Available online at www.sciencedirect.com

ScienceDirect

IFAC Papers Online CONFERENCE PAPER ARCHIVE

IFAC-PapersOnLine 48-29 (2015) 247–252

Remote Interoperability Protocol: A bridge between interactive interfaces and engineering systems. ⋆

Jesús Chacón * **Gonzalo Farias** ** **Hector Vargas** **
Antonio Visioli *** **Sebastián Dormido** *

* *Universidad Nacional de Educación a Distancia, e-mail: jchacon@bec.uned.es.*
** *Pontificia Universidad Católica de Valparaíso, Chile.*
*** *Dipartimento di Ingegneria Meccanica e Industriale, Università degli Studi di Brescia, Italia.*

Abstract: The process of building remote or virtual laboratories to be deployed via Internet usually involves communication between different software tools. Very often, there is a separation between the software which interfaces with the model or real system, and the software responsible of providing the student with an interactive and visual representation of the data provided by the engineering system. Abstracting the way these two elements communicate with each other from the particular implementation, the requirements are frequently the same: connection and session control, data transmission control, user interaction handling, etc. This work describes a generic protocol to interoperate remotely any kind of engineering software. The solution proposes to encapsulate all the communication issues into an interoperability API that can be implemented in many different systems. In order to show the flexibility of such API, an implementation to interoperate MATLAB from Java user interfaces via JSONRPC is explained in detail.

Keywords: control education, interactive, teaching, computer interfaces, communication protocols

1. INTRODUCTION

Networks such as the Internet are widely distributed in society, connecting people across of the world. The connectivity between many different computational devices is increasing every year and, nowadays, the scientific community and engineering companies invest economical and research effort in the development of technologies and standard for the *Internet of Things* (IoT, sometimes Internet of Everything). Under this term, there is a network of physical *things* or objects with connectivity to enable data communication with other devices. Within the IoT, there are countless devices that can be sensed or controlled via Internet, encouraging the integration between physical world and computer-based systems.

Educators can also benefit from this situation offering their students new ways to access learning resources without time and location constraints. And, in fact, in the last years the scientific community has devoted great efforts to apply the advances in these fields to engineering education (Heck, 1999; Dormido, 2004), yielding a plethora of online interactive tools, presented either as simulations or as remote experiences in real plants. But, as in IoT, one of the concerns is the scalability and interoperability, in part due to the variety of devices and the intense use of machine-to-machine (M2M) communications. Among all the opportunities that these technologies offer, three of them present features of special interest to teaching engineering: network communications, visualization, and interactivity.

Visualization and interactivity have proved to be crucial aspects when designing virtual labs, i.e. simulations that are to be used for pedagogical purposes, in the field of control engineering. The graphical capabilities of computers, using images or animations, can help students to understand more easily the key concepts of the system under study. Moreover, interactivity allows students to simultaneously see the response of the systems to any change introduced by the user (Dormido et al., 2005a; Sánchez et al., 2002). These features add to engineering simulations rich visual content and the possibility of an immediate observation of system response, which turns a simulation into a natural and human-friendly way to learn, helping the student to get useful practical insight into engineering systems fundamentals.

Currently, there are many tools available to build simulations of a wide range of systems in control engineering. These tools provide great capabilities that allow to study the behaviour of the systems under different scenarios, as well as countless hardware drivers to connect any kind of device. In control enginering, MATLAB (The MathWorks, 2015) is the *de facto* standard software tool. It is a technical and numerical computing environment, which was born as a tool to manipulate matrix. But there are many

⋆ This work has been funded by the National Plan Project DPI2012-31303 of the Spanish Ministry of Science and Innovation and FEDER funds.

10.1016/j.ifacol.2015.11.244

other tools, either commercial products such as LabVIEW, Sysquake, Maple, Mathematica, open source tools like Scilab, Maxima, Octave, or even programming languages with numerical computing libraries such as NumPy/SciPy for Python, and Numeric.js for JavaScript. Though some of these tools have support to develop graphical interfaces, usually they are far from being ideal for unexperienced programmers to develop interactive tools.

