the data from the lab by sending/receiving messages. Thus, the model must be able to receive and identify the different types of message. Minimal requirements includes four data services, in charge of receive the messages and then:

  - Check, parse and apply the changes sent by *sendActuatorData*.
  - Check, parse and read data from sensors when a *getSensorData* call arrives.
  - Check, parse and configure sensors.
  - Check, parse and configure actuators.

## 5.4. Timeline in a VRL session

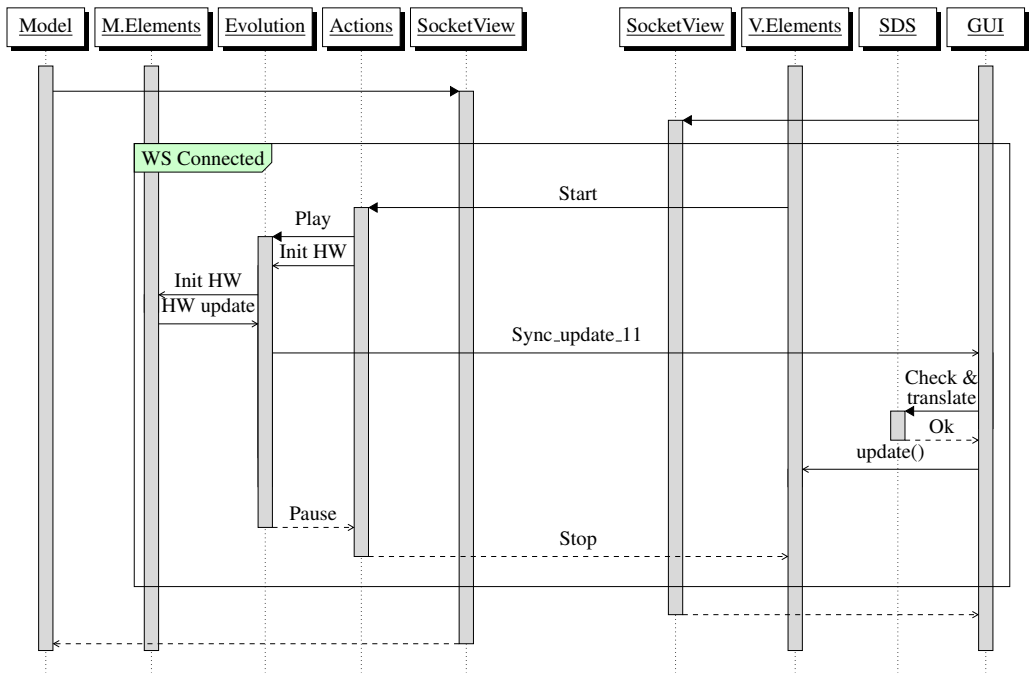



**Figure. 5.25** A sequence diagram of a generalized synchronous exchange of data when initializations and first updates are done (inside the WS connected box)

Last sections have established the basis to understand the foundations of parts of the Java & Javascript enabled version and how they work together inside the final application of a laboratory. Each part has been treated alone, just making references to the main needs to fit in the global architecture. This section presents an EjsS view and model interacting together from the first load to the end of the lab session, focusing in the messages exchange and timeline.

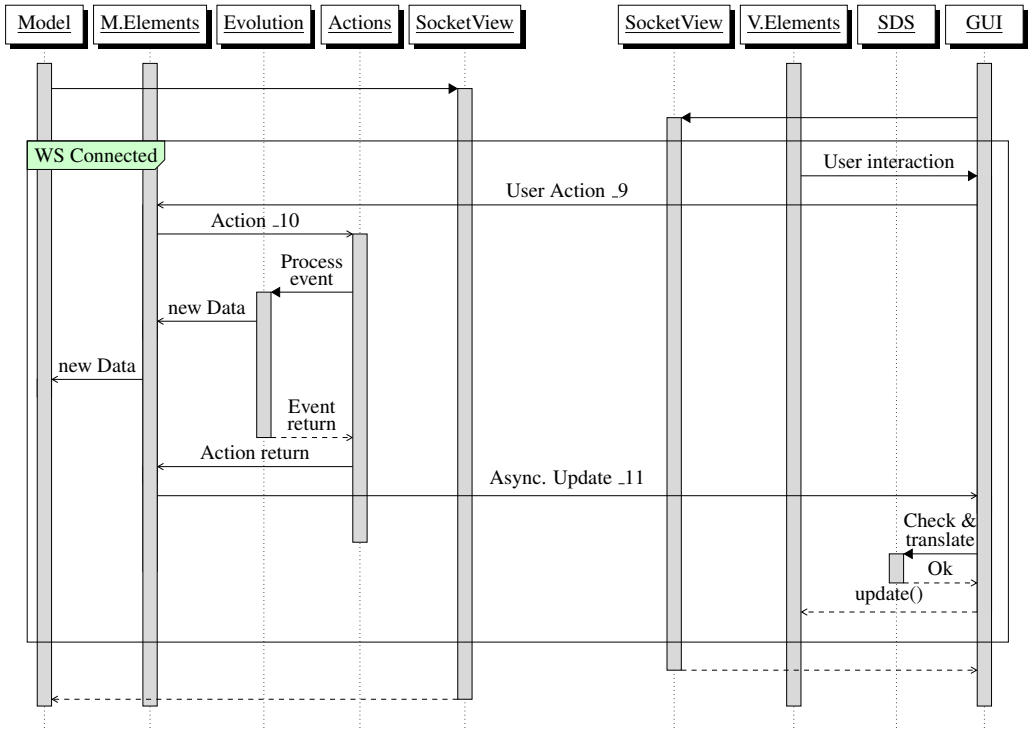



**Figure. 5.26** A sequence diagram of a generalized asynchronous exchange of data when initializations and first update are done and the model is running (inside the WS connected box)

Next step-by-step list shows all the tasks of a full lab session ordered by time. It starts from the launch of the model to the disconnect signal from user. Numbers of the list can be correlated with numbers inside all the time sequence diagrams presented before. Figures 5.25 and 5.26 present the exchange of synchronous and asynchronous messages. Since wrapper and SDS structure are defined inside the same view element, Figures 5.27 and 5.28 contain detailed processes to add complementary information to the diagram.

1. Model side: The **Initialize model** order can be done from a service, manually or any other method available inside the server side.

2. Initialize the model will trigger the **initialization of the Websocket server**. It will listen, waiting for a connection.

3. Client side: the user loads the web-page. It implies to **initialize the GUI**.

4. Initializing the GUI produce the **initialization of the Websocket client**. The GUI try to connect with a model at a IP address and port given by the developer. Websocket handshake uses initially the HTTP protocol in the session establishing phase.

5. If GUI and model are available, client sends an HTTP GET request containing the Websocket Upgrade Request. When the **connection is accepted**, the **communication established**.

6. Model side: the **evolution is initiated**. The evolution will run the first steps of the code or will initialize the equation solvers.

7. Client side: as the view needs the metadata to work properly, it will **ask for metadata**. If the client needs the information, it will also asks for the sets of sensors and actuators. When the metadata answer returns, the **initialize the SDS** structure is called. It triggers the auto-generators, creating the methods and models needed to a coherent data exchange.

8. Client side:The GUI makes the **first request of data**. This first request includes all the variables which are going to be included in the updates. Model side: the request triggers the **first update**, containing the values of sensors and actuators. As those may differ from the initial values, the view will be likely reshaped.

9. From this point, the model and client can be continuously sending messages or the model can be waiting for a message to run the application or to initialize hardware. If the system is not started it will wait for the play/start action, and when the message arrives, the model will trigger a method call. When hardware is initiated periodic sensor readings will produce model updates. For a student, the session with the real lab starts here, because the hardware data is available and the interactions will cause changes. Read, write and action messages are going to be sent/received, following the procedures illustrated in Figure 5.27 in the client side and 5.28. Life cycle of messages follow this sequence: arrives-precheck-routing-checking-extraction-translation-events for incoming messages and event-extraction-translation-formating-routing-send for outgoing messages:

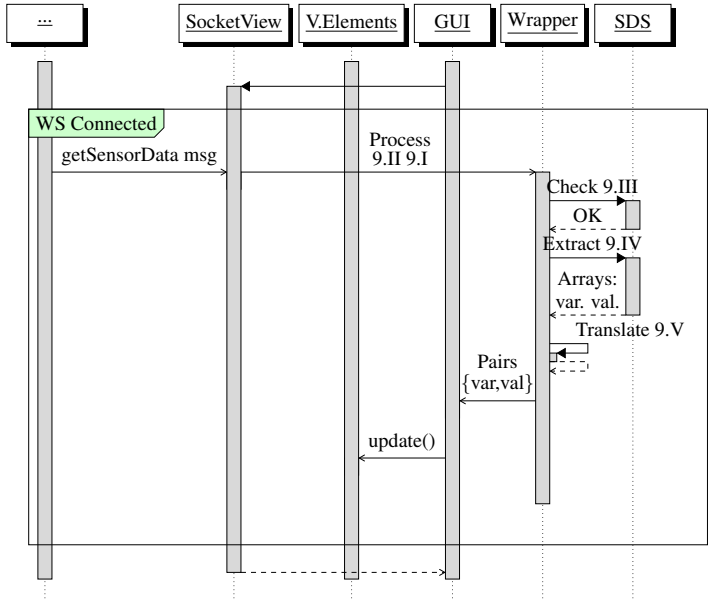



**Figure. 5.27** A sequence diagram representing the client side receiving a message from an EjsS model (inside the WS connected box)

I **Precheck** tests the message to verify that is JSON formatted.

II As the message is valid, now we can distinguish the type of message verifying if *method* of *API* fields are there. As soon as the message is identified a **routing** process starts.

III Each process do a **checking** to verify the presence of required fields.

IV A valid and readable message can be processed, to do that we need to **extract** all the information needed. This step may be different when using non-EjsS models.

V EjsS or model can exchange information, but each one may use a different format to understand changes and events, therefore a **Translation/-formating** is needed.

VI User interactions, actions and model events may trigger asynchronous messages during the lab session.

10. Model side: Any **action** which arrives the model may trigger an update in the model and therefore an update message to all connected views. This update is an asynchronous message. Figure 5.26 shows one of these asynchronous updates inside the sequence diagram.

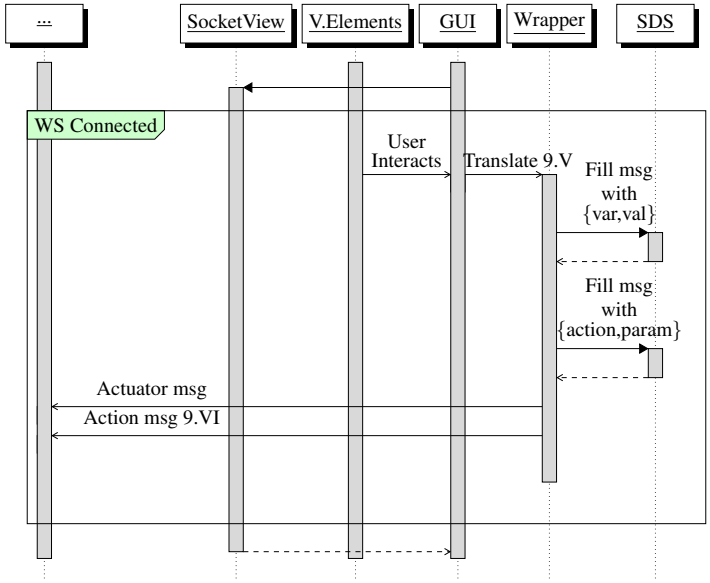



**Figure. 5.28** A sequence diagram representing the model side receiving a message, from the client, triggered by an user interaction (inside the WS connected box)

11. Periodically, other update messages are sent, these are synchronous messages. Figure 5.25 shows one of these synchronous updates inside the sequence diagram.

12. When the user wants to disconnect or if the model waits to much time without receiving messages the session will be finished. When the session is closed, the client will be disconnected.

13. The final task is to reinitialize the model and wait again for a new user.

Figure 5.7 contains an example of sequence diagram of a small lab session. Interactions and updates generated are examples to illustrate the process, but the session can exchange data during a full laboratory practice. Frequency and sizes will be treated at Chapter 7, where the advanced user-available features are described.

## 5.5. EjsS Editor

This new editor of the J&JS enabled version is similar to the original editor. Figure 5.29 shows that this new version preserves the main structure of the previous version, with some minor differences. The elements inside the user window have been briefly described in Chapter 1 and on the references given. In this regard, the top panel of the editor still shows the classical *View* panel to create a Java view, that will optionally run in the server. Additionally, the editor contains the *HtmlView* tab, where the developer can build the XHTML graphical interface for the final user. Both views can be built by using the same drag and drop elements of their respective editors.

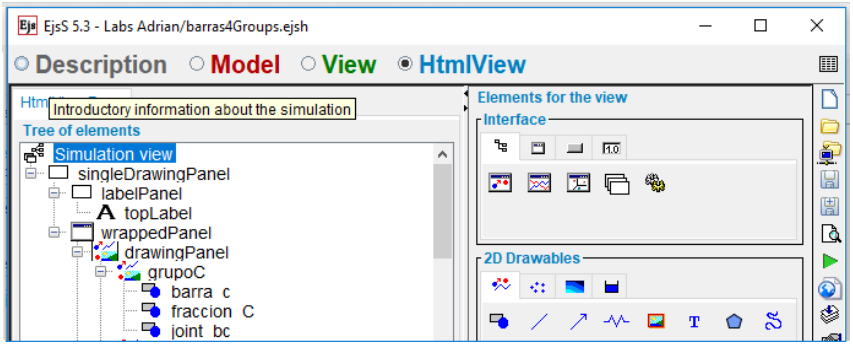



**Figure. 5.29** Main view of the EjsS editor, in Java + Javascript enabled version

However, the J&JS enabled version contains an additional configuration element which helps to set-up connections, the SDS element. These configuration allows to set: the server IP, port, wrapper and variables included in the application. The EjsS editor contains the model element, and using the configuration panel is also possible to make some automated tasks, like adding variables.

### 5.5.1. SDS element

Figure 5.30 shows the SDS element, which can be found inside the *Software Links* folder, added to the model within the configuration window. Having a common *shape* in configuration windows has a double goal, first, it is easier to the developers to understand and handle the configuration of various elements. Second, move closer to a unique element or grouping elements with similar capabilities. Similar configuration windows can be found also in LABView element and RIP element.

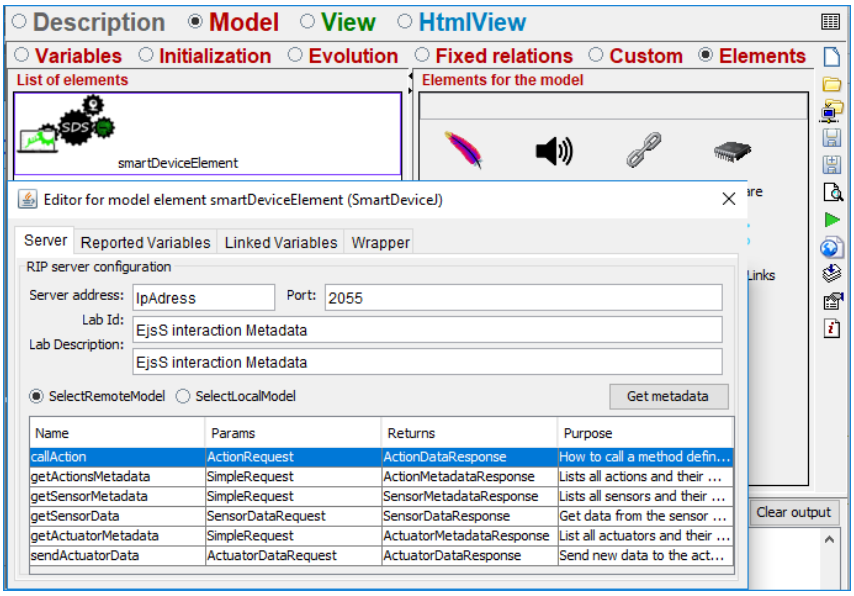



**Figure. 5.30** Configuration window of the SDS element when added to a VRL

The configuration window is divided in tabs. Each tab allows the user to perform different actions in order to understand the remote system, configure the connection parameters, obtain variables and edit the wrapper. The element provides two modes of operation: local and remote. The local mode is designed to run the application in the same computer, without configuring other connection parameters and with no change in the wrapper. The remote mode gives to the developer a higher control of the element. On this point, if *Local model* selector is ticked the numbers of tabs is reduced to one, but, when *remote model* is selected there are four tabs:

- **Server** is the main tab of the configuration window and contains two types of elements. First, editable elements: *Server Address*, *Port*, *SelectRemoteModel*/*SelectLocalModel* and *Get metadata* are editable. Second, information from the server: *Lab Id*, *Lab Description* and *Listed API Operations*.
  - *Server Address* is the remote or local direction of the laboratory, where our model is waiting for user connections, by default localhost (allows to do development tests).
  - *Port*: The remote port number where the server is listening, by default 2055.
  - *Lab Id*: A string that identifies the lab with a name but not a unique identification. If there is no ID inside the editor, it will take *EjsS model*

as name. If it is a non-EjsS model the ID will depend on the developer of the model.

- *Lab Description*: A string that provides a brief textual description of the lab. If description is not edited by the developer, it is replaced by the default value: *EjsS interaction Metadata*.
- *SelectRemoteModel/SelectLocalModel*: This selector allows to specify whether the model and the view are going to be defined in the same editor (local option) or not.
- *Get metadata*: This button tries to establish a connection with the model, asks for metadata, sensors and actuators and finally disconnects. All the information gathered is updated in the configuration window.
- *Listed API Operation*: In the example of Figure 5.30 all the operations described in metadata are listed. For an EjsS model these methods fixed (by default) are: *callAction, getActionsMetadata, getSensorMetadata, getSensorData, getActuatorMetadata* and *sendActuatorData*. For each one there are also 3 cells to be fulfilled by the developer:
  1. *Parameter* needed to call the method, which is the same as metadata model which defines the structure of the JSON message.
  2. *Returns* cells are the metadata models returned from the model when the operation is completed.
  3. *Purpose* provides additional information about the functionalities of the operation. If it is a non-EjsS model it will be written by the developers, if not, is a fixed description.

- **Reported Variables/Linked Variables**: The laboratory reports a set of variables that can be read or written. The element, by default, links the variables from that set with their counterparts defined in the view. If these variables are not created, the element will create a new list with them. The *reported variables* tabs tell us the names of the sensors, actuator and variables inside the lab. Figure 5.31 contains an example of tab with the info from the server.
  - *Server Writable Variables*: These field is a table of actuators (inputs) and variables (public).
  - *Server Readable Variables*: These field is a table of sensors (outputs) and variables (public).

- **Wrapper**: This tab provides an editor to adapt the default wrapper with user-specific code. It also allows to make file operations as *Save Wrapper/Load*

*Wrapper*: Allows to re-load the original file, templates or to save the changes made in the code of the wrapper.

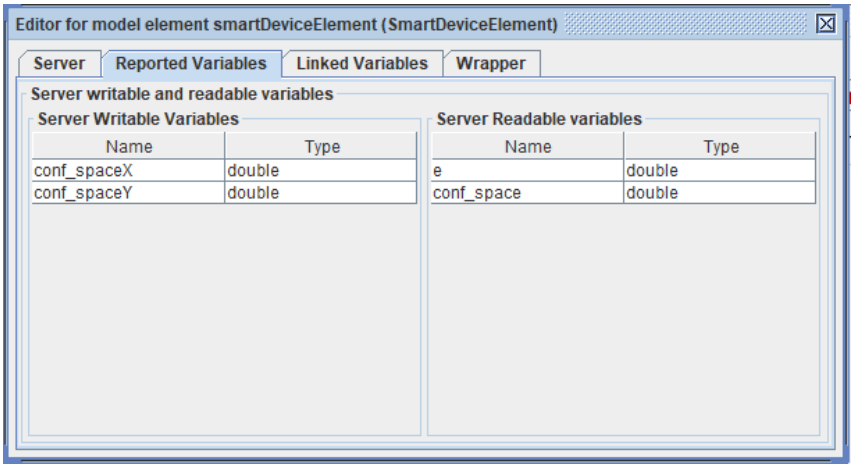



**Figure. 5.31** Configuration window of the SDS element when added to an example

### Wrapper edition

If the user has not changed the wrapper file, the corresponding tab will show the actual EjsS default code with all the needed methods. These methods are valid to connect with most labs, but in some cases (e.g. non-EjsS models) it might be necessary to modify the default behavior. In this sense, it is possible to load other files:

- Empty skeleton: As said in previous sections, it is possible to edit a wrapper from scratch using an empty template with named methods but without valid coding inside.
- Minimal template: Minimal template contains some code, see Figure 5.18.
- Default/other files: It is possible to reload the original version of the file or load a pre-saved versions, if needed.

### Variable generator

EjsS associates information resources (such as sensors or actuators) with variables of the model. In this situation, the tools uses a process to include all the

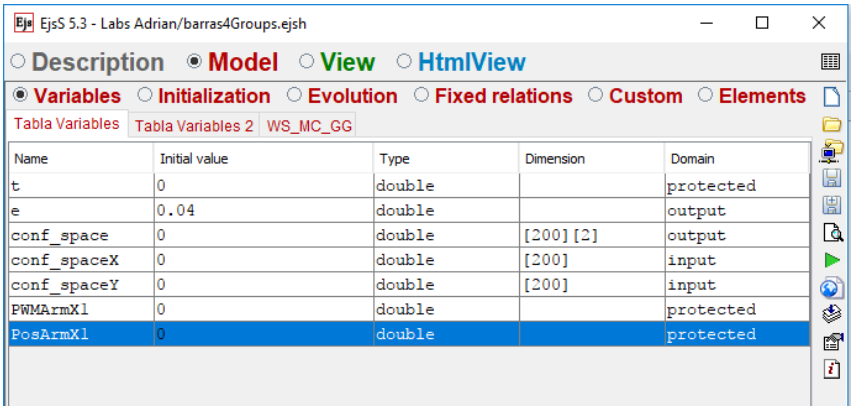



**Figure. 5.32** Autogenerated variables created by SDS element when used inside an example

needed variables to the model. Those variables support any of the possible processes of reading/writing, and also have listeners associated to identify events that result in model changes. The exchange task can also be done in an automated process or the values can be sent manually from any: methods, evolution, properties or visual element.

Figure 5.31 shows the set of sensors/actuators of an arbitrary example, we can see that all the variables that needs to be exchanged are inside two boxes. The variables allow read or write, as sensors (read-only) or actuators (write-only). We can find two readable: *e* and *conf_space* and two writable: *conf_spaceX* and *conf_spaceX*. In the left side of the image we can see the variables tab, each table has a name chosen by the developer but the AUTO_SmartDevice table is the auto-generated one. Figure 5.32 contains the auto-generated variables, which are defined maintaining the type specified by the model.

# 6. Advantages and limitations of implementation

EjsS J&JS enabled version implements the proposed solution by decoupling the model/view pair, adding the communication capabilities and using Java and Javascript as programming languages. Developing VRLs using this solution, maintains the same functionalities of other EjsS versions and adds new capabilities. Considering the characteristics of the communication and architecture, the VRL development is enhanced, although some limitations also appear as result of the change in the main architecture.

## 6.1. Advantages

Laboratory sessions in hands-on laboratories are commonly done by groups of students. Group work improves the skills of students and allows the teacher to use available systems by many users at the same time. From an educational point of view, the teacher can explain and carry-out the experience with the students. Remote experimentation can take advantage of grouping students by using multiuser and/or multiview laboratories in EjsS J&JS.

### 6.1.1. Multiuser sessions

The nature of remote experimentation introduces some technical issues when tackling with multiuser sessions, that were difficult to solve in previous versions of EjsS. Multiuser session are lab practices where there are more than one user connected to the same system, each one with one of two roles: controller or viewer. Configuration of EjsS tool ensures that a single user can view and control the lab. If there is more than one client, anyone can control and view the system, as the number of views is not restricted. To prevent concurrent changes in the model, two roles are specified: controller and viewer. The first one, is in control of all

the writable changes and can read any of the variables. The first one, the viewer, can only read variables and see how the changes made by the controller affect the application.

Previous versions of EjsS have also the possibility to make collaborative session by using peer-to-peer connections, which can result in a complex network architecture.

### 6.1.2. Multiview sessions

Mutiview or Collaborative sessions can be done using the restrictions regarding the viewer/controller scheme, but it is also possible to have different views for the same model. The purpose of having different GUIs can be to adapt the interface to different levels of knowledge, complexity, permissions or learning objectives. The J&JS version allows simultaneous users with different GUI connected to the same model.

Multiview along with collaborative sessions are useful advantages that can improve the leaning process. Together, they can provide complex lab experiences where the student has a limited framework to interact and the teacher has a full GUI to control all the parameters of the application.

The current version of the tool supports making this type of sessions, but the process is not automated and the teacher capabilities must be programmed on each application.

### 6.1.3. Reutilization

One of the goals of the research is to reuse previous models without coding all applications from scratch. The reutilization feature is possible by loading the *old* file inside the tool, creating the HTML GUI and changing the parts of the code where external or Java libraries were used. The HTML view can be similar to the *old* application, but this is not the best approach, because all the variable links and calls would be applied, sometimes consuming too much network resources.

Chapters 8 and 9 contains two examples of applications that illustrates the procedure described in *How to update a Java lab to Java & Javascript*. This document is included in the Appendix D. Next section describes restrictions related to the development and use of the tool.

## 6.2. Limitations

It is also important to consider the limitations of the EjsS J&JS enabled version. Most constraints are related to the connectivity of the labs: data flow through Internet, access to resources, the frequency of messages and delays in communications, etc. The next subsections discuss the main limitations of the tool.

### 6.2.1. Methods or dynamic calculus

Methods and variables are recovered at the moment of packaging the application, custom methods are extracted from the *custom* tab and the set of methods proper to the model and view are automatically added. In this sense, any other functionality or method defined in the model but not defined in *custom* tab cannot be dynamically found, and therefore, neither can be used. As all the coding is packaged with the model, including those inside the properties of the view, using methods to calculate properties can slow down the application. For example, the developers may choose to define the *Position X* property inside a *2D Shape* as

$$3 * (otherPosition + 10)$$

It looks like a simple calculus, but it implies a delay, and, if the view is going to be updated many times, can affect the performance of the GUI.

### 6.2.2. Access to resources

Accessing to server or local files and sharing these files with the view or the model can not be done by default and implies coding the process to send binaries. Take as an example, an application that needs to send sound files, captured by a microphone in a pneumatic cylinder lab. In this situation, the EjsS the model can capture the sound, but by default, the model cannot included it inside a SDS JSON message. To send this file, the user need to add the code to decode and serialize the sound. It is possible to send binary files by using the capabilities inside the Smart Device Specification, but is not defined inside EjsS.

### 6.2.3. View elements association

Method calls between Javascript GUIs and Java model are processed as described is last sections. The optional Java view, used to control the system from the server, adds some restrictions to the development. The limitation appears when a method call from a view element needs to be reported to the analogue one in other

view. When an action arrives to the model, it uses reflection[1] to call the corresponding model/custom/view method. Reflection allows the model to inspect the names of all the members of a Java class, then, is possible to find the view elements defined in the Java view. An action to modify a view element and called from the model implies and update in Java and Javascript view. Then, reflection must be applied in both views. A call like

$$\_view.trail.addPoint(0.0, 0.0)$$

produces an update in the element called *trail*, but, both must have the same name in the Java and Javascript views.

As the model is the central node, therefore, the Java view and the Javascript one do not know each other. In this situation, there is no real association between Java and Javascript views, and the *trail* element must be present in both and using the same name.

This effect only appears in the situation described in last paragraph and is not common. In remote experimentation, as described in this document, the model tasks usually rest on servers. Commonly, neither the developers nor the teachers are close to these servers when a lab session begins. Therefore, the Java view would probably not be used during experimentation time. If the developer or teacher need to use the Java and the Javascript views simultaneously they can benefit from the multiuser capabilities of the tool. However, if the use of the Java view is a essential requirement, the multiview capabilities will became unusable.

