A Framework for the Support of the SCORM Run-Time Environment

Gennaro Costagliola, Filomena Ferrucci, Vittorio Fuccella
Dipartimento di Matematica e Informatica, Università di Salerno
Via Ponte Don Melillo, I-84084 Fisciano (SA)
{gcostagliola, fferrucci, vfuccella}@unisa.it

Abstract

In order to allow interoperability among Learning Management Systems there is a need for a standard environment in which the Learning Objects could be launched and exchange data with the Learning Management System. The functionalities of such an environment are often referred to as Computer Managed Instruction and are at present defined in several specification documents issued by the producers of the main standards and guidelines. The most famous of them is SCORM Run-Time Environment, produced by ADL. In this paper we propose an Object Oriented framework which allows the development of Learning Management Systems with the support of Computer Managed Instruction functionalities. The framework can be opportunely instanced to obtain an environment in which Learning Objects, compliant with different versions of the specifications, can be launched without incurring incompatibility problems.

1. Introduction

Among the specifications produced by ADL SCORM [1], some, such as *Learning Object Metadata* and *Content Packaging*, have reached quite a good maturity level and have been adopted in software systems. Unfortunately, we cannot say the same for the *Run-Time Environment (RTE)* [2] specification, which has not reached the same success, probably due to its intrinsic difficulty in being understood adequately and implemented properly. A second possible cause could be that the specifications are still being developed, even by other guidelines producers, and are subject to frequent modifications, causing several incompatibility problems.

The RTE specification proposes a standard environment in which the *Learning Objects (LO)* could be launched and exchange data with the *Learning Management System* (*LMS*). The way in which the *RTE* works 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. A *SCO* (*Sharable Content Object*) is a *LO* that has a software module, called *ECMAScript*, which allows it to communicate with the *LMS*. The *API Instance* module, provided by the *LMS*, exposes an interface to the *SCO*, through which it can exchange data with the *LMS* server. Furthermore, the *API Instance* must handle error conditions which can happen during the communication. Another important subject treated in the specifications is the *Data Model*, that is the set of data on which the data exchange between the *SCO* and the *LMS* must be based.

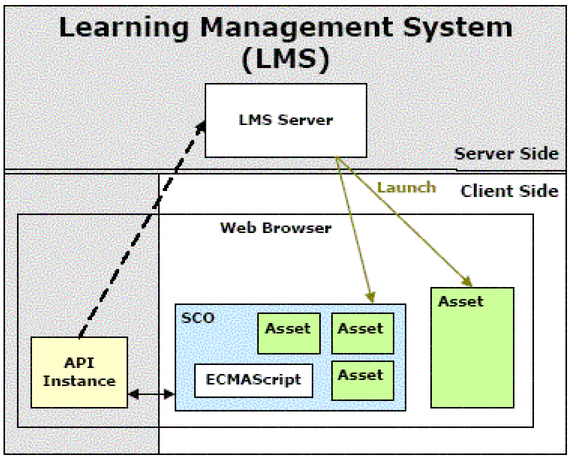


Figure 1 - SCORM RTE Architecture

The definition of such a model is actually proposed in several specification documents, such as *CMI Guidelines for Interoperability [3], IEEE CMI [4] and SCORM Run Time Environment [2]*. We will often refer to the functionalities proposed in such documents, comprising the specifications of all the cited producers, as *CMI functionalities*, where the acronym *CMI* is an abbreviation of *Computer Managed Instruction*. Among all the specifications, the *SCORM RTE*, which resumes and re-elaborates the proposals of other producers, is the most comprehensive.

The framework described in this paper has been implemented to assist the work of *LMS* developers. It allows the overcoming of the difficulties connected to the adoption of the *CMI* functionalities: a little work is necessary to obtain an environment in which *LOs*, which are compliant with several issues and versions of the specifications, can be launched without incurring incompatibility problems. A common solution to such problems is to implement the LMS systems conformant to the up-to-date version of the specification and to upgrade the old LOs to make them compatible. This approach clearly requires a bigger effort, compared to a solution that allows all the versions of LOs to run in the same environment.

