SCORM Run-Time Environment As a Service

Gennaro Costagliola, Filomena Ferrucci, Vittorio Fuccella

Dipartimento di Matematica e Informatica, Università di Salerno
Via Ponte Don Melillo, I-84084 Fisciano (SA)
+39 089 963319 Fax: +39 089 963303

{gcostagliola, fferrucci, vfuccella}@unisa.it

ABSTRACT

Standardization efforts in e-learning are aimed at achieving interoperability among *Learning Management Systems* (*LMSs*) and *Learning Object* (*LO*) authoring tools. Some of the specifications produced have reached quite a good maturity level and have been adopted in software systems. Some others, such as *SCORM Run-Time Environment* (*RTE*), have not reached the same success, probably due to their intrinsic difficulty in being understood adequately and implemented properly. The *SCORM RTE* defines a set of functionalities which allow *LOs* to be launched in the *LMS* and to exchange data with it. Its adoption is crucial in the achievement of full interoperability among *LMSs* and *LO* authoring tools. In order to boost the adoption of *SCORM RTE* in *LMSs*, we propose a *Service Oriented Architecture (SOA)*-based reference model for offering the *SCORM RTE* functionalities as a service, external to the *LMS*. By externalizing functionalities from *LMSs*, our model encourages the independent development of e-learning system components, allowing e-learning software producers to gain several benefits, such as better software re-use and easier integration and complexity management, with a consequent cost reduction. The proposed model is validated through a prototype system, in which a popular *LMS*, developed with *PHP* language, is enhanced with the support of *SCORM RTE* functionalities, provided by an external Web service based on Java technology.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures – *Domain-specific architectures;*
K.3.1 [**Computing Milieux**]: Computers and Education – *Computer-managed instruction (CMI)* .

General Terms

Design, Standardization.

Keywords

Service Oriented Architecture, SOA, SCORM Run-Time Environment, Computer Managed Instruction, CMI, Learning Objects

ICWE'06, July 11-14, 2006, Palo Alto, California, USA.
ACM 1-59593-352-2/06/0007.

1. INTRODUCTION

In recent years, great efforts have been made to define standards, reference models and guidelines for e-learning. These efforts are aimed at obtaining a stronger interoperability among *Learning Management Systems (LMSs)*. In the context of these systems, the term interoperability refers to the possibility of running *Learning Objects* (*LOs*) produced with any authoring tool on any *LMS* compliant to the standard specifications. Once full interoperability among *LMS* and authoring tools is achieved, it will be easier to share *LOs*, and, consequently, re-use them, with remarkable time and resource saving for the content developers.

Some of the specifications produced, such as *Learning Object Metadata* and *Content Packaging*, have reached quite a good maturity level and have been adopted in software systems. Some others, such as *SCORM Run-Time Environment* [1], have not reached the same success, probably due to their intrinsic difficulty in being adequately understood and properly implemented [2]. The difficulty concerning the adoption of standard specifications has been the main motivation for the investigation of approaches which insure the re-use of standard functionalities [3]. To this extent two main solutions have been explored:

1. Providing *LMS* developers with frameworks and reference implementations of standard functionalities.
2. Proposing architectures and reference models to adopt in real systems in order to establish a widely accepted decomposition for e-learning systems. Once established, these models should facilitate the independent development of the identified components.

Reference implementations give scarce opportunities for software re-use, since their components are tightly coupled with the whole system of which they are a part. Frameworks overcome this problem, being loosely coupled with the system in which they are instanced. In previous work, we proposed a solution for adopting *SCORM RTE* based on a suitable framework, called *CMIFramework* [4]. Several problems still arise with frameworks. First of all, in most cases they are adoptable only in systems developed with the same technology: an O-O framework developed in Java cannot be used in a *.NET* or *LAMP*-based *LMS*. Secondly, even though the use of a framework allows for the easy extensibility of a system with new functionalities and has more customization margins, when instanced in a system, frameworks become part of it, increasing its size. The drawbacks in this case are related to the maintenance, testing and workload of the resulting system, since most enterprises, educational organizations cannot afford high systems handling [5].

Among the architectural models proposed for e-learning systems, solutions based on *Service Oriented Architecture (SOA)* are more and more widely adopted. Offering a way to externalize functionalities from the *LMS*, they allow *LMS* producers to gain several benefits, such as better software re-use and easier integration and complexity management, with a consequent cost reduction. Furthermore, these solutions are language independent and interoperable. Basing our findings on a literature survey, we can argue that the efforts produced so far have been devoted to demonstrating the importance of adopting *SOA* in e-learning systems, to offer high-level decompositions and to show how to span functionalities among the identified components. Offering functionalities as services external to the *LMS* often poses technical and practical problems depending on the specific service offered. The lack of existing systems or prototypes based on the proposed architectures prevents us from effectively validating them. Furthermore, there is no agreement on the decomposition. As a consequence, we are quite far from obtaining a standardized architectural model of a generic and comprehensive e-learning system, which could effectively help in the re-use of functionalities. A more effective method could be to follow a bottom-up approach in the definition of this model, concentrating the efforts on defining how to offer a single set of functionalities using a component external to the *LMS*.