### 6.2.4. Communication speed and delays

In remote experimentation, the performance of the application is a key element, as VRLs are teaching tools that should serve to motivate students in their learning, and not frustrate them due to performance problems. Besides, the way a lab is perceived is also important for teaching, for remote labs must behave in a similar way to classroom labs. The user experience when using the lab is affected by many factors. Analyze how in the communication speed and delays change with the lab configuration in not immediate, therefore some test must be done before the deployment of the lab. These experimental tests focus on measuring the time required by the lab to complete cycles of communication and message parsing between a web browser and the remote lab. On the one hand, the remote lab is usually located in a university's lab and it is available for students through an online course managed by an LMS server. On the other hand, student is located in a home network connected

[1] https://www.oracle.com/technetwork/articles/java/javareflection-1536171.html

to the Internet. In this situation, it is not the same to control a servo-motor as to control a heatflow system, because the amount of data and the timing constraints to be exchanged is different.

Considering a situation in which the bandwidth of data to be exchanged is constant, different combinations of the lab configuration parameters (i.e. message rate, message length, and number of variables) produce a change in the response time of the global system. In this regard, using many variables to be exchanged or just a few to prevent the slow down can affect the user experience or the quality of the data. Next chapter describes some additional features to prevent this kind of problems and useful to study the behavior of the systems.

# 7. New Elements and additional features

EjsS J&JS enabled version has been developed along with other additional features and tools related with the creation of virtual and remote laboratories. The capabilities have been included inside the enabled version of last chapter and are made to enhance the user experience and facilitate the configuration.

## 7.1. Performance and parameterization

As the number of variables is not completely relevant for most labs because their processing is a low time-consuming task at the model side, it could be problematic when using EjsS. The message rate and message length are the key configuration parameters that determines the response time and, therefore, the quality of service of VRLs. Due to the different characteristics of remote labs and final users devices and networks, the best configuration parameters cannot be established a priori and must be managed by an external architecture that monitors the network and lab status in real time.

### 7.1.1. Minimum information exchange

While the main purpose of the tool is making easier the development of VRLs in education and research, delay-related problems and network status can affect the normal behavior of a system. HTTP-based labs have shown some disadvantages when used alone to develop VRLs, until the emergence of some architectural styles, as REST is.

Those issues are related with the delays when trying to build hard or soft real-time systems. In this regard, though Websocket performs better in the field of real-time experimentation, it is not perfect, and new labs must be tested before their deployment.

Most of networked lab applications need a steady flow of information using a connection free of errors and reliable data delivery. Therefore, as the bandwidth is limited, developers have to design their applications to only exchange the variables needed. While EjsS contains processes to control the number of view updates per second and the number of model steps between updates, the remote experimentation area commonly faces a problem regarding to coupled effects:

- Message **length**: The total number of bytes inside a single message. As the total size of one Websocket message is virtually unlimited [1], the total length affects in the parsing time and processing time at both sides: server and client.

- Message **rate**: The frequency established to send messages to the client is at least the number of view updates defined in the EjsS editor. Each sent message needs some time to *navigate* to the receiver and some more time to verify the arrival. Moreover, if the rate of messages is high, it is possible to find buffering delays at the receiver side.

As said before, both elements imply significant changes in the application performance, although tests done by other authors, reveal that Websockets show a significant performance drop when sending small messages and could be countered when the message size increases [30]. To prevent those situations, the number of view updates per second and the number of model steps are key parameters in order to obtain a good performance. To prevent delays, EjsS collects events close in time to send them in the same message. Therefore, the synchronous messages can be adapted to fit the time requirements by changing usual EjsS parameters.

Finally, asynchronous messages can also imply a drop in performance if those are not following the same restriction. In this regard, developers must design their application trying to collect also the requests for readable elements and/or the messages to change writable elements.

### 7.1.2. Reconfigurable elements

The proposal presented in this work benefits from the Smart Device Specification by including it in laboratories, enabling interaction and configuration by sending simple messages. While adding the SDS in laboratories implies the development of new services (as the metadata services) the software tool EjsS (used to develop the labs) contains new functionalities to automatically include the SDS in the lab.

---

[1] https://tools.ietf.org/html/rfc6455#section-5.2

Therefore, the presented architecture used in VRLs enhances the flexibility in the management of lab resources. Using the servo-motor (see Chapter 9) as example, it contains the following set of configurable elements:

- **Sensors**: The remote lab is equipped with two sensors. One of them measures the angular position of the motor. The other one, a tachometer, measures the angular speed. External entities can send messages to change:
  - Position or speed readings: The model can be configured to send measures from only one sensor (either position or speed) or from both (position and speed). To be used in different lab experiences, or to be activated in a running experiment.
  - Actual rate for reading from the sensor (as long as it is within the limits of the sensor).
- **Step time or update rate**: It is possible to modify the step time, which is the number of number of cycles per second inside the model. This affects the frequency of synchronous messages between the view and the model. This setting, however, does not have any effect on asynchronous messages, which depends on user interactions, such as dragging elements.
- **Actuators**: The actuators, as the sensors, allow a degree of adaptability to be configured in an efficient way. The servo-motor lab is equipped with one single actuator, the motor. In spite of the electrical signal of the motor, there are some other configurable elements:
  - The *Access roles* (controller and viewer): Controllers can change the value of the input signal in the motor. Viewers can only obtain information regarding the actual state of the actuator but are not allowed to modify it.
  - *Nominal update interval*: Is the minimum time between subsequent changes in the actuator, in the servo-motor laboratory the available interval is narrow because of the fast dynamics of the system: $[0.1, 1.0] sec$.

Each configurable element can be modified by sending configuration JSON messages.

However, the configuration must be available inside sensor/actuator descriptions or information about them. The architecture allows to measure the performance in real time, adapting the configuration dynamically to enhance the experience of the user with the remote lab.

### 7.1.3. Quality of Service

The behavior of some elements can be changed using the parameters presented to configure sensors and actuators. The purposes of sensor/actuator re-configuration are up to the developers, but usually are one of two. First one, how the software controls the system.

It relies on acquiring and actuating rates/delays to prevent effects like aliasing. As this parameters can be discovered while the system is being developed usually can be configured in a previous step of the GUI or model design. Second one, how is the user experience, regarding the possibility to understand the behavior of the lab and also to enhance the interactivity of the lab.

The implementation of the solution does not include the measurement of the response time in real time. However, it can be done manually and it allows to adapt dynamically the remote lab configuration. As every lab is different from others and the configuration elements are distinct, the process is not included inside the main code of the tool. Therefore, to implement this feature, the user needs to modify both, the GUI and the model, including a task to measure these times.

## 7.2. J&JS as elements

The exchange of data when decoupling model and view is possible using the J&JS enabled version. Although, limitations related to the methods defined by the user at the client side can restrict the usability of the tool, if the lab owner needs to define those. To complement the version, two new model elements have been developed:

1. An element to add capabilities to Java labs.

2. An element to add capabilities to Javascript labs.

Both elements can not add the same capabilities that are included by default inside the J&JS version, therefore, non-programmer users can find difficulties to use these elements inside their applications.

### 7.2.1. Java SDS and Javascript SDS Elements

The Java element is designed to be used with the Java enabled version of EjsS. This element creates a service which is listening to the messages from other SDS enabled devices which want to connect.

The capabilities added by this element are less than in the J&JS enabled version, thus, most of the capabilities must be implemented by hand.

The Javascript element is designed to be used with the Javascript enabled version of EjsS. The element is fully functional and adds almost all the client capabilities described in chapter 4.1. The element has the same icon and configuration window but it can not include some enhancements, like automatically building the first get message to configure the auto update from the model.

The Java and Javascript elements are meant to be used by developers with some programming skills. To make both available to non-programmer users some capabilities must be added. At the moment of writing this document, to add these capabilities is a future work but, both elements can establish a stable communication to exchange data and interact with a other pre-existent labs.

# Part III

# DEVELOPED LABS

# 8. Virtual Labs

This and next chapters present the labs which have been developed. These laboratories are virtual or remote, however, in some cases both types are available. This chapter presents the VLs divided by the version of EjsS used: Javascript or J&JS. The VLs are described using three different focal points:

- **Physical description of the experimental device**: The VLs are usually based in real equipment and systems, this section gives a description and some mathematical background. To lighten the description of the labs, some parts of the mathematical background of the system are included as appendices.

- **The virtual lab GUI**: This section describe the user interface and additional details about the development, like the equations used to simulate the behavior.

- **Learning objectives**: The purposes of the lab can be many, but this section gives a glimpse about some activities that can be achieved.

Developers and teachers around the world are continuously involved in the creation of new resources to enhance teaching. On this subject, UNED is up-to-date, having many online courses and web-accessible resources. During the research and development of the tool described in previous chapters some new virtual labs have been updated or created to be used inside the UNILabs website.

EjsS, in both versions JS and J&JS, has been used to create user interfaces and models to controls the laboratories. In each VRL, EjsS plays similar roles in order to achieve a good experience for the user. As said in Chapter 1, to replicate a real lab, the GUI needs to be: graphical, dynamic, interactive and representative of the system to be studied. The following virtual labs represent real systems that allow users to carry out lab experiences inside the control engineering and physics area.

## 8.1. Virtual labs developed in Javascript

EjsS Javascript version allows deploying VRLs inside a Webpage. Thus, it is used as a first option to develop labs from scratch, as it eases the creation and deployment process. *The two coupled electric drives* and *The vibrating wires* experiments have been developed using this enabled version.

### 8.1.1. Two coupled electric drives

Coupled electric drives apparatus is designed to allow students at all academic levels to investigate basic and advanced principles of control, including control of multi-variable systems. The real system is a compact bench designed by TecQuipment[1] to show the engineering problems regarding the control of multi-variable systems.

#### Physical description of the experimental device

Industry contains many production examples where coupled drives systems are needed. The applications of the coupled drives go from textile area to the production of plastic filaments. The system presented contains two electric Direct Current (DC) motors and a pulley connected by a flexible belt. The pulley is mounted in a swinging arm, which is called *jockey*, and measures the speed and the tension of the belt.

The aim of the lab is to help student to understand control topics regarding the control of more than one output using more than one input. On this point, the problem is to control the belt speed and the tension of the belt by regulating the voltage in both motors. As the system has two motor voltages and two outputs (speed and tension of the belt), it is a 2x2 Multiple Input Multiple Output (MIMO) system.

The equipment can be used in-place, without any extra hardware, to learn control techniques and to study in depth an example of multivariable system. But in order to prepare the students and to widen the access to this system, remote and virtual labs have been created.

Using the real system as a reference, a simulation application has been developed, in order to be representative of the real system and to be highly configurable to the student.

Appendix A contains the steps to obtain a mathematical model of the coupled electric drives. The model obtained, for this MIMO system, can be translated into

[1] https://www.tecquipment.com/es/

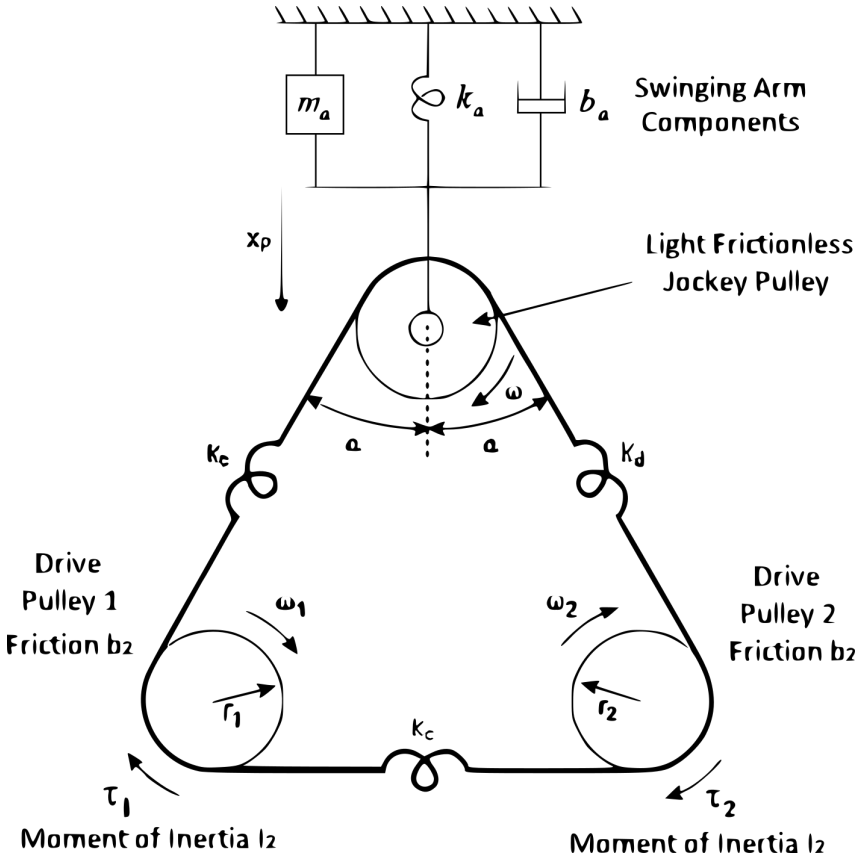



**Figure. 8.1** Equivalent dynamic components of coupled electric drives [113]

the form of transfer functions inside 8.1 and 8.2.

$$\begin{pmatrix} \dot{\theta}_p \\ x_p \end{pmatrix} = \begin{pmatrix} \omega_p \\ x_p \end{pmatrix} = \begin{pmatrix} G_\omega & G_\omega \\ -G_x & G_x \end{pmatrix} \begin{pmatrix} \tau_1 \\ \tau_2 \end{pmatrix}$$

$$\mathrm{G}_\omega(s) = \frac{0.5}{Is+b} \tag{8.1}$$

$$\mathrm{G}_x(s) = \frac{4600}{(s^2+1.6s+790)(s^2+111s+1)} \tag{8.2}$$

Where $x$ is the jockey pulley vertical position, $[I_1, I_2, I_p]$ are the motors and pulley inertia, $m$ is the jockey pulley mass, $[b_1, b_2]$ are the motors friction, $[b_{pt}, b_{pa}]$ are the translational and angular pulley friction, $[\theta_1, \theta_2, \theta_p,]$ are the angular positions,

[$k$,$k_0$] are the belt and spring stiffness and $r$ is the radius of the pulley and the motors, which are assumed to be equal. The transfer functions define both outputs depending of the inputs. Last equations correspond to:

- The velocity of the belt (equation 8.1): Contains a first order pole, which depend on the parameters used in the system (see Table 8.1).
- The tension of the belt (equation 8.2): Two complex and two real poles are identified in the transfer function.

| m | r | I | b | k | $k_0$ | $b_{pt}$ | $\alpha$ |
|---|---|---|---|---|---|---|---|
| 0.35 | 0.03 | $8 \cdot 10^{-4}$ | $9 \cdot 10^{-2}$ | 50 | 200 | 0.5 | $\pi/6$ |
| kg | m | $kgm^2$ | $Nm/s$ | N/m | N/m | N/s | |

**Table 8.1** Coupled electric drives parameters

The values regarding the real equipment are used in the model and are included in Table 8.1.

### The virtual lab GUI

Figure 8.1 contains a graphical representation of the coupled drives system. The scheme is simplified using some assumptions, that can be established: the jockey as

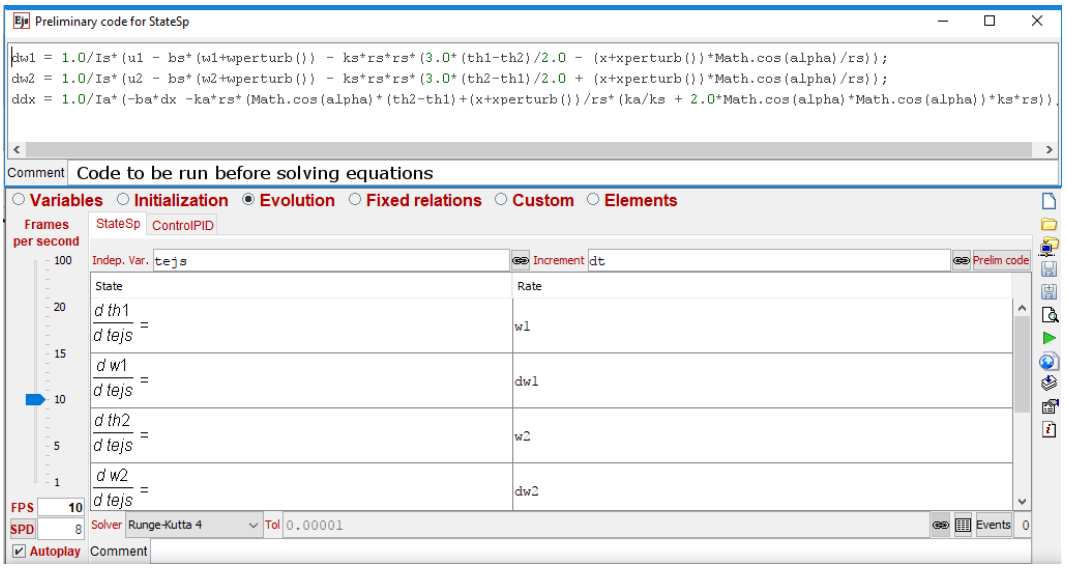
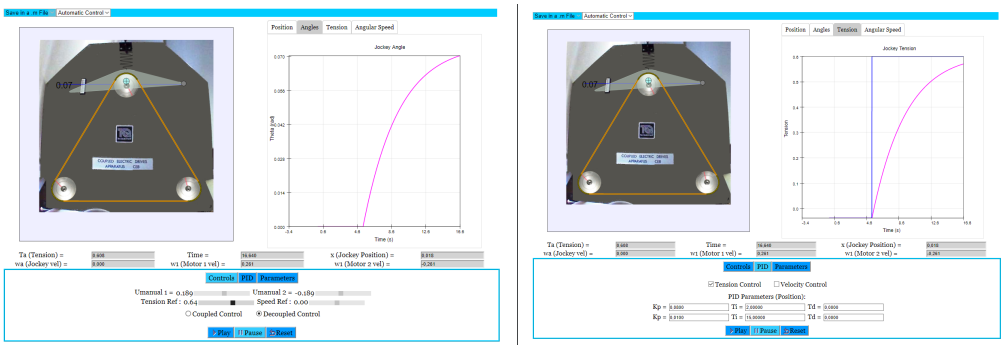



**Figure. 8.2** Javascript editor showing equations and code to be run before running the equation solving algorithms

a swinging arm with three elements, the elastic band as spring-like couplings and frictionless pulley.

Using the EjsS editor, simulation can built over the equations of the model. The simulation can be defined using the differential equations (Appendix A), leaving the precoded algorithms to numerically solve them. (see Figure 8.2)

The virtual laboratory of the coupled electric drives contains different parts inside the GUI window. The GUI distribution is common to many labs made in the *Informática y Automática* department. The simulations window is usually divided into three sections: a menu, the system 2D graphical representation and control, and the evolution graphs and indicators. Figures 8.3, 8.4 contains a visual example of all the functionalities working during a common lab experience.



(a) Controls tab and plot of the angular velocity for a manual control.

(b) PID tab for a coupled automatic tension control and the output.

**Figure. 8.3** Control and PID views of the virtual laboratory application of the coupled electric drives.

- Menu: In the top left corner of the VL's user interface there are four buttons:
  - *Save Files*: Used to save four different graphs: the belt's position, velocity, and tension, and the motors' angles. From this menu, users can also save the numeric data in a .m file for later analysis with Matlab.
  - *Control*: This menu button allows modifying the mode in which the experiment runs: in a manual way (open loop) or with an automatic control (closed loop with a controller).
- System representation and control: In last section Figure 8.1 shows a 2D representation of the system (two motors at the bottom, a flexible belt, and a

jockey pulley at the top). The visual representation is included in the laboratory GUI as shown in Figure 8.3. Under this visualization zone the application shows three tabs:

- In the *Controls* tab (Figure 8.3.a), students can change the velocity of the motors using the $U_1$ and $U_2$ sliders or introducing the values directly in the editable numeric fields at the right. In automatic mode, the $U_1$ and $U_2$ sliders are disabled and the user can change the tension and velocity references (the two inputs of this system), and select whether to make a coupled or a decoupled control for these variables.
- The *PID* tab (Figures 8.3.b and 8.3.c) shows three or six fields to adjust the PID parameters of tension and/or velocity controllers, depending on the kind of control that is selected: coupled or decoupled. In coupled mode (Figure 8.3.b), students have to choose the variable they want to control, either the tension or the velocity of the belt. In decoupled mode (Figure 8.3.c), both PID controllers are activated and the user can change their six parameters.
- The *Parameters* tab (Figure 8.3.c) provides the possibility of studying the system's time response by introducing a sinusoidal input signal. In

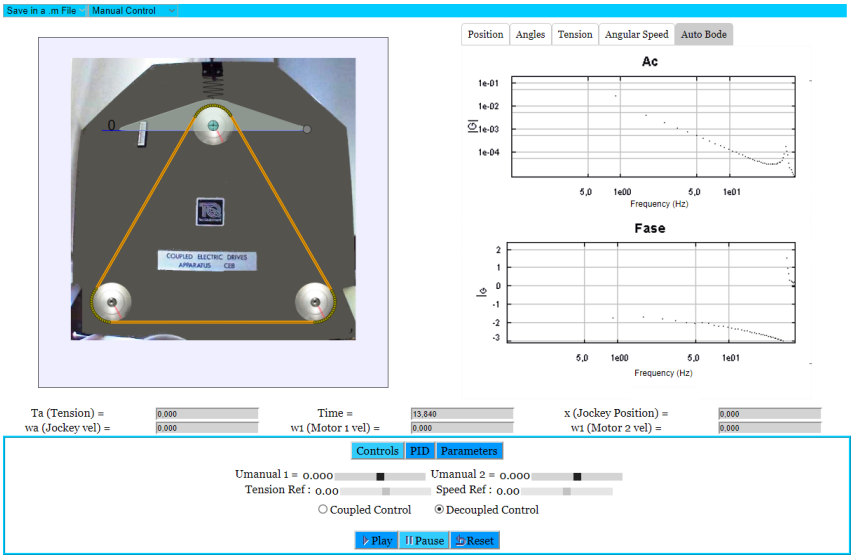



**Figure. 8.4** AC Signals tab in *AutoBode* mode. The one on the left shows the configuration of a frequency sweep of fifty points. The one on the right offers the graphs of the magnitude and phase values gathered in the sweep.

addition, selecting the *"Auto-Bode"* allows to configure an automatic *"ac sweep"*, obtaining a graph like the one in Figure 8.4.

Finally, the VL has three more buttons (in its bottom-left part) for controlling the execution of the simulation: *play*, *pause* and *reset*.

- Evolution graphs and indicators: The right part of the virtual lab application offers a graphical representation of the most important variables in different tabs: position, motor/pulley angles, velocity, and tension of the belt and estimations of gain and phase.

The Javascript virtual lab of the coupled electric drives is nowadays included inside UNILabs, inside the open course[2] and inside the *Laboratory practices of the master in systems engineering and control* course [3].

### Learning objectives

The simulated lab gives to the student a schematic representation of the coupled electric drives. The lab is initiated to be controlled manually. In this situation the simulated motors are not powered, waiting for user interactions. The laboratory practice can be done by following the steps defined by the teacher or by the student. The following paragraphs explain the main activities to be performed with this web-lab.

- **System Identification**: One of the VL's main purposes is to identify by gathering useful information about the order of the system and the values of zeros and poles. This activity may be performed acquiring data from all these steps in the simulation:

  1. The open loop analysis returns information of the dominant pole, as happens with first order systems, and the over-damped step response delimits the value of damping factor, as happens with second order systems.
  2. A most accurate damping factor and the natural frequency can be obtained by introducing a disturbance in the system. This allows determining the value of the complex pole pair.

---

[2] https://unilabs.dia.uned.es/course/view.php?id=24&lang=en
[3] https://unilabs.dia.uned.es/course/view.php?id=4&lang=en

3. The frequency response analysis provides detailed information of the transfer function. This can be used for checking results of previous activities, for explaining the differences with a second order system, or for fitting with a better model (identifying the other two real poles).

- **PID Tuning**: When the automatic control mode is selected in the upper menu (as described in the previous Section), a closed loop is established for controlling the velocity or tension of the belt by means of a PID controller. In coupled mode, students can select the control variable (either velocity or tension) in this MIMO system, and may change the proportional, integral and derivative parameters of the PID controller ($k_P$, $T_I$, $T_D$) in order to obtain a response that satisfies the specifications asked in the practice guide. In decoupled mode, students work with two SISO systems and, therefore, with two PID controllers. Then, an analysis of the output signals (stored in .m files), allows to obtain an approximated model of the system.

- **Frequency Response Analysis**: The model can introduce a sinusoidal input inside the simulated system. It can be defined by modifying the values of the frequency ($\omega$) and amplitude ($A$):

$$u_{in} = Asin(\omega t)$$

  By introducing such an input and registering the magnitude and phase estimations from the *AC Signals* graph in the evolution window, students can obtain the magnitude-frequency and phase-frequency graphs (i.e. a Bode plot). An analysis of this plot helps with the system identification problem since it provides information about poles and zeros of the open and closed loop transfer functions of the system. The VL is also provided with an option to automatically obtain the Bode plot and the data array. A student can process the data (stored in .m files) and obtain the plot of Figure 8.4, which shows a magnitude-frequency graph in open loop and in closed loop (for coupled and uncoupled control) in simulation mode.

- **Disturbances Analysis**: The 2D visualization allows introducing an isolated disturbance, by clicking and dragging the pulley, to obtain the natural frequency and damping ratio of the system. This can be used for studying the double complex pole of the system in order to identify the transfer function in the virtual laboratory.

- **User Defined Controller**: As an advanced task, the user can write different controllers to use in the control loop. This tool extends the use of the labora-

tory either for investigation or learning purposes. The code of the controller can be written by using an special Moodle environment, Blockly[4]. Using the Moodle plugin, the user can use visual blocks to code Javascript tasks [114, 115, 116]. By using this option the students can:

- Add features to the basic web-lab built-in PID controller.
- Analyze and compare the behavior of well-known controllers for learning purposes.
- Develop/test a controller for investigation purposes.

### 8.1.2. The vibrating wires

The vibration nature is a key element when studying physical phenomena in systems. In this sense, students of STEM areas must learn about the basis of this concept and methodologies to obtain information from sinusoidal signals. A common lab in Physics courses studies the wave motion on a fixed string.

The vibrational string is included inside one of the subjects of the first course on UNED Physics degree. Commonly is a hands-on laboratory where the student can excite and acquire data directly from the oscilloscope. Virtual and remote labs provide an alternative for students that cannot access to the experimental setup. The full setup is described in [117].

This virtual experiment gives the student an application to learn the dependence between many physical parameters. It is known that the fundamental frequency depends on the parameters of the string:

- Density, as the inverse of the square root.
- Length, as the inverse.
- Tension, as the square root.

This fact allows the student to learn about experimental procedures and data analysis using different representations of data (linear, inverse, logarithmic). Each collection of data implies fitting the data-point to well known curves, in order to acquire experience in laboratory data analysis.

---

[4]https://opensource.google.com/projects/blockly

**Physical description of the experimental device**

The laboratory is designed to control some of parameters that have a known dependence with the natural frequency. The student can learn about the existent dependence between parameters: a linear dependence between the period and the length of the string, an inverse dependence between the frequency and the radius, a square root dependence between the tension and the frecuency, and so on. The key goal of the experimental device is to establish a set-up with a fixed set of parameters which produce different behaviors. The virtual GUI is designed to look like the real set-up, containing also the same capabilities. Therefore, the student can explore the interface and repeat the experimental process as many times as needed in the virtual lab.

The virtual lab simulates the behavior of the vibrating strings, for this purpose the system is described in the form of equations in the additional documentation given to the students. The mathematical basis are also explained in the Appendix B, from where we can extract an equation:

$$z_1(t) = A_1 \cos(2\pi f_1 t) \quad , \tag{8.3}$$

where the fundamental frequency $f_1$ is given by

$$f_1 = \frac{1}{2Lr}\sqrt{\frac{T}{\pi\rho}} \quad , \tag{8.4}$$

where $r$ is the radius of the string, $L$ is the length of the string, $T$ is the tension and $\rho$ its volumetric density.

Equation 8.3 gives the dependence between the parameters that are controllable and the fundamental frequency. The set of parameters which can be modified depends on the experimental device and the available materials.

Figure 8.5 shows the real experimental device which is used in the remote lab. As the idea is to replicate the real set-up, some of the elements are simulated in the GUI. The graphical representation of the lab includes some of the elements but some are overlooked in the virtual version, to maintain a clean view of the simulated process.

There are five different strings (2) made of different materials (copper, kanthal, constantan, and nickel) and with diameters ranging from 0.3 mm to 0.5 mm. One of the ends of the strings (with the exception of the central string) is fixed on the aluminum structure of the device, while the other end of the strings is connected to a dynamometer (10) that measures the tension along the string. In the case of the

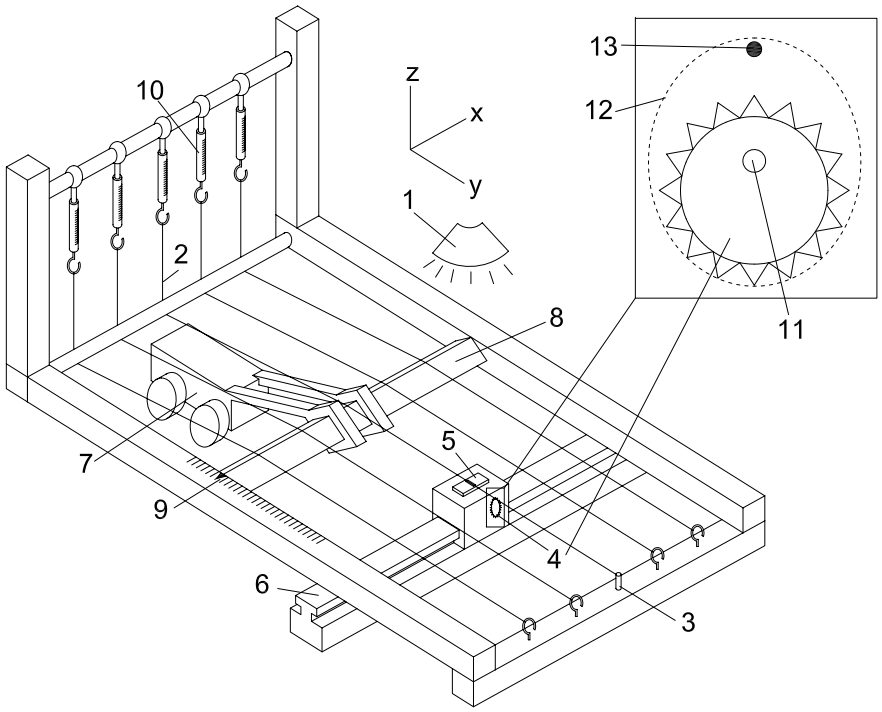



**Figure. 8.5** Fully developed experimental device, consisting of the following elements: (1) DC LED light source, (2) strings, (3) stepper motor, (4) LEGO gear connected to LEGO servomotor, (5) light sensor, (6) linear stage, (7) LEGO carrier, (8) mobile aluminum rod, (9) rule and length indicator, and (10) dynamometers. The close-view figure shows a frontal view of the string plucking element consisting of (11) the rotation axis of the LEGO gear, (12) trajectory described by the LEGO gear perimeter, and (13) string under study.

central string, the fixed end is connected to the axis of a stepper motor (3), so that the tension can be controlled. A LEGO carrier (7) is used to displace an aluminum rod in close contact with the strings (8) along the $y$ axis, in such a way that the length of the vibrating part of the strings can be changed from 380 mm to 550 mm. The signal acquired from the vibration of the wires is obtained mathematically and shown to the user in order to download and/or analyze it. The movement, acquiring and calibration is done by real equipment, which is shown and described in appendix 9.

### The virtual lab GUI

The GUI is made using the Javascript enabled version of EjsS and is done to give the student the control of the experimental environment (tension, length...) as in a hands-on laboratory.

The experiment is simulated by creating a laboratory environment which can emulate the experimental device and the available tool in a classic hand on lab. In this regard, the application window is divided into three sections: a 2D and 3D graphical visual representation of the system, the controls panel and the plots/graph panel. To create the signals obtained from the oscilloscope, which will be plotted in the graph panel, the VL uses equation 8.3. Each string is defined internally as a collection of parameters, which can be used inside the equation. The variables in charge on defining the string can be easily changed, allowing the developer to change its response in a fast way.

- *Visual representation (2D/3D)*: The top-left side of Figure 8.6 shows a 3D model of the system in which the user can observe the basic parts of the real structure of the vibrating string laboratory.
- *Controls panel*: Using the controls, buttons and sliders of this panel shown in top-right side of Figure 8.6, the user is helped to go through the experimental protocol, highlighting each step and giving tool-tips to make it easier.
- *Plots/Graphs panel*: This part of the interface, at the bottom part of Figure 8.6, allows the user to see data in different plots and graphs. The vibrating string laboratory plots the light intensity versus, first, the position of the linear stage (see section 5 for details in this calibration procedure), and second, time (in order to obtain the frequency of the fundamental normal mode).

Figure 8.6 shows the basic structure of the virtual laboratory when the .xhtml is served to the client. The virtual lab also allows students to get familiar with the available interaction and the protocol, in order to be prepared to the remote version of the lab

### Learning objectives

The simulated graphical representation allows the student for the visualization of a general view of the experimental setup as well as closer view of the dynamometers and the length of the strings. The student can initialize the experiment by clicking in connect, to emulate the starting sequence which is done in the remote lab. When the simulation is started, and the experimental experience begins, each subsequent step is colored in cyan, to guide the student through the process:

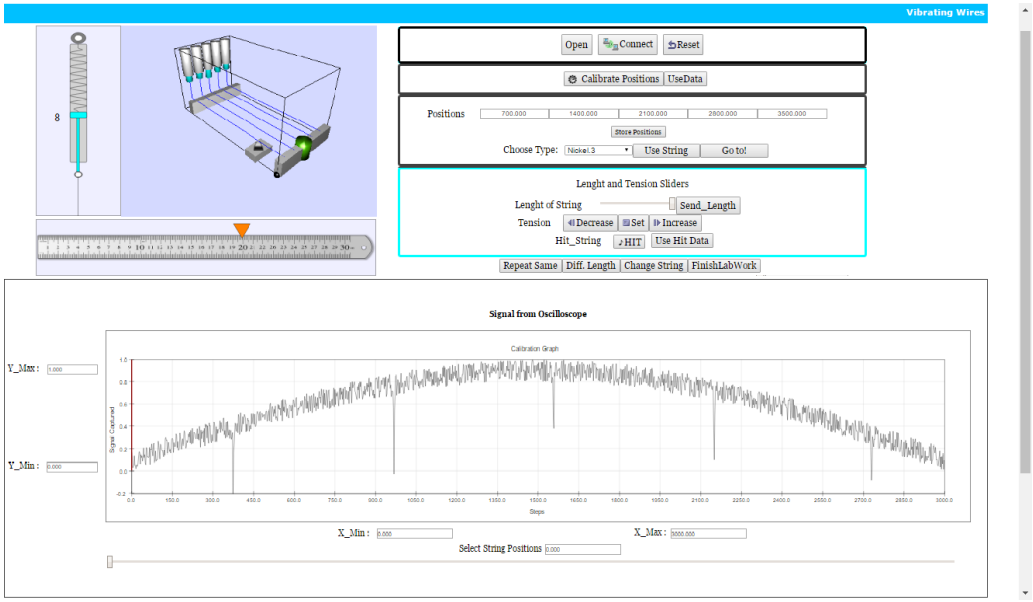



**Figure. 8.6** The virtual vibrating string system laboratory.

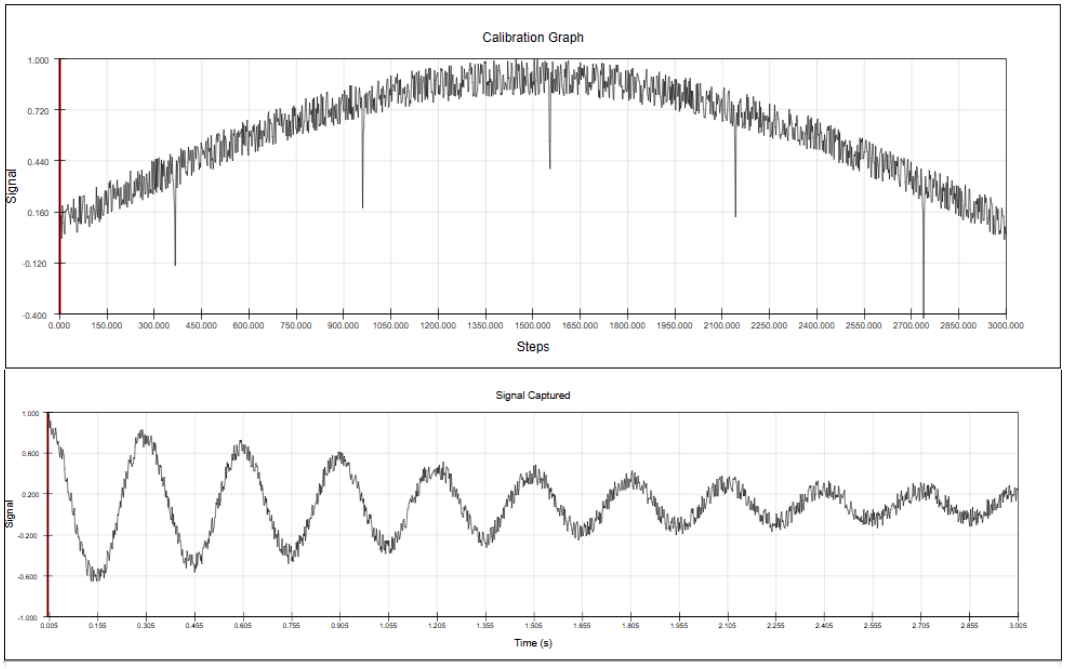



**Figure. 8.7** a) Light intensity versus position of the light sensor along the $x$ axis. b) Light intensity versus time after the gear hits one of the strings.

1. **Location of the strings' positions:** A complete sweep is simulated along the whole range the strings positions. The effect of the light sensor continuously measuring the light intensity to capture the shadows of the strings. Then, the student can plot the function light intensity versus position along the $x$ axis. Where the noise and shadows are obtained numerically. Figure 8.7.a shows that the light intensity is roughly symmetric with a local maximum at the center which represents a simulated light source placed at the center of the device. Thus, the student can determine the position of each string from the five local minima in the light intensity.

2. **Positions must be written down**: The GUI gives five field to write the positions obtained from the calibration procedure. The step of obtaining the positions must be accurate in the remote version, therefore, the simulation reward the accuracy by giving best results in next steps.

3. **Selection of the string and the length to explore:** After this calibration, the sensor can be displaced over to the positions of the strings by choosing the string. Then, the linear stage shown in the 3D representation moves to the string position.

4. **Selection of the tension (if needed):** If the string selected by the student is the central one, the tension can be varied. In that purpose, by clicking an *increase tension* (or *decrease*) control, the tension is increased (or decreased) in steps of approximately 0.05 N. The student observes both the measurement of the dynamometer and a color representation of the tension in the string. The user is able to change the tension as long as it is maintained below 10 N, presenting to the user the real limit of the remote string.

5. **Execution of the experiment and data acquisition:** When the previous steps have been completed, the simulation is ready to execute the experiment in those conditions selected by the student (considering the parameter of the string selected). Then, by clicking the corresponding control, the VL runs the plucking the string by generating the wave for the user. Figure 8.7.b shows the results of this simulated wave from one experiment with a correctly measured position. If the position is not accurate, the signal becomes noisy, even being impossible to identify the wave. The student has to analyze these data, calculating the frequency of the fundamental normal mode, $f_1$.

6. **Analysis of the results and comparison with theory:** Once the student has completed experiments under different physical conditions, the dependence relation between $f_1$ and the physical parameters of the string ($T$, $L$, $\rho$) can

be established. Then, the student should be able to discuss the experimental errors and the validity of the theoretical model.

## 8.2. Virtual labs developed in Java & Javascript

### 8.2.1. The servo motor lab

The servo motor lab is a well known example in control engineering areas. This system together with the use of EjsS has a long history within the *Informática y automática* department [118, 119, 120, 60, 121]. The system can be used to achieve multiple goals, and for that reason the servo-motor virtual and remote labs are included in three experimental courses of the UNILabs website:

- An open course with free access Labs on control[5].
- Labs of a course on control, inside the degree in informatics engineering. [6].
- A course of laboratory practices of the Master degree in Systems Engineering and Control[7].

#### Physical description of the experimental device

The servo motor system consist in a electrical driven motor which contains a sensor device to measure the angle of the external wheel and a tachometer to measure the speed. This was a system designed by *La École polytechnique fédérale de Lausanne*[8], and the main purpose is to control its position or speed. The system additionally includes optional parts, to enhance different effects:

- The main axis of the motor also includes a configurable load. This loads consist in a set of small disks which can be attached to the axis. Therefore, the load can be changed between different courses of different classrooms to change the dynamics of the servo-motor.
- Bottom part of the disc also includes a magnetic brake. The damping effect over the servo disc can be also modified, moving up and down the magnetic source. The magnetic brake represent a viscous friction, which can be also modeled mathematically.

---

[5] https://unilabs.dia.uned.es/course/view.php?id=24
[6] https://unilabs.dia.uned.es/course/view.php?id=4
[7] https://unilabs.dia.uned.es/course/view.php?id=7
[8] www.epfl.ch/index.en.html

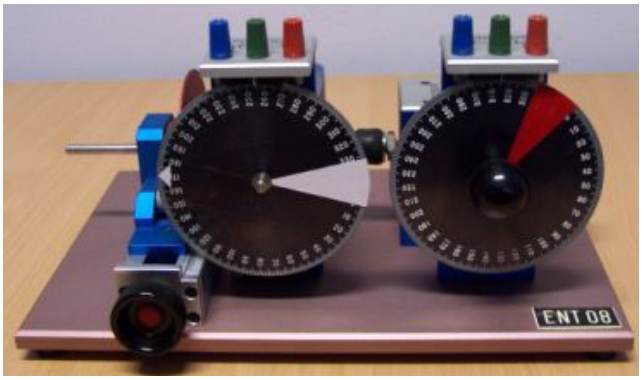



**Figure. 8.8** Real servo motor system with it protractor disc.

The input of the servo-motor is a variable voltage, which is applied in the DC motor. The outputs are also voltages, from the position and tachometer. These voltages can be easily translated to radians per second and angular position. Figure 8.8 shows the real servo motor lab.

The mathematical model used to simulate the system is the same to be used as reference for the students. To make it easier, there are some assumptions that can be made. Both, the assumptions made and the mathematical model that relates the input (voltage) and the output (position) is shown in Appendix C. The model shows that the transfer function that will be analyzed by the students is shown in equation 8.5.

$$\frac{\theta(s)}{E_a(s)} = \frac{K_1}{s((L_aB_m + R_aJ_m)s + R_aB_m + K_1K_b)} \tag{8.5}$$

**The virtual lab GUI**

The virtual lab simulates the behavior of the servo-motor lab, for this purpose the system is described in the form of differential equations.

$$e_a(t) - R_a i_a(t) = L_a \frac{di_a(t)}{dt} + K_b \frac{d\theta(t)}{t} \tag{8.6}$$

$$J_m \frac{d^2\theta(t)}{dt^2} + B_m \frac{d\theta(t)}{dt} = K_1 i_a(t) - T_L \tag{8.7}$$

The differential equations are shown in Eqs 8.6 and 8.7. Where $e_a$ is the armature voltage, $e_f$ is the field voltage and $i_a$, $i_f$ their corresponding currents. $R_a$ and $L_a$

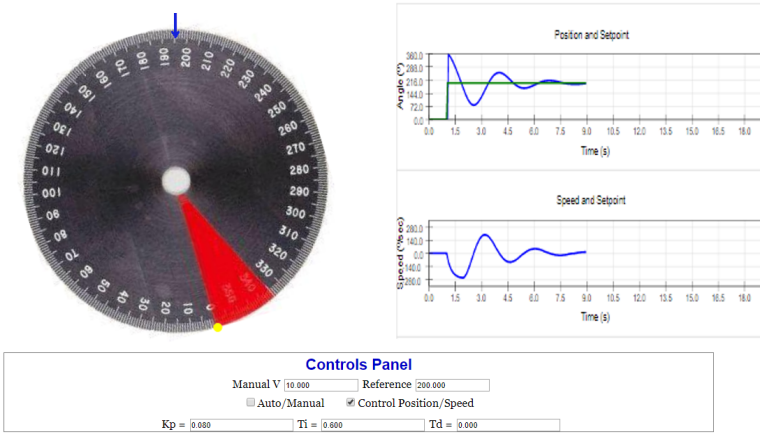



**Figure. 8.9** GUI of the servo-motor lab

are the resistance and inductance of the armature. $K_b$ is the counter-electromotive force constant and $K_1$ is the torque constant. $J_m$ and $B_m$ are the moment of inertia and the viscous friction coefficient. $T_L$ is the torque associated with the load of the motor.

The available interactive elements gives the user means to control different experimental modes:

- Manual operation:The system can be controlled manually to change the input voltage to the motor. The signal in the simulation has the same limits as the real servo-motor.

- Automatic operation: This represents the controlled variable. $K_P$, $T_I$, and $T_D$. Proportional gain, integral time, and derivative time of the built-in PID controller. The output to control can be changed between the **speed** and the **position**.

### Learning objectives

The simulated graphical representation allows the student in the visualization of a general view of the experimental setup. The angular position of the motor is represented as a protractor disc, Figure 8.9. When the user loads the webpage, the laboratory starts. Using the VL the student can perform common tasks in control engineering courses. Each one of the activities can be performed using one of the two available outputs.

- **System Identification**: One of the VL's main purposes is to identify the system or to gather useful information about the order of the system and the

values of zeros and poles. The open and closed loop analysis can provide to the student enough data to build a theoretical model. The quality of the model will affect consequent steps, as the student may use this model to design controllers or to be prepared for the remote version.

- **PID Tuning**: When the automatic control mode is selected in the upper menu (as described in the previous Section), a closed loop is established for controlling the velocity or position of the servo-motor by means of a PID controller. Students can select the control variable (either velocity or position) and may change the proportional, integral and derivative parameters of the PID controller ($k_P$, $T_I$, $T_D$). in order to obtain a response that satisfies the specifications asked in the practice guide. Then, an analysis of the output signals, allows to obtain an approximated model of the system.

- **Disturbances Analysis**: The 2D visualization allows introducing an isolated perturbation, by clicking and dragging the top yellow point in the servo. These single perturbations can be used to study and obtain information about the viscosity term, which have been shown in the physical description.

- **User Defined Controller**: As in the coupled drives example, advanced users can write different controllers to use in the control loop. The code of the controller can be written by using an special Moodle environment, Blockly[9]. Using the Moodle plugin, the user can use visual blocks to code Javascript tasks [114, 115, 116]. By using this option the students can:

  - Add features to the basic web-lab built-in PID controller.
  - Analyze and compare the behavior of well-known controllers for learning purposes.
  - Develop/test a controller for investigation purposes.

### 8.2.2. Planar parallel robots lab

Planar robots are a subgroup of robots which are designed to develop their movements inside a fixed plane, they can develop very accurate movements at high speed and payloads compared to serial robots. The kinematics problems involved in the movement can be modeled using algebraic expressions which can be simulated using mathematical tools. Parallel robots intersects with the planar group, and are also known to have an increased accuracy to bear high loads. In this regard, ”the teaching of parallel robotics cannot be treated as a mere continuation of

[9]https://opensource.google.com/projects/blockly

serial robotics because there are significant conceptual differences between these two types of manipulators" [53]. Therefore, kinematics problems behind of parallel robots can not be directly obtained from the serial robotics kinematics, additional resources, as a EjsS simulation, can be helpful to improve these theoretical concepts.

### Physical description of the experimental device

Inside the robotics area, actual robots are constructed using a variety of joint types. The system which is going to be explored in this subsection is just using planar revolution joints, where the rotation axis are always perpendicular to the plane of the robot. All the arms are connected between and considered rigid segments, where the beginning or the end of the arms could be fixed, a bond with other arm, or the end-effector. The movement through the plane depends on the position of the joints, and the angle between one segment and the next defined the status of the arm.

The example presented in this sections is a planar parallel robot which is simulated using EjsS J&JS enabled version. The main purpose of developing this lab inside this architecture is to decrease the computational cost of the simulation. As the mathematics behind the kinematic problem inside these systems may imply heavy computations the architecture presented can be helpful, splitting the costs between the model and the view. Usually real robots are defined in a fixed design, where the main parameters, as the length of the arms, can not be modified. On the one hand, we are considering ideal simulated robots inside a Cartesian plane, without flexing arms or loose joints. On the other hand, their movements are restricted by architectural constraints and by developers-defined restrictions, like maximum/minimum joint angles.

The model used in this work correspond to one Degree Of Freedom (DOF), planar robot with 4 rotational joints (4R). It is called the *4R linkage* and has been studied deeply along the time in the robotics area [122, 123, 124, 125, 126, 127, 128].

The 4R linkage is a closed chain of joints, which has two fixed pivots as shown in Figure 8.10. The arm distances are fixed, as the system contains only rotational joints. The mathematical model can be obtained using common kinematics analysis [122].

The analysis uses as origin the point marked with an $\boldsymbol{O}$ in Figure 8.10. $\theta$ is the input angle and $\Phi$ the is output angle. The analysis of the mathematical relationships

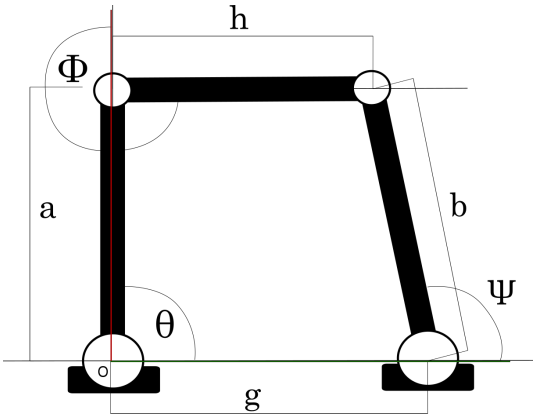



**Figure. 8.10** An schematic representation of the 4R linkage, where main angles and lengths are shown

between these angles leads to equations 8.8 and 8.9.

$$\Phi(\theta) = arctan\left(\frac{B}{A}\right) \pm arccos\left(\frac{C}{\sqrt{A^2+B^2]}}\right) \tag{8.8}$$

$$\mathrm{A}(\theta) = 2abcos(\theta) - 2gb$$

$$\mathrm{B}(\theta) = 2absin(\theta)$$

$$\mathrm{C}(\theta) = g^2 + b^2 + a^2 - h^2 - 2agcos(\theta)$$

$$\mathrm{A}(\theta)cos(\Phi) + B(\theta)sin(\Phi) = C(\theta) \tag{8.9}$$

$$\mathrm{A}(\theta) = 2ah - 2ghcos(\theta)$$

$$\mathrm{B}(\theta) = 2ghsin(\theta)$$

$$\mathrm{C}(\theta) = b^2 - a^2 - g^2 - h^2 + 2agcos(\theta)$$

Solving both equations to a single value of $\theta$ imply that there are two solutions. Then, each $\theta$ value correspond to two different positions of the planar robot as is shown in Figure 8.11.

Both equation that contains the relationship between the angles in the planar robot has geometric limitations. The limitation is expressed in the form of a condition which restricts the values of A, B and C on Equation 8.9:

$$\mathrm{A}^2 + B^2 - C^2 \geq 0 \tag{8.10}$$

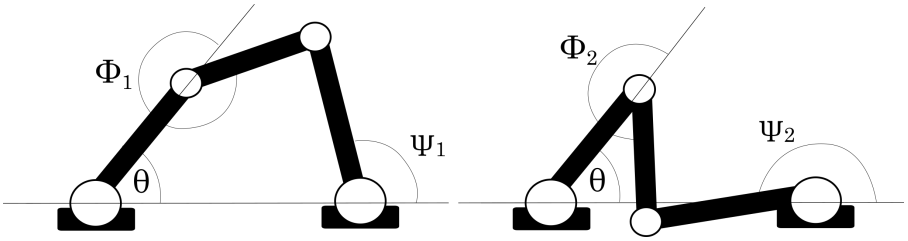



**Figure. 8.11** Visual representation of the two solutions behavior which is shown in equation 8.9 and 8.8

If restriction shown in Equation 8.10 is not satisfied, the linkage cannot be assembled. In this regard, the limits of the angles are defined by the solution given when the restriction is equal to zero. The solutions returns the maximum and minimum theta angles, and therefore the range of the input $\theta$ angle. In the same way, the limits over the output $\Phi$ angle can be calculated, returning the output range.

Both restrictions over the input and output define a workspace for the planar robot. The range of movement of the robot depends on the lengths of the arms, the distance between the fixed points and the restrictions imposed by the designer.

### The virtual lab GUI

Parallel planar robots simulation, as other VRLs, are intended to enhance the learning process by giving the students flexible systems to test the limits and behaviors of their robots. In this sense, architectural and user-defined restrictions can be edited to take advantage of the characteristics of virtual labs.

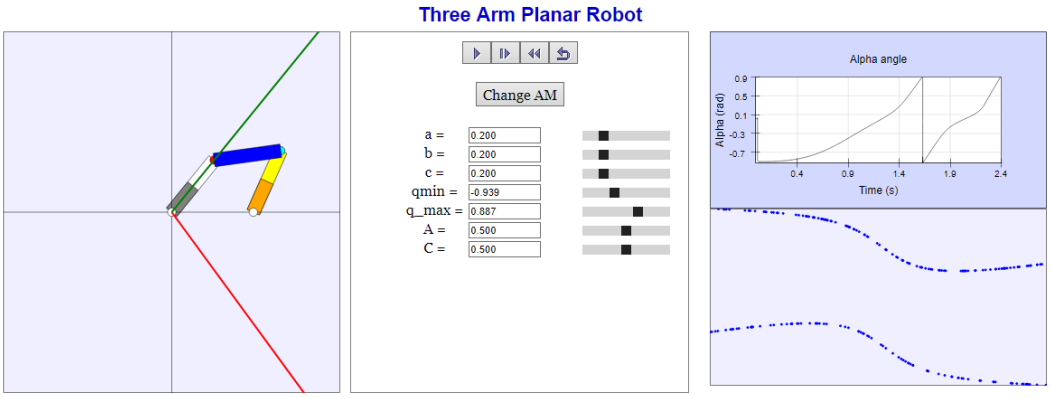



**Figure. 8.12** Virtual lab of the three arm planar robot

Figure 8.12 shows the GUI of the 4R linkage. The elements which conform

the robot structure are editable and interactive, giving the user total control over the configuration. The available fields and buttons give the student means to visualize and explore different characteristics:

- Direct interaction with the schematic robot: The movable joints (red and white circles) can be clicked and dragged by the user. When the position of the red joint is changed, the system reacts calculating the actual positions of the joints and redrawing. It is also possible to modify the relative position of the fixed points by dragging the white circle in the right part of the robot. All this calculations are made in the server side, giving a light application for the student, and returning the whole set of angles.

