IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received June 8, 2016, accepted June 28, 2016, date of publication July 7, 2016, date of current version October 31, 2016.

Digital Object Identifier 10.1109/ACCESS.2016.2587805

Advanced LMS Integration of SCORM Web Laboratories

ILDEFONSO RUANO, PABLO CANO, JAVIER GÁMEZ, AND JUAN GÓMEZ, (Member, IEEE)
Group of Robotic Automation and Computer Vision, E.P.S. de Jaén, Universidad de Jaén, Jaén 23071, Spain

Corresponding author: I. Ruano (alonso@ujaen.es)

This work was supported by the Research Project under Grant DPI2011-27284, Grant PID30-2014-16, and Grant PI10-AGR-6616.

ABSTRACT E-learning and its hybrid counterpart—B-learning—have become standard tools in higher education. The development of information and communications technologies (ICTs) and their application to education has made this possible. Among ICT technologies, the learning management system (LMS) has been the most important development. Teaching labs have also undergone a technological evolution, enabling them to be used online while interacting closely with LMSs. This paper presents a set of procedures that ease the integration of Java and JavaScript laboratories with LMSs in order to obtain adaptive learning contents, which are presented customized to students based on their likes and results. It describes innovative tools implemented by us as a Java package, a JavaScript library, and associated resources that achieve an advanced Lab-LMS integration in a shared content object reference model (SCORM) environment. These resources simplify communications between the elements of an SCORM content package, including the embedded Web laboratory, and the LMS where they are hosted. This paper describes the tools, procedures, and discusses its possibilities and advantages, and shows the results of several use cases of Web Labs delivered to undergraduate engineering students at University of Jaén that prove the validity of the proposal obtaining adapted learning and good results.

INDEX TERMS Adaptive learning, E-learning tool, E-learning standard, laboratory, learning management system, student experiment.

I. INTRODUCTION

Experimental work is essential for science and engineering students. Lab work helps improve students' practical understanding of the theoretical knowledge covered during classes. In engineering degrees, it is fundamental for students to obtain knowledge beyond pure theory. However, this is not the only positive characteristic of labs; there are several secondary advantages such as increases in student motivation [1].

The application of technical innovations in Information and Communications Technologies (ICTs) to teaching has enabled students to be able to carry out experimental work remotely through an Internet connection via online labs. When online labs are used with a web browser, or use web protocols, they are known as "WebLabs." Currently, WebLabs are an essential resource in education, particularly in engineering disciplines. WebLabs where students work with a simulation based on a model of the real system are called virtual labs (VL); those where students work with real systems are called remote labs (RL); and those where students work with both simulations and real systems are called hybrid labs. Additionally, the term Virtual or/and Remote Lab (VRL) can be used to refer to any combination of the above three types [2], [3]. In all cases, the student must run some software that acts as a graphical user interface (GUI) for the web-embedded lab [4], [5].

Currently, the most prominent component in the application of ICT to remote learning or e-learning is the Learning Management System (LMS). LMSs are ubiquitous in higher education institutions in the world and have become a fundamental part of the teaching and learning infrastructure. LMSs are virtual spaces accessible through a web browser that, among other things, can identify students, offer various communication tools (e.g., student to teacher and teacher to student), host teaching resources, allow the submission of papers, assess the knowledge progression of the students, offer evaluation tools, and present their results [6].

In 2008, [7] noted the convergence of labs with LMSs as one of several tendencies in the potential evolution of teaching. Subsequently, different points of view have been

contributed by several studies regarding this particular subject [8]–[12]. An advanced Lab-LMS integration facilitates the development of laboratories adapted to students as, since the LMS identifies the learner, it allows to adjust the contents and customize the learning path based on the likes and results obtained by the learners while they work in the WebLab.

When WebLabs are used for teaching purposes, they can be regarded as e-learning content. The Shared Content Object Reference Model (SCORM) [13] is a set of standards for the creation of e-learning content that is compatible with the majority of existing LMSs. There are several versions of SCORM, of which SCORM 1.2 and SCORM 2004 are most popular. The Tin Can API—also known as xAPI or experience API—has been called the ''next generation SCORM'' [14], however, its specification is quite different from SCORM, a change that carries new concerns. Nowadays, xAPI is still under development; currently, SCORM is still the most widely used e-learning format among the various LMSs.

This paper describes a series of tools developed by the authors to facilitate the creation of Java WebLabs (remote, virtual or hybrid). The most important tools consist on a SCORM package and a JavaScript library that allow an easy WebLab-LMS advanced integration. The main objective of these tools is to facilitate communication between a WebLab and the LMS to achieve student-adapted WebLabs. To validate this proposal, results that were obtained from a series of WebLabs developed using these tools are also presented. Figure 1 shows a screenshot of one of these embedded labs in a web page of a SCORM package.

The main features of WebLabs developed using these tools are:

1. Integrated pedagogical structure. The lab software, developed using Java, can be integrated into the SCORM package of the WebLab, along with a series of additional resources that can help maximize learning effectiveness for the students using the lab.
2. Multimode labs. The labs included in the WebLab can be remote, virtual or hybrid.
3. Familiar framework. WebLabs are presented to the students within the LMS they regularly use at their institution, which is a familiar framework.
4. WebLab-LMS integration. Communication with the LMS can be programmed using the proposed tools in a SCORM format, thus achieving advanced lab integration. Having an LMS container fully integrated with the VRL enables the following three features:
5. Customizable/Adaptive. Each user can interact with the lab in a customized way. For instance, different experiments or workflows for a single experiment can be presented on a per user basis.
6. Automated evaluation. Automated evaluations can be performed and their results stored in the LMS for posterior visualization.
7. Security. The LMS is responsible for performing user identification and also provides to the WebLab the inherent security given to all content included in the LMS.
8. Remotely accessible. The lab is accessible via web browsers. Students can perform the experiments from different locations and check their results afterward, while teachers can visualize the results of all the students.
9. Shareable and reusable. The lab can be exported and presented through any SCORM-compatible LMS.

The rest of the work is organized as follows: Section 2 analyses the lab software of a WebLab, Section 3 presents a brief introduction to SCORM and the possibilities for communication with the LMS, and Section 4 demonstrates how the proposed tools simplify interactions between a Java/JavaScript SCORM-embedded WebLab and the LMS. Section 5 shows some use cases for these tools, and Section 6 concludes the paper and discusses future work.

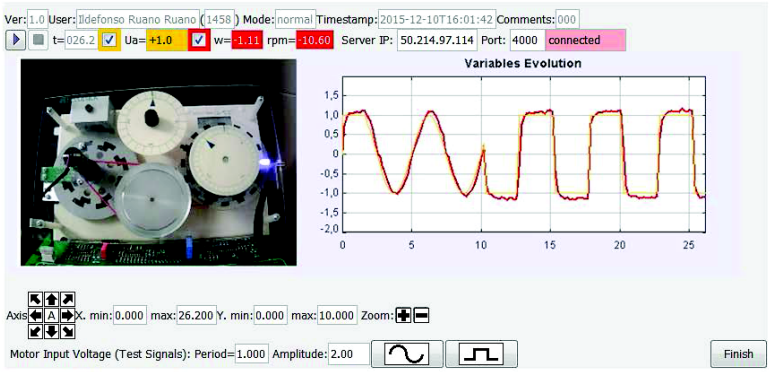
II. LABORATORY APPLICATION

A WebLab is an application that—at minimum—offers a GUI for the student and some other elements, depending on the particular type of lab. For remote labs, the WebLab must provide a means of communication for the student to interact with the remote system, while virtual labs must offer simulation capabilities.