This paper is aimed at describing how the *SCORM RTE* functionalities can be offered as a service, through the definition of a *SOA*-based reference model. The *SCORM RTE* addresses an important issue, namely the traceability of the student learning process. In particular, to enable the traceability of a student's activities, it defines the format of messages exchanged between the *LO* and the *LMS*. It is worth noting that the effectiveness of the e-learning paradigm can be heavily affected by the quality of the traceability process. Indeed, the collected information can be exploited to personalize knowledge contents, thus improving learning performances and the welfare of the students. Moreover, to carry out an accurate evaluation of each student, instructors can benefit from some information on course attendance, such as the time spent in completing a lesson or a test.

The high cost of implementing the *RTE* specifications suggests the necessity to externalize its functionalities from the *LMS*. Having a reference model that explains how to achieve this, can be useful for *LMS* producers to avoid such costs and to develop the *LMS* independently from the external module, which can be provided by third party efforts.

Starting from a technical discussion of the requirements of the model, we propose a high-level decomposition of an *LMS* system in order to establish the separation of roles between the basic *LMS* and the identified external service. Then, a decomposition at a lower level is presented, in order to be helpful for the developers who need to understand which modules they have to implement in their system to support our model. Finally, the proposed model is validated through a prototype system, in which a popular *LMS*, developed with *PHP* language, is enhanced with the support of *SCORM RTE* functionalities, provided by an external Web service based on Java technology.

The rest of the paper is organized as follows: the next section presents a summary of the *SCORM RTE* specifications. Section 3 outlines the proposed model. The prototype system is presented in section 4. In section 5 several works related to ours will be discussed. Some final remarks and some comments on future work conclude the paper.

2. THE SCORM RUN-TIME ENVIRONMENT

The *SCORM RTE* defines a set of functionalities which allow *LOs* to be launched in the *LMS* and to exchange data with it. Several documents from other producers of standards and guidelines for e-learning, such as *AICC* [6], and *IEEE LTSC* [7], propose a very similar model, even though several differences are present among the documents issued by different producers and often among different versions of the same specification. Almost all of them are aimed at defining the following common aspects regarding the *LO – LMS* communication:

- *Launch*: the set of rules under which an *LO* can be launched in a Web-based environment
- *API*: the interface of methods to be invoked by an *LO* in order to communicate with the *LMS*
- *Data Model*: the data set on which the communication is based

According to the *SCORM*, only a limited set of *LOs* can communicate with the *LMS*. These *LOs* are called *SCOs*, and their communication capability is due to the fact that they contain a specialized software module, called *ECMAScript*, which consists of several *Javascript* functions in the *ECMAScript* standard format.

The core of the *RTE* specification contains the description of the *SCO - LMS* communication mechanism. The way in which it takes place is shown in figure 1, which depicts a Web based scenario where a *SCO* has already been launched in a Web browser window and the *LMS* runs within a Web Server.

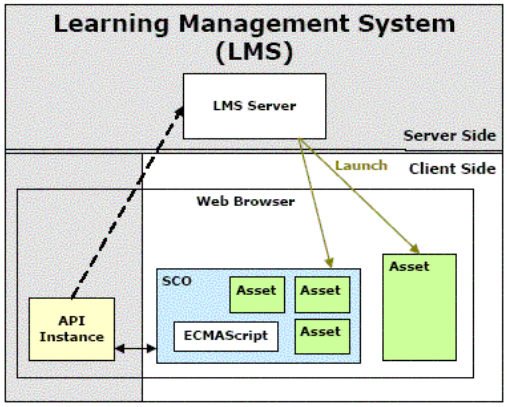

Figure 1 - SCORM RTE Architecture

The *SCO*, equipped with the *ECMAScript* module, can communicate with another module running on the client side: the *API Instance*. The latter, even though it runs on the client side, must be provided by the *LMS*. Therefore, it has often been implemented through a browser plug-in, an *Active-X* object, or, more frequently, through a Java applet. Java applets technology fits the needs of the *RTE* model well, since it can provide a module deployed on a server (the *LMS*), but running on the client (the Web browser). The *API Instance* module exposes an interface of methods to the *SCO*. By invoking them, the *SCO* can exchange

data with the *LMS* server. In practice, the *API Instance* works as a broker between the *SCO* and the *LMS*, since the former lacks the capability to connect with the *LMS* server directly, due to its nature of a plain document readable through a Web browser.

The *SCO* has the duties of starting and terminating the communication session and of leading the data exchange with the *LMS*. On the *LMS* side, an instance of the communication data must be kept. As mentioned before, the *SCO* can perform the communication invoking several *ECMAScript* methods exposed by the *API* Instance. With reference to the 2004 version of the *SCORM*, the methods for starting and terminating the communication are, respectively, *initialize*() and *terminate*(). The methods to set and get the run-time data on the *LMS* are, respectively, *getValue(<element_name>)* and *setValue(<element_name>, <value>)*.

