Widget-Based Approach for Remote Control Labs

Evgeny Bogdanov * **Christophe Salzmann** * **Denis Gillet** *

* *Ecole Polytechnique Fédérale de Lausanne, Switzerland*
(e-mails: evgeny.bogdanov,christophe.salzmann,denis.gillet@epfl.ch)

Abstract: The paper presents a novel approach to conduct laboratory experiments in relation with the Control class taught at EPFL. The existing laboratory interface built as a Java applet, which allows students to access experimentation devices locally or remotely, is split into a set of light-weight Web applications or widgets. While achieving the same basic functionalities, these widgets provide user customization and extension. They are more flexible and add off-the-shelf interaction with other widgets and with the surrounding Web environment. The paper shows how remote experimentation devices are integrated within the *Graasp* platform, which provides knowledge management and enables collaboration among students. The educational scenario illustrates the technical challenges, and the solutions to tackle them are proposed and explained.

Keywords: flexible education, remote labs, Web-based learning, widget, collaboration

1. INTRODUCTION

Flexible education is a recent approach used by institutions to transfer knowledge from universities to students. In Gillet et al. (2005) the authors stated that both a flexible access to experimentation resources and availability of collaboration facilities are required. The flexible access to experimentation is provided to students with the help of virtual and/or remote laboratories. In addition, Web collaboration facilities are provided to support students while conducting an experiment. For example, when they produced resources such as simulation results or measurements, they can be saved in a shared space to be accessed by other students. These two major requirements shape the available solutions for remote experimentation laboratories.

Many of the existing solutions are developed as complex monolithic stand-alone frameworks that handle both the remote experimentation aspect and the support for collaborative work. However, according to Salzmann et al. (2012), this monolithic structure is difficult to adapt to varying user requirements, evolving curriculum and new technologies. In Salzmann and Gillet (2007) the authors explain that the high development overhead and the difficulties associated with the integration of the remote experimentation within existing Learning Management Systems (LMS) has refrained the spread and the acceptance of the common monolithic solution.

Despite the limitations of existing solutions, the need and justifications for virtual and remote laboratories are still present. Salzmann et al. (2012) proposed a new paradigm to divide the current monolithic solution into smaller universal components (Web widgets) that users can re-aggregate dynamically to form a personal environment. Similarly, some intelligence and the flexibility are added to a remote experimentation server to provide more autonomous actions and to support a wider range of clients and protocols.

The paper shows how this novel approach is used at Ecole Polytechnique Fédérale de Lausanne (EPFL) within the Automatic Control course. The existing remote lab implemented as a monolithic Java applet is split into a set of light-weight widgets according to OpenSocial apps specification [1] . Afterwards, the widgets are combined together to provide the same functionality that existed in the Java applet solution. These settings will be used within Automatic control course in the fall 2012 semester at EPFL.

Turning monolithic application into a modular one with OpenSocial apps has many advantages. First, it brings more flexibility to teachers and students, since they are able to assemble modules on their own or replace some modules at their will. They can even extend the proposed set of modules with other modules such as a chat tool for communication or a collaborative formula editing tool. Additionally, maintenance and development costs are greatly reduced, since universities do not have to maintain a standalone application but rather relatively simple modules and can reuse already existing modules. Second, the widgets use Web protocols which makes them portable between different Web browsers (including mobile devices). Moreover, users do not have to install additional plugins such as Flash or Java. The last but not least, OpenSocial apps used as a modular unit represent one of the standards for widgets. This allows to bring a remote experimentation widget bundle into other Learning Management Systems that students may already use (e.g., Moodle [2] or Sakai [3]). Furthermore, widgets can save and retrieve data to/from their Web platform. This feature helps users to directly

[1] http://docs.opensocial.org/display/OSD/Specs
[2] http://moodle.org
[3] http://sakaiproject.org/

store data into LMS or other platforms where they run experiments and avoid additional authentication steps.

The paper is organized as follows. The section 2 introduces the scenario used in Control course. In the next section 3, we discuss the architecture supporting the course. In the section 4 we details the technical challenges we faced, and the section 5 concludes the paper.