The aim of this paper is to describe an interoperability protocol for remote and virtual laboratories, based on previous work (Sánchez et al., 2002; Dormido et al., 2005b; Farias et al., 2011; Chacon et al., 2015). The basic needs of an online interactive learning tool are captured in an easy-to-use and easy to implement protocol to help interconnect human interfaces to engineering simulations.

With this protocol, the development of an interactive learning tool for engineering can be decoupled in two tasks. On the one hand, the model of the engineering simulation or the interface to the process is created using a specialized software like MATLAB, LabVIEW or whatever the developer feels comfortable with. On the other hand, the human interface can be created using another software tool specialized in the design and implementation of graphical interfaces for non-expert programmers as, for example, Easy Java Simulations (EJS, (Esquembre, 2015)). This two components can be easily connected through a well-defined, simple and effective protocol, having a high-degree of independence.

In this way, not only the development effort is rationalized, because many software components can be reused and shared between different interactive tools, but also help the educator focus on the design of the experience and the visualization, while devoting less effort to the low-level implementation details. Moreover, if an engineering system adhere to the protocol, it is even possible to use it without knowing these implementation details, but only using the interface to interact with it. This enable a new paradigm of design, where physical resources can be shared between different universities, having the same or different views and experiences to carry out with the system, depending on the particular needs of each user. Despite from that, there are other cases where it can be useful to reuse a model or a system interface. For example, many academic plants are deployed with some sample software developed in MATLAB or LabVIEW, to demonstrate the capabilities of the system or to propose learning experiences.

The paper is organized as follows. Section 2 describes briefly the creation of standard engineering simulations. In Section 3 a communication protocol to connect engineering software with an interactive human interface is presented. Section 4 adapts the communication protocol to perform remote operation of engineering simulations. Finally main conclusions are discussed.

2. BUILDING SIMULATIONS

2.1 Overview

Any interactive simulation that uses a model previously created on an existing engineering software can be conceptually separated on two differentiated components. The first one is the *system*, a real process or a simulation created with a standard engineering software that models the process of interest. The second one is the *interactive view* through which users can observe and manipulate the system. The client application adds to the underlying (external) engineering application an upper layer that allows students to interactively manipulate and visualize the response of the system. The external application is controlled by the client application and provides it data for visualization. But, no matter what kind of software is being used, there always are some common needs to communicate the engineering system and the interactive view, such as the initial configuration, a way to read/modify the value of the variables or states (for visualization and manipulation) and also, depending on the system, a way to control the execution, pause, or more complex commands. These actions can be abstracted from the particular implementation, and captured into a generic interoperability protocol. Ideally, the subscription to the protocol should allow a complete decoupling between the two components (the system and the view) in a bidirectional way: on the one hand, an existent model or system can be accessed through different views, for example, to carry out different learning experiences on the same system or to present different visualization depending on the kind of user. On the other hand, an interactive view can be shared by different systems. For example, to connect with different systems depending on their time availability. Focusing on the interconnection between MATLAB and EJS, one have to cope with several problems. First, there are different approaches to control MATLAB from Java, such as compiling the program with Builder JA to create a Java wrapper, or use a third-party library like *JMatlink* or *matlabcontrol* that allow to control the MATLAB engine/GUI (see (Altman, 2012) for more details). Then the EJS/Java code must be implemented to solve the specific problem for the laboratory in development. In case there is a need to change the component on which the connection relies (due to compatibility issues, a substitution of the engineering software, etc.), the EJS application must be modified, requiring additional effort to adapt the laboratory. Encapsulating all the interconnection issues into a reusable component, the development can be rationalised, and only the components actually involved in the change have to be adapted. The communication protocol described in the next section shows a simple, but powerful application programming interface (API) that any engineering software should conform to, in order to provide all the features required to effectively implement the interoperate approach.

3. REMOTE INTEROPERABILITY PROTOCOL

In a common scenario, the client connects to the resource and acquire basic information about the capabilities of the system, such as: *can I control the execution or only behold the evolution of the system? what methods can I invoke? which variables can I obtain from the system? and which can I modify?*. After the negotiation phase, the client has obtained sufficient knowledge about the ways to interact with the system. Taking a close look to the functionalities that the engineering system must support, they can be divided into different categories: