A Middleware for the Integration of Third-party Learning Tools in SOA-based Learning Management Systems

Supporting Instance Management and Data Transfer

Jorge Fontenla González, Manuel Caeiro Rodríguez, Martín Llamas Nistal
Department of Telematic Engineering, University of Vigo
ETSE Telecomunicación, Campus Universitario
36310 Vigo, Spain
{jfontenla, manuel.caeiro, martin.llamas}@det.uvigo.es

Elio Sancristobal, Manuel Castro
Electrical and Computer Engineering Department
UNED - Spanish University for Distance Education
Madrid, Spain
{elio, mcastro}@ieec.uned.es

Abstract— **The widespread adoption of broadband Internet connections and the need of institutions such as universities or enterprises to provide their staff with continuous education have led to a fast adoption of Learning Management Systems. These systems typically provide a centralized environment where students can communicate, carry out experiments, etc. However, the rapid growth of these platforms together with the unlimited need for learning tools, mainly in engineering education contexts (e.g. simulators, communication tools), are hindering their development. The natural alternative is to decouple these tools from the Learning Management Systems themselves, taking advantage of the *Software as a Service* distribution model. To perform such a decoupling a middleware is required to allow the integration and use of an external tool by the Learning Management System. In this paper a proposal for such middleware is described, with a special focus on the part devoted to manage instances and the transfer of data.**

Keywords- Data-transfer interface; Hard integration; Instance Interface; Learning Management Systems; Middleware; Software as a Service.

I. INTRODUCTION

Engineering education is experiencing great changes during the last years. Many of these changes have been promoted by the adoption of new technologies, broadly Web-based applications. Learning Management Systems (LMSs) are playing a key role in this scenario. Some of the best-known examples are Moodle [1], Blackboard [2] and .LRN [3]. These systems typically facilitate the control of educational activities, providing a centralized environment to organize and provide information, to support the communication between teachers and students, to enable the interchange of documents, to answer online questionnaires, etc. Nevertheless these LMSs are too generic. The "one size doesn't fit all" problem is very notorious here, since their functionalities are generally designed for the support of a general educational approach based on the delivery of contents. Nevertheless, many engineering subjects require not just the delivery of contents, but also the performance of experiments, practical developments and collaborative works among students. During the last years these activities have been supported in several ways by technology-based solutions developed outside of the LMSs: simulators, remote labs, agent-based environments, games, immersive environments, etc. As a result, a plethora of tools and services is currently required for the support of engineering education e-learning courses, but in general they are not available in LMSs.

The previous problems have been identified as LMSs' tailorability and extensibility deficiencies that need to solved. Up to date, some solutions have been proposed, but with limited success. For example, Moodle and Blackboard have capabilities to extend their own functionalities: the so-called "extensions" [4]. However, in these systems integration of external tools is considered only a supplement. As a result, it is possible to include a new tool in Moodle or Blackboard, but the integrated tools do not work in coordination with the core LMS. On the one hand, LMSs lack any means to monitor and control the way users work with tools. This is the case, for example, when we want to track students' activities. On the other hand, LMSs are not able to control the interaction of the users with the external tools.

The tailorability and extensibility deficiencies found in existing LMSs, together with the essential need of extensions, have led us to the conception of a middleware to enhance and facilitate the integration between LMSs and third-party tools. This middleware is based on a *Software as a Service* (SaaS) distribution model that allows the LMS to use third-party tools exposed as Web Services. In this paper we give an overview of the different communication protocols, components and software stacks involved. A comprehensible description is out of the scope of this article, and therefore we put the focus on two key parts of the middleware: the Instance Interface (those elements to create and manage the instances of a third-party tool), and the Data-Transfer Interface (those devoted to

transfer data elements between the LMS and the tool). Their development has been based on the study of existing solutions and the application of a rigorous analysis of requirements.

This paper is organized as follows. Section 2 analyzes different ways in which LMSs can be extended according to the level of functionality achieved, and puts the data-transfer middleware in this context. Section 3 poses a typical scenario where the need of creating and configuring instances and transferring data between a LMS and a tool is clear. Next, Section 4 gives a brief description on the middleware and the way it has been structured to accomplish integration of third-party tools, and Sections 5 and 6 focus specifically on the Instance Interface and the Data-Transfer Interface. Section 7 provides a proof of concept, and Section 8 ends up with some conclusions.

II. Integration of Third-Party Tools

The basic problem of this research has been how to develop an LMS whose functionality can be extended at a minimum cost. In addition, the new functionality has to be appropriately integrated with the previous one of the LMS. This problem can be considered in a broader context, as the general problem of extending a Web application.

At this point we have considered two different alternatives to integrate third-party tools in a Learning Management System, which are also considered in [5]:

- Soft integration of third-party tools. The LMS functionality can be extended through a hyperlink to an (external) third-party component. Once the user clicks on it, the graphical user interface of the tool is displayed. From this point, users are operating a system that the LMS cannot control by any means. Therefore, a new functionality is included but it does not work in coordination with the core LMS, resulting in a very "soft" integration.
- Hard integration of third-party tools. It includes the soft integration, but providing the LMS with comprehensible control over the integrated tools. We describe in the next paragraphs our proposal for such a comprehensible control.

In soft integration a third-party application can be "inserted" in the page of the LMS, providing its users with a unified environment to carry out their tasks. However, the only part of the application that can be controlled by the LMS is a link to the tool. The LMS does not have any means to supervise and alter the behaviour experienced by its users on the application. This functionality can be enough in some cases, but not always. Hard integration, on the other hand, allows the LMS not only to link the application, but also to supervise and alter the behaviour of the tool as required.

As discussed in [6], the control of the operation of the tool to achieve hard integration involves the following issues:

1. Creating a working account (i.e. an instance) for each user at the tool. For example, in a "Hydrodynamics" course an instance can be created for a student at a fluids simulator.
2. Transferring from the LMS to the tool all those data that the user may need in order to carry out his/her tasks. In the case of a fluids simulator, the LMS may send the boundary conditions required for the hydrodynamics problem.
3. Establishing some access permissions over these data and the tool functionality. In our example, the student may be assigned execution permissions over the part of the application responsible for launching the simulation.
4. Subscribing to events result of the manipulation of the tool. For example, the LMS may be interested in knowing when the student launches a simulation, and hence it has to subscribe to the corresponding event type.
5. Authorising the user to access the instance. In our example, the student may not have a working account at the fluids simulator, in whose case the LMS has to grant him/her access as guest user. Otherwise, if the student is not involved in the subject he/she should not be granted access.
6. Altering the behaviour of the tool according to the information provided by the events triggered. For example, if the result of the fluids simulation is correct the LMS may order the simulator to give a verbatim explanation of the physical laws involved in the problem.

Figure 1 summarizes the difference between hard and soft integration in terms of the six aspects mentioned above.

III. An API for the Integration of Tools

The highest level of integration can be provided by the architecture depicted in Figure 2. The picture corresponds to a refinement of the TCP/IP protocol stack. Unlike the typical TCP/IP stack, the Application layer is further decomposed into three additional layers:

- **High-level entities**. The LMS and the third-party tool. They provide the bulk of the learning functionalities, but employ the integration managers to interact and complement each other.
- **Integration managers**. A set of classes and interfaces used by both high-level entities that enable to control and supervise the behaviour of the third-party tool by the LMS. In other words, these integration managers carry out tasks that enable to achieve hard integration. Each integration manager deals with a different task.
- **Integration protocols**. A set of protocols that allow the integration managers at the LMS and the tool to communicate.

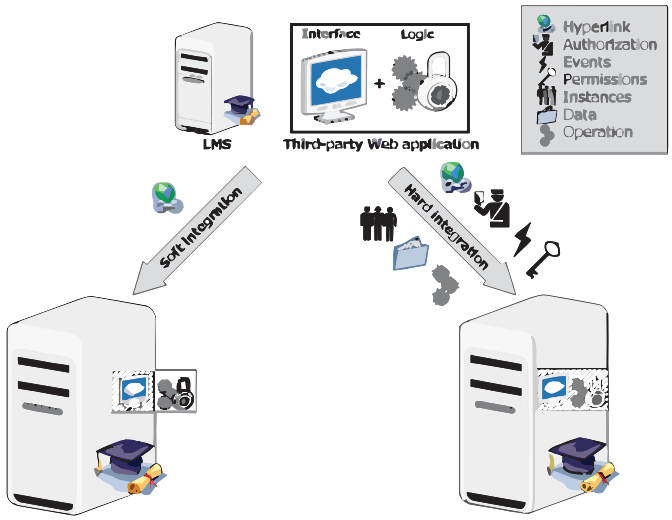

Figure 1. Difference between hard integration and soft integration.

As in the standard TCP/IP stack, each layer perceives a direct communication with an analogous layer at the remote host. Therefore, integration protocols communicate with analogous integration protocols, integration managers communicate with analogous integration managers, and high-level entities communicate with high-level entities, see Figure 2.

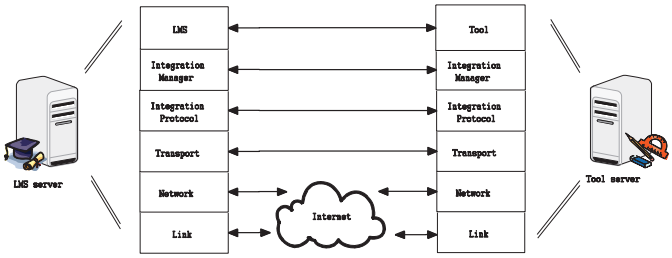

Figure 2. Hard integration architecture.

The conceptual layer diagram of Figure 2 is complemented by a class diagram representing the classes, components and interfaces that build up the layers of both the LMS and the Tool. Figure 3 provides such diagram just for the case of LMS. The diagram of the Tool is symmetrical due to the abovementioned property of the TCP/IP protocol stack of direct communication between layers.