The framework is easily configurable to allow the developer to decide the data formats and the interfaces to support and it provides the developer with some extension points (*hot spots* [5]) useful in the event of further modifications in the specifications.

The paper is organized as follows: the next section presents several works related to ours. In section 3 and 4 the framework will be presented, in particular, its functionalities and its architecture will be, respectively, analyzed. Section 5 is devoted to present an instance of the framework. Some final remarks and some comments on future work conclude the paper.

2. Related Work

The importance of the *CMI functionalities* is witnessed by the several e-learning systems (*Lotus LearningSpace*, *Blackboard* and *Moodle)* and authoring tools (*Macromedia Authorware* and *Toolbook Instructor*) which adopt them. Some other products are not fully conformant to the *SCORM* specification. However, they have a proprietary mechanism to support several *CMI functionalities*. A noteworthy example is *WebCT.*

Several authors engaged with standardization in e-learning claim the importance of the *CMI* functionalities and of the standard adoption in general [6], [7]. In particular, in [6], the *RTE* specification is identified among the most promising potentiality of the *SCORM*.

In order to simplify the duty of the developers, some reference implementations have been developed. The most important of them is *SCORM Sample RTE*. In [8] an implementation of the *SCORM* adapted to present contents on mobile devices is presented. Beside the implementation based on Java, such as the ones cited so far and as the further product described in [9], some systems have been developed using the *Microsoft .Net* framework, such as *DotNetSCORM* and the one described in [10], which implements the *API Instance – LMS* communication using Web Services.

In order to boost the adoption of the *CMI* specifications, instead of reference implementations, framework and libraries could be more suitable. An experiment of this type is shown in [11], which proposes a library of re-usable components and testing tools for *WBT* systems. A wider-ranging work, which has as an objective the development of a framework for the adoption of the whole *SCORM* model, is described in [12]. Another framework for the support of several functionalities not directly connected to the adoption of standards and guidelines in e-learning systems is presented in [13].

Despite recognizing the importance of supporting all of the different existing specifications, none of the cited works propose a valid solution to the problem of the incompatibility between *LO* and *LMS* supporting different specifications or different version of the same specification. The *SCORM* web site claims that “the *ADL Technical Team* is currently developing several conversion utilities that can be used to update *SCORM* Version 1.2 conformant content to *SCORM* Version 1.3 conformance”. Some third party conversion utilities are already available on the Internet and some authoring tools, such as the one proposed in [14], have been developed to be easily upgraded in order to support the latest version of the specification. An *LMS* developed with our framework allows the content developers to avoid the effort of updating all of the existing *LOs*.

3. Main Functionalities

As mentioned before, the reference model has been subject to several modifications through the many documents that discuss *CMI* functionalities. The most important ones take into account the definition of the following aspects: the *API interface*, the error handling model and the *Data Model* elements. The main configuration features and the extension points of the framework concentrate above all on these aspects. In particular, our framework supports the following main functionalities:

- Full implementation of all the *CMI* specifications produced so far;
- Caching of the *API Instance – LMS* communication;
- Server side persistence of the *Data Model* instances;
- Support for more than one *API Instance* interfaces and for their related error handling systems;
- Support for more than one *Data Model;*

3.1 Communication Caching

The way in which the *API Instance – LMS* communication takes place follows a consolidated set of rules. The framework supports these rules and, moreover, it implements a sort of communication caching: the *Data Model* instance, initialized by the *LMS*, is sent to the *API Instance* before the communication takes place. Later on, all the read and write operations are performed locally. Only at the end of the communication, the instance is sent back to the *LMS*, avoiding delays in the communication due to possible slow communications.

3.2 Server Side Persistence

The specifications produced consider some cases in which the *Data Model* instances could be persisted, to be possibly loaded and re-used in more than one communication sessions. In order to support the feature described before, the framework has a support for the persistence of *Data Model* instances on the server side.