- Parameter edition: The fields available give a wide range of configurations, which can be divided in three categories

  - Length edition: The arms have been defined with a variable size. The sizes are controlled by *a, b* and *c* fields. Each change in the length may imply the recalculation of the actual status of the robot. As said before, if some limitations are exceeded, the linkage cannot be assembled. In this situation, the GUI reacts imposing a limit in the slider or the field. Although, if the user insist, the limit will be exceeded, and the robot will appear as broken to give feedback about the system limitations.
  - User defined limitation over the input angle: The GUI uses $\alpha$ as input angle, which correspond to $\theta$ int he mathematical analysis. The parameters $q_{max}$ and $q_{min}$ parameters define the user restrictions in the form of maximum and minimum angles (in radians).
  - Arm split rate: Figure 8.12 shown that grey and yellow arms are divided in two halves, the $A$ and $B$ parameters control where this division is located. This divisions is now just a visual reference to help the student to identify the position where the blue arm can be blocked. On this point, the system will not react when the user crosses this limitations.

- Dynamic workspace update: The right part of the GUI contains plots. The bottom plot is reserved to show the workspace of the actual robot configuration, $\alpha$ versus $\gamma$ (which is the output angle $\Phi$). The configuration space is calculated in the model when any interaction is received, returning 200 points containing the ($\alpha$, $\gamma$) pairs.

- Dynamic movement using the full input range: The mathematical analysis defines the range for the input and the output. On this matter, the GUI makes

a simulation of the movement adding a physical work over the input joint. The evolution obtained can be studied in the corresponding graph.

**Learning objectives**

The introduction to parallel robotics can be done by using this type or virtual lab, which has the educational goals of familiarizing students with the following:

1. **Inverse kinematics:** It is as a "start-up" in the understanding of parallel robotics and is carried out by analyzing the movement of a set of parallel structures in terms of the restrictions imposed by each link.

2. **Direct kinematics:** The inverse kinematic solution in parallel robotics is usually easier to understand than the forward kinematic problem. The inverse kinematics of the 4R robot are analyzed; the students should be able to derive the equations themselves. Next, the student checks and visualizes all the feasible solutions using the virtual lab. This session is particularly interesting for students since they can observe the different configurations of the robot in rapid succession to achieve the same position and orientation.

3. **Workspace and singularities:** It focuses on the forward kinematic problem. The students are asked to solve the geometric constraints of the 4R robot in terms of the joint coordinates. The students should come to understand the complexity of the forward kinematic problem, in contrast to the inverse kinematic problem, for parallel robots. The VL allows students to find the relationship between the direct and inverse kinematic problems.

These concepts are fundamental for an introduction to parallel robotics. Other concepts, such as dynamics are not introduced here, to focus in the kinematics. Although, some basis can be explained using the the applied physical work in the input joint and with the angle versus time graph.

Experimenting with the virtual lab, and configuring these parameters helps students comprehend the concepts involved. As the data and information in obtained by its own, student can develop complex analysis regarding the behavior of the system. Regarding this, topics such as the workspace and singularities analysis, are easier to understand when the various parameters involved can be modified.

### 8.2.3. Heatflow lab

The heatflow lab is a well known example in control engineering areas. This system and the use of EjsS has also a long history within the *Informática y automática* department [129, 18, 58, 130, 121]. The system can be used to achieve

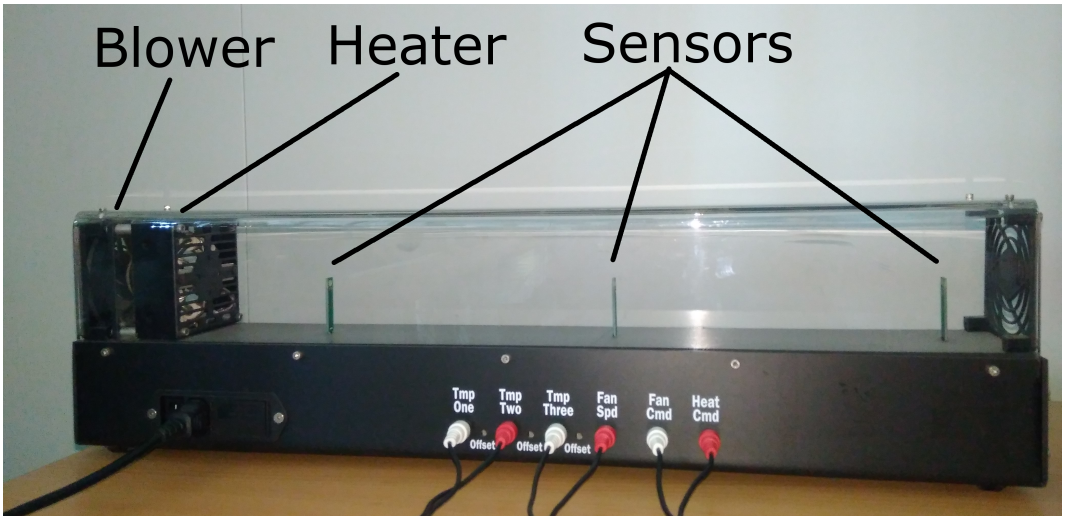



**Figure. 8.13** The heatflow system and its components: The three platinum temperature sensors, the blower and the heater grid

multiple goals and is also suitable to perform advanced control practices. As the servo motor-lab, the heatflow VRLs are included in three experimental courses of the UNILabs website:

- An open course with free access Labs on control[10].
- Labs of a course on control, inside the degree in informatics engineering. [11].
- A course of laboratory practices of the Master degree in Systems Engineering and Control[12].

### Physical description of the experimental device

Quanser Consulting[13] designed the heat-flow system we chose for the laboratory. This system consists of a partially isolated transparent tunnel with a heating grid, a blower, and three platinum sensors along the length of the tube: S1, S2, and S3. By controlling the power of the heater $V_h$, manually or in a automated process, the user can observe and analyze the outputs shown by the sensors. The fan speed

[10]https://unilabs.dia.uned.es/course/view.php?id=24
[11]https://unilabs.dia.uned.es/course/view.php?id=4
[12]https://unilabs.dia.uned.es/course/view.php?id=7
[13]https://www.quanser.com/solution/control-systems/

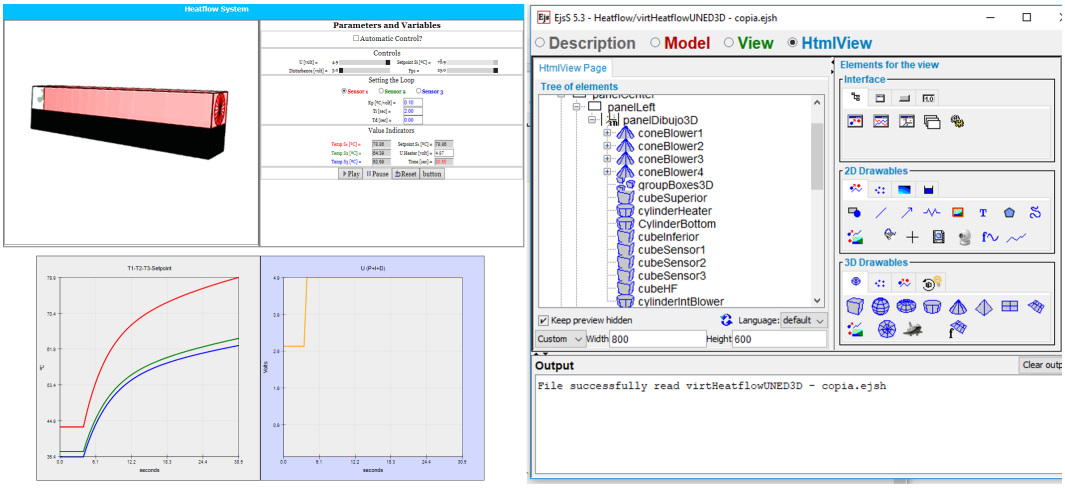



**Figure. 8.14** GUI of the heatflow along with the EjsS editor

can be also changed by changing the input signal $V_b$, and the real speed is provided by its internal tachometer, producing the $V_t$ signal.

The heatflow system is used to study: concepts related to the temperature flow control in presence of delays and identification techniques. Figure 8.13 shows the laboratory system with the main components of the heatflow device. Using system identification techniques, shown in [58], the transfer function obtained can be expressed as in equation 8.11.

$$G(s) = \frac{K_p(1+\tau_3)\exp(-\tau_d s)}{(1+\tau_1 s)(1+\tau_2 s)} \tag{8.11}$$

Where the gain $K_p$ (degrees C/volt), the lags, and the delays depend on which of the three sensors the user selects for closing the temperature control loop.

### The virtual GUI

Figure 8.14 shows the GUI of the VL developed with EjsS to control the plant. The virtual version contains a the main view divided into different parts. The graphical representation of the system is a schematic 3D representation to facilitate visualization of the heat-flow dynamics. It allows to know the actual state of the laboratory, as it gives visual feedback about the temperatures inside the transparent tube, the heater and the movement of the fan. To obtain this feedback, the inner air's color changes according to its temperature and the grid surrounding the heater is also colored. The bottom-left part of this panel shows several tabs that let users modify different experimentation parameters as:

- Manual operation: The system can be controlled manually to change the input voltage of both: the fan and the heater. The response is shown inside the graphics, at the right side of the GUI.

- Automatic operation: This represents the controlled variable. $K_P$, $T_I$, and $T_D$. Proportional gain, integral time, and derivative time of the built-in PID controller. The output to control can be changed between one of the three available sensors $S_1$, $S_2$ or $S_3$.

Using the built-in differential equation editor, the model is implemented using the information obtained from [58]. The model is designed to work as the real one, changing the analog signals $V_h$ and $V_b$ and students can close the control loop in any of the available sensors.

**Learning objectives**

The heatflow VL simulates the system dynamics using the knowledge gathered about the model. The GUI gives to the user the possibility of introducing new voltage values or to close the control loop over one of three sensors. The voltage of the blower can be changed an is used as a source of perturbations. Visual elements provide the possibility to perform many different laboratory practices. The activities can be divided into three different categories:

- **System Identification**: In this situation the model is complex, but some tasks regarding the system identification can be achieved. The information gathered may provide useful information about the order of the system and the values of zeros and poles. Even so, the knowledge level required to run and understand the system result widened, as the heatflow transfer function can be simplified. The different levels of complexity are translated to different qualities of the model, allowing the teacher to introduce and study delayed systems.

- **PID Tuning**: A closed loop can be established for controlling the temperature in one of the sensors. In all the cases, the controller used is a PID. Students can select the sensor and configure the proportional, integral and derivative parameters of the PID controller: $k_P$, $T_I$, $T_D$. These parameters can be changed in order to gather new information about the model or to satisfy the requirements of the laboratory practice.

- **Disturbances Analysis**: The system allows the introduction of perturbations in the form of variable airflow. Modifying the values of the blower voltage the

system can simulate the effect of the new air being introducing and flowing along the tube.

- **User Defined Controller**: As in last examples, advanced users can write different controllers to use in the control loop. The code of the controller can be written by using an special Moodle environment, Blockly[14]. Using the Moodle plugin, the user can use visual blocks to code Javascript tasks [114, 115, 116]. By using this option the students can:
  - Add features to the basic web-lab built-in PID controller.
  - Analyze and compare the behavior of well-known controllers for learning purposes.
  - Develop/test a controller for investigation purposes.

[14] https://opensource.google.com/projects/blockly

# 9. Remote Labs

This chapter presents RLs which have been described following the same structure of the last chapter. Each RL description is divided in three parts:

- **Server side description**: The RLs are usually based in real equipment and systems, this section gives a description and some mathematical background. To lighten the description of the labs, some parts of the mathematical background of the system included as appendices.

- **The remote lab GUI**: This section describe the user interface and additional details about the development, like the equations used to simulate the behavior.

- **Learning objectives**: The purposes of the lab can be many, but this section gives a glimpse about some activities that can be achieved.

Some labs have been updated from previous labs, and some have been developed from scratch. To represent the real lab which is the server side, the GUI has to be: graphical, dynamic, interactive and representative of the system to be studied.

The following remote labs allow the user to carry out lab experiences inside the control engineering and physics area. Some labs have been described mathematically in Chapter 8, then, the model description is not included, but can be consulted in the appendices.

## 9.1. Remote labs developed in Javascript

*The two coupled electric drives* and *The vibrating wires* experiments have been developed using this version.

### 9.1.1. Two coupled electric drives

The two coupled electric drives lab has been presented in its virtual version in previous chapter. Thus, the mathematical and theoretical modeling have been presented along with some common structures which are also used in the remote version. To maintain the global structure help students in the leaning process. A similar GUI is used to avoid a process of understanding and knowing new interfaces. Although, some changes are inevitable, because controlling a remote system is more restrictive than virtual versions. The coupled electric drives RL is a fast dynamics system, where the time needed to reach stable states is short. Fast dynamics imply that the student can make many changes and gather more data than is a slower system. But, the student needs a higher level of knowledge to carry out the experience than in other RLs due to the limitations of the real elastic band and the speeds of the motors.

#### Server side description

The architecture of previous remote labs in UNED were usually based on three software tools: EjsS, LabVIEW (Laboratory Virtual Instrument engineering Workbench) and the JIL server [131, 132, 21, 133]. In this line, the effort of the present lab focuses in the use of open standards and technologies to reduce the economical cost and efforts required for building RLs. In this sense, the *Remote Hardware Interoperability (RHI) Protocol* protocol [31] provides a well defined interface to get state measurement and control and handle user requests. The interoperability API relies on two standard *remote procedure calling* protocols (RPC): *XML-RPC* and *JSON-RPC*. This is somewhat similar to the *smart device paradigm* [98] which is used in the implementation of the EjsS J&JS.

In summary, as a general approach, RLs development can be decomposed into the middleware tier that implements the RHI protocol, and the hardware control, which usually must perform several tasks: *Communication*, *Data-Acquisition and Control*, and *Data Logging*.

In the remote lab, the server tasks are carried out by a low-cost development platform: the *BeagleBone Black* (BBB) board. The BBB is a low-cost, community-supported development platform for developers and hobbyists. This board has many interesting I/O capabilities, such as *USB* client and host, *Ethernet*, *HDMI*, *GPIO* with *PWM* and even built-in support for *I2C* and *SPI*. It is shipped with a pre-installed *Angstrom Linux* distro, which is optimized for embedded systems, but it is compatible with several Linux distros, such as *Debian* or *Ubuntu*, and even with *Android*.

The *JIL server* functions have been assumed by *Node.js*, a lightweight and ef-

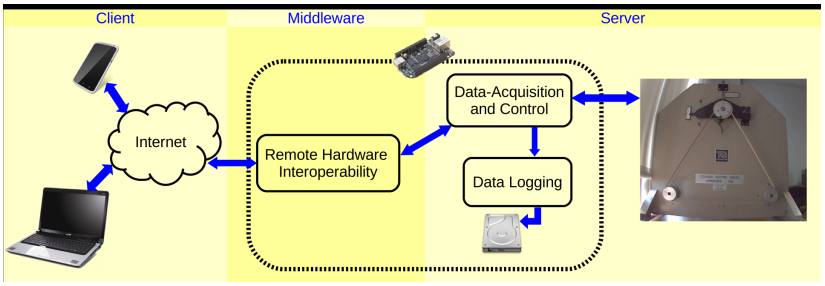



**Figure. 9.1** The architecture of the remote lab.

ficient platform which is built on *Chrome's JavaScript runtime*. This platform has been chosen because it is a ready-to-use tool, built-in in the installation of any BBB, and, as its website claims, it is for "easily building fast, scalable network applications".

The motivation for changing the PC by the BBB is twofold. On the one hand, it is much cheaper and it can, at least for this purpose, do the same function. On the other hand, the equipment used in the laboratory requires space, and PCs are much more bulky. Though it is possible to replicate the previous communication architecture based on LabVIEW with a BeagleBone Black board due to its low-cost and computational capabilities.

The middleware and the server tiers are implemented in the BBB. The framework is divided into three subsystems: *Communication*, *Data-Acquisition and Control*, and *Data Logging*.

The *communication* subsystem provides an implementation of the RHI protocol. Since there is a current trend to propose client in Javascript for compatibility with tablets, smartphones and similar devices, the use of the JSON format facilitates the integration with javascript applications.

The *data-acquisition and control* subsystem directly interfaces with the plant hardware, reading the velocities from the motors and the tension from the belt, and sending the control actions (received from the user interaction with the EjsS application in the client side) to the motors. Finally, the *data-logging subsystem* consists of a low priority loop which writes to disk the system events (sensor readings, user commands, etc).

With respect to the connection between the plant and the board, the signal measured from the sensors and the control voltages admitted by the motors are within the range (-15$V$, 15$V$). However, the analogs inputs of the BBB board admit only a range of (0$V$, 1.8$V$), while the outputs only can provide values in the range (0$V$, 3.3$V$). So to adapt the levels, a *signal-conditioning* block was designed (basically an amplifier with gains chosen according to the ranges, see Figure 9.2).

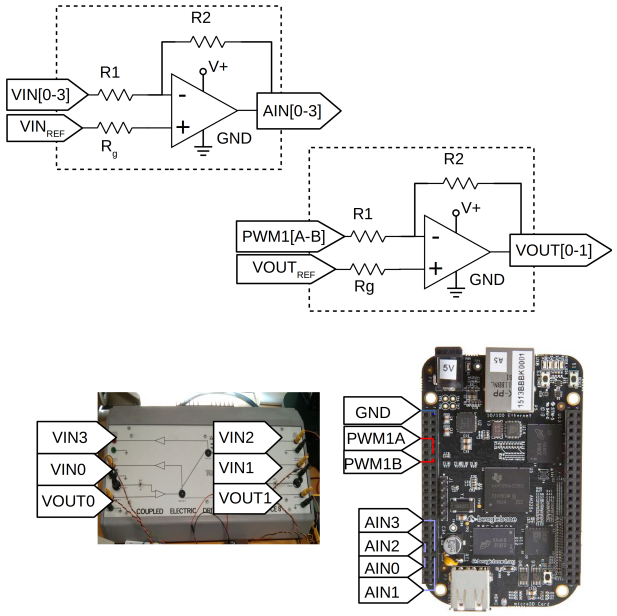



**Figure. 9.2** The plant and the board work at different voltage levels, so an op-amp based circuit was added to each I/O to obtain the adequate ranges.

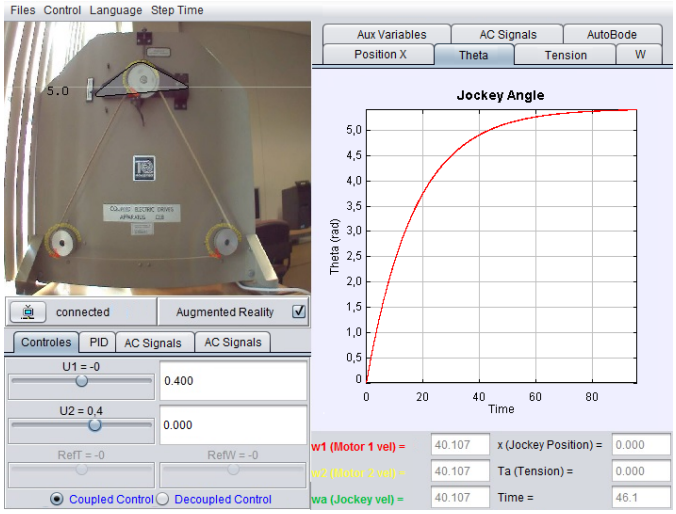



**Figure. 9.3** The remote lab application.

## The remote lab GUI

The client interface for the remote lab is still implemented as an EjsS application, similar to the virtual laboratory described in the previous Section (see Figure

9.3). Therefore the thoroughly change of the server is transparent to the user. As seen in Figure 9.3, the GUI for the RL is almost identical to the VL one. Therefore, just a minimal explanation is required in this regard to show the differences.

- *Step Time*: This menu is not possible in the remote version, as long a the step time is not controllable in a real system.
- System web-cam and control: The web-cam image contain a view of the coupled drives system, but to enhance the visualization of the dynamics, additional semitransparent figures have been added as augmented reality feedback. The high speed of motor and pulley makes almost impossible to appreciate its movement. Thus, the augmented reality feature is overlapped to the video image and help he user to identify the movement of the pulley and the motors.
- In the *Controls* tab. Students are also in control of the velocity of the motors, but remote version can not afford different rotation directions, neither big differences between speeds. In automatic mode, the $U_1$ and $U_2$ sliders are disabled and the user can change the tension and velocity references (the two inputs of this system), and select whether to make a coupled or a decoupled control for these variables.
- The *Controller* tab is not available in the remote version, as seen in the previous chapter some Blockly tools[1] included in Moodle can be used to write user-defined code in VRLs.
- The *AC Signals* tab: The main capability for making an *"ac sweep"* is available in the remote lab. Although, the real device is limited to a narrow interval of frequencies. For this reason, and due to the fragility of the elastic band, these fields and option are not available for students.

**Experimental protocol**

Students who have obtained an accurate model of the coupled drives system in the virtual laboratory will be able to program better controllers or tune better PID parameters to get the desired response in both, virtual and remote versions. In addition, they get better prepared for the remote laboratory where the system identification is not easy. If the students want to obtain a model for the real system they must take into account the results in virtual application, and then, fit the parameters of their models using the outputs of the remote lab. Visual elements provide

[1] https://opensource.google.com/projects/blockly

the possibility to perform many different laboratory practices. The activities can be divided into three different categories:

- **System Identification**: In this situation the model is complex, but some tasks regarding the system identification can be achieved. As the VL is a simulated lab, the user can study the similarities and differences between both. The information about the system response can be gathered to compare with the expected behaviors. This activity may be performed acquiring data from all the steps in the remote experiment like the open and closed loop responses.

- **PID Tuning**: When the automatic control mode is selected in the upper menu (as described in the previous Section), a closed loop is established for controlling the velocity or tension of the belt by means of a PID controller. In coupled mode, students can select the control variable (either velocity or tension) in this MIMO system, and may change the proportional, integral and derivative parameters of the PID controller ($k_P$, $T_I$, $T_D$) in order to obtain a response that satisfies the specifications asked in the practice guide. In decoupled mode, students work with two SISO systems and, therefore, with two PID controllers. Then, an analysis of the output signals (stored in .m files), allows to obtain an approximated model of the system.

- **User Defined Controller**: As in last examples, advanced users can write different controllers to use in the control loop. The code of the controller can be written by using an special Moodle environment, Blockly[2]. Using the Moodle plugin, the user can use visual blocks to code Javascript tasks [114, 115, 116]. By using this option the students can:

  - Add features to the basic web-lab built-in PID controller.
  - Analyze and compare the behavior of well-known controllers for learning purposes.
  - Develop/test a controller for investigation purposes.

### 9.1.2. The vibrating wires

The vibrating wires lab has been presented in its virtual form in the previous chapter. Thus, the mathematical and theoretical modeling have been presented along with some common structures which are also used in the remote version. Maintaining the global structure, the students can follow a similar path to perform

[2]https://opensource.google.com/projects/blockly

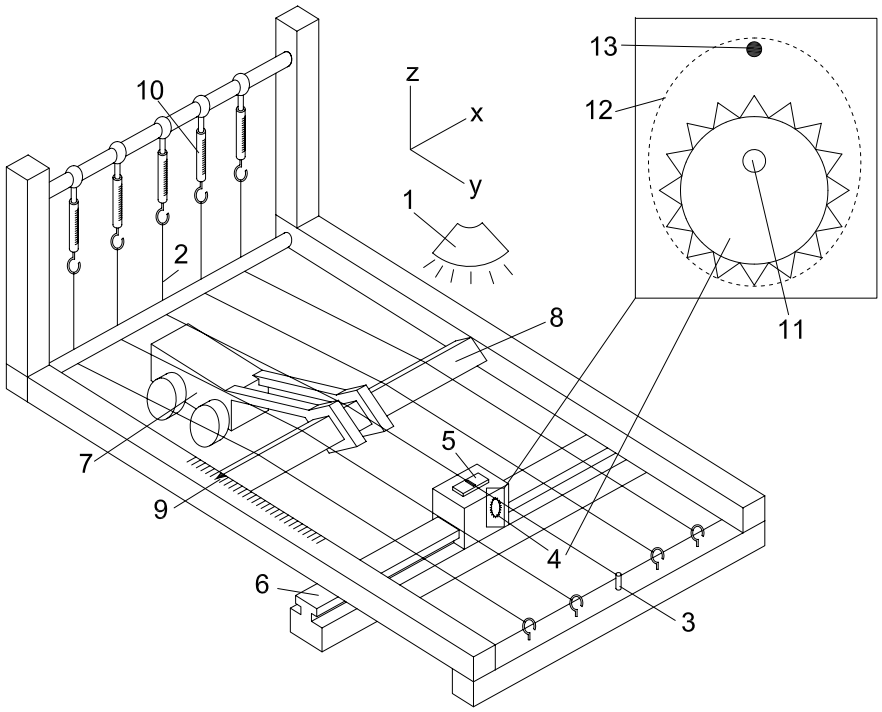



**Figure. 9.4** Fully developed experimental device, consisting of the following elements: (1) DC LED light source, (2) strings, (3) stepper motor, (4) LEGO gear connected to LEGO servomotor, (5) light sensor, (6) linear stage, (7) LEGO carrier, (8) mobile aluminum rod, (9) rule and length indicator, and (10) dynamometers. The close-view figure shows a frontal view of the string plucking element consisting of (11) the rotation axis of the LEGO gear, (12) trajectory described by the LEGO gear perimeter, and (13) string under study.

the lab experiences. Both graphical user interfaces are almost equal, therefore, the experience gained in the virtual version can be applied in the remote one.

### Server side description

With LEGO$^{TM}$ kits it is remarkably easy to introduce experimentalists into robotic designs. The development of Arduino boards also allows one to control stepper motors, light diodes, and so, in an easy manner. These tools can be widely used to construct future laboratories, compatible with the requirements of a given experimental setup and allowing the students to measure, analyze and extract conclusion with laboratories developed in the cloud.

A schematics of the device is shown in Figure 9.4 (which is the same Figure from 8). The real version contains five different strings (2) made of different materials (copper, kanthal, constantan, and nickel) and with diameters ranging from 0.3 mm to 0.5 mm. One of the ends of the strings (with the exception of the central string) is fixed on the aluminum structure of the device, while the other end of the strings is connected to a dynamometer (10) that measures the tension along the string. In the case of the central string, the fixed end is connected to the axis of a stepper motor (3), so that the tension can be controlled. A LEGO carrier (7) is used to displace an aluminum rod in close contact with the strings (8) along the $y$ axis, in such a way that the length of the vibrating part of the strings can be changed from 380 mm to 550 mm. This length is measured by means of a rule and an indicator attached on the mobile rod (9). A DC-LED (Galaxy 1000) light source (1) illuminates the system from above, and a linear stage (RS 340-3749) (6) is setup below the strings along the $x$ axis. Two elements are attached on the top of this linear stage: i) a light sensor (Phywe 08734-00) covered by an opaque cap with a 0.3 mm slit oriented along the $y$ axis (5), and ii) a LEGO gear connected to a LEGO servo motor (4). As can be seen in the close view of Fig. 8.5, the rotation axis of the LEGO gear (11) does not coincide with its center, so that the perimeter of the gear roughly describes an ellipse when the LEGO servo motor rotates (12). The position of the opaque cap and the gear along the vertical direction has been fine-tuned in such a way that, first, the cap of the light sensor is placed less than two millimeters below the horizontal plane formed by the strings, and second, the apex of the gear perimeter trajectory coincides with the horizontal plane formed by the strings (13). The stepper motor and the linear stage are controlled by two identical drivers (EasyDriver), and An Arduino I/O boardcard is used to send the convenient digital signals. A power supply (Lendher 3003D) provides the current required by both drivers, and a second identical power supply is used for the DC-LED light source. An oscilloscope (PicoScope 2203) is used to read the measurement from the light sensor. A LabVIEW code has been developed to control all of the above mentioned elements within the JIL server [131, 132, 21, 133] which have been used successfully in other labs. To connect both sides (LabVIEW on the server side and EjsS on the client side) the lab architecture includes a JIL server, [31]. JIL uses the XML-RPC protocol to encode the messages and allow data exchange between both sides, as shown in figure 9.5.

### The remote lab GUI

The GUI for both the virtual and remote lab is built using EjsS in the Javascript enabled version. It gives the student the control of the experimental environment

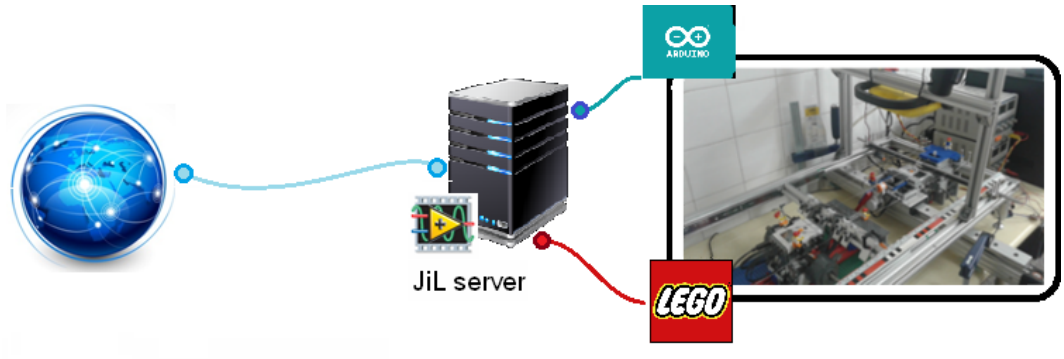



**Figure. 9.5** VRL communications architecture

(tension, length...) as in a hands-on laboratory.

Figure 9.6 shows the basic structure of the virtual laboratory when the .xhtml is served to the client. As it was said in previous sections, the virtual laboratory is based on a simulation of the system behavior and the GUI. The virtual lab also allows students to get familiar with the available interaction and the protocol, in order to be prepared to the remote version of the lab. The application window is divided into three sections: the web-cam, the controls panel and the plots/graph panel.

- *Web-cams*: In the remote version, this panel contains a panel which can be used to observe one of three web-cams. Using the obtained images, the student take measures from the tension and length of each string, or observe the device with a general view of the lab. Figure 9.6 contains and example of the remote GUI running.

- *Controls panel*: As in the virtual GUI, the user is helped to go through the experimental protocol, highlighting each step and giving tool-tips to make it easier.

- *Plots/Graphs panel*: The vibrating string RL plots the light intensity versus, first, the position of the linear stage (see section 5 for details in this calibration procedure), and second, time (in order to obtain the frequency of the fundamental normal mode). Both obtained using the oscilloscope device which is connected to the light sensor.

### Experimental protocol

Three CCD cameras allow the student for the visualization of a general view of the experimental setup as well as a close view of the measurement elements

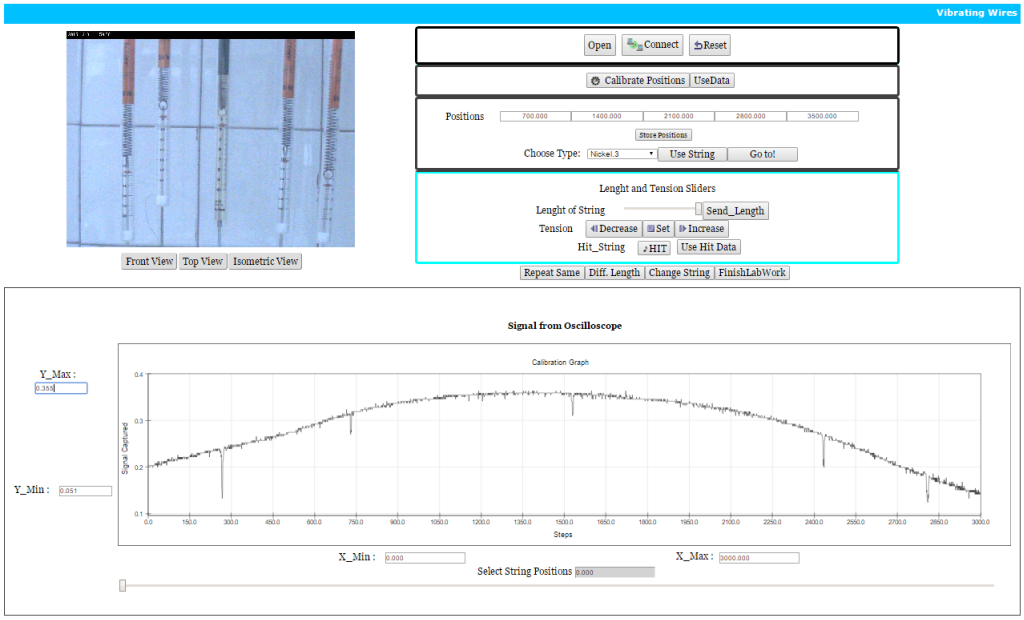



**Figure. 9.6** The Remote vibrating string system laboratory.

(dynamometers and length indicator). Once the student connects to the remote controller, an automated initialization procedure is executed by the device: The DC-LED is turned on, and the linear stage and the LEGO carrier are displaced to their initial positions, determined by means of two LEGO limit switches. After this initialization, the student has to proceed as follows:

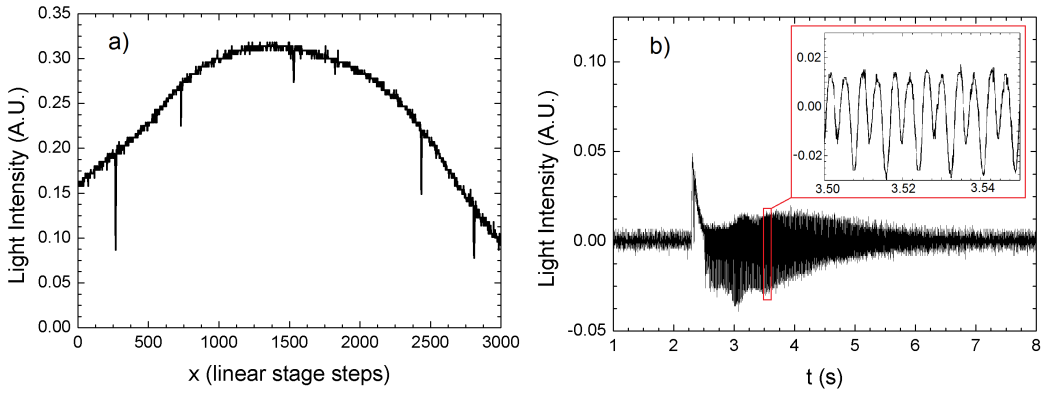



**Figure. 9.7** a) Light intensity versus position of the light sensor along the $x$ axis. b) Light intensity versus time after the gear hits one of the strings. The inset graph represents a close view of the results from $t = 3.5$ s to $t = 3.55$ s.

1. **Location of the strings' positions:** A complete sweep is performed by the linear stage along its whole range of displacement, while the light sensor attached on its top is continuously measuring the light intensity. Therefore, the student can see the function light intensity versus position along the $x$ axis. As can be seen in Fig. 9.7a, the light intensity is roughly symmetric with a local maximum at the center. This is because we use a single light source placed at the center of the device in order to avoid multiple shadows produced by multiple light sources. Thus, the student can determine the position of each string from the five local minima in the light intensity.

2. **Selection of the string and the length to explore:** After this calibration, the linear stage is displaced to the string selected by the student, and the LEGO carrier moves backward or forward to reach the desired length. The student still needs to obtain a precise measure of the positions of the wires, but the real linear stage helps the student by performing an automated fine-tuning. It ensures that the thin slit of the light sensor is placed exactly below the string shadow.

3. **Selection of the tension (if needed):** If the string selected by the student is the central one, the tension can be varied by the stepper motor. In that purpose, by clicking an *increase tension* (or *decrease*) control, the stepper motor rotates a fixed number of steps in the clockwise (or counterclockwise) direction, so that the tension is increased (or decreased) in steps of approximately 0.05 N. The student observes the measurement of the dynamometer by means of one of the CCD cameras, and is able to change the tension as long as it is maintained below 10 N to avoid breakage of the string.

4. **Execution of the experiment and data acquisition:** When the previous steps have been completed, the device is ready to execute the experiment in those conditions selected by the student. Then, by clicking the corresponding control, the light sensor starts to measure the light intensity, and the LEGO gear attached on the top of the linear stage performs a 360° rotation, plucking the string when it reaches its highest position. The Fig. 9.7b shows the results of an actual experiment as an example. The instant in which the gear hits the string and the relaxation dynamics are clearly observed. The student has to analyze these data, calculating the frequency of the fundamental normal mode, $f_1$.

5. **Analysis of the results and comparison with theory:** Once the student has completed experiments under different physical conditions, the dependence

relation between $f_1$ and the physical parameters of the string ($T$, $L$, $\rho$) can be established. Then, the student should be able to discuss the experimental errors and the validity of the theoretical model.

## 9.2. Remote labs developed in Java & Javascript

### 9.2.1. The servo motor lab

The servo-motor lab has been presented in its virtual version in last chapter. Thus, the mathematical and theoretical modeling have been presented along with some common structures which are also used in the remote version. To maintain the global structure help students in the leaning process. A similar GUI is used to avoid a new process of understanding and knowing the GUI. Although, some changes are inevitable, because controlling a remote system is more restrictive than virtual versions. The servo-motor RL is a fast dynamics system, where the time needed to reach stable states is short. Fast dynamics imply that the student can make many changes and gather more data than a slower system. Therefore, the student needs less level of knowledge to carry out the experience. But, these students which have studied the model obtained in the VL can make smarter decisions, taking advantage of the available time.

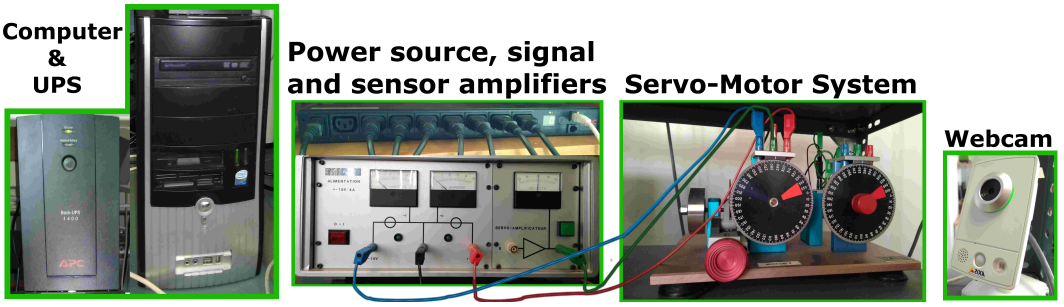



**Figure. 9.8** Components of the remote servo motor lab.

#### Server side description

The labs which are rebuilt using the EjsS J&JS enabled version follow a common architecture. Remote labs in UNED were usually based on three software tools: EjsS, LabVIEW (Laboratory Virtual Instrument engineering Workbench) and the JIL server [131, 132, 21, 133]. In the server PC, there is a LabVIEW VI (Virtual Instrument) which is plant-dependent and implements a local control. The server PC

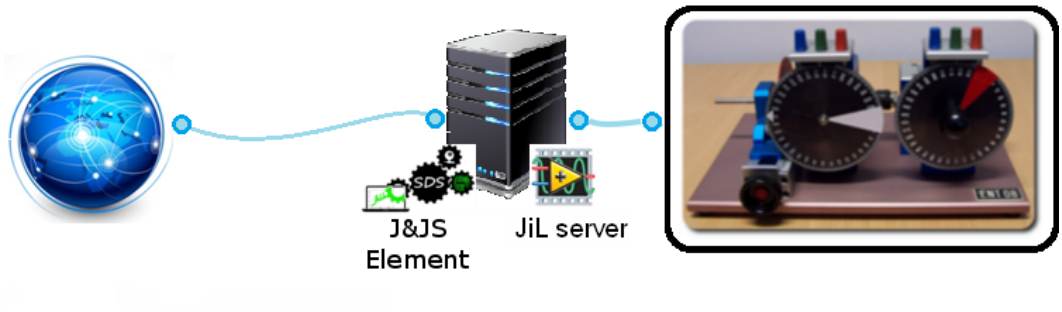



**Figure. 9.9** Software components of the remote servo motor lab.

runs the JIL server as the main interface between the local EjsS model application and the LabVIEW VI. The model application establish also the communications between the server side and the final user. This local architecture is described in Figure 9.9

The Figure 9.8 shows the servo-motor lab with all its components at the server side. It is divided into three main elements:

- Servo-motor device, presented in Chapter 8.
- Computer and Uninterruptible Power Supply (UPS): Computer contains the software which is needed to establish the exchange with the hardware and with the client. In this sense, the computer carry out all the server side tasks:
  - Runs the model, which is on charge of listening to connections from clients, building and translating the messages from the client and run the code which is triggered by user interactions (Chapter 5 contains additional details). The model establishes a connection between the clients and the LABView software.
  - Runs the LABView software: LABView is in charge of connecting with the acquisition board to perform control and sensing tasks. LABView is also in charge of exchanging information with the model, which adds the communication capabilities using the SDS. Control of the system is done inside the LABView software, as the acquisition board and servo-motor hardware are intended to be used with this software. LABView is a powerful tool that enhances the capabilities of the hardware.
- Webcam: Is an IP-camera which is connected to the same network as the computer and is used to provide a visual reference of the system. Figure

shows a image which is similar to the one shown in the VL, but in the remote version the user observes the real marked disk.

### The remote lab GUI

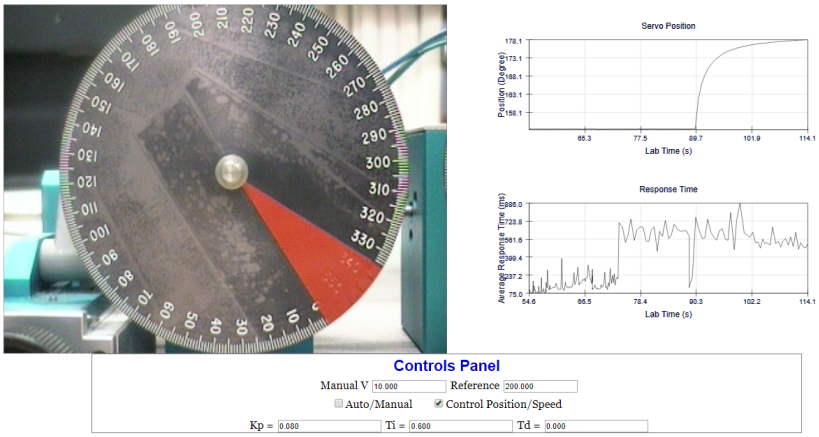



**Figure. 9.10** Components of the remote servo motor lab.

Figure 9.10 shows the GUI of the RL developed with EjsS to control the plant. The remote version of the GUI is similar to the one presented in the VL, but, there are some differences. The graphical representation of the system has been changed by a web-cam image, which contains a view of the motor system. The RL is exchanging information to know the actual state of the laboratory in the server side and it is plotted and visualized in the graphs and field in the GUI.

### Experimental protocol

The remote lab is almost equal to the simulated version of the lab. Although, some minimal details must be commented, as the experiences return real, then, different data. When the user loads the webpage, the laboratory must be started manually by connecting to the server. Using the RL the student can perform common tasks in control engineering courses and use their experience from the virtual version of the servo-motor lab.

- **System Identification**: RL's identification can be done in two ways. The first one implies a comparison step, where the student must compare both, the virtual and remote responses. The second step involves obtaining an accurate model for the real motor. This can be done by gathering information from the

open and closed loop analysis. The first step implies qualitative comparison to prepare student for the subsequent modeling.

- **PID Tuning**: The closed loop is used for controlling the velocity or position of the servo-motor by means of a PID controller. The student can tune the PID by hand by changing the proportional, integral and derivative parameters of the PID controller ($k_P$, $T_I$, $T_D$). The user can perform this task with no knowledge about the system, but, in order to obtain the best response, the student can also use the acquired knowledge to design the perfect PID for each situation, following the goals of the practice guide.

- **Disturbances Analysis**: The real lab cannot introduce an isolated disturbance in the motor, but in the manual control mode, the user can introduce signals to reproduce part of this behavior.

- **User Defined Controller**: It can be done as in the virtual version, using the Moodle plugin which uses Blockly to edit the controllers of the lab.

The set of activities presented for the servo-motor lab are enough to cover the possibilities for this lab. Although, more complex lab experiences can be imagined, then, the RL not limited to this set. The system is configured to exchange just a few variables needed to make these educational laboratory practices, but there are much other information that can be exchanged. On this subject, experiments for advanced students, regarding the system identification for example, are also possible. The server side is not the idealized version of the lab where anything in controllable. Such kind of server side is possible, but not useful for education. Although, the combination of $model + LABView$ gives a wide range of controllable parameters. And as the lab is using J&JS enabled version of EjsS, adding these variables or making new calculations in the model is easier than in previous versions.

### 9.2.2. Heatflow lab

The reader has a glimpse about the heatflow system, as the mathematical and theoretical modeling have been presented previously in Chapter 8. In the virtual version the simulation can be configured to use a bigger step time. This change results in a simulation where the time runs faster than with little step times. In the remote version the time is not a editable variable and the heatflow lab is a system characterized by slow dynamics. This means that the system needs a significant amount of time to reach stable states. Slow dynamics imply that the student needs a higher level of knowledge to carry out the experience than in the case of the

servo motor lab. Therefore, the student can make smarter decisions to improve the available time.

### Server side description

The Figure 9.11 shows the heatflow lab with all its components at the server side. Is divided in three main elements:

- Heatflow system: It contains three sensors, a heater grid and a blower inside a semi-isolated plastic tube.
- Acquisition board: Connected to both, the computer and the heatflow, to acquire the data from sensors and to control the signals of the blower and the heat grid actuators.
- Computer and UPS: Computer contains the software which is needed to establish the exchange with the hardware and with the client. In this sense, the computer carry out all the server side tasks:
  - Runs the model, which is in charge of listening to connections from clients, building and translating the messages from the client and run the code which is triggered by user interactions (Chapter 5 contains additional details). The model establishes a connection between the clients and the LABView software.
  - Runs the LABView software: LABView is in charge of connecting with the acquisition board to perform control and sensing tasks. LABView is also in charge of exchanging information with the model, which adds the communication capabilities using the SDS. Control of the system is done inside the LABView software, as the acquisition board and heatflow hardware are intended to be used with this software. LABView is a powerful tool that enhances the capabilities of the hardware.
- Webcam: Is an IP-camera which is connected to the same network as the computer and is used to provide a visual reference of the system. As the heatflow lab is not too *visual* some capabilities have been added to the GUI to improve the feedback.

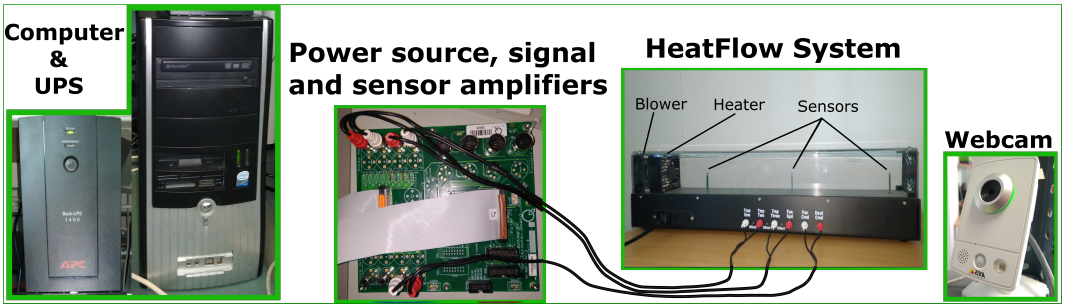



**Figure. 9.11** Components of the remote heatflow lab.

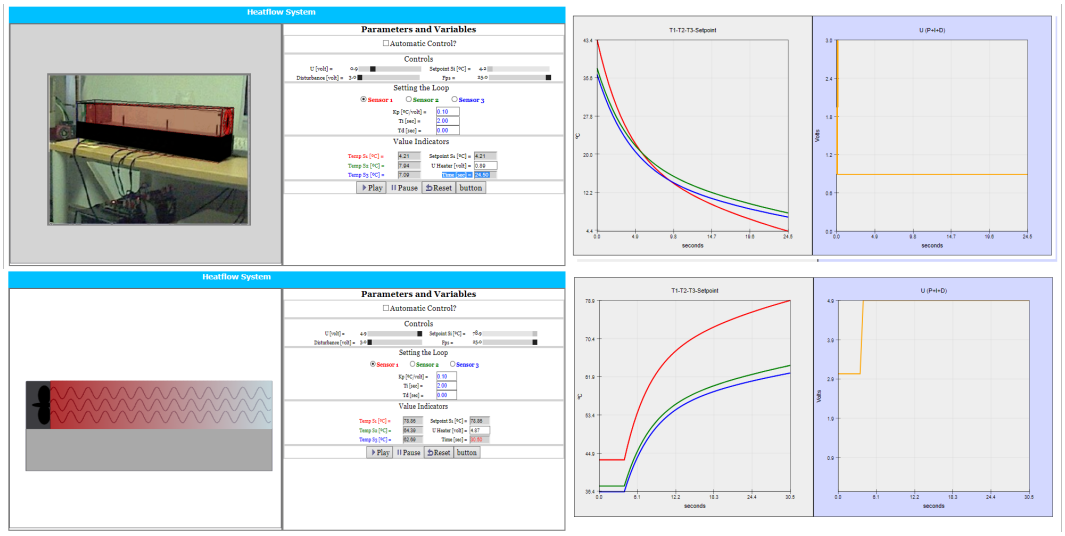



**Figure. 9.12** GUI of the heatflow RL with the augmented reality feature activated to see the temperature in the airflow inside the plastic tube. As 3D graphics consume many computation resources, a 2D simplified view is also available, but without the augmented reality capabilities

## The remote lab GUI

Figure 9.12 shows the GUI of the RL developed with EjsS to control the plant. The remote version of the GUI is similar to the one presented in the VL, but, there are some differences. The graphical representation of the system has been changed by a web-cam image. The RL is exchanging information to know the actual state of the laboratory in the server side. The bottom-left part of this panel shows several tabs that let users modify different experimentation parameters as:

- Manual operation: The system can be controlled manually to change the input

voltage of both: the fan and the heater. The response is shown inside the graphics, at the right side of the GUI.

- Automatic operation: This represents the controlled variable. $K_P$, $T_I$, and $T_D$. Proportional gain, integral time, and derivative time of the built-in PID controller. The output to control can be changed between one of the three available sensors $S_1$, $S_2$ or $S_3$.

The web-cam image contains a view of the heatflow, but to enhance the visualization of the dynamics, an additional image has been added in a augmented reality manner. The additional 3D representation gives some visual feedback about the temperatures inside the transparent tube, the heater and the movement of the fan. Some of the RL incorporate the augmented reality feature by which a graphical representation of the system is overlapped to the video image. In the heatflow system this feature is particularly useful because users can appreciate the heating process inside the duct since the colour of the air changes from blue to red gradually. The augmented reality feature can give the user the feeling of physical presence in the lab and some visual feedback.

**Experimental protocol**

The RL of the heatflow is a real system where the dynamics are slow. The knowledge gathered in its virtual version may play a key role in the experimental process in the real lab. Visual elements provide the possibility to perform many different laboratory practices. The activities can be divided in four different categories:

- **System Identification**: In this situation the model is complex, but some tasks regarding the system identification can be achieved. As the VL is a simulated lab, the user can study the similarities and differences between both. The information about the system response can be gathered to compare with the expected behaviors. The heatflow is a delayed system and can be studied in many different ways in the control systems theory.

- **PID Tuning**: As in the VL, the control loop can closed using one of the three available sensors. The RL is configured by default with PID parameters which are not optimized. In this regard, students can modify the selected sensor and configure the proportional, integral and derivative parameters of the PID controller: $k_P$, $T_I$, $T_D$. The characteristics of the output and control signals are more restrictive in the RL, because of their natural limitations.

- **Disturbances Analysis**: The VL gives also control over the voltage of the blower, changing the airflow crossing the plastic tube. It allows to see the real effects of introducing perturbations. As is a real system, the blower has a minimum voltage to be operational and a maximum voltage, where the airflow is maximum.

- **User Defined Controller**: As in last examples, advanced users can write different controllers to use in the control loop. The code of the controller can be written by using an special Moodle environment, Blockly[3]. Using the Moodle plugin, the user can use visual blocks to code Javascript tasks [114, 115, 116]. By using this option the students can:

  - Add features to the basic web-lab built-in PID controller.
  - Analyze and compare the behavior of well-known controllers for learning purposes.
  - Develop/test a controller for investigation purposes.

Though these are the main activities that can be performed over the heatflow system, the RL is not limited to this set. The system is configured to exchange just a few variables needed to make these educational laboratory practices, but there are much other information that can be exchanged. On this point, experiments for advanced students, regarding the system identification for example, are also possible. The server side is not the idealized version of the lab where anything in controllable. Such kind of server side is possible, but not useful for education. Although, the combination of $model + LABView$ gives a wide range of controllable parameters. And as the lab is using J&JS enabled version of EjsS, adding these variables or making new calculations in the model is easier than in previous versions.

[3]https://opensource.google.com/projects/blockly

# 10. Conclusions and Future Work

## 10.1. Conclusions

The outcomes of the research done in this Thesis can be divided into software, developed VRLs results.

### 10.1.1. Software results

Software results comprises two main contributions. The first one is the study and analysis of the VRLs needs and its problematic. On this subject, the VRLs design has been divided in three topics:

- Hardware to network.
- Network to device:
  - Architectures.
  - Protocols.
  - Data formats.
- Programming languages.

Similarly, the problematic associated to the architecture of the VRL has been studied and divided in three main problems:

- Controller location.
- Model and GUI coupling.
- Computational needs of the system.

The second one shows a collection on minimal requirements for a solution, taking in considerations the state of art and the problems presented. Following this requirements, a new enabled version of EjsS has been implemented. The tool makes easier the development of VRLs by automating the GUI and model creation. The implemented solution is adaptable to other lab structures and the level of standardization given by the data format make the VRLs usable by other developers or institutions.

The study of the problematic when developing VRLs is of particular interest since the problems are associated with many different areas of study. Thus, usually it is not possible to find general solutions, as each problem is inside a wide area of knowledge.

Therefore, a general design of a solution has been proposed to eliminate or reduce common problems when developing VRLs inside an online course. In this sense, the solution gives general guidelines to conform laboratories that can be run in mobile devices and be compatible with other laboratories, tools and devices by sharing a common structure.

It is always desirable to check the theoretical solution inside a real environment. In this regard, EjsS tool is the perfect framework to implement the solution as to test it in the development of VRLs. A significant effort has been dedicated to ease the use of the tool, encapsulating all the low-level issues presented at the client side into an *EjsS Model Element*, and adding a wrapper to enhance the edition by the user. Then, the implementation of the solution inside a real tool in important, an so, the labs developed using this tool.

### 10.1.2. Developed VRLs results

To illustrate and validate the software results, a collections of virtual and remote labs for education are presented in this Thesis. These VRLs have been done using the different versions of EjsS and in different areas like: Physics, Robotics and control engineering:

- **Physics area** labs studies physical phenomena in systems. On this point, students of this area must learn about the basis of the system and methodologies to obtain information. The Thesis has presented two version of the vibrating wires lab: the virtual and the remote. In this lab the student can excite a wire and acquire data directly from the oscilloscope to study the nature of vibrations.

- **Robotics area**: The kinematics problems involved in the movement of robots can be simulated using mathematical tools such as EjsS and can be helpful to

improve these theoretical concepts. The *Planar robots lab* gives the students a flexible system to test the limits and behaviors of their robots.