In this diagram the high-level entity is represented as a UML component without any further class decomposition. The reason is that a high-level entity is a complex system, involving many classes and packages whose relationships are not relevant for our purposes.

Beginning with the diagram of Figure 3, the LMS invokes the methods of the integration managers to alter the behaviour of the tool (e.g. a method for granting write permissions over a file, for subscribing to a particular event). Each integration manager deals with a different aspect of hard integration, up to the six different aspects enumerated in Section 3.2. The LMS does not have to worry about the code structure of the integration managers but only about the methods they provide and the functionality of each of them. Therefore, each integration manager realizes an interface, which is exposed to the LMS.

The integration managers of the LMS use the integration protocols to communicate with the tool. The integration protocols, both those at the LMS or at the tool, carry out the effective exchange of messages, dealing with the logic of the messages sent between both entities (sequencing, detection of invalid messages, etc.).

When the message has been delivered to the integration protocol of the tool it is passed to the corresponding integration manager, which alters the behaviour of the tool.

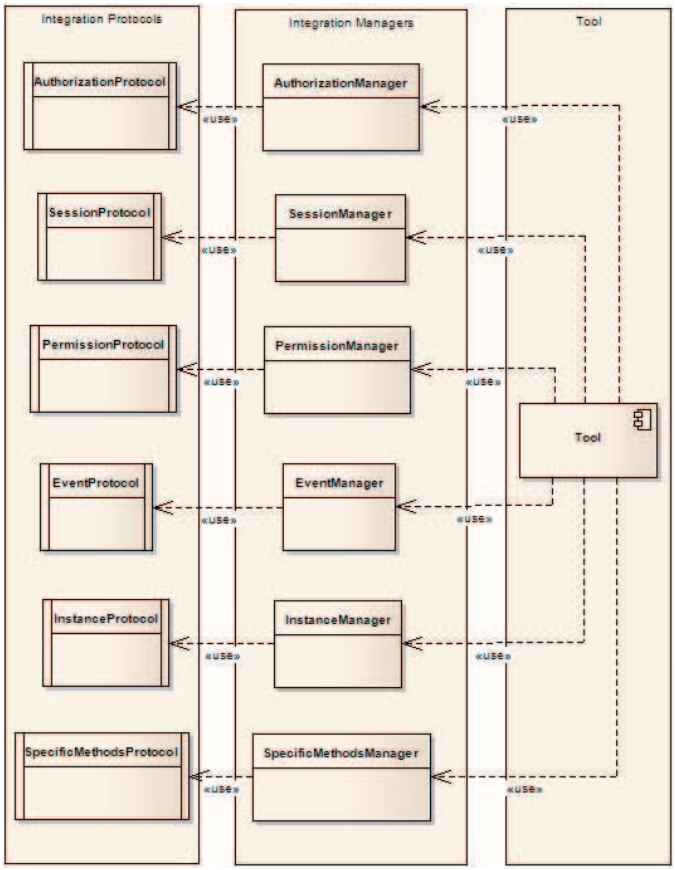

Figure 3. UML class diagram of the LMS.

In this article we will describe only the Instance Manager-Protocol and the Data-Transfer Manager-Protocol. Taking this framework as a starting point, the only missing points are the following:

- Firstly, an enumeration of the methods supported by the Instance Manager and the Data-Transfer Manager.
- Secondly, a description of the way the Instance Protocol and the Data-Transfer Protocol work.

In order to specify the functionalities of both pairs Manager-Protocols (or simply “Interfaces”) we will firstly describe a typical use case in Section IV involving the creation

and management of instances in a remote tool, and the transfer of data between a LMS and the tool. This use case will help us to develop the abovementioned logic in Sections V and VI. The descriptions carried out in Sections IV, V and VI follow a constructive approach, i.e. they represent the same creation process taken by the authors when developing the full middleware.

IV. A TYPICAL SCENARIO

For our purposes (defining the Instance Interface and the Data-Transfer Interface) we will consider a LMS hosted and managed by the University A. This LMS offers its students the possibility to use a collaborative tool hosted and maintained by the University B.

Students of a certain course in a Telecommunications Engineering degree use the University A's LMS. At a certain part of the course, students are expected to work in groups to accomplish a task. The tool at University B provides the desired functionalities. Therefore, the LMS creates one instance of the tool per working group. The work of one group should not interfere with another group's (e.g. a group should not cheat trying to copy another group's work). In this way, when the members of the group get access to the tool they join their particular instance where they find a copy of the wording of their assignment along with appropriate documentation.

At a certain point of the schedule the teacher wants to qualify the pieces of work created by each group. At this time, the access to the instances needs to be blocked. When the teacher finishes the qualification instances are unlocked again to enable students continue working. This unlock may be followed by the transfer of the wordings of new assignments.

During all the process it is feasible that some students may have some tasks validated by previous activities. In this case they do not need to perform the assignments and therefore their membership to their respective groups should be revoked. Similarly, some new students may be assigned to some existing group by some reason.