3.3 Support for More *Api Instances*

The interface exposed to the *SCO* by the *API Instance* has been subject to some changes which have regarded, particularly, the name with which the *API Instance* object is identified in the Web page the method prototypes definition. Even though such modifications could seem trivial, they have been the main source of incompatibility between *LMS* and *LO*. To face this problem, we chose to give the framework the chance to expose more than one interface to the *SCO*. Such interfaces must be specified in the framework configuration and, for each of them, a Java interface must be written by the developer.

The error handling system is strictly dependent on the *API Instance* interface and, in fact, for each invocation of one of its methods, a set of exceptions can occur. During the evolution of the specifications, the code and the name bound to each of them, have been subject to changes. Moreover, time after time, new error conditions have been defined. An exception mainly occurs when wrong values for the parameters are passed to the invoked methods or when the *API Instance* state does not allow the invocation of a given method. In the event of an error condition, the method execution is interrupted and an atypical value is returned to the *SCO*. The framework provides a large set of predefined *checks* and it can be extended adding new check definitions. The internationalization support assures that the name and the text of for the error messages are viewed in the language selected in the browser's settings. The *Validator* framework [15], provided by *Apache Group*, has been used for aiding the error handling process.

3.3 Support for More *Data Models*

The data model on which the *SCO-LMS* communication is based has been in the centre of the debate since its first definition by *AICC*. Later on, several more data models have been defined. Even in this case, the framework has been designed to support more than one data model. For each data model, the elements that constitute it, can be defined by the users, setting their identifier and type in the configuration. Moreover, other information concerning them can be defined, such as: the access rules to the element, some constraints on its value and some eventual dependencies on other elements. Finally, it is possible to define some derived elements, calculated on the basis of the values of other elements. In this case it is necessary to extend the framework as well, writing the code of the method that calculates the derived element.

The framework configuration is easily performed by editing some *XML* files.

4. Architecture

The framework is composed of two main components: a client side one and a server side one. The former is composed of Web browser's modules (Java applets), archived in a *JAR* (*Java ARchive*) compressed file. The latter is a Java library of modules to be used in Web based applications.

4.1 The Client Component

The client component provided by the framework is, in practice, the *API Instance*, shown as a black box in figure 1. Its internal architecture is detailed in figure 2. The remaining client modules shown in figure 1 are internal to the *SCO*, so they are not part of the framework.

The *API Instance* provided by the framework is composed of a set of intercommunicating Java Applets. A main module, called *Master API* is present, together with a set of *API Satellite* modules. The former is responsible for the communication with the LMS. Each of the *API Satellite* is responsible for exposing a given *API interface* to the *SCO* and for the related error handling. Each *API Satellite* is identified by a name (identifier) in the *DOM* tree of the HTML page which contains them. The example shown in figure 2 shows an instance of the framework which implements three API interfaces (*SCORM 1.2*, *SCORM 1.3* and a third, user defined, interface) and in which a *SCORM 1.3* compliant *SCO* has been launched. Two *SCOs* respectively conformant to the versions 1.2 e 1.3 of the *SCORM*, will try to interact with the two *API Satellite* objects respectively called *API* and *API_1484_11*.

It is possible to insert the client component of the framework in the *JSP* pages of the *LMS*, through the use of a *TagLib* defined ad hoc.

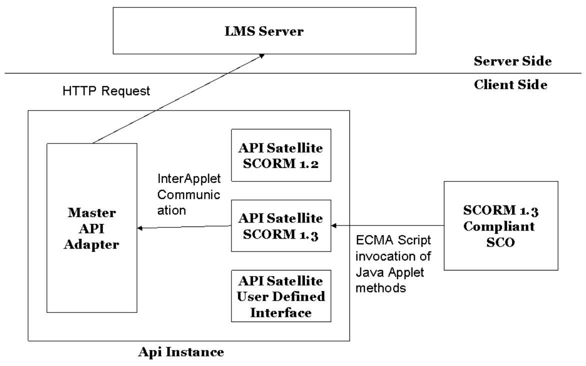


Figure 2 - The Client Component

4.2 The Server Component

The server component is composed of two main modules, both of them implemented as Java Servlets: *ConfigurationManager* and *LMSCMI*. The former is responsible for reading the configuration from the *XML* files and for sending it as serialized objects on demand to the *Master API*. The *LMSCMI* module constitutes the core of the server component. It is responsible for handling the server side duties of the *SCO-LMS* communication. Its behavior has been defined through the methods of an abstract class, such that it can be customized for *LMS* implementations. The customizable features are the actions which take place before the start of the communication and after the end of it. In the first case, the *Data Model* instance must be initialized with some data, often kept by the *LMS*. In the second, the instance itself, received from the client, must be used to update the *LMS* data.