2. CONTROL COURSE TEACHING SCENARIO

Undergraduate students enrolled in engineering programs at EPFL are asked to perform hands-on laboratory sessions during the last semester of their bachelor program in the context of an automatic control laboratory course. These laboratory sessions aim at studying experimentally the behavior of dynamical systems. These experiments deal with different thematic modules and are conducted by students in groups of 2 to 4. Each group is supposed to handle a report at the end of the semester. The average number of students who took the course in the last 5 years is about 150. They can access the laboratory experiments directly on campus once a week, or remotely, 24 hours a day, 7 days a week.

The typical laboratory setup introduced to practice position and speed control is a servo drive where students first identify the system and then design a PID controller. This setup consists of a DC motor equipped with a digital encoder. The motor drives a brass disk acting as the load. The angular position is measured with a digital encoder connected to the motor axle. Along the same axle, an enlarged rotating disk permits an easy visualization of the motion. This enlarged disk and the rotating load motion are captured by a video camera. The whole hardware has been designed in such a way that it is fully controllable from the connected computer. Similarly, the hardware state can be diagnosed from the connected computer. For example, in addition to the required disk position and speed measurements, diagnostic signals informing about the power status can be read from the connected computer. Likewise, additional actuators have also been added. For example, a second motor acting as a generator is placed along the main motor axle to generate a perturbation that can be controlled remotely by switching the generator load. Also, the main power can be switched on and off from the connected computer to save energy when not used. Before the sessions an instructor prepares a list of recommended widgets and guidelines. An additional material like reference to the ex-cathedra course is also recommended.

In the first module students will identify the electrical drive by applying a step to the servo drive and measure its response. Students will need two widgets to perform this task. The first widget is a signal generator that can apply a given voltage to the servo drive. The second widget is an oscilloscope that displays both the applied signal and the servo drive response. Once the successful response is measured, it can be saved in the user environment by a simple drag and drop action. The saved measurements are automatically shared with the other members of the group.

In the next module students will identify the servo drivers parameters, namely, the time constant and the static gain. To do so they will use another widget that can display both the measured response and a simulated one. The saved response is drag and dropped from the student environment to the widget. Students will change the parameters of the simulated signal by dragging the simulated response until it superposes with the measured step response. At this time the simulated parameters are equal to the system parameters.

In the following step students implement a PI controller to control the rotational speed of the servo drive. The design of the controller is based on the Ziggler-Nichols method. Once the Kp and the Ti parameters are defined, students validate them by adding a controller widget to the initial signal generator and oscilloscope configuration. By adding the controller widget to the configuration, the servo drive automatically switches from open-loop to close loop mode. Additional measurements can be saved to the user environment.

Historically these laboratory sessions were first conducted within an in-house developed LMS, namely, *eMersion* (Gillet and Fakas (2001)) and later with *eLogbook* (Rekik et al. (2007)). While the possibility to remotely access the laboratories were praised by students, both environments suffer limitations. Beside the fact students had to learn a new environment, one of the main limitations reported was the lack of integration of tools students were currently using. For example, to reinforce collaboration among users we introduced a custom chat application (Salzmann et al. (2008)). It turned out that it was barely used since students had all their contacts in another chat applications. The assessment of the formers monolithic environments showed limitations that would be difficult to alleviate without a complete rethinking of the current LMS environment. This reflexion lead to the deconstruction of the former environments into various functionalities served by smart devices (Salzmann and Gillet (2008)) on the server side and presented with the help of widgets on the client side. This structure permits the development of a richer and more versatile user experience. Within *Graasp* [4], the aggregation of these functionalities combined with functionalities found elsewhere can form a Personal Learning Environment (PLE) tailored to users' needs.

3. REMOTE LAB ENVIRONMENT

This section describes the proposed architecture to support the Automatic Control teaching scenario at EPFL. First, it describes how a monolithic Java Applet is split into a modular set of widgets. Second, it introduces the platform that is used to run the remote experiment as widgets and behaves as data storage and a collaboration environment.