Suddenly, some unexpected behaviour is experienced at the tool provided by the University B (e.g. server crashes). In view of this fact the LMS requests a full backup of all the data generated by the groups. If University C provides a tool with similar functionalities to University B's the LMS creates new instances in University C's tool for each group and restores the backup copies in them.

In sections V and VI we use this scenario to explain the requirements we have identified for the Instance and Data-Transfer Interfaces. We adhere to a strict software engineering approach, firstly identifying the requirements of a solution, then analyzing existing technologies against these requirements, and finally coming up with a proposal that overcomes the identified limitations.

V. THE INSTANCE INTERFACE

The first of the interfaces that will be described is the Instance Interface. This Interface is devoted to the control and management of the instances of a tool. We understand by *instance of a tool* a working environment along with a graphical user interface, several files to manipulate, and a set of users allowed to access it.

A. Requirements

On the basis of the use cases included in the scenario described in Section IV we can enumerate the following requirements for a data-transfer solution:

- **Interoperability**: the LMS should interoperate with a Web tool even if they are in different network domains. This is necessary as in the general case the Web tool and the LMS are located at different network domains.
- **User-oriented instances**: as pointed in Section IV, it should be possible to add specific users to specific instances of the tool. Particularly, depending on the kind of tool one instance could be assigned one or many users.
- **Disjoint instances**: the membership of one instance should not affect either the membership or the state of other instances.
- **Dynamic reconfiguration**: it should be possible to remove participants from an instance or add new ones during runtime.
- **LMS-controlled membership**: it is the LMS which decides which user is assigned to each instance of the tool. In Section IV it is the LMS which creates the working groups.
- **LMS-controlled instances**: the LMS should be able to create, delete, suspend temporarily and resume instance depending on the requirements of the course. In Section IV the LMS suspends the instances during the teacher's evaluation.

B. Related Works

Previous to the development of an interface we perform a study of the state of the art. Unlike other interfaces such as the Data-Transfer Interface in Section VI or the Authorization Interface [7], the study of the state of the art in the field of instance management did not bring out significant results. There are not systematic approaches to control and manage instances, which in part is understandable due to the vast diversity of tools and requirements involved.

Given this heterogeneity we decided to concrete our research and investigate the state of the art of the most common learning tools [8]. This research has been fruitless in many cases, but in the fields of chats and conferences we came across several specifications to define how to create instances and manage them once created.

RFC2811 [9] addresses the creation and control of chat rooms or "channels" (i.e. concrete instances of the chat tool). In its simpler version, new channels are created whenever its first user joins it. This user is assigned the role of "channel founder" and has unlimited privileges over the channel. Since the channel is created other users are free to join it unless the founder explicitly puts them in a black list. Other parameters that can be set by the founder include silencing the room, setting a welcome message, or establishing a user limit for the

channel. When all users have left, the channel is finally deleted.

RFC4353 [10], on its turn, specifies how to use the SIP protocol in order to initialize, modify and terminate media sessions between multiple participants or "conferences". The RFC regards a specific conference as an instance of a multi-party conversation. The document addresses common instance operations (creating and destroying an instance, and adding, removing and listing participants) and specific conferencing operations (adding and removing media, and recording a conversation).

Table I summarizes the behaviour of these two specifications against our requirements.

TABLE I. COMPARATIVE ANALYSIS OF RFC2811 AND RFC4353.

	RFC2811	**RFC4353**
Req. 1	YES	YES
Req. 2	YES	YES
Req. 3	YES	YES
Req. 4	YES	YES
Req. 5	NO	NO
Req. 6	NO	NO

We see that these RFCs have been written to describe services that, although they behave well against our requirements, they have not been thought to be operated by a third-party system but directly by (one of) its users. Therefore, our Instance Manager offers similar functionalities to these RFC, but providing the LMS with full control of the creation, deletion and state of the instances. Eventually, this control could be transferred by the LMS to the users.

C. The Instance Manager

As described in Section III, the approach we have followed is the decomposition into a Instance Manager providing a known interface, and a Instance Protocol responsible for the actual transfer of data (see Figures 2 and 3). Both the Manager and the Protocol have to be implemented at both the LMS and the tools. Table II summarizes the methods and input and output parameters of the Instance Manager.

TABLE II. METHODS OF THE INSTANCE MANAGER.

Method	**Input parameter**	**Output parameter**
createInstance	name	URI
suspendInstance	URI	result
resumeInstance	URI	result
addUser	URI, userID	result
removeUser	URI, userID	result
deleteInstance	URI	result

Instances are uniquely identified by its URI (Unified Resource Identifier). All operations receive the URI of the instance as an input parameter (except the method for creating an instance, which returns the URI of the new one). The field *result* contains a verbatim message containing the success or not of the operation (e.g. "OK", "Error - Instance not found"). Finally, each LMS is responsible for assigning its users a unique *userID* to avoid colliding usernames at the tool.

D. The Instance Protocol

The specifications described in Section V-B describe complex and very specific protocols to manage instances from an external entity, which do not adapt to our scenario. Therefore, a light Instance Protocol has been specifically designed to work with the Instance Manager.