The application should be developed using a web-compatible technology, as students will access the WebLab through a web browser. Traditionally, the most common technologies employed in e-learning have been Java, JavaScript and Flash; however, vulnerabilities found in Flash and, recently, in Java, along with the lack of support of these technologies in some browsers and devices such as mobile phones and tablets leave JavaScript as the best alternative. A large number of labs have been created in Java, and it still constitutes a viable technology for developing new labs as long as best practices are followed and applications are digitally signed. However JavaScript is the best option today. In addition, although the LMS provides security to all its content, the lab application should be developed taking into account aspects of web security to prevent an attack.

The integration level of a Learning Management System (LMS) with a WebLab can be classified according to the following proposed classification, which reflects increasing levels of integration:

- Type 1: Referenced Lab (Minimal integration). The lab is referenced through a link in the LMS and the lab GUI is hosted in a web page in some other location.
- Type 2: Host LMS. The lab is accessed through the LMS, but the LMS merely acts as a host for the WebLab. There is no communication between the LMS and the lab, which is executed completely independently of the LMS. This integration level can identify users who access the WebLab after they have been identified in the LMS.
- Type 3: Fully integrated VRL. The WebLab is located in the LMS itself or is implemented as a remote application



How to pass this Remote Laboratory?

The maximum score that can be achieved with these practices is 30 points.
You must reach at least 15 points to complete and pass the Remote Laboratory.
Once you have completed the practices in the laboratory you must press the "Finish" button that appears in the EJS application to store your score in the LMS.

FIGURE 1. Partial snapshot of a WebLab with advanced integration.

that might even be running on a mobile device. The lab software establishes communication with the LMS at runtime using a protocol that may have been developed ad hoc [15], [16] or may be based on an e-learning standard such as SCORM, the eXperience Application Programming Interface (API), Learning Tool Interoperability (LTI) [17], OpenSocial [18] or Computer-Managed Instruction 5 (cmi5) [19]. This integration level not only supports identification of students accessing the lab, but can also use this identification to customize lab execution based on the individual student; the work carried out by each student can be controlled; and self-evaluations can be performed and their results registered in the LMS [20].

For a lab to enjoy advanced integration with an LMS (type 3), a fundamental requirement is to establish communication between them. At the most basic level, this communication should allow the LMS to send the student ID and receive information regarding the work of the student in the lab. This information should be stored in an LMS database accessible to both the student and appropriate tutors/tutors. This communication between the lab application and the LMS can be implemented in three ways:

- Using an LMS extension that provides a communication interface that can be used by the lab software to access and manipulate the elements in the LMS database. Currently available LMS are quite heterogenic, so extensions are usually not generic; they must be developed specifically for each LMS, with the corresponding complexity and work burden. Another issue with this approach is that labs developed for use with a specific extension cannot be used with a different one. This limits labs to use only with the LMS for which they were originally designed.
- Using a standard supported by the LMS in the lab application that allows lab-LMS remote interaction. These standards can be implemented in any programming language and executed on any type of device, even mobile phones; it is not required to be located in the LMS, and a web browser is not required. There are several possibilities for implementation such as using the LTI specifications, xAPI (also known as Tin Can API or TCAPI), cmi5 or OpenSocial. LTI is not in widespread use except in some academic environments. It requires the proper use of LMS-compatible APIs during WebLab development [21]. The xAPI is more popular. The required implementation approach is similar to LTI, although xAPI requires using an LRS (Learning Record System) that must be integrated with the LMS [22]. Using cmi5 is currently not a viable approach, as the standard is still under development. However, it is a promising alternative for the near future, as it has been developed based on both SCORM and xAPI and will overcome some of the drawbacks of each. The Open Social standard, backed by the World Wide Web Consortium (W3C), would also allow LMS-lab data exchange, however, it is designed for more general social network communications; therefore, using an e-learning-specific standard seems more appropriate. Still, several works discuss using OpenSocial with labs [23], [24].
- Embedding the WebLab application in an LMS-supported element that implemented a communications interface for accessing and manipulating the LMS database. This scheme exposes LMS functionality to the WebLab application and is the proposed approach of this work. Just as SCORM content packages are capable of communicating with the LMS in which they are embedded, lab applications embedded in a SCORM

package have the same advantages. This approach also presents some limitations, because data interchange is limited to the SCORM data model. However, this approach does guarantee the reusability of the WebLab in any other SCORM-compatible LMS—and most of the currently available LMSs are SCORM compatible [25].

When a WebLab can communicate with the LMS it is embedded in and obtain the current user's ID, a range of possibilities opens. Customized execution of the lab software is simplified, and student-dependent frameworks and personalized experiments can be developed. When activity supervision is included and the work of each student is evaluated, lab performance and evaluation results can be stored in the LMS.

III. SCORM COMMUNICATIONS WITH LMS

SCORM is a set of standards arranged in three subareas: Content Aggregation Model (CAM), Sequencing and Navigation (SN) and Run Time Environment (RTE). CAM specifies the way that contents are described and packaged, SN defines the navigation between the different parts of the content, and RTE specifies how content should be launched and how to establish communication with the LMS. When a student logs into an LMS and opens SCORM content, the LMS manages the request, offering the objects that make up the content (Assets and SCOs) through HTTP. An SCO is the smallest logical unit capable of communicating with the LMS—usually a web page. The SCO can include assets, which are objects that do not communicate with the LMS on their own, such as videos, texts, sounds, etc. From a practical point of view, an SCO is usually implemented as a web page (HTML). The RTE data model of SCORM defines the information that can be shared between an SCO and an LMS: version 2004v4 also includes specifications for data that can be shared related to sequencing and navigation. The format for these data is predefined and standardized and includes, among others, student information such as their names and IDs in the LMS, preferences, personal comments, interactions between the user and the SCO, as well as completion and success statuses.

A set of predefined functions organized into an API enable an SCO to share SCORM data with an LMS. When an LMS loads a window into the SCORM browser it offers a Document Object Model (DOM) (a DOM structure defines the object organization of a web page) object instance as a means for enabling communication. This way, using the SCORM API functions, an SCO can store and retrieve information through the LMS. Each communication must be initialized by the SCO. The API contains 8 functions that can be grouped into 3 categories:

- Session functions that indicate the beginning and end of a SCO-LMS communication session: *Initialize()* and *Terminate()*, respectively.
- Data transference functions for retrieving or storing LMS data model information such as *GetValue(data)*, *SetValue(data, value)*, and *Commit(data)*.
- Helper functions used for auxiliary communications related to error management such as *GetLastError()*, *GetErrorString()* and *GetDiagnostic()*.

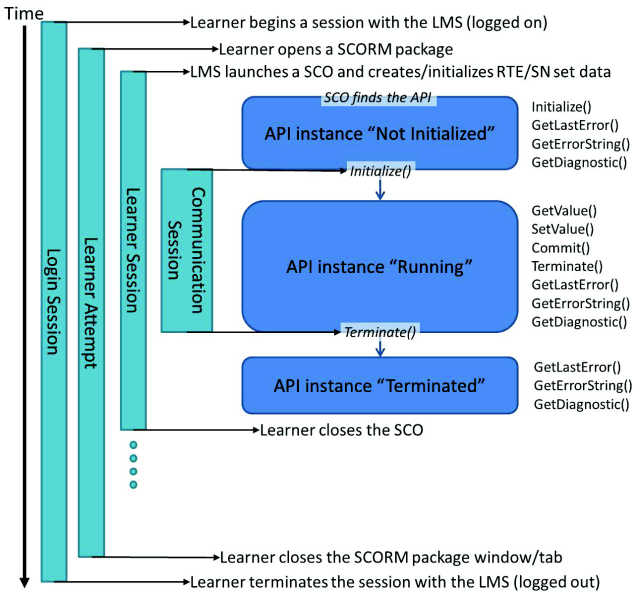


FIGURE 2. Example of a user session and API SCORM instance states.

The state model of a SCORM communication session defines 3 states for an API instance (Figure 2):

- Not Initialized. This state occurs after the SCO has been launched by the MLS and has found the API instance, but has not yet called *Initialize()*. The only functions available in this state are the helper functions and *Initialize()* itself.
- Running. This state extends from the point when the SCO successfully executes *Initialize()* until *Terminate()* is called. Every function except *Initialize()* can be called in the Running state.
- Terminated. This state is in effect from the point at which *Terminate()* is successfully executed. Only helper functions are available in this state.

Typically, a JavaScript file included in the main HTML page is used to facilitate finding the API instance and using its functions,. This file, called API Wrapper, usually encapsulates the 8 original SCORM API functions, offering simpler alternative functions. When a WebLab application is embedded in a SCO web page using JavaScript, the functions included in the API Wrapper file are directly available to the WebLab so it can retrieve LMS data and store data it generates (Figure 3).

Although conceptually simple, the communications programming of the communications is intricate. Each element of the data model must be known before it can be accessed. SCORM 2004v4 defines 24 high-level elements, many of which are composite and contain sub-elements that can appear in multiple ways. Similarly, the RTE model defines 69 elements and sub-elements in the RTE model,

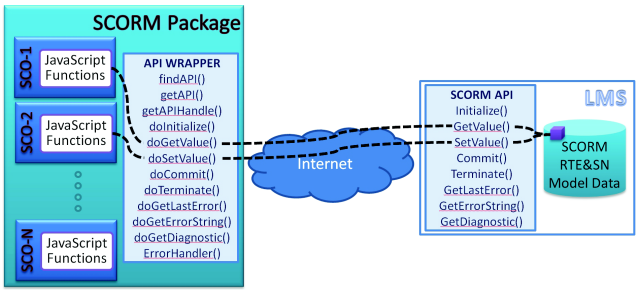


FIGURE 3. Using JavaScript functions from the SCO to manipulate SCORM model data.

while there are 5 in the SCORM Navigation model. For instance, the element "Comment from learner" allows the lab to store a character string in the LMS from the SCO. A SCORM-compatible LMS should allow storage of 250 comments, at minimum. Each comment is an element composed of the comment itself (string of characters), a localization that indicates the application area (string of characters), and the date and time when the comment was created or modified (SCORM2004 Timestamp). Furthermore, there is an element that defines sub-elements implemented by the LMS for the comments and another that quantifies the number of comments stored in the model database. In contrast, SCORM 1.2 includes only 49 items, but they have similar characteristics.

IV. TOOLS FOR LMS INTERACTION THROUGH SCORM

When using SCORM to create e-learning contents, some complications derived from the implementation of the 3 sub-specifications arise. This problem can be solved using templates that are modified to obtain the desired contents. The solution works well with the CAM and SN sub-specifications but, when the idea is to establish advanced interactions with the LMS, it is not that simple. Authors of this work have developed some tools and utilities that facilitate and simplify the use of advanced SCORM-LMS communications:

- Test SCORM packages: These contents test SCORM1.2 and SCORM2004 communications to know the model data supported by the LMS.
- Templates of SCORM packages: These templates are used to develop other contents from them.
- Java package (*scormRTE.jar*): It is used to facilitate the JavaScript code-LMS communications based on SCORM RTE.
- JavaScript library (*RTE.js*): It is used to facilitate the JavaScript code-LMS communications based on SCORM RTE.
- Help documents: Several documents containing summary tables of the elements of the SCORM models and related methods/functions showing their availability in SCORM2004 and SCORM1.2. These tables are helpful when using the *scormRTE.jar* package/*RTE.js* library to develop WebLab applications in Java/JavaScript, when adapting existing software, when implementing communications with the SCORM-based LMS and to switch between SCORM2004 and SCORM1.2.

These resources are available at:
http://dv.ujaen.es/docencia/goto_docencia_fold_682679.html

This site also includes advanced examples for integrating remote and virtual WebLabs. The most important resources that have been developed by the authors will be described in the next subsections: the scormRTE.jar Java package and the RTE.js JavaScript library.

A. THE scormRTE.jar JAVA PACKAGE

As discussed in the previous section, SCORM specifications allow access to the RTE and SN data models directly from JavaScript. However, direct access is not possible if the lab application was developed in Java. Instead, access requires a Java package called *Netscape.Javascript* that allows the Java program access to the SCORM API through the web page DOM [26]. A tool has been developed to simplify this process, consisting of a Java package called *scormRTE.jar* [27] that offers a wide range of resources for managing SCO-LMS communications from Java and direct access to the SCORM data model from the LMS to store and retrieve information. The advantages that this package offers are twofold. First, Java developers of the WebLab application do not have to employ JavaScript, because the classes offered by the *scormRTE.jar* package hide it. Second, the package simplifies access to the elements of the model, because it includes custom methods to access both simple and composite elements.

The only additional requirements for the student PC are the Java Runtime Environment (JRE) and a Java-compatible web browser. Figure 4 shows the UML for a deployment diagram after a user has been identified through the LMS and has accessed a SCORM module. The diagram includes the various entities in use on both the server and the client when a lab application has been embedded in a SCO of a SCORM module using *scormRTE.jar*.

The *scormRTE.jar* package includes two classes: *ScormRTE* and *ScormRTE12*; the first was developed to be used when the WebLab is embedded in a SCO developed using SCORM 2004v4, while the second is appropriate for SCORM 1.2. *ScormRTE* includes 74 variables related to the elements of the SCORM models, 30 of them are array types, two of which are two-dimensional. Of the variables, 69 are related to the 69 items of the runtime environment, while only 5 belong to the SCORM navigational data model. On the other hand, *ScormRTE12* includes 49 variables that are related to elements of the SCORM RTE model, 17 of these are arrays, of which 2 are two-dimensional arrays. SCORM 1.2 does not contain any sub-specification for sequencing and navigation, so all the variables represent items of the RTE model.

The methods exposed by these classes can be classified into 7 categories:

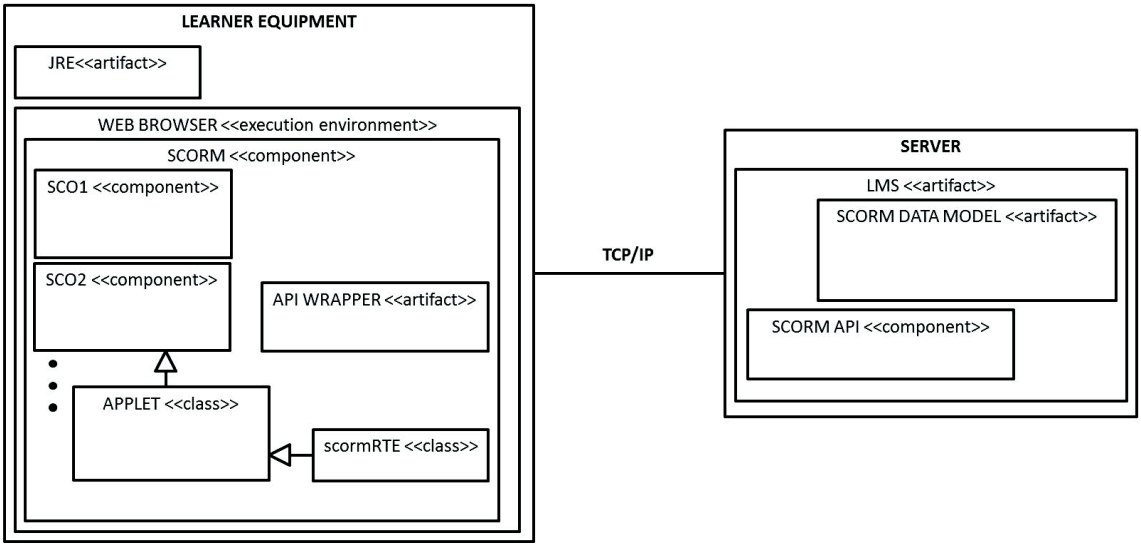


FIGURE 4. UML deployment diagram of an example using of the *scormRTE.jar* Java package.

1) METHODS FOR JavaScript INVOCATION (javaToRTE(data))

These methods are called from Java to invoke the JavaScript functions contained in the API wrapper including the SCORM session methods, the LMS data-retrieval and data-storage methods, and the support methods.

2) CONSTRUCTOR METHODS

These are Java constructor methods that allow the creation of *scormRTE* and *scormRTE12* objects before using their methods and variables. These methods, in addition to creating the objects, also initialize some of their properties with information retrieved from the LMS that might be required for the WebLab. The retrieved data are all constants and include the SCORM version, the name and id of the student, the method for accessing the SCO and the maximum time allowed to complete the SCO.

3) SCORM SESSION METHODS

These are methods that establish and end SCORM communication sessions for the embedded SCO. These methods are not required if the JavaScript session functions from the SCO are used instead. In this sense (between JavaScript and the *scormRTE.jar* package), any combination is supported. For example, the SCO can launch SCORM communication by calling JavaScript functions or through the Java lab application using the corresponding method in *scormRTE*, and either the SCO or the application can end the communication.

4) METHODS TO RETRIEVE DATA FROM THE LMS

These methods allow reading data from the SCORM model inside the LMS and use the JavaScript invocation method described above. For example, the method *doGetValue(data)* from the API Wrapper internally calls the function *GetValue(data)* from the SCORM API. All the functions are customized to retrieve a specific data type, thus simplifying their use and making it unnecessary to know all the model characteristics. Figure 5 shows an example of one of these methods, namely, *rteGetLearnerName()*. When this method is called from Java code in a lab application, the JavaScript method *doGetValue("cmi_learner_name")* is invoked from the HTML of the SCO in which the Java application is embedded. This function, in turn, calls the function *doGetValue("cmi_learner_name")* from the SCORM API provided by the LMS that launched the page. When *doGetValue("cmi_learner_name")* is executed, the LMS obtains the name of the student who logged into the LMS and accessed the SCO, and that name is assigned to the Java variable "name." In addition, a series of helper methods have been developed that—when called from Java—read the different items of an element from a single code line. This simplifies the Java code of the lab application. As an example, when the method *rteGetCommentById(id)* is invoked from a Java application, it calls the methods *rteGetCommentsLearnerTs(id)*, *rteGetCommentsLearner(id)* and *rteGetCommentsLearnerLoc(id)* to obtain the comment stored in the id position of the data model kept by the LMS, as well as its timestamp and location, in a manner similar to the example shown in Figure 3. These three data items are then stored in the corresponding variables of the *scormRTE* class instance used in the application.

5) LMS DATA STORAGE METHODS

These methods allow writing SCORM model data to the LMS. Similar to the data retrieval methods of the LMS, they use the JavaScript invocation method presented above. In this case, the methods execute the function *doSetValue(data, value)* from the Wrapper API, which calls the function

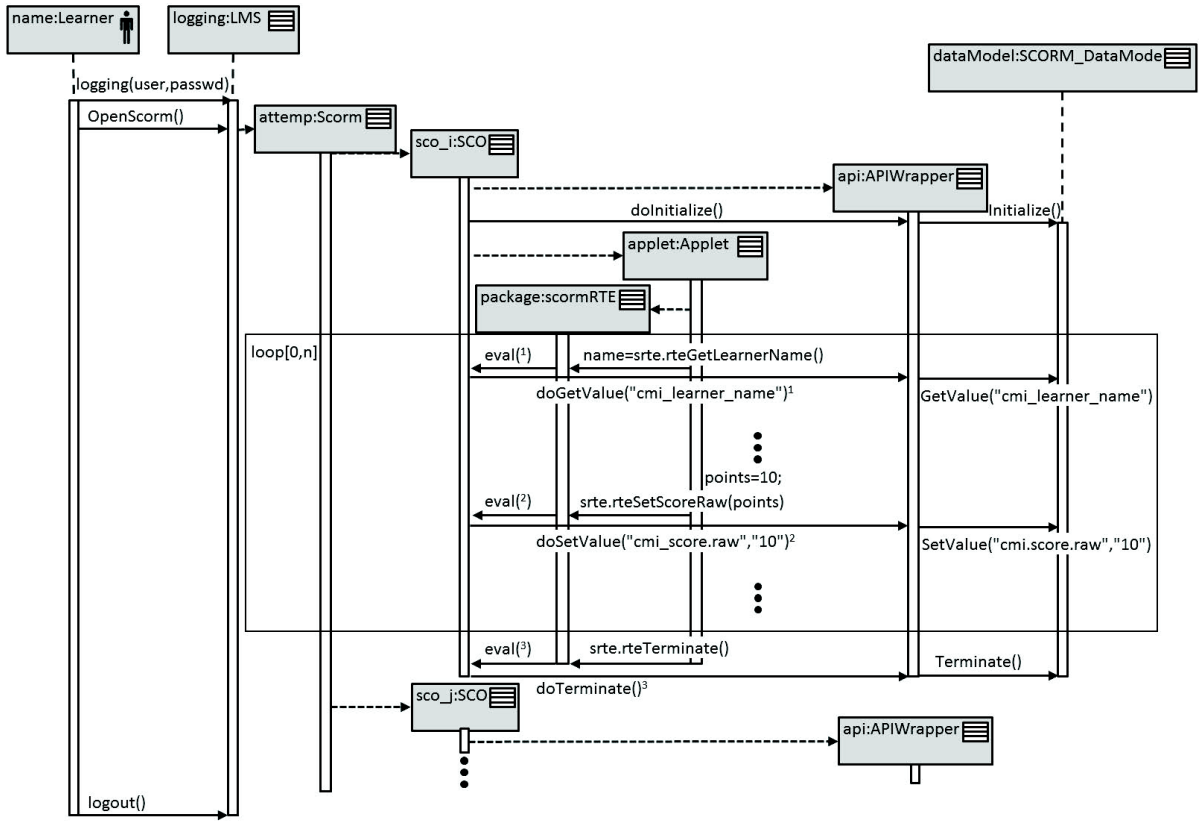


FIGURE 5. Example of UML sequence diagram with entities and their related methods.

SetValue(data, value) of the SCORM API to store the information generated by the Java lab application in the LMS. These methods are also customized to store specific data types from the SCORM data model, facilitating their use and reducing the need to know detailed characteristics of the data model. Figure 5 also includes an example of using data storage functions in the LMS from a Java applet embedded in a SCO. The example shows a case where the score attained by the student is points=10. To store this value in the corresponding element of the database managed by the LMS for the RTE model, the specific element of the *scormRTE.jar* package is called (*rteSetScoreRaw()*). This method calls the JavaScript *eval()* method to invoke the function *doSetValue()* of the Wrapper API, passing the parameter to be stored and its new value. The function *doSetValue()* calls the function *SetValue()* of the SCORM API provided by the LMS to store the value "10" in the element *cmi.score.raw*. Other methods in this category also simplify the process of storing composite elements of the RTE model in the LMS. These methods employ methods of exposed by the particular composite element classes to store subelements of the composite one. For instance, the method *setCommentsLearnerNew(comment, location)* calls the methods *setCommentsLearner(comment)*, *setCommentsLearnerLoc(location)* and *setCommentsLearnerTs()*, which store the comment of a user, the location related with the comment, and the timestamp, respectively.

6) METHODS FOR COMMUNICATION SUPPORT

These methods call support methods of the SCORM API that allow diagnostic and error checking of states generated during data transfers. These can be called from Java to check the state of the connection and the result of the data transfers.

7) DATA FORMAT AND COMMUNICATIONS HELPER METHODS

These methods simplify the handling of the data formats required by the SCORM models. For instance, the method *fechaHoraScorm()* returns a string with the current date and time in the SCORM 2004 format: "*yyy-MM-ddTHH:mm:ss*" or the time in the format used by SCORM 1.2: "*HH:mm:ss.SS*", depending on the class instance in use.

As shown in the last example, a uniform method name for both classes (*scormRTE* and *scormRTE12*) has been chosen. This allows the reuse, with only minor modifications, of the same WebLab in SCOs created using different SCORM versions (2004v4 and 1.2). The adaption from SCORM 1.2 to SCORM 2004v4 is simpler because it requires only changing the class declaration from *scormRTE12* to *scormRTE*. When the change is from SCORM 2004v4 to SCORM 1.2, in addition to changing the class type from *scormRTE* to *scormRTE12*, the code must be checked to find and replace SCORM 2004 capabilities that are not supported by SCORM1.2. This way, the WebLab programmer needs to be aware of and use

only the variables and methods of the *scormRTE.jar* package of the corresponding class.

B. THE RTE.js JavaScript LIBRARY

Java technology, including Java Applets, appeared in 1995. After twenty years many browser vendors have either announced timelines for the removal or removed standards based plugin support [28]. Nowadays the best alternatives to provide dynamic content to a Web page on the client side are the script technologies, which are based on source code embedded within an HTML document that are interpreted and executed by the user's web browser. The most famous, extended, used and recommended script language is JavaScript. Modern browsers are built with embedded JavaScript support. This means they are able to interpret and execute, directly on the client machine, the JavaScript code that is included in Web pages. For this reason, authors have developed the *RTE.js* JavaScript library. This library facilitates the integration of JavaScript code included in the Web pages of the SCORM package with the LMS where it is hosted. When the laboratory application has been developed using JavaScript, the *RTE.js* library can be used too. *RTE.js* works as a wrapper of the API wrapper; it simplifies the use and manipulation of SCORM data model. JavaScript is a powerful and flexible language that is not object-oriented but it allows emulate patterns and provides all the same object-oriented features [29]. *RTE.js* includes a function called *RTE* that has been defined to work as a class. The *RTE* function includes 83 variables and 146 functions that provide support for SCORM 2004v4 and SCORM 1.2, even though there are some version-exclusive elements (table 1).

TABLE 1. RTE function Elements.

SCORM versions	Model Data Direct relation	Variables	Functions
Common (SCORM2004/1.2)	Yes	45	71
	No	5	10
SCORM2004	Yes	29	60
SCORM1.2	Yes	4	5
	Total	83	146

To optimize the execution of the RTE.js library, the variables are declared but not defined until its first use. If a variable is not used in the SCORM session then it is not declared. Most of the variables have a direct correspondence with an element in the SCORM data model; there are only 5 auxiliary variables used internally that do not present this correspondence. There are 45 variables that are common for both versions of SCORM, 29 exclusive variables for SCORM 2004 and 4 exclusive variables for SCORM 1.2. 31 of the 83 variables are arrays, 13 of them are exclusive to SCORM2004 and the rest common to both versions. The 146 functions implemented in RTE perform tasks similar to those described in the previous subsection for the scormRTE.jar JAVA package and they can be classified in the same way. The authors have also created some additional auxiliary functions to simplify the communications programming code in the SCORM pages and the JavaScript laboratory applications.

The *RTE* function has only a parameter called "version" that indicates the SCORM version used in the code ("2004" or 1.2"). This allows to use the same code for both versions simply by changing the value assigned to the "version" parameter (Long as the developers had not used version-exclusive variables and functions, in this case they also have to make the changes involved with these elements). Figure 6 shows an example of the use of *RTE.js*.

V. USE CASE EXAMPLES

In the last few years, WebLabs developed with the tools presented in this work have been used at the University of Jaén, and are currently being migrated from Java to JavaScript. This section includes Java and JavaScript code examples from WebLabs used in several undergraduate engineering courses in the Escuela Politécnica Superior de Jaén. The Java code must be embedded as an applet in the web page of a SCORM SCO and the web page must have found the SCORM API offered by the LMS and started the connection with the LMS (function *doInitialize()* using the JavaScript API Wrapper). The JavaScript code must be in the web page of a SCORM SCO.

A. INITIALIZATION

Java: After defining the variables, an object of the class *scormRTE* must be created using the applet to be executed as a parameter and its default constructor. For example, if the objective is to create an object called *srte* in SCORM 2004, then the code is shown below:

```
1: srte = new ScormRTE (_getApplet());
```

For SCORM 1.2, the code is as follows:

```
2: srte = new ScormRTE12 (_getApplet());
```

The default constructor initializes some of the variables of the object that are now available as local variables.

JavaScript: The code is much simpler, simply a new RTE variable must be created indicating the SCORM version to use ("2004" or "1.2"):

```
1: var miRTE = new RTE("2004");
1: var miRTE = new RTE("1.2");
```

It is particularly important to obtain the student ID, because this allows customization of the WebLab's behavior and adaptive learning. Java code:

```
1: version=srte.version;
2: learnerName=srte.learnerName;
3: learnerId=srte.learnerId;
```

and JavaScript code:

```
1: version=miRTE.getVersion();
2: learnerName=miRTE.getLearnerName();
3: learnerId=miRTE.getLearnerId();
```

To obtain the login time to the WebLab, the value must be stored in a local variable using the following Java code:

```
1: dateTime=srte.fechaHoraScorm();
```

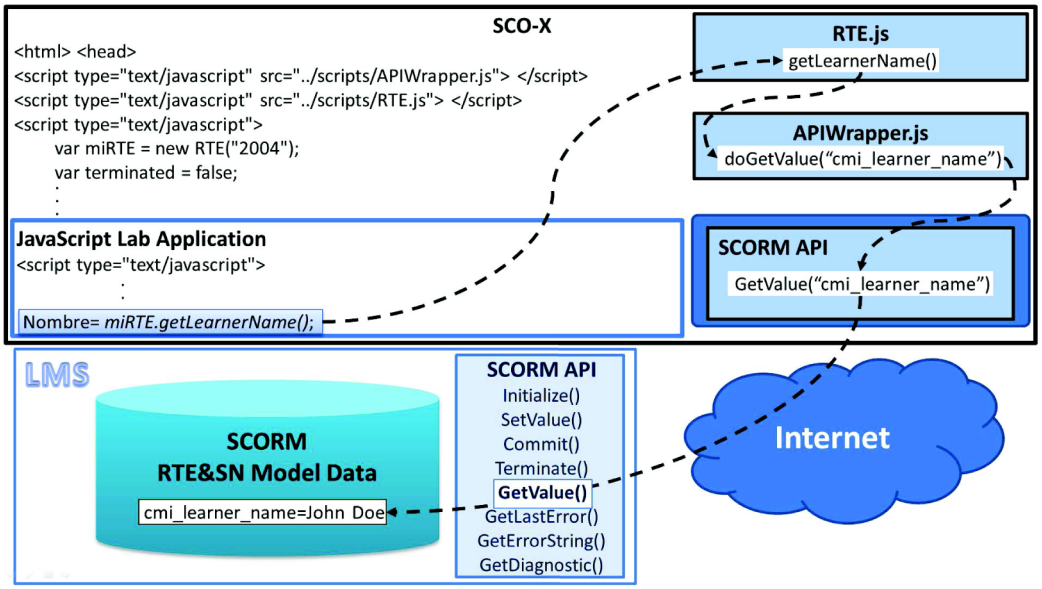


FIGURE 6. Example of JavaScript code embedded in a HTML SCORM page using the *RTE.js* JavaScript library to retrieve the name of the student from the LMS (it is stored in the cmi_learner_name element of the SCORM 2004 data model).

In JavaScript, there is a variable called *startTimeStamp* storing this information (*miRTE.startTimeStamp*).

B. SESSION DEVELOPMENT

SCORM includes elements in the RTE model for storing the following student preferences: language, content delivery speed, audio level and audio text. These elements of the data model, once obtained from the user, can be stored in the LMS. The following methods implement this functionality in Java:

1: srte.rteSetLearnerPreferenceAudio(audio);
2: srte.rteSetLearnerPreferenceLanguage(language);
3: srte.rteSetLearnerPreferenceSpeed(speed);
4: srte.rteSetLearnerPreferenceCaptioning (captioning);

This way, when a SCORM module allows multiple executions, subsequent executions can retrieve the student's preferences and execute the lab accordingly. The following initialization methods enable this functionality:

5*: audio=srte.rteGetLearnerPreferenceAudio();*
6: language=srte.rteGetLearnerPreferenceLanguage();
7: speed=srte.rteGetLearnerPreferenceSpeed();
8: captioning=srte.rteGetLearnerPreferenceCaptioning();

The *RTE.js* JavaScript library includes equivalent functions and, in addition, it includes two functions that can be used to set or get the user preferences all at once:

1*: miRTE.setLearnerPreferences(aud, lang, spe, capt);*
2*: p=miRTE.getLearnerPreferences(aud, lang, spe, capt);*

The user ID obtained during the initialization of the lab can also be used to slightly customize the experiments, so that they are not identical for all students and achieve student-adapted learning.

User comments can be used to store any textual information in the LMS. For example, they can be used to store partial work results and personal notes that can be retrieved later. Scorm1.2 supports storing only one comment object as a string, but it can be modified by appending more characters. Scorm2004 supports up to 250 string sub-elements, each of which has an associated timestamp and location. Depending on the SCORM version, the following functions are available for Java:

1: srte.rteSetCommentsLearnerNew (comment, location); //SCORM2004
1: srte.rteSetCommentsLearnerNew (comment); //SCORM1.2

and JavaScript:

1: miRTE.setCommentsLearnerAddAll (comment, location);

The stored comments can be accessed at a future time. To fetch a comment from Scorm2004, the comment ID is required. Three strings are returned in the specific arrays of the *scormRTE* object: the comment, the associated timestamp, and the location. The call to retrieve a comment is shown below for Java:

2: comment=srte.rteGetCommentsById (id);

and JavaScript:

2: comment=miRTE.getCommentsLearnerAll (id);

After this line executes, the comment (stored in the position specified by id), its corresponding timestamp and location can be accessed, respectively, using the following variables:

srte.commentsLearner[id], srte.commentsLearnerTs[id] and srte.commentsLearnerLoc[id] for Java and the *comment* variable (array) for JavaScript.

To fetch the comment from SCORM1.2, all that is required is to call the following function for Java:

2: comment=srte.rteGetCommentsLearner();

and JavaScript:

2: comment=miRTE.getCommentsLearner();

After this line executes, the stored comment is available in the variable *comment*.