3.1 Widgets as Modular Units

A Web widget is a small application that can be installed and executed within a Web page by an end user. Widgets are typically created in DHTML and JavaScript. Several widget standards exist (OpenSocial apps, W3C widgets [5]). We rely on OpenSocial apps standard for widgets. One of

[4] http://graasp.epfl.ch
[5] http://www.w3.org/2008/webapps/wiki/WidgetSpecs

the main reasons was the fact that the OpenSocial Foundation provides, in addition to the specification for widgets, a standard API to retrieve the user data from the widget container. Previously the Remote Experimentation Device (RED) for the Automatic Control course laboratory was implemented as a Java applet (Fig. 1). Due to the limitations of this approach it was decided to re-implement the remote lab framework as light-weight widgets (Salzmann et al. (2012)). The resulting implementation can be found in Fig. 2.

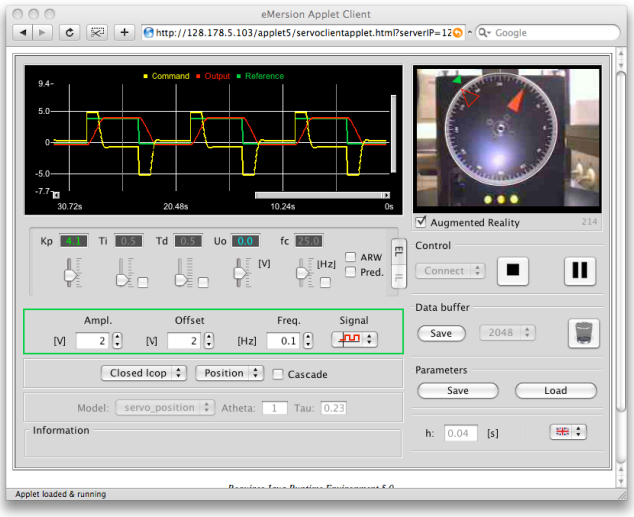

Fig. 1. RED device as a Java applet

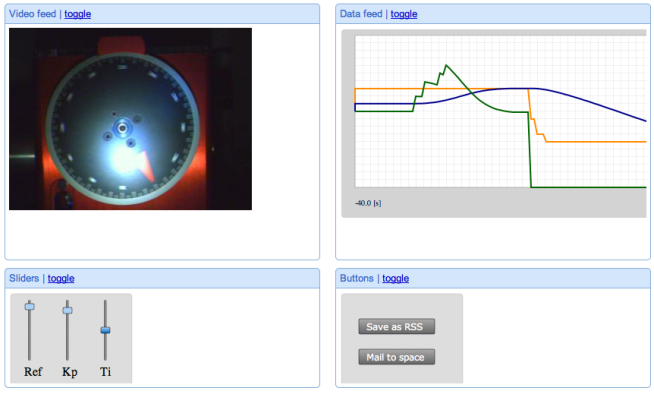

Fig. 2. RED device as a set of widgets

As it can be seen from the comparison of two pictures, the main functionalities are still the same, however, the widgets have many valuable advantages discussed in the introduction.

3.2 Graasp as Collaborative Widget Container

A set of widgets has to run in a widget container. We use *Graasp* (previously known as *Graaasp*) as a widget container (Bogdanov et al. (2010)). *Graasp* is a Web platform that has been developed to investigate the potential of social media in higher education for learning and knowledge management purposes (Bogdanov et al. (2012)). There are 4 types of entities in the platform: resources, applications, spaces and people. *Graasp* targets the management of people's spaces. It supports users in creating and sharing resources and applications with other people in the context of a space.

To support the Automatic Control course scenario, a teacher creates a space in *Graasp* containing resources relevant to the course. In addition, the teacher adds widgets for remote experimentation into this space. The resulting remote lab interface within *Graasp* is depicted in Fig. 3. Then students are invited into the space where they can find the resources needed for the course: pdf documents with lab descriptions and experiment protocols, the course assistants, the remote lab experimentation interface, etc.