- **Control engineering area** have well known examples to study automatics. In this regard, four labs have been developed or updated. These labs share similar learning objectives: System identification, PID tuning, disturbances analysis and the design of a controller.

  - *Two coupled electric drives remote lab* in both versions: virtual and remote. This lab is used to investigate basic and advanced principles of control on multi-variable systems. Additionally it offers the possibility to perform frequency response analysis.
  - *Servo motor lab*: The main purpose of this lab is to control its position or speed.
  - *Heatflow lab*: The main purpose of this lab is to control the temperature by closing the control loop using one of three sensors.

## 10.2. Future Work

Lines of further work can be divided into software, developed VRLs. On the one hand, the software ones:

- Add capabilities to the Javascript and Java elements, to make its interfaces simpler and ready-to-use.
- Build a Java library containing all the functionalities described to be reusable and ready to use in non-EjsS environments.
- Merge different elements into a common model element, sharing resources and capabilities. The final element will be able to connect with software, hardware or devices in a simpler manner.
- Consider other programming languages to build the library, to widen the usability of the solution.

On the other hand, in developed VRLs lines of future work we propose the completion or update on VRLs that are now in development or in its Java version:

- Complete a RL, that is now in development by using the J&Js version. The remote version for the planar robot (see Subsection labelVL-planarRobot): The virtual one is very useful to check and understand the basis of planar robots, but the remote will give a practical vision of the data acquiring, structural restrictions and direct and inverse problems.
- Complete the Maxwell disc laboratory, that are now in development by using the Javascript version:
  - Remote version of Maxwell disc laboratory.
  - Virtual version of the Maxwell disc.
  - A pneumatic linear actuator laboratory.
- Update the Mobile robots laboratory, that is now only available in its Java version, by using the J&Js version. The mobile robots lab was used before in its Java version [134, 135], now due to the their library usage and processing it is a good candidate to be used in the J&Js version.

# APPENDIX

# A. Coupled Electric Drives Mathematical Model

The mathematical model for the coupled electric drives system is well known [113] and can be obtained using the scheme in Figure A.1. Using the lagrangian mechanics, the kinetic energy, and the dissipation term of motors and pulley, it follows that:

$$\mathrm{T} = \frac{1}{2}m\dot{x}_p^2 + \frac{1}{2}I_1\dot{\theta}_1^2 + \frac{1}{2}I_2\dot{\theta}_2^2 + \frac{1}{2}I_p\dot{\theta}_p^2 \tag{A.1}$$

$$\mathrm{R} = \frac{1}{2}b_1\dot{\theta}_1^2 + \frac{1}{2}b_2\dot{\theta}_2^2 + \frac{1}{2}b_{pa}\dot{\theta}_p^2 + \frac{1}{2}b_{pt}\dot{x}_p^2 \tag{A.2}$$

Where $x_p$ is the jockey pulley vertical position, $[I_1, I_2, I_p]$ are the motors and pulley inertia, $m$ is the jockey pulley mass, $[b_1, b_2]$ are the motors friction, $[b_{pt}, b_{pa}]$ are the translational and angular pulley friction, and $[\theta_1, \theta_2, \theta_p,]$ are the angular positions. $\alpha$ is the half-angle between the elastic belts. $\alpha$ is not constant, due to the movement of the jockey, but, the total variation can be neglected in a first approximation. The elastic nature of the belt can be approximated to a spring so that the potential energy is:

$$\begin{aligned}\mathrm{V} = \frac{1}{2}[r(\theta_1 - \theta_2)]^2 + \frac{1}{2}k[r(\theta_1 - \theta_p) - x\cos(\alpha)]^2 \\ + \frac{1}{2}k[r(\theta_p - \theta_2) - x\cos(\alpha)]^2 + \frac{1}{2}k_0x^2\end{aligned} \tag{A.3}$$

Where [$k$,$k_0$] are the belt and spring stiffness and $r$ is the radius of the pulley and the motors, which are assumed to be equal. Neglecting the pulley inertia and angular dissipation and using equations (A.1), (A.2) and (A.3), we finally obtain the

following expressions where the torques $\tau_1$ and $\tau_2$ are the system inputs:

$$I_1\ddot{\theta}_1 + b_1\dot{\theta}_1 + kr^2[\frac{3}{2}\theta_1 - \frac{3}{2}\theta_2 - \frac{x}{r}\cos(\alpha)] = \tau_1 \quad \text{(A.4)}$$

$$I_2\ddot{\theta}_2 + b_2\dot{\theta}_2 + kr^2[-\frac{3}{2}\theta_1 + \frac{3}{2}\theta_2 + \frac{x}{r}\cos(\alpha)] = \tau_2 \quad \text{(A.5)}$$

$$m\ddot{x} + b_{pt}\dot{x} - kr\theta_1\cos(\alpha) + xk_0 + kr[\theta_1\cos(\alpha) + \frac{2x}{r}\cos^2(\alpha)] = 0 \quad \text{(A.6)}$$

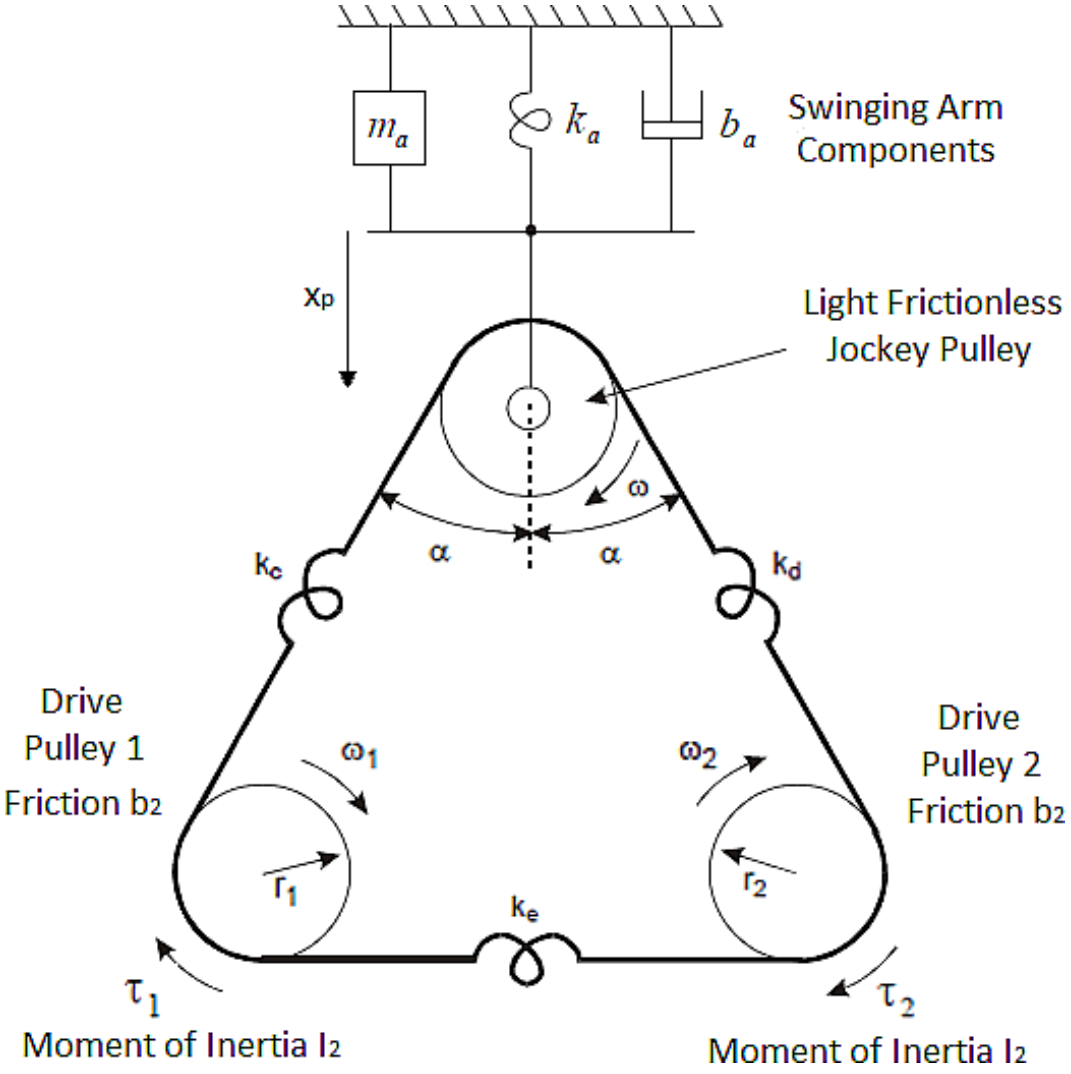



**Figure. A.1** Equivalent dynamic components of coupled electric drives [113]

The equations (A.4), (A.5) and (A.6) are used in the virtual laboratory, taking $I = I_1 = I_2 = I_p$, $b = b_1 = b_2$.

Dynamic equations in matrix notation could be written as:

$$M = \begin{pmatrix} I_1 & 0 & 0 \\ 0 & I_2 & 0 \\ 0 & 0 & m \end{pmatrix} \quad B = \begin{pmatrix} b & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & b_{pt} \end{pmatrix}$$

$$K = \begin{pmatrix} \frac{3}{2}kr^2 & -\frac{3}{2}kr^2 & -kr\cos(\alpha) \\ -\frac{3}{2}kr^2 & \frac{3}{2}kr^2 & kr\cos(\alpha) \\ -kr\cos(\alpha) & kr\cos(\alpha) & k_0 + 2k\cos^2(\alpha) \end{pmatrix}$$

$$M \cdot \frac{d^2}{dt^2} \begin{pmatrix} \theta_1 \\ \theta_2 \\ x \end{pmatrix} + B \cdot \frac{d}{dt} \begin{pmatrix} \theta_1 \\ \theta_2 \\ x \end{pmatrix} + K = \begin{pmatrix} \tau_1 \\ \tau_2 \\ 0 \end{pmatrix}$$

On this point, the problem is to control the belt speed and the tension of the belt by regulating the voltage in both motors. Then system inputs are the two motor voltages and the outputs are the speed and tension of the belt. Finally, with the Laplace transform and rewriting the output and input vector to match with the inputs and outputs of the system:

$$Z(s) = G(s) \cdot U(s)$$

$$U(s) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}$$

$$Z(s) = \begin{pmatrix} s/2 & s/2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \theta_1 \\ \theta_2 \\ x \end{pmatrix}$$

| m | r | I | b | k | $k_0$ | $b_{pt}$ | $\alpha$ |
|---|---|---|---|---|---|---|---|
| 0.35 | 0.03 | $8 \cdot 10^{-4}$ | $9 \cdot 10^{-2}$ | 50 | 200 | 0.5 | $\pi/6$ |
| kg | m | $kgm^2$ | $Nm/s$ | N/m | N/m | N/s | |

**Table A.1** Coupled electric drives parameters

Thus, using the Table A.1, to simplify the equations, the transfer functions of this MIMO system are:

$$\begin{pmatrix} \dot{\theta}_p \\ x \end{pmatrix} = \begin{pmatrix} \omega_p \\ x \end{pmatrix} = \begin{pmatrix} G_\omega & G_\omega \\ -G_x & G_x \end{pmatrix} \begin{pmatrix} \tau_1 \\ \tau_2 \end{pmatrix}$$

$$\mathrm{G}_\omega(s) = \frac{0.5}{Is+b} \tag{A.7}$$

$$\mathrm{G}_x(s) = \frac{4600}{(s^2+1.6s+790)(s^2+111s+1)} \tag{A.8}$$

Equation (A.7) (the belt's velocity) matches with a first order system, while in equation (A.8) (the belt's tension) we get a transfer function with two real and two complex poles. This transfer function has been obtained making some assumptions due to the dimensions and geometry of the real system:

- Model the elastic band as springs, mass and friction.
- Neglect the pulley inertia.
- Neglect the angular disipation.
- Neglect the variation in $\alpha$.
- Assume equal inertias for both motors.

The system parameters (see Table A.1) have been obtained from a bench designed by TecQuipment[1], therefore, the final results in this Appendix are an approximation to this system. Later simulations done using the differential equations can consider or not the assumptions presented, giving to the developers different levels of fidelity to the real system.

[1] https://www.tecquipment.com/es/

# B. Vibrating Strings Mathematical Model

Consider a string of length $L$, volumetric density $\rho$ and mass $M$, that oscillates in the $YZ$ plane under a constant tension $T$. Fig. B.1 shows a forces diagram on an infinitesimal length $dy$ of the string. This infinitesimal portion of the string has a mass $dm = \mu dy$, with $\mu = M/L$ the linear density of mass. Net tensions produced on the string are

$$F_y = T\cos(\alpha + d\alpha) - T\cos\alpha \ , \tag{B.1}$$

$$F_z = T\sin(\alpha + d\alpha) - T\sin\alpha \ . \tag{B.2}$$

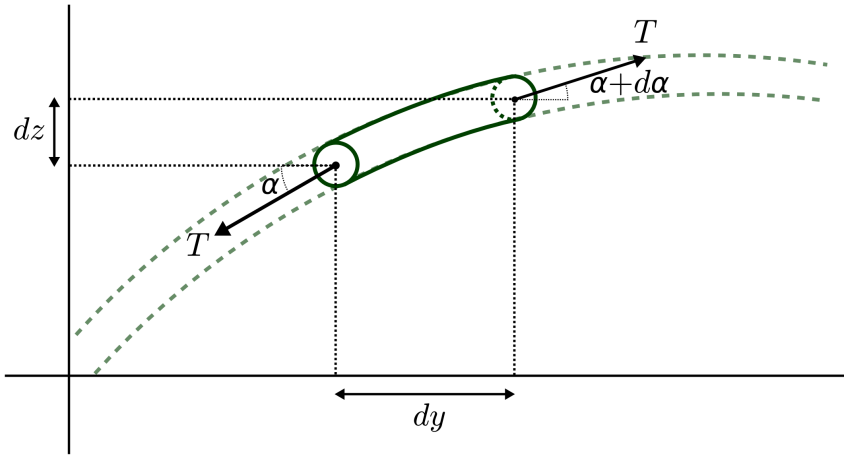



**Figure. B.1** In an infinitesimal portion of the string $dy$ appear two tensions, one at each end of the portion, so that under a small displacement assumption the horizontal net tension is null.

Under the assumption of a small displacement in the vertical direction, a first

order Taylor expansion of the forces gives

$$F_y = 0 \ , \tag{B.3}$$
$$F_z = T d\alpha \ . \tag{B.4}$$

Newton's Second Law gives, then

$$T d\alpha = dma \tag{B.5}$$
$$= (\mu dy)\frac{\partial^2 z}{\partial t^2} \ . \tag{B.6}$$

By relating the angle $\alpha$ with its $YZ$ components, taking derivatives and aproximating in Taylor's first order we obtain an equation for the infinitesimal angle

$$d\alpha = \frac{\partial^2 z}{\partial y^2} dy \ , \tag{B.7}$$

so as the equation that describes the wave motion is

$$\frac{\partial^2 z}{\partial y^2} = \frac{\mu}{T}\frac{\partial^2 z}{\partial t^2} \ , \tag{B.8}$$

which is the so-famous wave equation [136]. This equation describes the temporal evolution of a transversal wave propagating at a speed $v = \sqrt{T/\mu}$.

For a fixed-fixed string both ends are fixed, so that the displacement at these nodal points is zero. The temporal part of the solution to the wave equation can be written as a linear combination of normal modes

$$z(y,t) = \sum_{n=1}^{\infty} A_n \sin\left(\frac{n\pi y}{L}\right)\cos(\omega_n t)\, e^{-n\gamma t} \ , \tag{B.9}$$

where

$$\omega_n = \frac{n\pi}{L}\sqrt{\frac{T}{\mu}} \ . \tag{B.10}$$

Here, $\gamma$ is a damping coefficient. Fig. B.2 shows the vertical oscillation of the string as a function of the position $x$, for the first four normal modes. Each $n$ mode has $(n+1)$ fixed nodes (positions where there is no displacement) and $n$ anti-nodes (positions with maximum displacement).

Equation B.9 shows that the bigger the normal mode, the higher the damping factor. Eventually, all the $n > 1$ modes vanish and the only surviving term gives a "stationary" wave

$$z_1(y,t) = A_1 \sin\left(\frac{\pi y}{L}\right) \cos\left(\frac{\pi}{L}\sqrt{\frac{T}{\mu}}t\right) e^{-\gamma t} \quad . \tag{B.11}$$

Note that the amplitude of the perturbation eventually goes to zero as a consequence of its proper damping coefficient $-\gamma$.

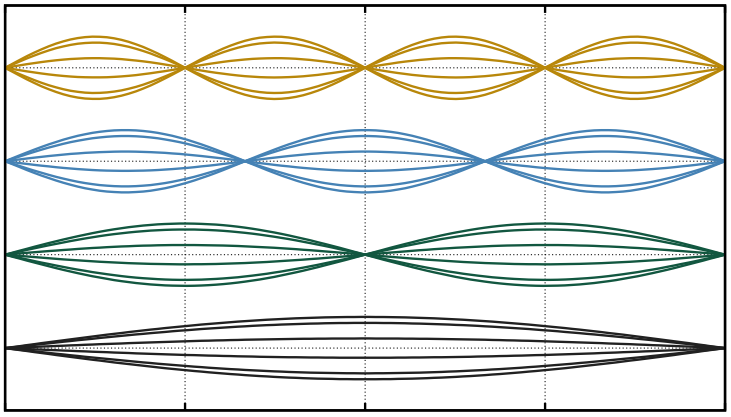


**Figure. B.2** First four normal modes of vibration for a fixed-fixed string. Each $n$ mode has $(n+1)$ points where the oscillation is zero. The wave length of mode $n$ is $\lambda_n = 2L/n$.

At a fixed position $y = L/2$ we have

$$z_1(t) = A_1 \cos(2\pi f_1 t) \quad , \tag{B.12}$$

where the fundamental frequency $f_1$ is given by

$$f_1 = \frac{1}{2Lr}\sqrt{\frac{T}{\pi\rho}} \quad , \tag{B.13}$$

where $r$ is the radius of the string and $\rho$ its volumetric density. If we select a control parameter (as could it be the tension, for example), by measuring the frequency $f$

of the string as a function of this parameter we may stablish a relationship of the kind

$$f = \alpha T^{\beta} \ , \tag{B.14}$$

so as for different tensions we may perform a least squares method to obtain the constants $\alpha$ and $\beta$ and, therefore, find the density of a string just knowing the length of the string and its radius, i.e.,

$$\rho = \frac{1}{4\pi(Lr\alpha)^2} \ . \tag{B.15}$$

# C. Servo Motor Mathematical Model

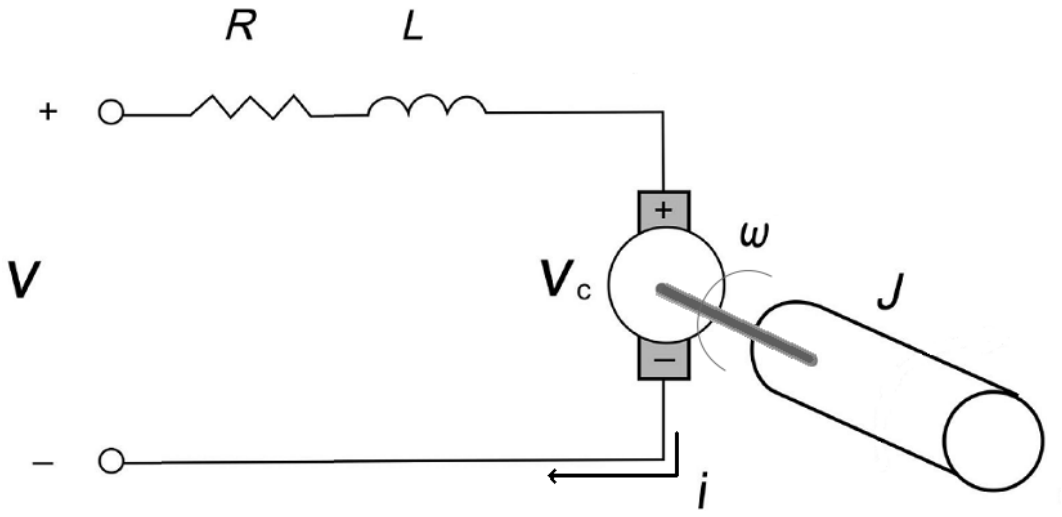



**Figure. C.1** Components of the remote servo motor lab.

The servo-motor system is DC motor with two separeted windings, the armature and field. When power is applied, the polarities of the energized field and the armature windings are misaligned, and the rotor will rotate until them are almost aligned. Figure C.1 shows a basic scheme to represent the equivalent electrical circuit.

In this regard, the voltages which can be applied are $e_a$, the armature voltage, and $e_f$, the field voltage. Lets assume that:

- The magnetic flux is proportional to the electrical current flowing across the excited winding.
- The torque is proportional to the magnetic flux and electrical current through the armatures winding.

Both relations can be expressed like:

$$\phi(t) = K_f i_f \tag{C.1}$$

Where $i_f$ is the field current, $\phi(t)$ is the magnetic flux, and $K_f$ is the constant that relates both magnitudes.

$$\mathrm{T}_m(t) = K_m \phi(t) i_a(t) \tag{C.2}$$

Where $i_a$ is the armature current, $T_m$ is the torque and $K_m$ is a constant value.

Maintaining constant the field and armature voltages the magnetic flux is also constant. Then, the torque can be obtained as a proportional relation with the field current as:

$$\mathrm{T}_m(t) = K_m K_f i_f i_a(t) = K_1 i_a(t) \tag{C.3}$$

Where $K_1$ is the torque constant, combination of the constants of both contributions to torque. If the magnetic flux is constant, the counter-electromotive force has only one contribution due to the change in the axis angle. This relation is expressed like:

$$\mathrm{e}_b(t) = K_b \frac{d\theta(t)}{t} = K_b \omega \tag{C.4}$$

Where $K_b$ is the counter-electromotive force constant. The circuit from Figure C.1 is an approach to the real model. Thus, the corresponding circuital equation can be expressed like:

$$\mathrm{e}_a(t) = L_a \frac{di_a(t)}{dt} + R_a i_a(t) + e_b(t) \tag{C.5}$$

Where $R_a$ and $L_a$ are the resistance and inductance of the armature. The resultant torque obtained using a common mechanics analysis result in three components: Friction torque, loading torque and total torque from motor. Next equation shows the relation between these three:

$$\mathrm{J}_m \frac{d^2\theta(t)}{dt^2} = T_m(t) - B_m \frac{d\theta(t)}{dt} - T_L \tag{C.6}$$

Where $J_m$ and $B_m$ are the moment of inertia and the viscous friction coefficient and $T_L$ is the torque associated with the load of the motor. Combining the gathered

expression of the torque and circuital analysis is obtained a relation between the output angle and the field voltage:

$$\mathrm{e}_a(t) - R_a i_a(t) = L_a \frac{di_a(t)}{dt} + K_b \frac{d\theta(t)}{dt} \tag{C.7}$$

$$\mathrm{J}_m \frac{d^2\theta(t)}{dt^2} + B_m \frac{d\theta(t)}{dt} = K_1 i_a(t) - T_L \tag{C.8}$$

Both equations represent the behavior of the system, although, choosing the parameters carefully it can be simplified. If we also assume that the motor has no load, the Laplace transform and the transfer function can be easily obtained like:

$$\mathrm{E}_a(s) - R_a I_a(s) = sL_a I_a(s) + sK_b\theta(s) \tag{C.9}$$

$$\mathrm{s}^2 J_m\theta(s) + sB_m\theta(s) = K_1 I_a(a) \tag{C.10}$$

$$+\frac{\theta(s)}{E_a(s)} = \frac{K_1}{s((L_aB_m + R_aJ_m)s + R_aB_m + K_1K_b)} \tag{C.11}$$

Where $e_a$ is the armature voltage, $e_f$ is the field voltage and $i_a$, $i_f$ their corresponding currents. $R_a$ and $L_a$ are the resistance and inductance of the armature. $K_b$ is the counter-electromotive force constant and $K_1$ is the torque constant. $J_m$ and $B_m$ are the moment of inertia and the viscous friction coefficient. $T_L$ is the torque associated with the load of the motor.

# D. Document: How to change a lab from Java to Java&Javascript

## D.1. First steps

### D.1.1. Locate your files

Any developer who wants to use the Java &Javascript enabled version to reuse a Java lab needs at least two thing:

- The original file, with extension .ejs, written in Java.
- A EjsS distribution with the Java & Javascript version enabled. To obtain it, the user can write to jacobo.saenz@bec.uned.es and ask for a copy.

Once the file is located and the EjsS editor is ready to be used, the next step is to open EjsS and the file.

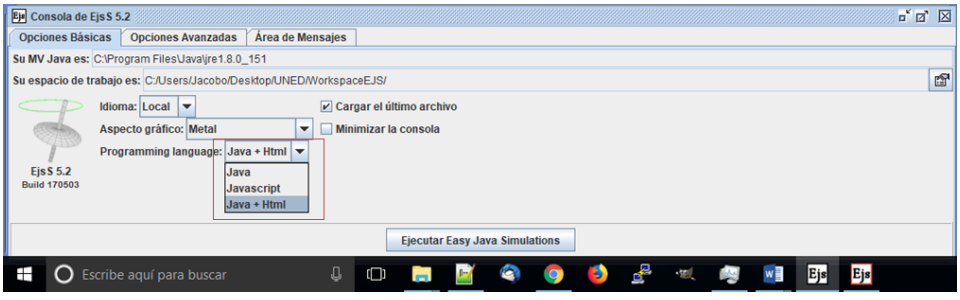



**Figure. D.1** How to run the new version of EjsS

### D.1.2. Open your file

First, EjsS must be initiated, EjsS will prompt the last enabled version opened in your computer. To open the Java & Javascript version the user must follow the next steps:

- The editor contains two windows: The main editor and the console. To open J&JS version the user need to change some console parameters in the *Main Options* tab.
- Inside this tab, the user will found a drop menu: *Programming language* (see Figure D.1). The *Java+HTML* option must be selected.
- The next step is run *Easy Java Simulations* by clicking the corresponding button in the console window.
- A new editor will be opened like the one in Figure D.2. Therefore, the developer is ready to open the .ejs file in the new editor.
- A soon as the developers loads the file a warning pop-up is shown, see Figure D.3. The answer to the question about to use other editor to open the file must be *No*.

Figure D.2 shows a ejs file opened with the new editor. At this point, the developer has to save the file.

Some error messages may be prompted in the console, as the file is in a different version is quite common.

### D.1.3. Save with new extension

The last step to do after changing the application or the GUI is to save the file using the right extension: **.ejsh**. To do that is recommended to use the *save as* button of the editor. Probably, the tool will ask if we want to write with this extension, we must anser *Yes*.

## D.2. Prevent problems and create the GUI

The tool can help with the change in the EjsS application, but the process is not immediate, the developer must make some changes to obtain no-errors and to prevent bad behaviors of the GUI. First step is to delete the sarlab element: the *Nucleo*. This element is not used at this point, then the best option is to delete it.