SCORM2004 defines a sub-specification that controls sequencing and navigation between the SCOs that compose the SCORM package contained in the Java lab. This way, the next element to be shown to the user (e.g., SCO) can be determined based on the score obtained by the student when the active communication session ends (i.e., *Terminate()* is executed). The following Java code snippet defines the actions of the LMS after executing *Terminate()* based on the student's score: if it is less than 5, the whole SCORM package is shut down, otherwise the next SCO is presented:

```
1: navReq=(points<5)? "exitAll":"continue";
2: srte.rteSetNavRequest(navReq);
```

and JavaScript:

```
1: navReq=(points<5)? "exitAll":"continue";
2: miRTE.setNavRequest(navReq);
```

C. FINALIZATION

Before ending WebLab execution, it is convenient to store the result of the work of the student in the LMS so that the information is available for teachers and students afterwards. Typical stored information includes the score, whether the student has completed all the tasks and whether the student passed the lab. The following snippet illustrates how to achieve this for Java:

```
1: scoreRaw=srte.rteSetScoreRaw(points);
2: completionStatus="completed";
3: successStatus="passed";
4:completionStatus=srte.rteSetCompletionStatus (completionStatus);
5: successStatus=srte.rteSetSuccessStatus(successStatus);
```

and JavaScript:

```
1: miRTE.setScoreRaw(points);
2: completionStatus="completed";
3: successStatus="passed";
4:miRTE.setCompletionStatus (completionStatus);
5: miRTE.setSuccessStatus(successStatus);
```

The communication session between the SCO and LMS must be closed before stopping applet execution. This can be performed either from the SCO or from the Java applet, as explained above. However, it is best practice to close the communication from the Java applet as follows:

```
1: srte.rteTerminate();
```

To close the communication session from JavaScript this code must be used:

```
1: miRTE.terminate();
```

These code snippets were used to create several remote and virtual lab applications using Easy Java Simulations (EJS) [30]. These VRLs have been used in several different WebLabs. Two of these (''Modelling of a Dynamic System: DC Motor'' and ''PID Control of a Dynamic System: DC Motor'') have been implemented, along with some other resources, as pilot test cases for the course ''Industrial Automation.'' This course is taught in various undergraduate engineering programs at the Polytechnic School of Jaén. During 2014, 273 students used the WebLab ''Modelling of a Dynamic System: DC Motor''. Those who passed the requirements obtained an average score of 8.1 out of 10 (1.7 SD) in the automatic evaluation included in the WebLab. A survey filled in by the students on their opinion of the Weblab yielded an average score of 3.67 out of 5 (SD 0.96). In 2015, the students were split into two groups to work with two different versions of the WebLab ''PID Control of a Dynamic System: DC Motor''. This was an optional activity, but the WebLab was completed by 106 students and it showed a 30% knowledge gain. The survey yielded a score of 3.78 (SD 0.79) and 3.85 (SD 0.67) out of 5 for each version of the WebLab. All the evaluations performed in the pilot tests were quite successful, both in terms of learning effectiveness and student opinion of the WebLabs. A complete description and analysis of these examples can be found in [25] and [31].

VI. CONCLUSIONS AND FUTURE WORK

This work presented a set of innovative tools developed by the authors that simplify the communication between applications embedded in SCORM modules and the LMS that hosts them. The applications can be programmed using Java or JavaScript languages by using different tools: *scormRTE.jar* package and *RTE.js* library respectively. Standard SCORM-LMS communications allow the creation of interactive e-learning content customizable for each student and capable of storing student results. However, their use may be complex for the creators of teaching materials, because the SCORM-LMS communications demand advanced knowledge of Web programming and JavaScript. The proposed *scormRTE* Java package greatly simplifies the communication tasks by hiding the native JavaScript interface from the developer. The future of the scormRTE Java package is the *RTE.js* JavaScript library that offers the same features in a JavaScript environment and can be used by SCORM pages and laboratory applications. This paper presented various characteristics and communication requirements of the proposed tools and provided some examples and ideas for customizing the execution of WebLabs, tracking the work of the students, controlling access to the elements that compose the SCORM module and storing the final score of each student. This way the student-adapted learning is easier to achieve. Because the information is stored in the LMS, all the data generated by the WebLab can be made accessible to the appropriate students and teachers, facilitating their tasks and producing more effective learning. These elements illustrate the convenience of the tools in developing of new Java/JavaScript-based WebLabs or in adapting existing WebLabs to achieve adaptive learning thanks to the complete integration with the LMS.

Our Future work will include converting the Java-based WebLabs to JavaScript-based WebLabs.

REFERENCES

[1] Z. Nedic, J. Machotka, and A. Nafalski, "Remote laboratories versus virtual and real laboratories," in *Proc. 33rd Annu. Frontiers Edu. (FIE)*, vol. 1. 2003, pp. T3E-1–T3E-6, doi: 10.1109/FIE.2003.1263343.