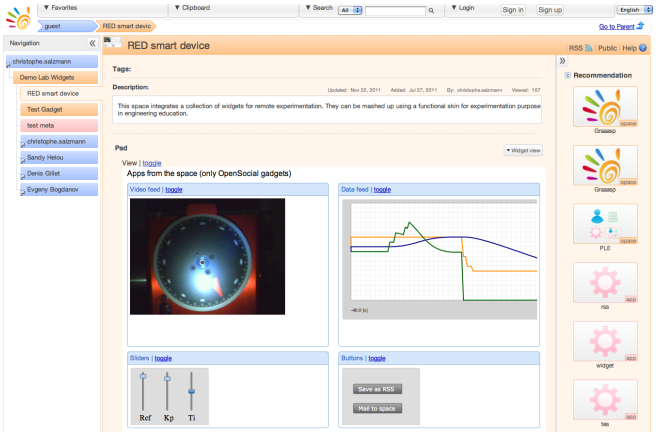

Fig. 3. RED device running in *Graasp*

When students conduct the experiment, they produce data that have to be later analyzed or might be used for the exam preparation. The data produced during the experiment can be transferred from widgets into the *Graasp* platform as resources. Later, students can share the produced data with their peers or tutors and analyze the data with other widgets. The features that enable communication between students over the produced artifacts are supported by *Graasp*. One important aspect of the whole installation is the ability of widgets to communicate with each other, with their hosting platform, and with external remote smart devices. In the next section we discuss the technical details enabling the communication.

A ubiquitous access to the remote lab experimentation is provided to students. This enables collaboration over produced data which supports the flexible education. In the Automatic Control course scenario we use *Graasp* platform to enable collaboration among teachers/students and for knowledge management. However, the proposed solution is not limited to *Graasp* but encompasses any OpenSocial-compliant widget container. In particular, it can be easily exported to an existing LMS. Fig. 4 shows how our installation setting runs within Moodle LMS.

4. TECHNICAL DETAILS

In order to support the Automatic Control course, several technical problems had to be tackled. This section discusses the problems and the solutions.

4.1 Apache Shindig as Widget Engine

Java applets require a Java Virtual Machine, and widgets require a widget engine to run in a Web page. The widget

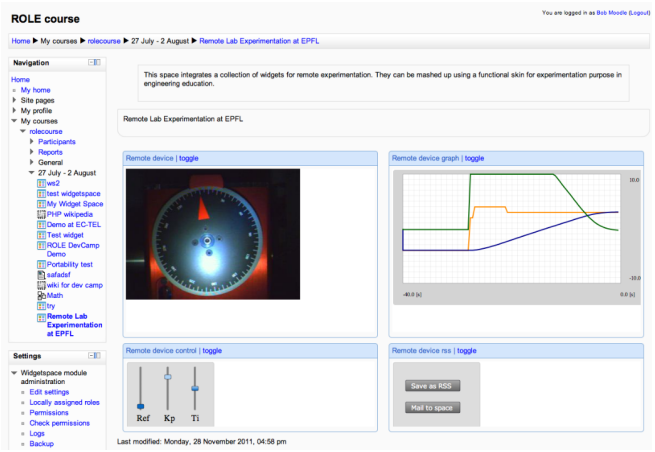

Fig. 4. RED device running in Moodle

engine is provided by a widget container (*Graasp* in our case). It is JavaScript based, which means that any recent Web browser supports widgets without any extension or a plug-in.

Graasp uses Apache Shindig [6] as the widget engine, which is an open source reference implementation of the OpenSocial API. OpenSocial currently lacks a concept that can map to a course or to a place shared by several people (e.g., group of students sharing an experiment space). To fulfill our specific needs of space and resource concepts, we extended OpenSocial APIs and provided an implementation within Apache Shindig. Since this concept is a relatively common scenario in collaborative platforms, it was proposed to the OpenSocial specification (Bogdanov et al. (2011)).

The resource extension allows to save digital artifacts from a widget to its hosting container. A similar concept exists in the OpenSocial specification, though at the moment it is limited to photo and video albums. The upcoming file API of the OpenSocial will satisfy our needs but for now we have to stick to our extension. To sum up, by using the extended Apache Shindig, *Graasp* can render OpenSocial widgets and through the OpenSocial API students can exchange data between widgets and *Graasp*.

4.2 Widget-to-widget communication

In the monolithic Java applet that was previously used for the course (Fig. 1), the different parts were interacting with each other. If a user moves the slider, the remote device reacts accordingly by updating the graphs and rotating the device in the applet. The applet is split now into modular widgets, however, these widgets still have to communicate with each other. So if the user updates the slider position in the slider widget (Fig. 2, left bottom widget), the other widgets should be notified about the change. To support data and events exchange between widgets, two inter-widget communication modes are needed: local and remote.

In the case of local inter-widget communication (Fig. 5.a), the data does not leave the browser and the remote services are not involved. Data is transferred from a source widget directly to the browser memory and then moved from there to a destination widget. In Zuzak (2011) the author classified the available techniques to achieve this type of communication. We have chosen OpenApp approach introduced by Isaksson and Palmér (2010) for local inter-widget communication due to its improved usability compared to other approaches and partial semantic interoperability features.

Even though local inter-widget communication is enough for the majority of scenarios, it limits several use cases. As example, consider a graph widget that can send data to an analysis widget. The graph widget displays normally a sparse representation of a complete data set for performance reasons. So if the graph widget is about to send the complete data set to the analysis widget, it inevitably includes an interaction with a server since the complete data set is located there. This type of communication is called remote inter-widget communication. Three implementation scenarios for this mode are detailed next. First, the source widget asks its server for all data and retrieves it from its remote service. Then the data is sent via local inter-widget communication to the destination widget (Fig. 5.c). Second, the source widget locally notifies the destination widget about data (Fig. 5.d). The source widget either provides a remote access point, where the data can be retrieved by the destination widget or asks the destination widget to update its state. In the latter case, the destination widget will receive updated data from its server. Third approach works even in the case where several widgets are open in different browsers (Fig. 5.b). The source widget either sends some data to its remote server or asks the server to notify the destination widget about the updated state of the source widget. Afterwards, the server sends the data or the new state to the destination widget. To propagate data from the server to a widget either HTTP POST requests can be used or different synchronous communication approaches (WebSockets, AJAX Comet, etc). It should be noted that by the server we mean either a single server or a federation of several servers that can communicate with each other.

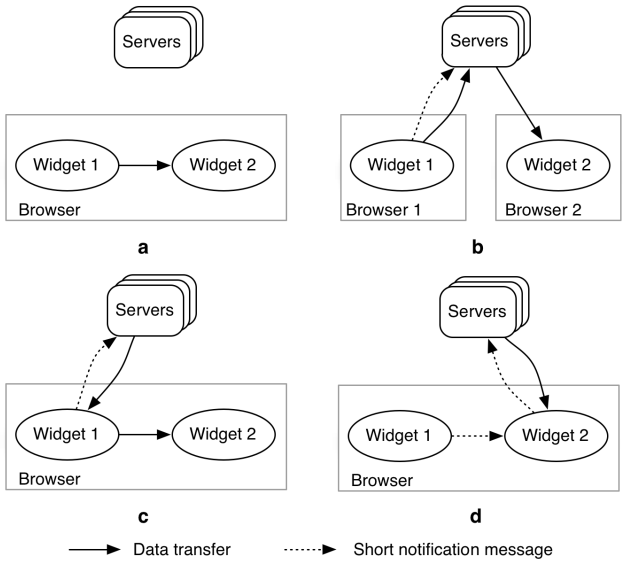

Fig. 5. Inter-widget communication

[6] http://shindig.apache.org/index.html

4.3 Widget-to-platform communication

For security reasons, widgets are run on the domain different from the domain of their hosting platform, which means they can not directly access (authentication is required) APIs of the platform to save and retrieve data. However, Apache Shindig provides a special security token for widgets and serves as a proxy service to exchange data between a platform and a widget. With this approach widgets can save and retrieve data from their hosting container without any additional authentication.

The OpenSocial API is common among different containers. If a widget accesses data from the container through this API, it is guaranteed the widget will work in other OpenSocial-compliant containers. Widgets can save data into container with AppData API. These data belong to widgets and can be further reused by them. For example, the last used state of the sliders for a widget can be saved as AppData. Next time the widget is loaded it can retrieve these data and set sliders into previously saved positions. The bigger pieces of data (measurements, reports) are normally saved as resources in *Graasp* with Resource API (see the section 4.1). Differently from AppData that always belong to a widget, resources do not belong to a specific widget. Resources can be moved from one place to another, shared among people, accessed by different widgets, etc.