The *API Instance* must handle error conditions which can occur during the communication, and notify the *SCO* about them by returning a specific value on a method invocation. Furthermore, the *API Instance* provides the *SCO* with further methods for obtaining information on the errors, in case any of them have occurred.

The *Data Model* is the set of data exchanged between the *SCO* and the *LMS* during the communication. For each element, the name, the data type, the access mode (*read only*, *write only*, *read/write*), the multiplicity and other information have been defined. This set of data includes, but is not limited to, information about the learner, interactions that the learner has had with the *SCO*, objectives, success status and completion status of the *SCO*. The set of data that can only be read by the *SCO (RO)* is typically information which must be passed from the *LMS* to the *SCO* to be shown to the user, such as the learner's name and identifier. The set of data that can be both read and written (*RW*) is information which must be available at the *SCO* at its launch and updated by the *SCO* at the end of the session. An example of this information is the progress level of the lesson. Finally, an example of data which can only be written (*WO*) by the *SCO*, is the time spent by the learner in the session. Generally, there is an instance of the *Data Model* (the run-time data) for each (learner, SCO) couple, if the learner has accessed the *SCO* at least once. The same instance can be shared throughout the session of the learner on the *SCO*, otherwise a new instance can be generated, according to the needs of the *LMS*.

3. THE ARCHITECTURE

This section defines the *SOA*-based architecture for offering the *RTE* functionalities. Our solution is valid for a generic *LMS*. A real-world application, based on our model, is contained in the next section. We propose a decomposition performed at two different levels: at a higher level, the separation of concerns between the *LMS* and the external service is specified; at a lower level, the modules composing each service are identified. Only the basic functionalities of the *RTE* model, such as the launch of *LOs* and the *LO*-*LMS* communication, together with basic *LMS* functionalities, such as the management of *LO*, are considered. Other services which can be found in a common *LMS* or other standard functionalities, which are not pertinent to our research, are not considered in this work. This choice does not prevent us from applying our model to wider systems.

3.1 Definition of the Services

The main objective of this phase is the definition of the services to build and of the logic encapsulated in each of them. Most of our work in this phase consists of establishing how to span the *RTE* functionalities among the identified services. Our aim is to alleviate the duties of the *LMS* as much as possible in the handling of *RTE* functionalities. Most of the work will be provided by an external service, which will be referred to as *RTE Service*.

In order to support the *SCORM RTE*, the basic functionalities of an *LMS* are the following:

- managing users (above all, learners and tutors) and keeping an *LO* database
- launching and dismissing *LOs* on learner's demand
- communicating with the *LO*, providing the learner's user-agent with an instance of the *API Adapter*
- handling the run-time data: the *LMS* must create an instance of it using names and types defined in the *Data Model*, keep it up-to-date during the communication and save it for future sessions.

The handling of users, including registration, authentication and authorization services, must be a duty of the *LMS*. Digital repositories of *LOs* can be external to the *LMS*. Other solutions integrate them on the same server as the *LMS* which launches them. We prefer to deal with the separate servers option because it is flexible enough to include the integrated one: once an external service is identified to keep *LOs*, it can still be placed on the same server as the *LMS*. We will refer to the service which keeps *LOs* and provides them to the *LMS* as *LO Repository service*.

According to the *RTE* model, among the operations provided to the learner by the *LMS*, there are the launch, the suspension, the resume and the dismissal of an *LO*. The communication between the *LO* and the *LMS* must start on the launch or resume events and must end on the suspend or dismiss events.

While it is quite clear that the *RTE Service* is in charge of hosting the server-side module which handles the communication with the *LO*, more doubts can arise as to which service should provide the *API Adapter* to the user-agent. The reader must recall from section 2 that it is up to the *LMS* to provide the *API Adapter* to the user-agent. This module must be downloaded and run on the client-side. Due to these requirements, a common solution is to implement the *API Adapter* as a Java applet, which can be packed in a *JAR* file and downloaded through the *HTTP* protocol. We will refer to the instance of the *API Adapter* running on the user-agent as *API Instance*. To avoid complications, the following reasons suggests the inclusion of the *API Adapter* as a module of the *RTE Service*:

- The *API Instance* must interact with the server-side module responsible for the communication. Putting the *API Adapter* on a separate service from this module gives no practical benefits and would compel us to define a standard protocol for the communication.
- A security limitation of Java applets prevents them from establishing network connections with other servers than the one from which they have been downloaded. This limitation, however, can be

overcome by using signed applets or changing user-agents security policies.

The last considerations concern how and where to keep the communication run-time data and, if they are kept by a service external to the *LMS*, how to make this data available to the latter during the communication. It is widely accepted that run-time data is not part of the *LMS* database. In the past, a poor design choice, adopted in some systems, was to design the *LMS* database in conformity with the *Data Model* of the *SCORM RTE*. This choice should be avoided for the following reasons: firstly, the *Data Model* has a hierarchical structure, which does not fit well with the relational model that is almost always used by *LMSs*; secondly, the definition of the data model has been subject to changes across the versions of the *SCORM* specifications. To be up-to-date, a re-engineering of the systems designed with the data conformant to the *Data Model* would have been necessary. In light of the previous observations, our choice is to keep the run-time data on the *RTE Service*. In the next section we will explain how to make the run-time data available to the *LMS* when needed.

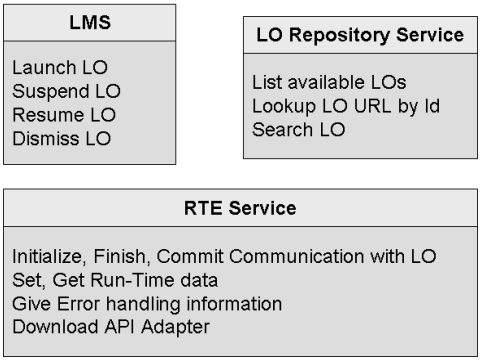

Figure 2 - Services Model

The above reasoning led us to identify the services model for *RTE* functionalities showed in figure 2. It identifies the services and the operations for each of them. Including only the *RTE* functionalities, the *LMS* must only supply the operations for the learner to make use of the *LOs*. The *LO Repository Service* provides the operations related to the administration of the *LO* repository, such as listing, searching and downloading of the *LOs* contained in it. The *RTE Service* is responsible for all the operations to perform the *RTE* communication with the *LO*, for making the run-time data available to the *LMS* and, finally, for making the *API Adapter* available for download to the learner's user-agent.

3.2 Low-Level Decomposition and Message Patterns Definition

The main objective in this phase is to define the low-level architectural decomposition of an *LMS* system which offers *RTE* functionalities, using the services identified in the previous section. The interactions among them, with the specification of the message exchange patterns, are shown.

Figure 3 shows the "actors on the scene" and their interactions. They are the *LMS*, the *RTE Service*, the *LO Repository Service* and the user-agent. The interactions among them are the following:

1. The channel through which the *User-Agent* downloads the *API Adapter* from the *RTE Service*
2. The channel for requests and responses from the User-Agent to the *LMS* to perform operations (launch, suspend, resume and dismiss) related to the *LOs*
3. The channel used by the *LMS* to locate the requested *LO* on the *LO Repository Service* and to forward the user-agent's request to the given *URL*
4. The channel used by the *API Instance* (running on the *User-Agent*) to perform the *RTE* communication with the *RTE Service*
5. The channel through which the *RTE Service* and the *LMS* communicate to allow the *LMS* to access run-time data when needed

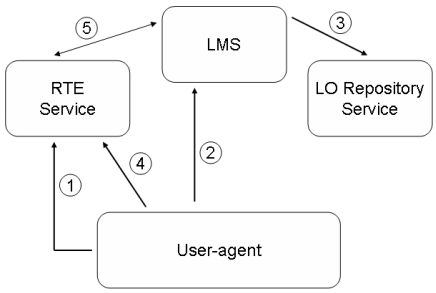

Figure 3 - Interactions Among Services

Channels from 1 to 4 can use a simple *HTTP* request/response message pattern. The message pattern for channel 5, instead, requires a more detailed explanation on the events which cause the *LMS* to access the run-time data. In our model, the run-time data is kept by the *RTE Service*. According to the *RTE* model, the run-time data can be read and written by the *LO* during the communication through the invocation of the methods *getValue*() and *setValue*() respectively, exposed by the *API Instance*. The run-time data must also be read and written by the *LMS*. This happens on the occurrence of several events, for the following reasons:

1. After run-time data is instanced and just before the communication starts, the data must be initialized with *LMS*-specific settings
2. After the communication is finished the LMS can read the run-time data to up-date its internal database with information gathered during the communication
3. Whenever a *setValue*() or *getValue*() or *commit*() is performed, the *LMS* could undertake some customized actions.

It is worth noting that, since the *RTE* communication is performed between the *API Instance* and the *RTE Service*, the *LMS* is unaware of the events listed above. Thus, the channel 5 is used to inform the *LMS* of the occurrence of these events. Due to our requirements, the most suitable message exchange pattern is the *event-driven* one: the *LMS* first registers at the *RTE Service*, sending a message to a module called *RTE Registry*, requesting notification for all the events. This registration should be performed whenever a user-agent asks for an *LO* to be launched. The *RTE Registry* must authenticate the *LMS* and reply with the authentication result. In case of success, the *RTE Service* sends a synchronous message to the *LMS* carrying the run-time data, on each of the previously identified events. This data can be read by the *LMS* and then sent, eventually modified, back to the *RTE Service* through a synchronous message again. To perform this

message exchange, the *LMS* must equipped with a service callback endpoint. We will refer to this module as the *LMS Callback Endpoint*. The communication between the *RTE Service* and the *LMS* can be based on *SOAP* formatted messages and must be *conversational*: some information, such as the learner's and *LO* identifiers, must be sent from the *LMS* to the *RTE Service* on the registration, and must be remembered later, when the following messages have to be handled. In other words, the messages must be part of a session.