5. A Case-Study: *MyLMS*

MyLMS is an *LMS* comprising of a minimal set of functionalities and only able to launch preloaded *SCOs*. It has been implemented in order to validate the framework described so far. *MyLMS* is able to launch *SCOs* conformant to the versions 1.2 and 1.3 of the *SCORM* specification.

The steps in which *MyLMS* developers have been involved in order to instantiate the framework comprises both server side and client side components programming. Lastly, the framework has been configured ad hoc.

5.1 Server Side Programming

The main server side programming activity has consisted in subclassing the *LMSCMI* servlet in order to tailor the server side behavior to the simple *MyLMS* requirements. This is a mandatory operation for the applications based on *CMIFramework*, because the two steps of initializing the *Data Models* and using the data from them after the communication termination depend on the design choices made during the development of the application, particularly on the database design choices. *MyLMS* just holds some information regarding the *SCO* completion state and its objectives in a relational database. Such an information must be used to initialize the *Data Model* instances and must be updated after the communication. These operations have been performed overriding the methods *BeforeInitialize* and *AfterTerminate*, respectively, adding simple pieces of *JDBC* code in them.

Few lines of further code writing have been necessary to define the classes which calculate the derived elements contained in the *Data Models*.

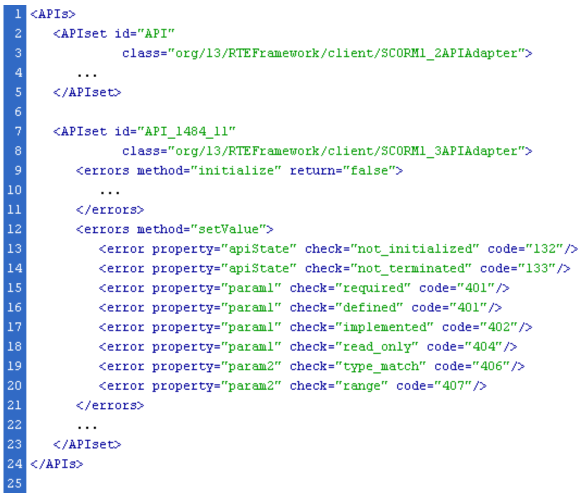
5.2 Client Side Programming

The client side programming has been limited to the definition of the interfaces exposed by the *API Instance* to the *SCO*. Using the *Java Reflection API*, the framework automatically generates a proxy for each of the interfaces. Since *MyLMS* does not use any tailored *check* for the error handling, no more classes needed to be written by the developers. To complete the client-side work, the *API Instance* object have had to be put in the *MyLMS’* HTML main page. This page is a dynamically generated JSP page. Few code lines of the TagLib provided ad hoc by the framework have been added in it.

5.2 Configuration

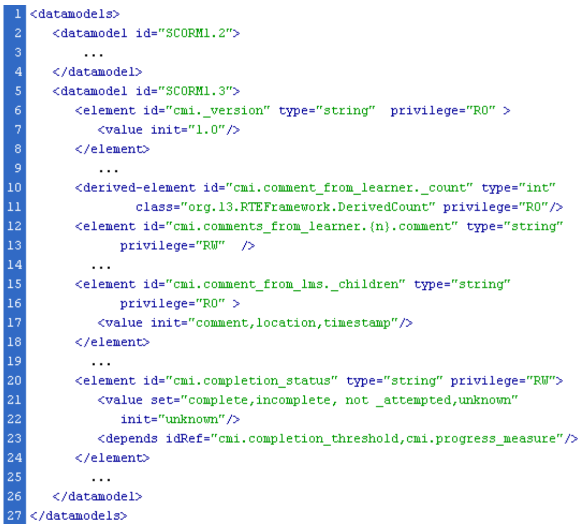
Two configuration files must be written by the MyLMS developers: the *apis.xml* configuration file, with the definition of the *API* Interfaces and the error conditions, and the *datamodels.xml* configuration file, which contains the definition of the data models.