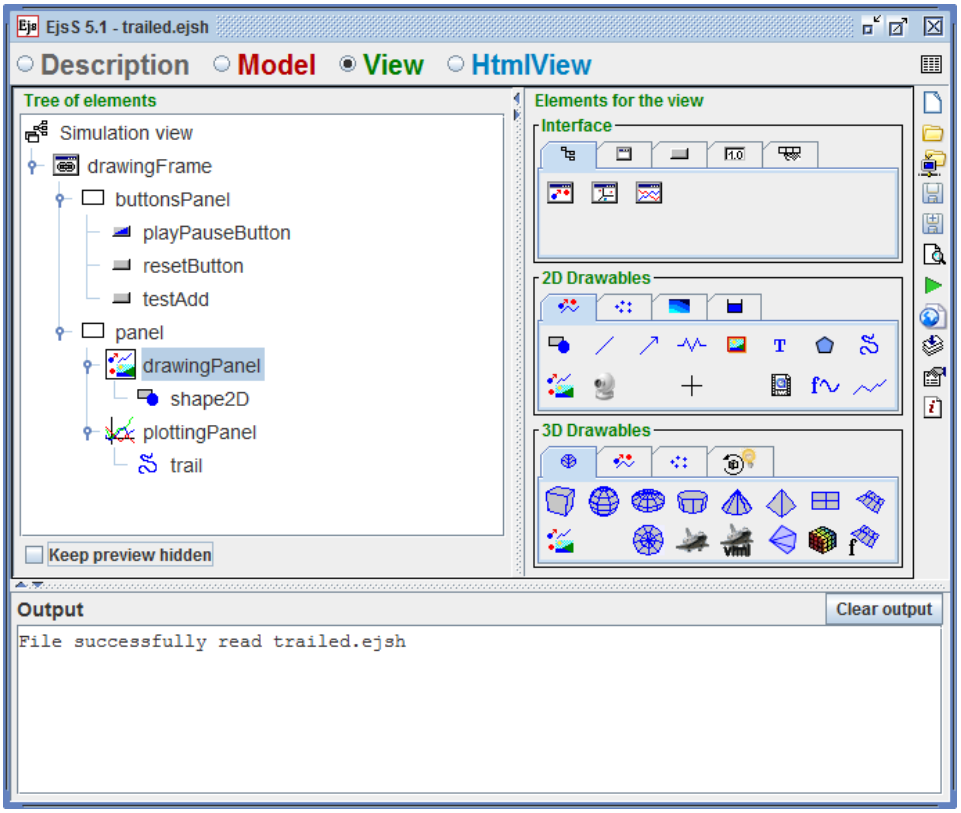



**Figure. D.2**

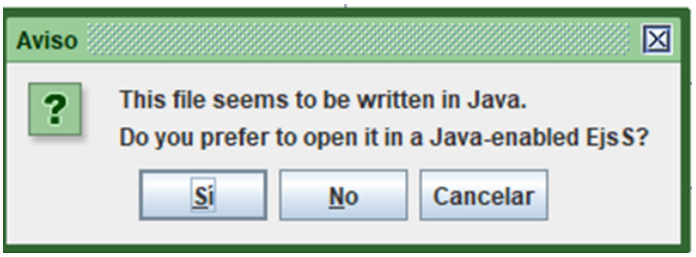



**Figure. D.3**

The next step is to add the SmartDeviceElement which will add additional functionalities to the application. Figure D.5 shows the element and the icon in the process to be added from the SoftwareLinks folder.

The SmartDeviceElement can be configured by double clicking it. Figure D.6 contains the configuration windows and some parameters. During the testing the IP must be *localhost*, and the port can be 2055, 8080 or any value higher than 1300 to

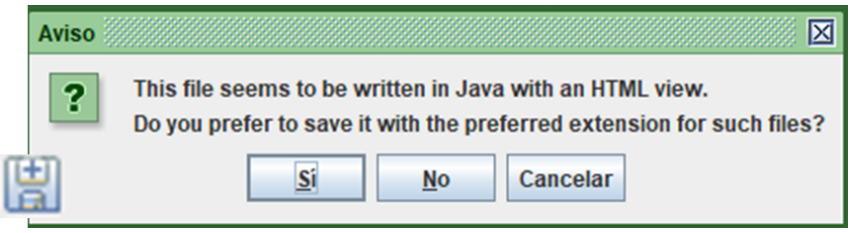



**Figure. D.4**

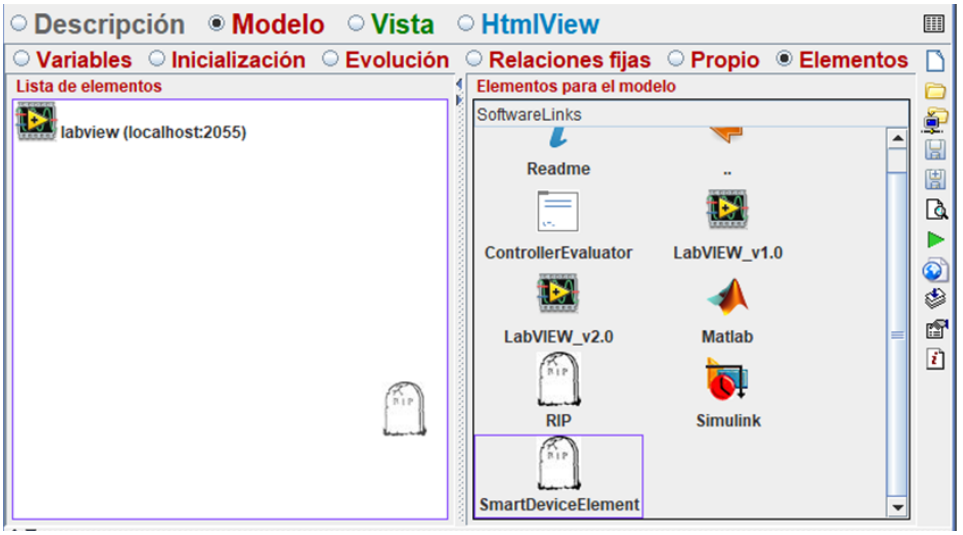



**Figure. D.5**

prevent access problems. Both values will be changed to use the ones selected by the developer.

## D.2.1. Reduce errors

In the best situation, the application will be runnable, and the developer can initialize the experiment. But usually some problems will appear regarding the call to the Java view. This view is going to be empty or will not be run, then, if possible the developer must reduce the calls in the code to the *_view* object. Other errors may be prompted, but usually are not directly related with the new editor.

In fact, _view calls are allowed, but some will produce errors. To _view to handle an interaction will not produce data, as _view is not in the user side. If these calls can be done in any other way, is a best practice to do it.

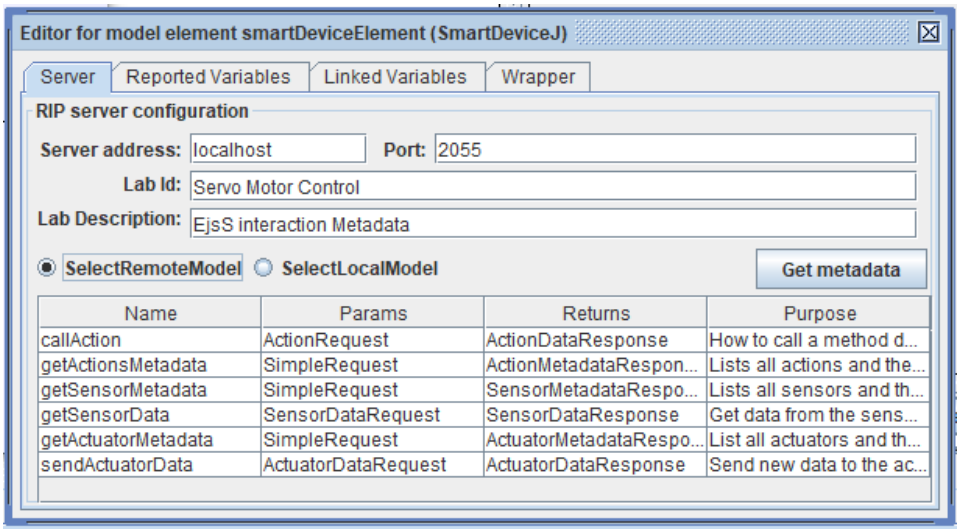



**Figure. D.6**

## D.2.2. Create you GUI

The first approach is to make our GUI directly in the main editor from the last steps. In this situation, the developer can create an HTML view just considering a couple of things:

- The developer can build and test its GUI by dragging and dropping the desired elements to the tree structure of the HTML-view.
- Is recommended to reduce the dynamic calculus, it means, to use variables and not formulas to calculate values of fields inside the HTML view element. For example coordinates changes or similar must be done in the code, and not in the view.
- Red field of the view elements (actions field like *on_Release*) must contain Java code, remember that the model will carry out with all the calculations of the view.
- All the variables must be initialized with a default value and the minimum and maximum values of the GUI must be specified to prevent not allowed values.

The HTML GUI created using this methodology will be the student interface, then, is important to test and double check all the changes in the web browser.

### D.2.3. Get ready to use it

All the test done are local, which mean that both sides are in the same machine. To use it in a networked environment the developer need to change IP and port in the configuration windows of the element. Once it is done, it can be placed in the server to do more testings. In the server side how the model in run in on charge of the developer, but, if the model in running, it will accept all the incoming connections.

If the GUI needs later changes, the Editor can be opened in the remote machine to modify the HTML view. This is the second approach to create or edit interfaces. If the model is well designed and contains all the methods the model can be maintained in the remote location. If not, the model will need to be replaced by the new one.

# E. Full Metadata

This Appendix contains the full metadata file which is used inside the EjsS models. It is defined using the JSON data format and follows the Smart Device Specification.

```
{
    "apiVersion": "1.0.0",
    "swaggerVersion": "",
    "basePath": "http://127.0.0.1:8800",
    "info": {
        "title": "Servo Motor Control",
        "description": "EjsS interaction Metadata",
        "contact": "jacobo.saenz@bec.uned.es",
        "license": "Apache 2.0",
        "licenseUrl": "http://www.apache.org/licenses/LICENSE-2.0.html"
    },
    "authorizations": {},
    "concurrency": {
        "interactionMode": "synchronous",
        "concurrencyScheme": "roles",
        "roleSelectionMechanism": ["race", "fixed role"],
        "roles": [
            {
                "role": "controller",
                "selectionMechanism": ["race"]
            }
        ]
    },
    "apis": [
        {
            "protocol": "websocket",
            "produces": [
                "application/json"
            ],
            "operations": [
                {
                    "method": "Send",
                    "nickname": "getSensorMetadata",
                    "summary": "Lists all sensors and their metadata",
                    "type": "SensorMetadataResponse",
                    "parameters": [
                        {
                            "name": "message",
                            "description": "The payload for the getSensorMetadata service",
                            "required": true,
                            "paramType": "message",
                            "type": "SimpleRequest",
                            "allowMultiple": false
                        }
                    ]
```

```
                    "authorizations": {},
                    "responseMessages": [
                        {
                            "code": 402,
                            "message": "Too many users"
                        },
                        {
                            "code": 404,
                            "message": "No sensors found"
                        },
                        {
                            "code": 405,
                            "message": "Method not allowed. The requested method is not allowed by
                                this server."
                        },
                        {
                            "code": 422,
                            "message": "The request body is unprocessable"
                        }
                    ]
                },

                {
                    "method": "Send",
                    "nickname": "getSensorData",
                    "summary": "Get data from the sensor with the given sensor identifier",
                    "type": "SensorDataResponse",
                    "parameters": [
                        {
                            "name": "message",
                            "description": "Returns captured values",
                            "required": true,
                            "type": "SensorDataRequest",
                            "paramType": "message",
                            "allowMultiple": false
                        }
                    ],
                    "responseMessages": [
                        {
                            "code": 401,
                            "message": "Unauthorised access. The authentication token is not valid"
                        },
                        {
                            "code": 402,
                            "message": "Too many users"
                        },
                        {
                            "code": 404,
                            "message": "No sensors found"
                        },
                        {
                            "code": 405,
                            "message": "Method not allowed. The requested method is not allowed by
                                this server."
                        },
                        {
                            "code": 422,
                            "message": "The request body is unprocessable"
                        }
                    ]
                }
            ]
        },
        {
            "protocol": "websocket",
            "produces": [
                "application/json"
            ],
```

```
            "operations": [
                {
                    "method": "Send",
                    "nickname": "getActuatorMetadata",
                    "summary": "List all actuators and their metadata",
                    "type": "ActuatorMetadataResponse",
                    "parameters": [
                        {
                            "name": "message",
                            "description": "The payload for the getActuatorMetadata service",
                            "required": true,
                            "paramType": "message",
                            "type": "SimpleRequest",
                            "allowMultiple": false
                        }
                    ],
                    "responseMessages": [
                        {
                            "code": 404,
                            "message": "No actuators found"
                        },
                        {
                            "code": 405,
                            "message": "Method not allowed. The requested method is not allowed by
                                this server."
                        },
                        {
                            "code": 422,
                            "message": "The request body is unprocessable"
                        }
                    ],
                    "authorizations": {}
                },
                {
                    "method": "Send",
                    "summary": "Send new data to the actuator with the given actuator identifier",
                    "notes": "The parameters go into a JSON object send over the websocket",
                    "type": "ActuatorDataResponse",
                    "nickname": "sendActuatorData",
                    "parameters": [
                        {
                            "name": "message",
                            "description": "Provides value for actuator control",
                            "required": true,
                            "type": "ActuatorDataRequest",
                            "paramType": "message",
                            "allowMultiple": false
                        }
                    ],
                    "responseMessages": [
                        {
                            "code": 401,
                            "message": "Unauthorised access. The authentication token is not valid"
                        },
                        {
                            "code": 402,
                            "message": "Too many users"
                        },
                        {
                            "code": 404,
                            "message": "No actuator not found"
                        },
                        {
                            "code": 405,
                            "message": "Method not allowed. The requested method is not allowed by
                                this server."
                        },
                        {
                            "code": 422,
                            "message": "The request body is unprocessable"
                        }
                    ]
                }
            ]
        }
    ],
```

```
    "models": {
        "Sensor": {
            "id": "Sensor",
            "required": [
                "sensorId", "fullName"
            ],
            "properties": {
                "sensorId": {
                    "type": "string"
                },
                "fullName": {
                    "type": "string"
                },
                "description": {
                    "type": "string"
                },
                "webSocketType": {
                    "type": "string",
                    "description": "the type of websocket. Websockets can either be binary or
                         textual.",
                    "enum": [
                        "text",
                        "binary"
                    ],
                    "defaultValue": "text"
                },
                "singleWebSocketRecommended": {
                    "type": "boolean",
                    "description": "If this field is set to true it means that the smart device
                         expects that a client opens a dedicated websocket for to read from this
                         value",
                    "defaultValue": false
                },
                "produces": {
                    "type": "string",
                    "description": "The mime-type of the data that is produced by this sensor. A
                         list of mime types can be found at http://en.wikipedia.org/wiki/
                         Internet_media_type",
                    "defaultValue": "application/json"
                },
                "values": {
                    "type": "array",
                    "items": {
                        "$ref": "Value"
                    }
                },
                "configuration": {
                    "type": "array",
                    "description": "The configuration consists of an array of JSON objects that
                         consist of parameter and type",
                    "items": {
                        "$ref": "ConfigurationMetadataItem"
                    }
                },
                "accessMode": {
                    "type": "AccessMode"
                }
            }
        },
        "Action":{
        "id": "Action",
            "required": [
                "actionId", "fullName","params"
            ],
            "properties": {
                "sensorId": {
                    "type": "string"
                },
                "fullName": {
                    "type": "string"
                },
                "description": {
                    "type": "string"
                },
```

```
                    "webSocketType": {
                        "type": "string",
                        "description": "the type of websocket. Websockets can either be binary or
                             textual.",
                        "enum": [
                            "text",
                            "binary"
                        ],
                        "defaultValue": "text"
                    },
                    "singleWebSocketRecommended": {
                        "type": "boolean",
                        "description": "If this field is set to true it means that the smart device
                             expects that a client opens a dedicated websocket for to read from this
                             value",
                        "defaultValue": false
                    },
                    "produces": {
                        "type": "string",
                        "description": "The mime-type of the data that is produced by this sensor. A
                             list of mime types can be found at http://en.wikipedia.org/wiki/
                             Internet_media_type",
                        "defaultValue": "application/json"
                    },
                    "params": {
                        "type": "array",
                        "items": {
                            "$ref": "Value"
                        }
                    },
                    "accessMode": {
                        "type": "AccessMode"
                    }
                }
        }

        "Value": {
                "id": "Value",
                "required": [
                    "name"
                ],
                "properties": {
                    "name": {
                        "type": "string"
                    },
                    "unit": {
                        "type": "string"
                    },
                    "type": {
                        "type": "string",
                        "description": "The data type of this value",
                        "enum": [
                            "integer",
                            "long",
                            "float",
                            "double",
                            "string",
                            "byte",
                            "boolean",
                            "date",
                            "dateTime",
                            "object",
                            "array",
                            "any",
                            "binary"
                        ]
                    },
                    "rangeMinimum": {
                        "type": "number",
                        "format": "double"
                    },
                    "rangeMaximum": {
                        "type": "number",
                        "format": "double"
                    },
```

```
                    "rangeStep": {
                        "type": "number",
                        "format": "double"
                    },
                    "lastMeasured": {
                        "type": "date-time"
                    },
                    "updateFrequency": {
                        "type": "number",
                        "description": "The frequency in Hertz of which the sensor value updates",
                        "format": "int"
                    }
                }
            },
            "ConfigurationMetadataItem": {
                "id": "ConfigurationMetadataItem",
                "required": [
                    "parameter", "type"
                ],
                "properties": {
                    "parameter": {
                        "type": "string",
                        "description": "The name of the configuration parameter"
                    },
                    "description": {
                        "type": "string",
                        "description": "This field can provide some more information on how this
                             parameter should be used."
                    },
                    "type": {
                        "type": "string",
                        "description": "The data type of that this configuration parameters expects, e.g
                             . number or string",
                        "enum": [
                            "integer",
                            "long",
                            "float",
                            "double",
                            "string",
                            "byte",
                            "boolean",
                            "date",
                            "dateTime",
                            "object",
                            "array",
                            "any",
                            "binary"
                        ]
                    },
                    "items": {
                        "type": "string",
                        "description": "This field should only be used when the type is 'array'. It
                             describes which types are present within the array",
                        "enum": [
                            "integer",
                            "long",
                            "float",
                            "double",
                            "string",
                            "byte",
                            "boolean",
                            "date",
                            "dateTime",
                            "object",
                            "any",
                            "binary"
                        ]
                    }
                }
            },
```

```
        "AccessMode": {
            "id": "AccessMode",
            "properties": {
                "type": {
                    "type": "string",
                    "enum": [
                        "push",
                        "pull",
                        "stream"
                    ]
                },
                "nominalUpdateInterval": {
                    "type": "number",
                    "format": "float"
                },
                "userModifiableFrequency": {
                    "type": "boolean",
                    "defaultValue": false
                }
            }
        },
        "SimpleRequest": {
            "id": "SimpleRequest",
            "required": [
                "method"
            ],
            "properties": {
                "authToken": {
                    "type": "string"
                },
                "method": {
                    "type": "string",
                    "description": "The method should be equal to the nickname of one of the
                        provided services."
                }
            }
        },
        "SensorMetadataResponse": {
            "id": "SensorMetadataResponse",
            "required": [
                "method", "sensors"
            ],
            "properties": {
                "method": {
                    "type": "string",
                    "description": "The method should be equal to the nickname of one of the
                        provided services."
                },
                "sensors": {
                    "type": "array",
                    "items": {
                        "$ref": "Sensor"
                    }
                }
            }
        },
        "SensorDataRequest": {
            "id": "SensorDataRequest",
            "required": ["method", "sensorId"
            ],
            "properties": {
                "method": {
                    "type": "string",
                    "description": "The method should be equal to the nickname of one of the
                        provided services."
                },
                "sensorId": {
                    "type": "string"
                },
                "configuration": {
                    "type": "array",
                    "items": {
                        "$ref": "ConfigurationItem"
                    }
                },
```

```
                    "accessRole": {
                        "type": "string",
                        "description": "This field contains one of the roles defined in the concurrency
                             roles list. If accessRole is not defined, the controller role is assumed."
                    }
                }
            },
            "ConfigurationItem": {
                "id": "ConfigurationItem",
                "required": [
                    "parameter", "value"
                ],
                "properties": {
                    "parameter": {
                        "type": "string",
                        "description": "The name of the configuration parameter"
                    },
                    "value": {
                        "type": "any",
                        "description": "The value to set the configuration parameter to. The type should
                              equal the type given in the metadata for this sensor."
                    }
                }
            },
            "SensorDataResponse": {
                "id": "SensorDataResponse",
                "required": [
                    "method", "sensorId"
                ],
                "properties": {
                    "method": {
                        "type": "string",
                        "description": "The method should be equal to the nickname of one of the
                             provided services."
                    },
                    "sensorId": {
                        "type": "string"
                    },
                    "accessRole": {
                        "type": "string",
                        "description": "This field contains one of the roles defined in the concurrency
                             roles list. If no roles are defined controller is returned. If the
                             observer is returned, the observerMode field will be available with extra
                             info on the status of the lab."
                    },
                    "responseData": {
                        "type": "SensorResponseData",
                        "description": "The data as measured by this sensor"
                    },
                    "payload": {
                        "type": "any",
                        "description": "This optional payload field can contain any JSON object that
                             provides extra information on this sensor or the current measurement."
                    },
                    "observerMode": {
                        "type": "ObserverMode",
                        "description": "This field is only available if the accessRole field returns
                             observer."
                    }
                }
            },
            "SensorResponseData": {
                "id": "SensorResponseData",
                "required": [],
                "properties": {
                    "valueNames": {
                        "type": "array",
                        "description": "An ordered array with all the value names of this sensor. The
                             same order will be applied to the data array and lastMeasured array.",
                        "items": {
                            "type": "string"
                        }
                    },
```

```
                "data": {
                    "type": "array",
                    "description": "An ordered array with all the data values of this sensor. Each
                        data element in the array should be ordered in the same position of its
                        corresponding value elements in the values array.",
                    "items": {
                        "type": "any"
                    }
                },
                "lastMeasured": {
                    "type": "array",
                    "description": "An ordered array with all the data values of this sensor. Each
                        data element in the array should be ordered in the same position of its
                        corresponding value elements in the values array.",
                    "items": {
                        "type": "date-time"
                    }
                }
            }
        },
        "Actuator": {
            "id": "Actuator",
            "required": [
                "actuatorId", "fullName"
            ],
            "properties": {
                "actuatorId": {
                    "type": "string"
                },
                "fullName": {
                    "type": "string"
                },
                "description": {
                    "type": "string"
                },
                "webSocketType": {
                    "type": "string",
                    "description": "the type of websocket. Websockets can either be binary or
                        textual.",
                    "enum": [
                        "text",
                        "binary"
                    ],
                    "defaultValue": "text"
                },
                "singleWebSocketRecommended": {
                    "type": "boolean",
                    "description": "If this field is set to true it means that the smart device
                        expects that a client opens a dedicated websocket for to read from this
                        value",
                    "defaultValue": false
                },
                "consumes": {
                    "type": "string",
                    "description": "The mime-type of the data that is consumed by this actuator. A
                        list of mime types can be found at http://en.wikipedia.org/wiki/
                        Internet_media_type",
                    "defaultValue": "application/json"
                },
                "produces": {
                    "type": "string",
                    "description": "The mime-type of the data that is produced by this actuator. A
                        list of mime types can be found at http://en.wikipedia.org/wiki/
                        Internet_media_type",
                    "defaultValue": "application/json"
                },
                "values": {
                    "type": "array",
                    "items": {
                        "$ref": "Value"
                    }
                },
                "configuration": {
                    "type": "array",
                    "description": "The configuration consists of an array of JSON objects that
                        consist of parameter and type",
                    "items": {
                        "$ref": "ConfigurationMetadataItem"
                    }
                },
```

```
                "accessMode": {
                    "type": "AccessMode"
                }
            }
        },
        "ActuatorMetadataResponse": {
            "id": "ActuatorMetadataResponse",
            "required": [
                "method", "actuators"
            ],
            "properties": {
                "method": {
                    "type": "string",
                    "description": "The method should be equal to the nickname of one of the
                        provided services."
                },
                "actuators": {
                    "type": "array",
                    "items": {
                        "$ref": "Actuator"
                    },
                    "description": "The list of actuator metadata elements"
                }
            }
        },
        "ActuatorDataRequest": {
            "id": "ActuatorDataRequest",
            "required": [
                "method", "actuatorId"
            ],
            "properties": {
                "authToken": {
                    "type": "string"
                },
                "method": {
                    "type": "string",
                    "description": "The method should be equal to the nickname of one of the
                        provided services."
                },
                "actuatorId": {
                    "type": "string"
                },
                "valueNames": {
                    "type": "array",
                    "description": "An ordered array with all the value names of this sensor. The
                        same order will be applied to the data array and lastMeasured array.",
                    "items": {
                        "type": "string"
                    }
                },
                "data": {
                    "type": "array",
                    "description": "An ordered array with all the data values of this sensor. Each
                        data element in the array should be ordered in the same position of its
                        corresponding value elements in the valueNames array.",
                    "items": {
                        "type": "any"
                    }
                },
                "configuration": {
                    "type": "array",
                    "items": {
                        "$ref": "ConfigurationItem"
                    }
                },
                "accessRole": {
                    "type": "string",
                    "description": "This field contains one of the roles defined in the concurrency
                        roles list. If accessRole is not defined the controller role is assumed."
                }
            }
        },
```

```
        "ActuatorDataResponse": {
            "id": "ActuatorDataResponse",
            "required": [
                "method"
            ],
            "properties": {
                "method": {
                    "type": "string",
                    "description": "The method should be equal to the nickname of one of the
                        provided services."
                },
                "lastMeasured": {
                    "type": "date-time"
                },
                "accessRole": {
                    "type": "string",
                    "description": "This field contains one of the roles defined in the concurrency
                        roles list. If no roles are defined controller is returned. If the
                        observer is returned, the observerMode field will be available with extra
                        info on the status of the lab."
                },
                "payload": {
                    "type": "any",
                    "description": "The payload can be useful for describing a result that is
                        returned, for instance by using the SensorResponseData model. Since
                        results can differ from acknowledgements to result data, the field is
                        optional and can contain any JSON object."
                },
                "observerMode": {
                    "type": "ObserverMode",
                    "description": "This field is only available if the accessRole field returns
                        observer."
                }
            }
        },
        "ObserverMode": {
            "id": "ObserverMode",
            "required": [],
            "properties": {
                "queueSize": {
                    "type": "integer",
                    "description": "Provides the length of the user waiting queue that want to get
                        control of the lab"
                },
                "queuePosition": {
                    "type": "integer",
                    "description": "Provides the position of the client who made this call in the
                        user waiting queue. This value should be positive and smaller or equal to
                        queueSize."
                },
                "estimatedTimeUntilControl": {
                    "type": "integer",
                    "description": "The estimated waiting time from now on until the client will get
                         controllerMode access. The time is expressed in seconds."
                }
            }
        }
    }
```

# Bibliography


[1] F. A. Silva. It innovative practices in secondary schools: Remote experiments [book news]. *IEEE Industrial Electronics Magazine*, 9(1):92–93, March 2015.

[2] A. Tait. Planning student support for open and distance learning. *Open Learning: The Journal of Open, Distance and e-Learning*, 15(3):287–299, 2000.

[3] J. Petrović, D. Tralić, and P. Pale. Learning benefits of online formative self-assessments. In *2015 57th International Symposium ELMAR (ELMAR)*, pages 243–246, Sept 2015.

[4] J. L. M. Núñez, E. T. Caro, J. S. López, and P. M. García. Education quality enhancement through open education adaptation. In *2014 IEEE Frontiers in Education Conference (FIE) Proceedings*, pages 1–4, Oct 2014.

[5] C. C. W. Hulls, A. J. Neale, B. N. Komalo, V. Petrov, and D. J. Brush. Interactive online tutorial assistance for a first programming course. *IEEE Transactions on Education*, 48(4):719–728, Nov 2005.

[6] R. W. Reid, B. E. Gardner, and R. E. Bitten. Use of electronic media for interactive space systems education. *IEEE Aerospace and Electronic Systems Magazine*, 24(2):34–43, Feb 2009.

[7] H. Q. Yu, C. Pedrinaci, S. Dietze, and J. Domingue. Using linked data to annotate and search educational video resources for supporting distance learning. *IEEE Transactions on Learning Technologies*, 5(2):130–142, April 2012.