A complete picture of all the *SOA* architecture, with the details of all the modules of the services mentioned so far, is shown in figure 4. For convenience, a layered architecture has been chosen to separate modules of the *Web-based Interface*, from those of the *Business Logic* and *Data* layers. The *Web-based Interface* layer contains both the Web resources, which can be accessed using a classical *HTTP* request/response message pattern and the deployed Web services.

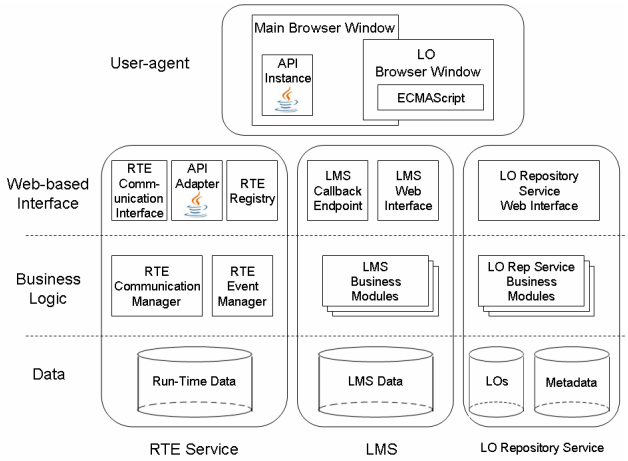

Figure 4 – Architecture

Before concluding, it is opportune to show a complete example using an interaction diagram. Let us consider the following situation: a learner, already logged on the *LMS*, requests an *LO* (in this example, an on-line test) to the *LMS*. The *LMS*, before launching it, registers to the *RTE Service*, and then forwards the request to the *LO Repository Service*. The *LO* is then downloaded by the *User-Agent* and the *RTE* communication starts (the *LO* invokes the *initialize*() method on the *API Adapter*). The *RTE Service*, through its *Communication Module*, receives the message, instances the run-time data and sends this instance using a *SOAP* message to the *LMS*. The *LMS* initializes the run-time data with the name of the learner and the scores to assign to each response of the learner on the test items. Once the learner has executed the test, the *LO* calculates the final score and sends it to the *RTE Service* using the *setValue*() method. The *RTE Service* sends the run-time data again to the *LMS*, which reads the score and saves it in its database with the learners' records. Later on, the *LO* is dismissed and the communication is terminated. The interaction diagram in figure 5 shows the interactions described in the example above. To keep it simple, the internal interactions of each service are omitted.

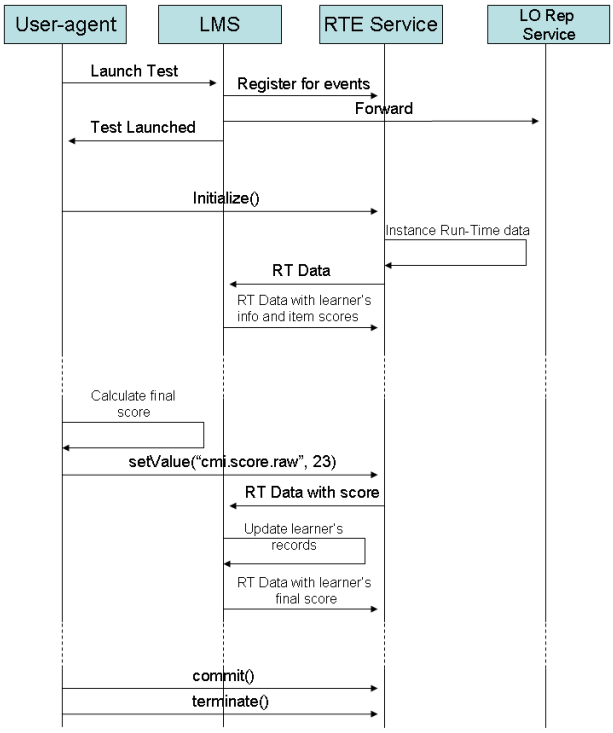

Figure 5 - Example of Interaction Diagram

4. CASE STUDY: A SCORM RTE MODULE FOR MOODLE

In this section we show how the reference architecture presented in the previous sections has been applied to add *SCORM RTE* functionalities to *Moodle* [8], a popular *Open Source LMS* developed using *PHP* server-side language. A prototype of the *RTE Service* has been implemented using *Java 2 Enterprise Edition (J2EE)* technology. The choice of such cross-technology system is not the fruit of coincidence, but has been made in order to show the language independency of our solution. Furthermore, the *RTE Service*, developed as a prototype, can be completed to offer its services to more than one *LMS*, based on whatever technology, at the same time.

4.1 The RTE Service

The *RTE Service* has been built as a *J2EE* Web Application, packaged in a *WAR* file. It can be deployed in any *J2EE* Web container.

The availability of *CMIFramework*, a framework for easily adopting *Computer Managed Instruction* functionalities in *LMSs* (developed at the University of Salerno) has allowed us to make little effort in developing the *RTE Service*. Among the others, *CMIFramework* provides the following components:

- An implementation of the *API Adapter* as a Java applet
- Full implementation of the modules involved in the *LO-LMS* communication
- Run-time data persistence handling module
- A module, implemented as a Java Servlet, which provides methods to override in order to handle the events of the communication.

Thanks to the availability of the above modules, it has been necessary to develop only the *RTE Registry* from scratch, as a Web Service, using *Apache Axis* [9]. *Axis SOAP* library has been used to compose the messages to carry run-time data to and from the *LMS*, on the occurrence of the events described before. To elaborate, the *RTE Event Manager* has been developed by overriding the *onInitialize*() and *onTerminate*() methods, provided by the server side module of *CMIFramework*. In these methods, the code to compose *SOAP* messages has been added. The information carried by these messages include: the event type, a session identifier, to keep a conversational state and the entire run-time data, represented as a list of (name, value) couples. It is worth noting that the caching of the communication has been used: in our implementation we have avoided the *API Instance* and the *RTE Service* to communicate on every single *setValue*() and *getValue*() method invocation. Instead, the run-time data has been changed locally on the *API Instance*, thus sending it to the *RTE Service* only on the termination of the communication.

4.2 Moodle: the LMS

Moodle comes with a mechanism to develop extensions to the basic *LMS*: a new module can be developed and integrated modifying a template provided with the Moodle documentation. Actually, a *SCORM* player for *Moodle* already exists, but it is entirely built as an internal module. Our prototype, however, is aimed at demonstrating how to provide *SCORM RTE* functionalities using an external service.

Moodle has an internal *LO* repository, thus, the operations of searching an *LO*, getting its *URL* and so on, are based on the simple invocation of *Moodle API* methods. Furthermore, the *forward* operation with which the *LMS* launches an *LO*, has been implemented as an action internal to the Web server which hosts the *LMS* system. The support for external *LO* repositories has been announced for the 2.0 version of *Moodle* and is expected for the end of 2006.

In light of the previous arguments, our development activity has consisted of the following two steps:

1. Preparing the environment in which the *LOs* are launched
2. Developing the *LMS Callback Endpoint* from scratch.

The activities related to the first point have consisted in simple *PHP* page coding: a *PHP* Web page has been created. The *API Instance* has been inserted in it as an applet to download from the Web server which hosts the *RTE Service*. Furthermore, this page has been designed to contain a form with the buttons to launch, resume, suspend and dispose a previously selected *LO*. The function which handles the launch operation, contains the code to send a *SOAP* message to register to the *RTE Service*, as described in the previous section. Applying a common pattern, suggested by the *RTE* specifications, the *LO* downloaded from *LMS* is launched in a child Window of the user-agent. In this way, the *API Instance* keeps running while the learner uses the *LO*.

The development of the *LMS Callback Endpoint* has been quite simple: a free library of *PHP* functions [10] has been used to manage the *SOAP* messages received from the *RTE Service*. A single function has been created to decode the message, read the event type, perform operations on the run-time data and send all the data back.

4.3 The LMS - RTE Service Communication

An interesting point concerning the communication between the *LMS* and the *RTE Service* is the handling of the conversational state. In our implementation we have adopted the 1.0 version of the *SOAP Conversation Protocol* [11]. This protocol makes it easy to conduct stateful conversations between two parties. Basically, the state is kept sending the following information in the header of *SOAP* messages:

- A *conversation Id*, in order to mark messages exchanged in the same conversation
- A *callbackLocation*, which is a URI that specifies the address from which the sender is listening to callbacks.

The callback location is sent only on the first message of the conversation, to provide the counterpart with the callback endpoint URI. The following code segments represent an extract from the *SOAP* messages sent by the *LMS* to the *RTE Service* to register for event notification and the response, in case of successful authentication. As the reader can see, they both carry the *conversation Id* in the header. The request carries the location of the callback endpoint, as well. In our simple prototype, the body of the request message specifies the authentication credentials of the *LMS*, while the body of the response message signals that the authentication is ok and the *LMS* will be notified of the occurrence of the *RTE* events.

```
<env:Envelope xmlns:env="…”>
  <env:Header>
    <StartHeader xmlns="…">
       <conversationID>
          1018048628974
       </conversationID>
       <callbackLocation>
          http://192.168.0.34/LMSCallbackEndpoint
       </callbackLocation>
    </StartHeader>
  </env:Header>
  <env:Body>
    <rte:registrationRequest xmlns:rte="…">
      <rte:LMSName>MyLMS</rte:LMSName>
      <rte:password>MyLMS</rte:password>
    </rte:registrationRequest>
  </env:Body>
</env:Envelope>
```

```
<env:Envelope xmlns:env="…”>
  <env:Header>
    <ContinueHeader xmlns="…">
       <conversationID>
          1018048628974
       </conversationID>
    </ContinueHeader>
  </env:Header>
  <env:Body>
    <rte:registrationResponse xmlns:rte="…">
      <rte:response>ok</rte:response>
    </rte:registrationResponse>
  </env:Body>
</env:Envelope>
```

The following code segments represent an extract from the *SOAP* messages sent from the *RTE Service* to the *LMS* on the *initialize*() method invocation event and its response. The messages are rather similar each other: they both contain the whole run-time data. In addition, the request carries the data of the event which caused the LMS to be notified.

```
<env:Envelope xmlns:env="…”>
```

```
  <env:Header>
    <ContinueHeader xmlns="…">
       <conversationID>
          1018048628974
       </conversationID>
    </ContinueHeader>
  </env:Header>
  <env:Body>
    <rte:eventNotify xmlns:rte="…">
        <rte:method>initialize</rte:method>
        <rte:runTimeData>
             <rte:element_name>
                cmi.learner_id
             </rte:element_name>
             <rte:value >
                556-00981
             </rte:value>


          <!-- more data -->

      </rte:runTimeData>
    </rte:eventNotify>
  </env:Body>
</env:Envelope>

<env:Envelope xmlns:env="…”>
  <env:Header>
    <ContinueHeader xmlns="…">
       <conversationID>
          1018048628974
       </conversationID>
    </ContinueHeader>
  </env:Header>
  <env:Body>
    <rte:eventNotifyResponse xmlns:rte="…">
      <rte:runTimeData>
          <rte:element_name>
             cmi.learner_id
          </rte:element_name>
          <rte:value >
             556-00981
          </rte:value>

          <!-- more data -->

      </rte:runTimeData>
    </rte:eventNotifyResponse>
  </env:Body>
</env:Envelope>
```

5. RELATED WORK

Some researchers propose a *SOA*-based architecture for defining a decomposition of a generic e-learning system [e.g. 3, 12, 13]. Authors in [12] propose a service architecture to integrate *LMS* and *Learning Content Management System* functionalities. All the identified modules are services that offer their functionalities using Web Services technology. Authors in [3] propose an architecture of a generic e-learning system, whose functionalities are provided by a set of Web Services, external to the main *LMS* application. In [13] a Grid-based layered architecture for the support of collaborative learning is proposed.

Other *SOA*-based architectures are more focused on the search of *LOs*, which may or may not use standard functionalities. In [14] a Web Services-based architecture is proposed in order to allow *LMS* servers to share learning-related information, such as learning material, learner data and learning strategies. Each of the previous category of information is kept by a different sub-system. According to [15] Web Services can be used in the field of content repositories, in order to obtain an infrastructure for the centralized search and discovery of *SCORM*-based learning contents. The work proposed in [16] is based on the *LTSA* [17] architecture, which is adapted to a *SOA*-based model. The authors intend to use this model to allow for a flexible integration of educational components. *LOs* can be discovered using the metadata annotation of the *LOM* [18] and then assembled together in a Web-services based platform.

Other work is more concerned with obtaining a standard environment based on the *SCORM RTE* model. A very technical paper is [19], where *SOAP* is used to perform the communication between the *LMS* and the *API Adapter*. There is no evidence that this could provide a better solution than using simple *HTTP* messages. An interesting matter concerns the launch of *RTE* compliant *LO* on *PDA* devices. For these environments, due to several hardware and software limitations, the architecture of the *SCORM RTE* is unsuitable. In [20], the authors claim that the use of Web Services should help to access the services provided by *SCORM API*. Unfortunately a finite and concrete solution for *RTE* service is postponed to further studies. In a previous work [21], we have proposed several modifications to the approach described by the *SCORM RTE*. The use of the *API Adapter*, which could not run in devices with limited capabilities, is substituted by the use of a suitable *Middleware* component in a Web Services-based architecture.

A work closely related with ours is [5]. It presents a framework for the adoption of the whole *SCORM* model in a *SOA*-based architecture. Most of the functionalities are provided by external services. A service which offers the functionalities specified in the *RTE* model is called *Tracking Service*. In the authors' opinion, such a service should be local to the *LMS*, for performance reasons. This argument is valid in their architecture, due to their decision to fuse *RTE* functionalities with other tracking functionalities. Otherwise, in our opinion, there would not have been valid reasons for preventing the externalization of the *RTE* functionalities from the *LMS*.

6. CONCLUSION

In this paper we presented a *SOA*-based architecture which can be adopted by *LMS* systems in order to support the *SCORM RTE* functionalities, using a service external to the *LMS*. We are confident that our proposal could represent a step ahead towards the definition of a more comprehensive standard architecture for an e-learning system built using loosely-coupled components. The availability of this standard architecture will allow the independent development of the components constituting the e-learning system, gaining all the benefits related to the adoption of this solution.