[2] S. D. Bencomo, "Control learning: present and future," *Annu. Rev. control*, vol. 28, no. 1, pp. 115–136, 2004, doi: 10.1016/j.arcontrol.2003.12.002.
[3] L. Gomes and S. Bogosyan, "Current trends in remote laboratories," *IEEE Trans. Ind. Electron.*, vol. 56 no. 12, pp. 4744–4756, Dec. 2009, doi: 10.1109/TIE.2009.2033293.
[4] J. Chacón, H. Vargas, G. Farias, J. Sánchez, and S. Dormido, "EJS, JIL server, and LabVIEW: An architecture for rapid development of remote labs," *IEEE Trans. Learn. Technol.*, vol. 8, no. 4, pp. 393–401, Oct./Dec. 2015, doi: 10.1109/TLT.2015.2389245.
[5] M. Kalúz, J. García-Zubía, M. Fikar, and L. Čirka, "A flexible and configurable architecture for automatic control remote laboratories," *IEEE Trans. Learn. Technol.*, vol. 8, no. 3, pp. 299–310, Jul./Sep. 2015, doi: 10.1109/TLT.2015.2389251.
[6] R. Medina-Flores and R. Morales-Gamboa, "Usability evaluation by experts of a learning management system," *IEEE Revista Iberoamericana Tecnologias Aprendizaje*, vol. 10, no. 4, pp. 197–203, Nov. 2015, doi: 10.1109/RITA.2015.2486298.
[7] C. Gravier, J. Fayolle, B. Bayard, M. Ates, and J. Lardon, "State of the art about remote laboratories paradigms—Foundations of ongoing mutations," *Int. J. Online Eng.*, vol. 4, no. 1, pp. 1–10, 2008.
[8] L. de la Torre Cubillo, "New generation virtual and remote laboratories: Integration into Web environments 2.0 with learning management systems," Ph.D. dissertation, Dptm. Informatica Automatica, UNED, Madrid, Spain, 2013.
[9] N. Abdellaoui, C. Gravier, B. Belmekki, and J. Fayolle, "Towards the loose coupling between LMS and remote laboratories in online engineering education," in *Proc. Educ. Eng. (EDUCON)*, Madrid, Spain, 2010, pp. 1935–1940, doi: 10.1109/EDUCON.2010.5492439.
[10] H. Vargas, J. Sánchez, C. A. Jara, F. A. Candelas, F. Torres, and S. Dormido, "A network of automatic control Web-based laboratories," *IEEE Trans. Learn. Technol.*, vol. 4, no. 3, pp. 197–208, Jul./Sep. 2011, doi: 10.1109/TLT.2010.35.
[11] E. S. Ruiz, *et al.*, "Virtual and remote industrial laboratory: Integration in learning management systems," *IEEE Ind. Electron. Mag.*, vol. 8, no. 4, pp. 45–58, Dec. 2014, doi: 10.1109/MIE.2012.2235530.
[12] M. Ožvoldová and P. Ondrušek, "Integration of online labs into educational systems," *Int. J. Online Eng.*, vol. 11, no. 6, pp. 54–59, 2015.
[13] *Advanced Distributed Learning Initiative (ADL). SCORM 2004 4th Edition, 2009*. [Online]. Available: http://www.adlnet.gov/scorm
[14] Rustici Software. (2016). *Experience API*. [Online]. Available: http://experienceapi.com/
[15] J. Sáenz, J. Chacón, L. De La Torre, A. Visioli, and S. Dormido, "Open and low-cost virtual and remote labs on control engineering," *IEEE Access*, vol. 3, pp. 805–814, 2015, doi: 10.1109/ACCESS.2015.2442613.
[16] R. Bose, "Virtual labs project: A paradigm shift in Internet-based remote experimentation," *IEEE Access*, vol. 1, pp. 718–725, 2013, doi: 10.1109/ACCESS.2013.2286202.
[17] IMS Global Learning Consortium. *Learning Tools Interoperability (LTI)*. [Online]. Available: https://www.imsglobal.org/activity/learning-tools-interoperability
[18] M. Häsel, "Opensocial: An enabler for social applications on the Web," *Commun. ACM*, vol. 54, no. 1, pp. 139–144, 2011, doi: 10.1145/1866739.1866765.
[19] *Advanced Distributed Learning Initiative (ADL). cmi5 (Computer-Managed Instruction 5)*. [Online]. Available: https://adlnet.gov/adl-research/performance-tracking-analysis/cmi5/
[20] I. Ruano-Ruano, J. Gómez-Ortega, J. Gámez-Garcia, and E. Estevez-Estévez, "Integration of online laboratories—LMS via SCORM," in *Proc. IEEE Int. Conf Syst., Man, Cybern. (SMC)*, Oct. 2013, pp. 3163–3167, doi: 10.1109/SMC.2013.539.
[21] P. Orduña *et al.*, "An extensible architecture for the integration of remote and virtual laboratories in public learning tools," *IEEE Revista Iberoamericana Tecnologias Aprendizaje*, vol. 10, no. 4, pp. 223–233, Nov. 2015, doi: 10.1109/RITA.2015.2486338.
[22] H.-D. Wuttke, M. Hamann, and K. Henke, "Integration of remote and virtual laboratories in the educational process," in *Proc. 12th Int. Conf. Remote Eng. Virtual Instrum. (REV)*, 2015, pp. 157–162, doi: 10.1109/REV.2015.7087283.
[23] L. Tobarra *et al.*, "Creation of customized remote laboratories using deconstruction," *IEEE Revista Iberoamericana Tecnologias Aprendizaje*, vol. 10, no. 2, pp. 69–76, May 2015, doi: 10.1109/RITA.2015.2418011.
[24] P. Orduña *et al.*, "Generic integration of remote laboratories in public learning tools: Organizational and technical challenges," in *Proc. IEEE Frontiers Educ. Conf. (FIE)*, 2014, pp. 1–7, doi: 10.1109/FIE.2014.7044025.
[25] I. R. Ruano, J. Gámez-García, and J. Gómez-Ortega, "Laboratorio Web SCORM de control PID con integración avanzada," Revista Iberoamericana de Automática e Informática Industrial, Tech. Rep. RIAI-D-15-00066R2.
[26] Oracle. (2013). *Invoking JavaScript Code from an Applet*. [Online]. Available: http://docs.oracle.com/javase/tutorial/deployment/applet/invokingJavaScriptFromApplet.html
[27] I. R. Ruano, J. Gámez-García, and J. Gómez-Ortega, "Building SCORM embedded WebLabs with LMS interaction," in *Proc. IEEE Frontiers Edu. Conf. (FIE)*, Oct. 2014, pp. 1–4, doi: 10.1109/FIE.2014.7043993.
[28] B. Evans, *Java: The Legend*, vol. 53. Sebastopol, CA, USA: O'Reilly Media, 2015.
[29] R. Harmes and D. Diaz, *Pro JavaScript Design Patterns*. New York, NY, USA: Apress, 2008.
[30] F. Esquembre, "Using easy Java simulations to create scientific simulations in Java," in *Proc. IEEE Region 8 Comput. Tool (EUROCON)*, vol. 1. pp. 20–23, Sep. 2003, doi: 10.1109/EURCON.2003.1247971.
[31] I. Ruano-Ruano, P. Cano-Marchal, J. Gámez-García, and J. Gómez-Ortega, "PID control WebLab with LMS integration using SCORM," *IFAC-PapersOnLine*, vol. 48, no. 29, pp. 301–306, 2015. doi: 10.1016/j.ifacol.2015.11.252

ILDEFONSO RUANO received the degree in telecommunication engineering from the University of Málaga, Spain, in 1996. Since 1996, he has been with the Telecommunication Department, University of Jaén, Jaén, Spain. His current research interest includes e-learning, online virtual and remote laboratories, learning management systems, and engineering education.

PABLO CANO received the degree in electrical engineering from the University of Seville, Seville, Spain in 2007, and the Ph.D. degree from the University of Jaén, Jaén, Spain, in 2015.

He was a Visiting Researcher with the Department of Automatic Control, University of Lund, Lund, Sweden, in 2013. Since 2015, he has been with the System Engineering and Automation Department, University of Jaén, as an Assistant Professor. His current research interests include automatic control applied to olive oil and elaboration process and education in engineering.

JAVIER GÁMEZ received the degree in electrical engineering and the Ph.D. degree from the University of Jaén, Jaén, Spain, in 2001 and 2006, respectively. From 2003 to 2004, he was a Visiting Researcher with the Department of Automatic Control, University of Lund, Lund, Sweden. Since 2005, he has been with the System Engineering and Automation Department, University of Jaén, as an Assistant Professor. He is currently the Director of the Transfer and Entrepreneurship Secretariat and the Director of the Research Results Transfer Office (OTRI). His current research interests include force control and sensor fusion in robotic manipulators, engineering education and automatic control applied to olive oil and elaboration process. Dr. Gámez was a recipient of a Formacion de Profesorado Universitario grant from the Spanish Ministry of Education and Science in 2002.

JUAN GÓMEZ (M'10) received the degree in electrical engineering and the Ph.D. degree from the University of Seville, Seville, Spain, in 1989 and 1994, respectively. In 2001, he became a Professor with the University of Jaén, Jaén, Spain. From 1987 to 2001, he was with the Departamento de Ingeniería de Sistemas y Automática, University of Seville, as a Research Assistant, then an Assistant Professor, and an Associate Professor. In 2001, he became a Professor with the University of Jaén, where he is the Rector. He has been responsible for several research projects on robotic systems, computer vision applied to fault detection, and automatic assembly, with some of them being directly applied to industry. His research interests include force control and sensor fusion in robotic manipulators, sensor planning, mobile robot and Education in Engineering. He has been serving as a Reviewer for different technical journals.