[8] L. Morgado, B. Fonseca, P. Martins, H. Paredes, G. Cruz, A. M. Maia, R. Nunes, and A. Santos. Social networks, microblogging, virtual worlds,

and web 2.0 in the teaching of programming techniques for software engineering: A trial combining collaboration and social interaction beyond college. In *Proceedings of the 2012 IEEE Global Engineering Education Conference (EDUCON)*, pages 1–7, April 2012.

[9] E. Ye, C. Liu, and J. A. Polack-Wahl. Enhancing software engineering education using teaching aids in 3-d online virtual worlds. In *2007 37th Annual Frontiers In Education Conference - Global Engineering: Knowledge Without Borders, Opportunities Without Passports*, pages T1E–8–T1E–13, Oct 2007.

[10] J. M. Pullen. Applicability of internet video in distance education for engineering. In *31st Annual Frontiers in Education Conference. Impact on Engineering and Science Education. Conference Proceedings (Cat. No.01CH37193)*, volume 1, pages T2F–14, Oct 2001.

[11] Y. Lai, S. S. Young, and N. Huang. A preliminary study of producing multimedia online videos for ubiquitous learning on moocs. In *2015 8th International Conference on Ubi-Media Computing (UMEDIA)*, pages 295–297, Aug 2015.

[12] J. P. Mathews, N. Haughton, S. Pisupati, A. W. Scaroni, and D. DiBiase. For an online course encompassing "traditional campus students": how, where, and when students work and engage with the course material. In *34th Annual Frontiers in Education, 2004. FIE 2004.*, pages F1D–12, Oct 2004.

[13] T. Rekha Liyanagunawardena and S. A. Williams. Elderly learners and massive open online courses: A review. *Interact J Med Res*, 5(1):e1, Jan 2016.

[14] A. Molinari. Learning management systems and the integration with social media services: a case study. In *Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2017*, pages 775–781. ACM, 2017.

[15] M. Hori, S. Ono, S. Kobayashi, K. Yamaji, T. Kita, and T. Yamada. Fusion of e-textbooks, learning management systems, and social networking sites: A mash-up development. In *International Conference on Genetic and Evolutionary Computing*, pages 377–386. Springer, 2015.

[16] D. Bitzer, P. Braunfeld, and W. Lichtenberger. Plato: An automatic teaching device. *IRE Transactions on Education*, 4(4):157–161, 1961.

[17] S. Dormido, J. Sanchez, H. Vargas, L. de la Torre, and R. Heradio. *Using remote labs in education: two little ducks in remote experimentation*, chapter UNED Labs: A Network of Virtual and Remote Laboratories, pages 253–270. Deusto Digital, 2012.

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

[19] Abul K. M. Azad, M. E. Auer, and V. J. Harward. *Internet Accessible Remote Laboratories: Scalable E-Learning Tools for Engineering and Science Disciplines*. IGI Global, Hershey, PA, USA, 1st edition, 2011.

[20] M. S. Shur T. A. Fjeldly. *Lab on the Web : running real electronics experiments via the Internet*. Wiley-IEEE Press, Hoboken, NJ, USA, 1st edition, 2003.

[21] M. Guinaldo, L. de la Torre, Ruben Heradio, and Sebastian Dormido. A virtual and remote control laboratory in moodle: The ball and beam system. In *10th IFAC Symposium on Advances in Control Education*, 2013.

[22] M. P. Kazmierkowski and M. Liserre. Advances on remote laboratories and e-learning experiences (gomes, l. and garcia-zubia, j., eds.) [book news]. *IEEE Industrial Electronics Magazine*, 2(2):45–46, June 2008.

[23] J. Ma and J. V. Nickerson. Hands-on, simulated, and remote laboratories: A comparative literature review. *ACM Computing Surveys*, 38, 2006.

[24] S. Abu Shanab, S. Odeh, R. Hodrob, and M. Anabtawi. Augmented reality internet labs versus hands-on and virtual labs: A comparative study. In *Proceedings of 2012 International Conference on Interactive Mobile and Computer Aided Learning (IMCL)*, pages 17–21, Nov 2012.

[25] Z. Nedic, J. Machotka, and A. Nafalski. Remote laboratories versus virtual and real laboratories. In *Frontiers in Education Conference*, volume 1, pages T3E–1–T3E–6 Vol.1, Westminster, CO, USA, Nov 2003.

[26] M. F. Aburdene, E. J. Mastascusa, and R. Massengale. A proposal for a remotely shared control systems laboratory. In *Proceedings Frontiers in Education Twenty-First Annual Conference. Engineering Education in a New World Order*, pages 589–592, Sept 1991.

[27] N. Ivaki and F. Araujo. Fault-tolerant bi-directional communications in web-based applications. In *2014 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, pages 833–836, Dec 2014.

[28] S. Agarwal. Real-time web application roadblock: Performance penalty of html sockets. In *2012 IEEE International Conference on Communications (ICC)*, pages 1225–1229, June 2012.

[29] V. Herwig, R. Fischer, and P. Braun. Assessment of rest and websocket in regards to their energy consumption for mobile applications. In *2015 IEEE 8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, volume 1, pages 342–347, Sept 2015.

[30] D. Skvorc, M. Horvat, and S. Srbljic. Performance evaluation of websocket protocol for implementation of full-duplex web streams. In *2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 1003–1008, May 2014.

[31] J. Chacon, H. Vargas, G. Farias Castro, J. Sanchez Moreno, and S. Dormido. EJS, JIL Server and LabVIEW: An architecture for rapid developments of remote labs. *IEEE Transactions on Learning Technologies*, 8, 2015.

[32] K. Alexander, C. Lee, and S. Chai. Declarative policy support for cloud application orchestration. In *2017 19th International Conference on Advanced Communication Technology (ICACT)*, pages 102–104, Feb 2017.

[33] A. Tomono, M. Uehara, and Y. Shimada. Improvement and evaluation of a method to manage multiple types of logs. In *2011 IEEE Workshops of International Conference on Advanced Information Networking and Applications*, pages 601–606, March 2011.

[34] P. Baumann, E. Hirschorn, J. Maso, V. Merticariu, and D. Misev. All in one: Encoding spatio-temporal big data in xml, json, and rdf without information loss. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 1–10, Dec 2017.

[35] M. Ganzha, M. Paprzycki, W. Pawłowski, P. Szmeja, K. Wasielewska, and C. E. Palau. From implicit semantics towards ontologies — practical considerations from the inter-iot perspective. In *2017 14th IEEE Annual Consumer Communications Networking Conference (CCNC)*, pages 59–64, Jan 2017.

[36] M. J. McGrath and J. Delaney. An extensible framework for the management of remote sensor data. In *SENSORS, 2011 IEEE*, pages 1712–1715, Oct 2011.

[37] T. Inoue, H. Asakura, H. Sato, and N. Takahashi. Key roles of session state: Not against rest architectural style. In *2010 IEEE 34th Annual Computer Software and Applications Conference*, pages 171–178, July 2010.

[38] X. Feng, J. Shen, and Y. Fan. Rest: An alternative to rpc for web services architecture. In *2009 First International Conference on Future Information Networks*, pages 7–10, Oct 2009.

[39] Y. Yan, Y. Liang, and X. Du. Controlling remote instruments using web services for online experiment systems. In *IEEE International Conference on Web Services (ICWS'05)*, page 732, July 2005.

[40] S. Kumari and S. K. Rath. Performance comparison of soap and rest based web services for enterprise application integration. In *2015 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 1656–1660, Aug 2015.

[41] B. Upadhyaya, Y. Zou, H. Xiao, J. Ng, and A. Lau. Migration of soap-based services to restful services. In *2011 13th IEEE International Symposium on Web Systems Evolution (WSE)*, pages 105–114, Sept 2011.

[42] S. Malik and D. Kim. A comparison of restful vs. soap web services in actuator networks. In *2017 Ninth International Conference on Ubiquitous and Future Networks (ICUFN)*, pages 753–755, July 2017.

[43] C. L. Chamas, D. Cordeiro, and M. M. Eler. Comparing rest, soap, socket and grpc in computation offloading of mobile applications: An energy cost analysis. In *2017 IEEE 9th Latin-American Conference on Communications (LATINCOM)*, pages 1–6, Nov 2017.

[44] M. W. Goldberg. Calos: First results from an experiment in computer-aided learning for operating systems. *SIGCSE Bull.*, 29(1):48–52, March 1997.

[45] M. A. Stegawski and R. Schaumann. A new virtual-instrumentation-based experimenting environment for undergraduate laboratories with application in research and manufacturing. In *IEEE Instrumentation and Measurement Technology Conference Sensing, Processing, Networking. IMTC Proceedings*, volume 2, pages 1418–1421 vol.2, May 1997.

[46] S. E. Poindexter and B. S. Heck. Using the web in your courses: the how-to's and the why's. In *Proceedings of the 1998 American Control Conference. ACC (IEEE Cat. No.98CH36207)*, volume 2, pages 1299–1303 vol.2, June 1998.

[47] Y. Piguet and D. Gillet. Java-based remote experimentation for control algorithms prototyping. In *Proceedings of the 1999 American Control Conference (Cat. No. 99CH36251)*, volume 2, pages 1465–1469 vol.2, June 1999.

[48] A. Bhandari and M. H. Shor. Access to an instructional control laboratory experiment through the world wide web. In *Proceedings of the 1998 American Control Conference. ACC (IEEE Cat. No.98CH36207)*, volume 2, pages 1319–1325 vol.2, June 1998.

[49] J. Sanchez, F. Morilla, S. Dormido, J. Aranda, and P. Ruiperez. Virtual and remote control labs using java: a qualitative approach. *IEEE Control Systems*, 22(2):8–20, Apr 2002.

[50] M. J. Moron, J. R. Luque, A. Gomez-Jaime, E. Casilari, and A. Diaz-Estrella. Prototyping of a remote monitoring system for a medical personal area network using python. In *2009 3rd International Conference on Pervasive Computing Technologies for Healthcare*, pages 1–5, April 2009.

[51] L. F. Z. Rivera, M. M. Larrondo-Petrie, and L. Ribeiro Da Silva. Implementation of cloud-based smart adaptive remote laboratories for education. In *2017 IEEE Frontiers in Education Conference (FIE)*, pages 1–5, Oct 2017.

[52] E. Besada-Portas, J. A. Lopez-Orozco, L. de la Torre, and J. M. de la Cruz. Remote control laboratory using ejs applets and twincat programmable logic controllers. *IEEE Transactions on Education*, 56(2):156–164, May 2013.

[53] A. Gil, A. Peidró, Ó. Reinoso, and J. M. Marín. Implementation and assessment of a virtual laboratory of parallel robots developed for engineering students. *IEEE Transactions on Education*, 57(2):92–98, May 2014.

[54] Wolfgang C., F. Esquembre, and L. Barbato. Open source physics. *Science*, 334(6059):1077–1078, 2011.

[55] A. Visioli and F. Pasini. A virtual laboratory for the learning of process controllers design. *IFAC Proceedings Volumes*, 39(6):458 – 462, 2006. 7th IFAC Symposium on Advances in Control Education.

[56] P. Casals-Torrens. Virtual laboratory for learning asynchronous motors in engineering degrees. *IEEE Revista Iberoamericana de Tecnologias del Aprendizaje*, 8(2):71–76, May 2013.

[57] R. Heradio, L. de la Torre, J. Sanchez, and S. Dormido. Making EJS applications at the OSP digital library available from Moodle. In *International Conference on Remote Engineering and Virtual Instrumentation*, pages 112–116, Porto, Portugal, Feb 2014.

[58] H. Vargas, J. Sánchez-Moreno, S. Dormido, C. Salzmann, D. Gillet, and F. Esquembre. Web-enabled remote scientific environments. *Computing in Science and Engineering*, 11:36–46, 2009.

[59] G. Farias, R. D. Keyser, S. Dormido, and F. Esquembre. Developing networked control labs a matlab and easy java simulations approach. *IEEE Transactions on Industrial Electronics*, 57:3266–3275, 2010.

[60] S. Dormido, H. Vargas, J. Sánchez, N. Duro, R. Dormido, S. Dormido-Canto, and F. Esquembre. Using web-based laboratories for control engineering education. International Conference on Engineering Education–ICEE 2007, 2007.

[61] J. Chacón, J. Sänchez, A. Visioli, and S. Dormido. Building process control simulations with easy java simulations elements. *IFAC Proceedings Volumes*, 46(17):138 – 143, 2013. 10th IFAC Symposium Advances in Control Education.

[62] I. Ruano, P. Cano, J. Gámez, and J. Gómez. Advanced lms integration of scorm web laboratories. *IEEE Access*, 4:6352–6363, 2016.

[63] F. Esquembre. Easy java simulations: a software tool to create scientific simulations in java. *Computer Physics Communications*, 156(2):199 – 204, 2004.

[64] F Esquembre. Facilitating the creation of virtual and remote laboratories for science and engineering education. In *3rd IFAC Workshop on Internet Based Control Education, IFAC-PapersOnLine*, volume 48, pages 49–58, 2015.

[65] J. Andujar Marquez, A Borrero, M.A. Márquez, and F. Esquembre. Connecting hardware to easy java simulations: from virtual experiments to remote/local labs. In *International Conference on Multimedia in Physics Teaching and Learning*, 2013.

[66] J. P. Hespanha, P. Naghshtabrizi, and Y. Xu. A survey of recent results in networked control systems. *Proceedings of the IEEE*, 95(1):138–162, Jan 2007.

[67] D. Yue, Q.L. Han, and C. Peng. State feedback controller design of networked control systems. In *Proceedings of the 2004 IEEE International Conference on Control Applications, 2004.*, volume 1, pages 242–247 Vol.1, Sept 2004.

[68] R. M. Murray, K. J. Astrom, S. P. Boyd, R. W. Brockett, and G. Stein. Future directions in control in an information-rich world. *IEEE Control Systems Magazine*, 23(2):20–33, April 2003.

[69] L. Zhang, H. Gao, and O. Kaynak. Network-induced constraints in networked control systems—a survey. *IEEE Transactions on Industrial Informatics*, 9(1):403–416, Feb 2013.

[70] E. Aranda-Escolástico, C. Rodríguez, M. Guinaldo, J. L. Guzmán, and S. Dormido. Asynchronous periodic event-triggered control with dynamical controllers. *Journal of the Franklin Institute*, 355(8):3455 – 3469, 2018.

[71] R. C. Martin. *Agile software development: principles, patterns, and practices*. Prentice Hall, 2002.

[72] A. M. Rosado da Cruz and S. Paiva. *Modern Software Engineering Methodologies for Mobile and Cloud Environments*. IGI Global, Hershey, PA, USA, 1st edition, 2016.

[73] E. Granado, W. Colmenares, O. Perez, and G. Cataldo. Remote experimentation using mobile technology. *IEEE Latin America Transactions*, 11(4):1121–1126, June 2013.

[74] F. Nayebi, J. Desharnais, and A. Abran. The state of the art of mobile application usability evaluation. In *2012 25th IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, pages 1–4, April 2012.

[75] D. López-de Ipiña, J. García-Zubia, and P. Orduña. 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, 2006.

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

[77] J. Bermudez-Ortega, E. Besada-Portas, J.A. Lopez-Orozco, J.A. Bonache-Seco, and J.M. de la Cruz. Remote web-based control laboratory for mobile devices based on ejss, raspberry pi and node.js. In *3rd IFAC Workshop on Internet Based Control Education, IFAC-PapersOnLine*, volume 48, pages 158–163, Brescia, Italy, November 2015.

[78] J. B. Bottentuit-Junior and C. P. Coutinho. Virtual laboratories and m-learning: learning with mobile devices. 2007.

[79] P.l Mach and Z. Becvar. Mobile edge computing: A survey on architecture and computation offloading. *arXiv preprint arXiv:1702.05309*, 2017.

[80] K. Kumar, J. Liu, Y. Lu, and B. Bhargava. A survey of computation offloading for mobile systems. *Mobile Networks and Applications*, 18(1):129–140, Feb 2013.

[81] F. Liu, P. Shu, H. Jin, L. Ding, J. Yu, D. Niu, and B. Li. Gearing resource-poor mobile devices with powerful clouds: architectures, challenges, and applications. *IEEE Wireless Communications*, 20(3):14–22, June 2013.

[82] S. Ou, K. Yang, and J. Zhang. An effective offloading middleware for pervasive services on mobile devices. *Pervasive and Mobile Computing*, 3(4):362 – 385, 2007. Middleware for Pervasive Computing.

[83] L. Liu, Z. Chang, X. Guo, S. Mao, and T. Ristaniemi. Multiobjective optimization for computation offloading in fog computing. *IEEE Internet of Things Journal*, 5(1):283–294, Feb 2018.

[84] K. Yang, S. Ou, and H. Chen. On effective offloading services for resource-constrained mobile devices running heavier mobile internet applications. *IEEE Communications Magazine*, 46(1):56–63, January 2008.

[85] G. Huerta-Canepa and D. Lee. A virtual cloud computing provider for mobile devices. In *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*, MCS '10, pages 6:1–6:5, New York, NY, USA, 2010. ACM.

[86] J. Yue, D. Zhao, and T. D. Todd. Cloud server job selection and scheduling in mobile computation offloading. In *2014 IEEE Global Communications Conference*, pages 4990–4995, Dec 2014.

[87] I. Giurgiu, O. Riva, D. Juric, I. Krivulev, and G. Alonso. Calling the cloud: Enabling mobile phones as interfaces to cloud applications. In *Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware*, Middleware '09, pages 5:1–5:20, Berlin, Heidelberg, 2009. Springer-Verlag.

[88] Y. Tao, Y. Zhang, and Y. Ji. Efficient computation offloading strategies for mobile cloud computing. In *2015 IEEE 29th International Conference on Advanced Information Networking and Applications*, pages 626–633, March 2015.

[89] W. S. Chin, H. Kim, Y. J. Heo, and J. W. Jang. A context-based future network infrastructure for iot services. *Procedia Computer Science*, 56:266 – 270, 2015. The 10th International Conference on Future Networks and Communications (FNC 2015) / The 12th International Conference on Mobile Systems and Pervasive Computing (MobiSPC 2015) Affiliated Workshops.

[90] F. Manco, J. Martins, K. Yasukata, J. Mendes, S. Kuenzer, and F. Huici. The case for the superfluid cloud. In *Proceedings of the 7th USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'15, pages 7–7, Berkeley, CA, USA, 2015. USENIX Association.

[91] P. Garcia Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, and E. Riviere. Edge-centric computing: Vision and challenges. *SIGCOMM Comput. Commun. Rev.*, 45(5):37–42, September 2015.

[92] B. Han, P. Hui, V.S.A. Kumar, M. V. Marathe, G. Pei, and A. Srinivasan. Cellular traffic offloading through opportunistic communications: A case study. In *Proceedings of the 5th ACM Workshop on Challenged Networks*, CHANTS '10, pages 31–38, New York, NY, USA, 2010. ACM.

[93] A. Mtibaa, K. A Harras, K. Habak, M. Ammar, and E. W. Zegura. Towards mobile opportunistic computing. In *2015 IEEE 8th International Conference on Cloud Computing (CLOUD)*, pages 1111–1114. IEEE, 2015.

[94] C. Salzmann, S. Govaerts, W. Halimi, and D. Gillet. The smart device specification for remote labs. In *12th International Conference on Remote Engineering and Virtual Instrumentation (REV)*, pages 199 – 208, 2015.

[95] M. Tawfik, C. Salzmann, D. Gillet, D. Lowe, H. Saliah-Hassane, E. Sancristobal, and M. Castro. Laboratory as a service (laas): A novel paradigm for developing and implementing modular remote laboratories. *International Journal of Online Engineering (iJOE)*, 10(4):13–21, 2014.

[96] M. A. Bochicchio and A. Longo. Delivering collaborative web labs as a service for engineering education. *International Journal of Online Engineering*, 8(2):4–10, 2012.

[97] Domenico Ponta, Anna Marina Scapolla, and Paolo Buschiazzo. Survey of remote laboratories using service oriented architectures. *International Journal of Online Engineering*, 5(1):34–39, 2009.

[98] C. Salzmann and D. Gillet. Smart device paradigm, standardization for online labs. In *Global Engineering Education Conference (EDUCON), 2013 IEEE*, pages 1217–1221, March 2013.

[99] H Saliah-Hassane, D Benslimane, I De La Teja, B Fattouh, LK Do, G Paquette, M Saad, L Villardier, and Y Yan. A general framework for web services and grid-based technologies for online laboratories. In *Proceedings of the Int'l Conf. Engineering Education and Research 2005*, pages 1–5, 2005.

[100] C. De Capua, A. Liccardo, and R. Morello. On the web service-based remote didactical laboratory: further developments and improvements. In *Instrumentation and Measurement Technology Conference, 2005. IMTC 2005. Proceedings of the IEEE*, volume 3, pages 1692–1696. IEEE, 2005.

[101] Y. Duan. Value modeling and calculation for everything as a service (xaas) based on reuse. In *Software Engineering, Artificial Intelligence, Networking and Parallel & Distributed Computing (SNPD), 2012 13th ACIS International Conference on*, pages 162–167. IEEE, 2012.

[102] A. Dubey and D. Wagle. Delivering software as a service. *The McKinsey Quarterly*, 6(2007), 2007.

[103] S. Bhardwaj, L. Jain, and S. Jain. Cloud computing: A study of infrastructure as a service (iaas). *International Journal of engineering and information Technology*, 2(1):60–63, 2010.

[104] H. Hacigümüs, S. Mehrotra, and B. Iyer. Providing database as a service. In *icde*, page 0029. IEEE, 2002.

[105] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos. Sensing as a service model for smart cities supported by internet of things. *Transactions on Emerging Telecommunications Technologies*, 25(1):81–93, 2014.

[106] F. Long. Software vulnerabilities in java. Technical Report CMU/SEI-2005-TN-044, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2005.

[107] Said M. *Extensive analysis of the software security vunerabilities that exist within the Java software execution environment*. PhD thesis, University of Wisconsin, 2008.

[108] J. Fonseca, N. Seixas, M. Vieira, and H. Madeira. Analysis of field data on web security vulnerabilities. *IEEE Transactions on Dependable and Secure Computing*, 11(2):89–100, March 2014.

[109] J. Gassen and J. P. Chapman. Honeyagent: Detecting malicious java applets by using dynamic analysis. In *2014 9th International Conference on Malicious and Unwanted Software: The Americas (MALWARE)*, pages 109–117, Oct 2014.

[110] N. Meng, S. Nagy, D. Yao, W. Zhuang, and G. Arango-Argoty. Secure coding practices in java: Challenges and vulnerabilities. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pages 372–383, May 2018.

[111] A. Sahu and A. Singh. Securing iot devices using javascript based sandbox. In *Recent Trends in Electronics, Information & Communication Technology (RTEICT), IEEE International Conference on*, pages 1476–1482. IEEE, 2016.

[112] E. Freeman, E. Freeman, B. Bates, and K. Sierra. *Head First Design Patterns*. O' Reilly & Associates, Inc., 2004.

[113] M. Readman and H. Hagadoom.

[114] D. Galán Vicente. *Towards the open experimentation with interactive laboratories*. PhD thesis, UNED. Universidad Nacional de Educación a Distancia (España), 2017.

[115] D. Galan, R. Heradio, L. de la Torre, S. Dormido, and F. Esquembre. Conducting online lab experiments with blockly. *IFAC-PapersOnLine*, 50(1):13474–13479, 2017.

[116] D. Galán, L. de la Torre, D. Chaos, E. Aranda, J. Sánchez, and S. Dormido. Entorno de experimentación para laboratorios en línea: el caso del péndulo de furuta. *Actas de las XXXIX Jornadas de Automática, Badajoz, 5-7 de Septiembre de 2018*, 2018.

[117] J. Tajuelo, J. Sáenz, J. A. de la Torre, L. de la Torre, I. Zúñiga, and J. Sánchez. On the fully automation of the vibrating string experiment. In *Online Engineering & Internet of Things*, pages 469–482. Springer, 2018.

[118] M. Guinaldo, J. Sánchez, H. Vargas, and S. Dormido. An Advanced Web-Based Control Laboratory for the Ball and Beam System. In *Control 2010*, Coimbra, 2010.

[119] D. Galan, R. Heradio, L. de la Torre, S. Dormido, and F. Esquembre. Automated experiments on EjsS laboratories. In *International Conference on Remote Engineering and Virtual Instrumentation*, pages 78–85, Madrid, Spain, Feb 2016.

[120] R. Pastor, J. Sanchez, and S. Dormido. Web-based virtual lab and remote experimentation using easy java simulations. In *Proceedings of the 16th IFAC World Congress*, 2005.

[121] J. Valiente, L. de la Torre, and J. Chacón. Master de ingeniería de sistemas y de control. 2014.

[122] J. M. McCarthy and G. S. Soh. Analysis of planar linkages. In *Geometric Design of Linkages*, pages 15–53. Springer, 2011.

[123] N. P Robson and J. M. McCarthy. The synthesis of planar 4r linkages with three task positions and two specified velocities. In *ASME 2005 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pages 425–432. American Society of Mechanical Engineers, 2005.

[124] X.J. Liu and J. Wang. Some new parallel mechanisms containing the planar four-bar parallelogram. *The International Journal of Robotics Research*, 22(9):717–732, 2003.

[125] D. . Ruth and J. M. McCarthy. The design of spherical 4r linkages for four specified orientations. *Mechanism and Machine Theory*, 34(5):677–692, 1999.

[126] A. P. Murray and P. Larochelle. A classification scheme for planar 4r, spherical 4r, and spatial rccc linkages to facilitate computer animation. *ASME Paper No. DETC98/MECH-5887*, 1998.

[127] J.P. Merlet, C. M. Gosselin, and N. Mouly. Workspaces of planar parallel manipulators. *Mechanism and Machine Theory*, 33(1):7 – 20, 1998.

[128] C. Gosselin and J. Angeles. Singularity analysis of closed-loop kinematic chains. *IEEE transactions on robotics and automation*, 6(3):281–290, 1990.

[129] S. Dormido, H Vargas, J Sánchez, R. Dormido, N. Duro, S. Dormido-Canto, and F Morilla. Developing and implementing virtual and remote labs for control education: The uned pilot experience. *IFAC Proceedings Volumes*, 41(2):8159–8164, 2008.

[130] H. Vargas, G. Farias, J. Sanchez, S. Dormido, and F. Esquembre. Using augmented reality in remote laboratories. *International Journal of Computers Communications & Control*, 8(4):622–634, 2013.

[131] J. Chacon, M. Beschi, J. Sánchez, A. Visioli, and S. Dormido. Experimental analysis of a remote event-based PID controller in a Flexible Link System. In *Workshop on Event-Based Systems, 19th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'2014)*, Barcelona, Spain, 2014.

[132] J. Chacón, M. Beschi, J. Sánchez, A. Visioli, and S. Dormido. An Experimental Framework to Analyze Limit Cycles Generated by Event-Based Sampling. In *19th IFAC World Congress*, Cape Town, South Africa, August 2014.

[133] L. de la Torre, J. Sanchez, S. Dormido, J.P. Sanchez, M. Yuste, and C. Carreras. Two web-based laboratories of the fisl@bs network: Hooke's and snell's laws. *European Journal of Physics*, 32:571–584, 2011.

[134] E. Fabregas, G. Farias, S. Dormido-Canto, M. Guinaldo, J. Sánchez, and S. Dormido Bencomo. Platform for teaching mobile robotics. *Journal of Intelligent & Robotic Systems*, 81(1):131–143, 2016.

[135] D. V Neamtu, E. Fabregas, B. Wyns, R. De Keyser, Se. Dormido, and C. M. Ionescu. A remote laboratory for mobile robot applications. In *Proceedings of Preprints of the 18th International Federation of Automatic Control World Congress,(IFAC), Milano, Italy*, volume 28, pages 7280–7285, 2011.

[136] A. P. French. *Vibrations and waves*. CRC press, 1971.