The Instance Protocol is based on the exchange of plaintext HTTP messages with specific headers. All the message exchanges take place exclusively between the LMS and the tool (i.e. the final user of the tool does not take part in them). HTTP is specially suited for the design of a client-server interface such as the Instance Interface, as it offers the capability to send parameters in the header of the messages.

The format of the headers follows a simple structure. Each parameter goes in a different extension header. For example, should the parameter URI be sent, the corresponding header would be something like *URI: www.mylearningtool.net/specificinstanceuri*. Additionally, there are two more headers. On the one hand, the *InstanceMethod* header specifies the method of the Instance Manager that is invoked by the LMS. On the other hand, the *RequestID* header identifies the request from any other request from this or any other LMS. This is useful for the LMS to match the incoming response with a previous request. Figure 4 depicts a sample message of the Instance Protocol.

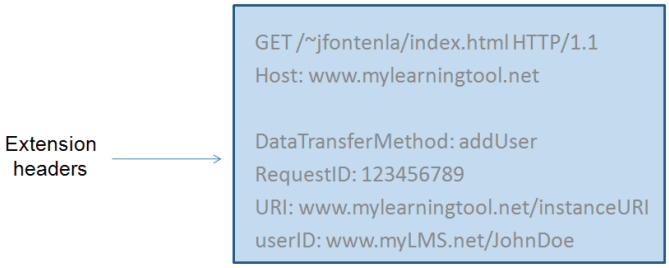

Figure 4. Sample message of the Instance Protocol.

VI. THE DATA-TRANSFER INTERFACE

In this section we describe the Data-Transfer Interface. As described in Section III, the approach we have followed is the decomposition into a Data-Transfer Manager providing a known interface, and a Data-Transfer Protocol responsible for the actual transfer of data (see Figures 2 and 3). Both the Manager and the Protocol have to be implemented at both the LMS and the tools, in order to allow the virtual direct communication among layers described in the TCP/IP protocol stack. It has been our intention to build a simple and

lightweight solution that could work in any kind of learning environments where two entities need to transfer data.

The structure of this section is analogous to that of Section V. Starting with an analysis of the requirements, we list some related works and study their behaviour against our requirements. Next, we get into some detail of a Data-Transfer Manager and a Data-Transfer Protocol solving the previous limitations.

A. *Requirements*

As with the analysis of requirements carried out for the Instance Interface in Section V, On the basis of the use cases included in the scenario described in Section IV we can enumerate the following requirements for a data-transfer solution:

1. **Interoperability**: the LMS must be able to interoperate with a Web tool even if they are in different network domains.
2. **Bidirectionality**: the solution must enable to transfer data either from the LMS to the tool, or from the tool to the LMS. Moreover, the communication can be started by any of these entities. For instance, in the example of Section IV the tool could fetch from the LMS an avatar for each student.
3. **Granularity and bulkability**: the solution should accept the transfer of single data elements (Granularity) as well as the transfer of multiple data elements in a single message (Bulkability). In the use case of Section IV, for example, transfers involve both single data elements (the wordings of the assignments) and full backup copy.
4. **Scheduled and on-demand transfers**: the transfer of data can be made either on demand, or in a scheduled fashion. In the latter case, it should be possible to specify at least the time of the transfer, and which data elements are going to be sent. For instance, in the scenario of Section IV the LMS could request backup copies from the tool of University B every hour.
5. **Bandwidth efficiency**: the solution should not be bandwidth intensive (e.g. in the case of frequent bulk copies). This includes the sending of incremental backups, including just those elements that changed since the previous copy.
6. **Transparency**: the user should not be aware of the data-transfer process that is taking place while he/she is using the tool.
7. **Integrity**: in the general case, the data being transferred is not fault-tolerant. Hence, the solution should provide some mechanisms to check the consistency of the data.
8. **Confidentiality**: the solution should provide mechanisms to treat the data as confidential when necessary.

B. *Related Works*

During the course of our work we studied several standards and technological products devoted to the transfer of data between two entities. Some of these products have relationship with the e-learning field, while some others are for general purpose. Aware that this is a very wide category, we chose the following three representative technologies for study: SCORM RTE, IMS SSP and W3C's Storage interface.

ADL Shareable Content Object Reference Model [11] (henceforth SCORM) is an effort to provide a set of specifications, guidelines and standards to meet the requirements for the Web-based delivery of reusable learning contents. One of the results of this effort is the SCORM Runtime Environment [12] (henceforth SCORM RTE), which details the requirements for launching content objects (henceforth SCOs) and establishing communication between LMSs and SCOs. The RTE is a middleware that is downloaded to the learner's Web browser along with the SCO itself. Whenever a SCO wants to begin a data transfer with the LMS invokes the *getValue* or *setValue* methods of the RTE. All the transfers are initiated by the SCO.

There are two main deficiencies in SCORM RTE that avoided us from choosing it as our data-transfer solution. Firstly, it provides a very limited set of methods, as it only allows the transfer of single data elements and not full backups. Secondly, it has been designed with a very concrete and simplistic scenario in mind (namely, a SCO making requests to a LMS following a client-server approach) which is slightly different from ours. We are considering tools in general and not SCOs uniquely.

IMS Shareable State Persistence [13] (henceforth IMS SSP) has been posed as an extension of SCORM to support the storage and retrieval of information from shared dataspaces called "buckets". The main goal of IMS-SSP has been to provide SCORM's SCOs the ability to share information among them which, according to the IMS, would allow to make more reusable content objects.

On addition to the methods *getValue* and *setValue* of SCORM RTE, IMS SSP considers the methods *appendData*, *getData* and *setData* that work over buckets. Unlike the methods of SCORM, these three methods allow not only to transfer single data elements but also collections of them with a single call. Nonetheless the operation of IMS SSP is similar to SCORM RTE's: it has been designed for scenarios where SCOs makes requests to an LMS by means of a runtime, and the LMS consequently replies to the request. There is no kind of bidirectionality between the entities involved. Other requirements of Section VI-A such as Bandwidth efficiency, Integrity or Confidentiality are neither considered.

Finally, W3C Storage [14] is a mechanism designed for storing user's data in general-purpose Web applications due to the W3C Consortium, allowing persistent storage lasting the current browser session. Storage has been developed as an interface to access a standardised set of methods in the Web application. These methods include *getItem*, *setItem*, *removeItem* and *clear* to operate over data elements represented as key-value pairs (both keys and values are strings). The invocation of these methods is carried out by some JavaScript code running in the browser. Despite Storage is designed for a general-purpose scenario involving a Web

application and a browser, its working principles remain the same with regards to SCORM RTE and SSP. The specification involves a client-server architecture where the browser makes requests to the Web application, and the latter performs some operations over the stored data and replies to the browser. On top of that, the set of methods work over specific data elements and do not allow any kind of planned transfers or incrementality. All these issues are a consequence of the fact that Storage is a quite simplistic solution for our needs, and therefore exhibits a poor behaviour against our requirements.

Table III summarizes the analysis of the three technologies considered in this section against our requirements.

TABLE III. COMPARATIVE ANALYSIS OF SCORM RTE, IMS-SSP AND W3C STORAGE.

	SCORM RTE	IMS SSP	W3C Storage
Req. 1	YES	YES	YES
Req. 2	NO	NO	NO
Req. 3	NO	NO	NO
Req. 4	NO	NO	NO
Req. 5	NO	YES	NO
Req. 6	YES	YES	YES
Req. 7	YES	YES	YES
Req. 8	NO	NO	NO

C. The Data-Transfer Manager

The underlying Data-Transfer Protocol is wrapped by the Data-Transfer Manager, which provides a standardised set of methods. Five methods, summarized in Table IV, have been considered to read the value of a data element (*getDataElement*), to overwrite it (*setDataElement*), to get an on-demand backup copy of the data of the instance (*getBackup*), to restore it (*restoreBackup*), and to schedule the transfer of future copies (*scheduleBackup*). These five methods can be divided into two groups: those that can be invoked by either the LMS or the tool (*getDataElement* and *setDataElement*), and those that can only be invoked by the LMS (*getBackup*, *restoreBackup*, *scheduleBackup*). An example of the first kind of methods takes place when the tool requests the name of the learner to the LMS using *getDataElement*, whilst an example of the latter could be an LMS requesting a backup copy of the data hosted at a tool by the use of *getBackup*, and transferring it to another tool using *restoreBackup*.

Table IV shows the methods along with their input and output parameters. All of them have an *id* input parameter, which refers to an identifier of the data involved (whether a single data element or a full backup). The parameter *data* contains the value of the data element or the backup being transferred, and has the same meaning whether considered as an input or output parameter. The parameter *result* contains a verbatim error code, if any (e.g. "The requested data element has not been initialized"). Finally the parameters *incremental*, *time* and *period* are specific for the request and scheduling of backup copies. The parameter *incremental* contains "false" or "true" if the backup is considered standalone or incremental. The former includes all the data at the tool, while the latter only contain those files that have changed since the last backup. This "incrementality" boosts the performance of a backup transfer, which is typically bandwidth-intensive. On its turn, the parameters *time* and *period* are devoted to schedule the periodical transfer of copies from a predefined point in time (e.g. the transfer of copies every 10 minutes from 9 pm).

TABLE IV. METHODS OF THE DATA-TRANSFER MANAGER.

Method	Input parameter	Output parameter
getDataElement	id	data
setDataElement	id, data	result
getBackup	id, incremental	data
restoreBackup	id, data	result
scheduleBackup	id, incremental, time, period	result

D. The Data-Transfer Protocol

A simple Data-Transfer Protocol has been specifically designed to work with the Data-Transfer Manager. The reason is that none of the technologies described in Section VI-B satisfies our needs to request, pack, transfer and restore data, and therefore a light ad-hoc Data-Transfer Protocol has been developed.