4.4 Widget-to-smart devices communication

Widgets work with remote devices which requires real-time updates in the browser. In other words, if a user changed a parameter and a device reacted to the change, the effect of the action should be seen immediately. The state of the remote device should be synchronized between the device and its representation in the browser. Similarly, to support collaboration among several people over the remote device, its state should be updated in real time between the collaborators' browsers. We use the WebSockets protocol to synchronize the state and push updates from the remote device to widgets.

5. CONCLUSION

The paper presented the laboratory experiment used in the Automatic Control course at the EPFL university. We proposed to divide the currently used Java applet through which students can work with the laboratory device remotely into a combination of light-weight widgets. While providing the same functionality, this decomposition tackles the disadvantages of the monolithic applet approach. We discussed where the main technical difficulties lie in the scenario implementation and how they can be overcome. In addition, we explained how the remote experimentation device can be integrated into different platforms (e.g, Moodle LMS) and provided an example of its use within collaborative knowledge management system called *Graasp*, that will be used at EPFL for the course.

At the moment, we have a working proof-of-concept setting that supports the Automatic Control course scenario. However, additional work has to be accomplished to fully realize the scenario before it can be given to students. First, the set of proof-of-concept widgets has to be brought into production-ready state by making them user-friendly and implementing fully the required functionality. Second, we considerably rely on inter-widget communication. However, in Isaksson and Palmér (2010) the authors stated that widget communication risks worsening the user experience, since it may cause the user interface to become more complicated and unintuitive. Thus, further research in this area and required adaptations are needed.

ACKNOWLEDGEMENTS

The research work described in this paper is partially funded through the ROLE Integrated Project; part of the Seventh Framework Programme for Research and Technological Development (FP7) of the European Union in Information and Communication Technologies, and the Personal Learning Environment (Phase 3) project of the AAA/SWITCH programme.

REFERENCES

Bogdanov, E., El Helou, S., Gillet, D., Salzmann, C., and Sire, S. (2010). Graaasp: a web 2.0 research platform for contextual recommendation with aggregated data. *CHI EA '10*, 3523–3528.

Bogdanov, E., Limpens, F., Li, N., El Helou, S., Salzmann, C., and Gillet, D. (2012). A social media platform in higher education. Accepted to Educon Conference.

Bogdanov, E., Salzmann, C., and Gillet, D. (2011). Contextual Spaces with Functional Skins as OpenSocial Extension. *Advances in Computer-Human Interactions*.

Gillet, D. and Fakas, G. (2001). eMersion: A New Paradigm for Web-Based Training in Mechanical Engineering Education. 8, 10–14.

Gillet, D., NguyenNgoc, A.V.N., and Rekik, Y. (2005). Collaborative Web-Based Experimentation in Flexible Engineering Education. *IEEE Transactions on Education*, 48(4), 696–704.

Isaksson, E. and Palmér, M. (2010). Usability and inter-widget communication in PLEs. *Proceedings of the 3rd Workshop on Mashup Personal Learning Environments*.

Moccozet, L., Benkacem, O., Ndiaye, B., Ahmeti, V., Roth, P., and Burgi, P.Y. (2011). An exploratory study for the implementation of a techno-pedagogical personal learning environment. *The PLE Conference 2011*.

Rekik, Y., Gillet, D., El Helou, S., and Salzmann, C. (2007). The eLogBook Framework: Sustaining Interaction, Collaboration, and Learning in Laboratory-Oriented CoPs. *International Journal of Web-based Learning and Teaching Technologies*, 2(3), 61–76.

Salzmann, C. and Gillet, D. (2007). Challenges in Remote Laboratory Sustainability. In *ICEE 2007*.

Salzmann, C. and Gillet, D. (2008). From online experiments to smart devices.

Salzmann, C., Gillet, D., Esquembre, F., Vargas, H., Sánchez, J., and Sebastián, D. (2012). Web 2.0 open remote and virtual laboratories in engineering education. In *Book chapter: Collaborative Learning 2.0: Open Educational Resources, by IGI Global*. In print.

Salzmann, C., Gillet, D., Scott, P., and Quick, K. (2008). Remote lab: online support and awareness analysis.

Zuzak, I. (2011). List of systems that enable inter-window or web worker communication. `code.google.com/p/pmrpc/wiki/IWCProjects`.