A prototype based on Web service technology has been developed, in which a popular *PHP*-based *LMS* uses an external service, built and deployed with *J2EE* technology, to offer *RTE* functionalities, thus showing the language independency of our solution. The *LO-LMS* communication caching mechanism allows us to significantly reduce the message exchange between the *LMS* and the external service, thus keeping the performances of the whole system high. A performance comparison between integrated systems and services-based systems is left for further studies, even if we think that the latter are inevitably destined to supplant the formers.

Future work is aimed at finding solutions to externalize other functionalities from the *LMSs*, starting from the standard ones, which lend themselves to be offered by components external to

the *LMS* and loosely-coupled with it. We think such kind of bottom-up approach is suitable to obtain a final environment that defines the functionalities that can be externalized and those that must be integrated into the *LMS*. To this extent, our proposal could be a step forward.

7. REFERENCES

[1] *The Scorm Run-Time Environment ver 1.3.1*, http://www.adlnet.org/scorm/history/2004/documents.cfm

[2] Nakabayashi, K., Kubota, Y., Yoshida, H., Shinohara, T., Design and Implementation of WBT System Components and Test Tools for WBT content standards, *Proceedings of ICALT '01* (Madison, USA, Aug 2001), 213-214

[3] Vossen, G., Westerkamp, P., E-learning as a Web service, *Proceedings of Seventh International Database Engineering and Applications Symposium*, (July 2003), 242 – 249

[4] Costagliola, G., Ferrucci, F., Fuccella, V., A Framework for the Support of the SCORM Run-Time Environment, *Proceedings of the 2006 International Conference on SCORM 2004* (Taipei, Taiwan, Jan 2006), 21-26

[5] Chu, C.P., Chang, C.P., Yeh C.W., Yeh Y.F., A Web-service oriented framework for building SCORM compatible learning management systems, *Proceedings of International Conference on Information Technology: Coding and Computing*, (2004), 156 - 161 Vol.1

[6] *CMI Guidelines for Interoperability AICC rev. 4.0*, http://www.aicc.org/docs/tech/cmi001v4.pdf, 2004

[7] *IEEE LTSC, WG11: Computing Managed Instruction*, http://ltsc.ieee.org/wg11/index.html

[8] *Moodle, A Free, Open Source Course Management System for Online Learning*, http://moodle.org/

[9] *Apache Web Services – Axis*, http://jakarta.apache.org/axis /

[10] *XML-RPC for PHP Homepage*, http://phpxmlrpc.sourceforge.net

[11] *SOAP Conversation Protocol ver 1.0*, http://dev2dev.bea.com/pub/a/2002/06/SOAPConversation.htm

[12] Xiaofei L., El Saddik, A., Georganas, N.D., An implementable architecture of an e-learning system, *Proceedings of IEEE Canadian Conference on Electrical and Computer Engineering* (May 2003), 717 - 720 vol.2

[13] Wang G.L., Li Y.S., Yang S.W., Miao C.Y., Xu J., Shi M.L., Service-oriented grid architecture and middleware technologies for collaborative e-learning, *Proceedings of IEEE International Conference on Services Computing* (July 2005), 67 - 74 vol.2

[14] Tamura Y., Yamamuro, T., Distributed and Learner Adaptive e-Learning Environment with Use of Web Services, *Proceedings of the International Conference on SCORM 2004* (Taipei, Taiwan, Jan 2006), 11-15

[15] Hussain, N., Khan, M.K., SCASDA: SCORM-based Centralized Access, Search and Discovery Architecture, *Proceedings of the International Conference on SCORM 2004* (Taipei, Taiwan, Jan 2006), 137-140

[16] Pahl, C., Barrett, R., A web services architecture for learning object discovery and assembly, *Proceedings of the 13th int. World Wide Web conference on Alternate track papers* (May 2004), 446-447

[17] IEEE LTSC, WG1, *Architecture & Reference Model*, http://ieeeltsc.org/inactive/arch/

[18] IEEE LTSC, WG12, *Learning Object Metadata*, http://ltsc.ieee.org/wg12/

[19] Shih, T.K., Chang, W. C., Lin, N.H., Lin, L.H., Hsu, H. H., Hsieh, C. T., Using SOAP and .NET web service to build SCORM RTE and LMS, *Proceedings of Advanced Information Networking and Applications* (Xi'an, China, Mar 2003), 408-413

[20] Lin, N.H., Shih, T.K., Hui-huang, H., Chang, H. P., Chang, H. B., Ko, W. C.; Lin, L.J., Pocket SCORM, *Proceedings of 24th International Conference on Distributed Computing Systems Workshops* (Tokyo, Japan, March 2004), pp. 274-279

[21] Casella, G., Costagliola, G., Ferrucci, F., Polese, G., Scanniello, G., A SCORM Thin Client e-learning Systems Based on Web Services, *To appear in International Journal of Distance Education Technology*