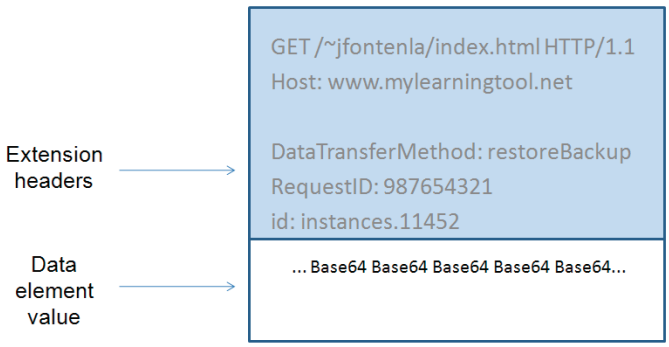

Figure 5. Sample message of the Data-Transfer Protocol.

As the Instance Protocol, the Data-Transfer Protocol has been designed as an extension of the HTTP protocol with special header extensions. Again, HTTP is specially suited for the design of a Data-Transfer Protocol, as it offers not only the capability to send parameters in the header of the messages (with the same format of the Instance Protocol), but also a payload to transfer binary data. On top of that, HTTP provides integrity as it works on top of TCP. In case that confidentiality

(e.g. when a learning tools wants to transfer the marks of a questionnaire to the LMS, at the transfer of a personal email account linked to an instant messaging application) or authenticity (e.g. to verify that the LMS managing the session data is the same that created the user account) were required, the Data-Transfer Protocol could use HTTPS instead.

Data are encoded using the BASE64 schema [15], and sent in the payload of the HTTP message. Full backup copies are packed as zip files prior to their transfer by means of the Data-Transfer Protocol. The use of this format is not as unusual as it may seem, as it is also the format used by Moodle to pack and transfer educational contents [16]. Should any problem occurred during the processing of the request, an error code would be returned for the Data-Transfer Manager, along with the habitual HTTP error code for the Data-Transfer Protocol. Figure 5 depicts a sample message of the Data-Transfer Protocol.

VII. Proof of Concept

In order to demonstrate the feasibility of the approach, a prototype consisting of a LMS and a learning Web tool has been designed. Acting as LMS we deployed an instance of Moodle (henceforth, "the LMS"). The choice of Moodle responds to its important presence in online universities, as well as to its opensource-ness. On the other hand, we deployed another instance of Moodle just to use one of its tools (a forum). The capabilities of the second instance of Moodle (henceforth, "the forum") to work as a full Learning Management System are not used, but only its forum. Whenever a user of the first instance of Moodle wants to use a forum, the one of the second instance is used instead. The forum of the first Moodle is "bypassed".

A full implementation of the six parts of the middleware described in Section III has not been made yet. Instead, we just implemented the Instance Interface, Data-Transfer Interface and Authorization Interface in both instances of Moodle.

A simple test, summarized in Figure 6, has been made. Firstly, the LMS used the Instance Manager to create a new instance of the forum (i.e. a new conversation thread) named "Waves". When the instance has been created, users must be added. In our case we added a student whose LMS user account had been previously created manually. This action does not grant instant access to the instance, but instead tells the forum that the instance will be visible to the user when he/she joins.

The next action carried out by the LMS was to post a welcome message giving some explanations about the wave equation. In terms of the middleware, this implied using the method *setDataElement* of the Data-Transfer Interface. During this proof of concept we tried to avoid the use of a complex agreed vocabulary in order to describe every data element hosted at the forum. Instead, we only used a data element called *newMessage*, but the Data-Transfer Interface is flexible enough to support any kind of vocabulary as long as it had been previously agreed by both parts.

Finally, when the user accessed the forum application (by means of the Authorization Interface, whose description can be found in [7]) found the post of the LMS as if it had been posted by any other user.

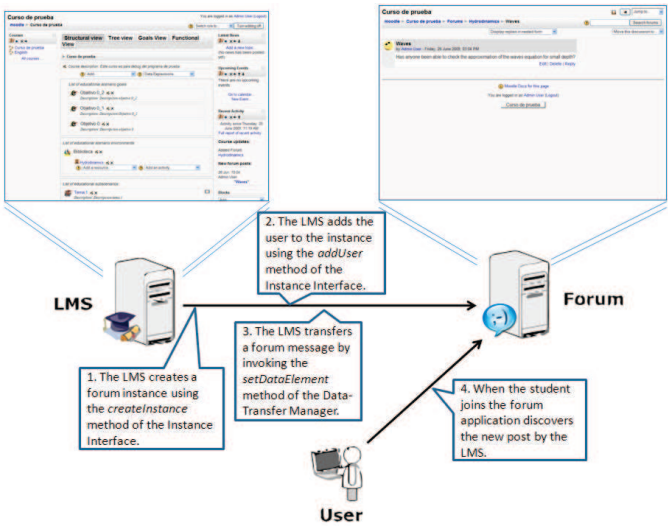

Figure 6. Proof of concept.

VIII. Conclusions

Current LMSs are playing an important role in providing engineering education. However, their possibilities are limited due a clear "one size doesn't fit all" problem. These limitations have been the starting point of our research. The work described in this paper tries to tackle these issues with tailorability and extensibility in mind by the use of the SaaS software distribution model.

The architecture described in this work is devoted to take advantage of the SaaS model while at the same supporting the concept of hard integration. At different with current solutions that only enable a soft integration of third-party tools, our architecture allows a harder level of integration. As a result, the LMS can extend its functionality, and can do it in a controlled way.

Our proposal not only implies the design of a new kind of e-learning system, but also an entirely new business model where the development of LMSs and educational tools follow separate (but complementary) ways. In the final term this business model implies more opportunities to bring students a better and more applied education.

One aspect of the Data-Transfer Interface we deliberately omitted is the use of some sort ontologies or agreed vocabularies in order to uniquely identify each resource hosted at the remote tool. Nonetheless, as pointed in Section VII the Data-Transfer Interface provides enough flexibility to support any kind of shared (semantic) knowledge. In fact, when we began to work on the interface we assumed as a starting point that there is already some kind of shared vocabulary, but no effective data-transfer mechanism between LMS and tools.

Currently we are working on the middleware to give full support to the other aspects of hard integration, apart from the

two interfaces described here and the Authorization Interface described in [7]. Nonetheless, the six parts we have considered are completely independent from each other, and they can be used in a standalone way if desired. The source code of the Data-Transfer Interface itself is available as open source and is freely available for anyone interested in its use.

ACKNOWLEDGMENT

This work has been funded by the Spanish Ministerio de Educación y Ciencia under the grant TIN2007-68125-C02-02, and by the Galician Consellería de Innovación e Industria under the grant PGIDIT06PXIB32 2270PR. Likewise, the authors would like to thank CYTED, by means of its ACCIÓN DE COORDINACIÓN 508AC0341 "SOFTWARE LIBRE EN TELEFORMACIÓN".

REFERENCES

[1] Web site of the Moodle project. Last accessed on November, 2009 at: http://moodle.org/

[2] Web site of the Blackboard project. Last accessed on November, 2009 at: http://www.blackboard.com/us/index.bbb

[3] .LRN Web site. Last accessed on November, 2009 at: http://dotlrn.org/

[4] Moodle modules and extensions. Last accessed on November, 2009 at: http://moodle.org/mod/data/view.php?id=6009

[5] M. Kyng, "Computers and Design in Context". *The MIT Press, 1997.*

[6] M. Caeiro, "PoEML: A separation-of-concerns proposal to instructional design", *Handbook of visual languages for instructional design: theory and practice, edited by L. Botturi and T. Stubbs, IGI Global, 2007*.

[7] J. Fontenla, M. Caeiro, M. Llamas, L. Anido, "Reverse OAuth - A solution to achieve delegated authorizations in single sign-on environments", *Computers and Security*. Last accessed in November, 2009 at: http://dx.doi.org/10.1016/j.cose.2009.06.002. Forthcoming publication.

[8] Edutools. CMS: Product Comparison System. Last accessed in November, 2009 at: http://www.edutools.com/compare.jsp?pj=4&i=550

[9] RFC 2811, "Internet Relay Chat: Channel Management". Last accessed in November, 2009 at: http://www.ietf.org/rfc/rfc2811.txt

[10] RFC 4353, "A Framework for Conferencing with the Session Initiation Protocol (SIP)". Last accessed in November, 2009 at: http://www.ietf.org/rfc/rfc4353.txt.

[11] ADL SCORM specification. Last accessed in November, 2009 at: http://www.adlnet.gov/Technologies/scorm/SCORMSDocuments/2004%204th%20Edition/Documentation.aspx

[12] ADL SCORM Runtime Environment specification. Last accessed in November, 2009 at: http://www.adlnet.gov/Technologies/scorm/SCORMSDocuments/2004%204th%20Edition/Documentation.aspx

[13] IMS Shareable State Persistence specification. Last accessed on November, 2009 at: http://www.imsglobal.org/ssp/

[14] W3C Storage. W3C working draft. Last accessed on November, 2009 at: http://www.w3.org/TR/webstorage/

[15] RFC3548, "The Base16, Base32, and Base64 Data Encodings". Last accessed on November, 2009 at: http://www.ietf.org/rfc/rfc3548.txt

[16] Moodle Docs Web site, "Tools for creating SCORM Content". Last accessed on November, 2009 at: http://docs.moodle.org/en/Tools_for_creating_SCORM_content

[17] AICC CMI Guidelines for Interoperability. Last accessed on June, 2009 at: http://www.aicc.org/docs/tech/cmi001v3-5.pdf

[18] G. Gross, "Google, IBM Promote Cloud Computing". *PC World, 2007*.

[19] Responsive Open Learning Environments. "Survey of learning-related services". Last accessed on November, 2009 at: http://www.role-project.eu/wp-content/uploads-role/2009/09/role-deliverable-31.pdf

[20] S. Wilson, B. Olivier, S. Jeyes, A. Powell, T. Franklin, "A Technical Framework to Support e-Learning." *JISC, 2004*. Last accessed on November, 2009 at: http://www.jisc.ac.uk/uploaded_documents/Technical%20Framework%20feb04.doc