In figure 3, an extract of the *apis.xml* configuration file is shown. The definition of the interfaces 1.2 and 1.3 of the *SCORM API* is performed, respectively, in line 2 and 7, through the use of the *APIset* element. The *id* attribute is the identifier with which the *API Satellite* objects will be inserted in the *JSP* Page of the *LMS*. For each method of the *API* interface, the set of errors must be defined. The complete list of error *checks* for the *setValue* method is shown from line 11 to line 21. Let us analyze some of them: with the code in line 13 we are checking that the *setValue* method is only called when the API is in an *initialized* state. More precisely, the code tells the framework to raise an exception if the *check not_initialized* performed on the *APIstate* variable reveals an error condition, which is marked with the code *132*. Finally, in line 20 we are telling the framework to check that the second parameter passed to the method is in the range specified for it in the other configuration file *datamodels.xml*.



```
 1 <APIs>
 2    <APIset id="API"
 3             class="org/l3/RTEFramework/client/SCORM1_2APIAdapter">
 4       ...
 5    </APIset>
 6
 7    <APIset id="API_1484_11"
 8             class="org/l3/RTEFramework/client/SCORM1_3APIAdapter">
 9       <errors method="initialize" return="false">
10          ...
11       </errors>
12       <errors method="setValue">
13          <error property="apiState" check="not_initialized" code="132"/>
14          <error property="apiState" check="not_terminated" code="133"/>
15          <error property="param1" check="required" code="401"/>
16          <error property="param1" check="defined" code="401"/>
17          <error property="param1" check="implemented" code="402"/>
18          <error property="param1" check="read_only" code="404"/>
19          <error property="param2" check="type_match" code="406"/>
20          <error property="param2" check="range" code="407"/>
21       </errors>
22       ...
23    </APIset>
24 </APIs>
25
```

Figure 3 - Extract from the *apis.xml* config file



```
 1 <datamodels>
 2    <datamodel id="SCORM1.2">
 3        ...
 4    </datamodel>
 5    <datamodel id="SCORM1.3">
 6       <element id="cmi._version" type="string"  privilege="RO" >
 7          <value init="1.0"/>
 8       </element>
 9          ...
10       <derived-element id="cmi.comment_from_learner._count" type="int"
11               class="org.l3.RTEFramework.DerivedCount" privilege="RO"/>
12       <element id="cmi.comments_from_learner.{n}.comment" type="string"
13             privilege="RW"  />
14         ...
15       <element id="cmi.comment_from_lms._children" type="string"
16             privilege="RO" >
17          <value init="comment,location,timestamp"/>
18       </element>
19         ...
20       <element id="cmi.completion_status" type="string" privilege="RW">
21          <value set="complete,incomplete, not _attempted,unknown"
22             init="unknown"/>
23          <depends idRef="cmi.completion_threshold,cmi.progress_measure"/>
24       </element>
25         ...
26    </datamodel>
27 </datamodels>
```

Figure 4 - Extract from the *datamodels.xml* config file

An extract of the *datamodels.xml* configuration file is shown in figure 4. The definition of the data models 1.2 and 1.3 of the *SCORM* is performed, respectively, in line 2 and 5, through the use of the *datamodel* element. Only the latter of them is described in detail, showing some of its elements, in the figure. The element *cmi._version* is defined in lines 6 to 8. This element is a string, which can only be read (the *privilege* attribute is set to *RO*) by the SCO, and whose value is initialized to *1.0* before the communication starts. An example of the definition of a derived element is shown in lines 10 and 11. This element is an integer whose value is the length of a collection and it is calculated by the *calculate*() method of the class *org.l3.CMIFramework.DerivedCount*, provided by the framework. The element we are talking about is *cmi.comment_from_learner._count*, which keeps the size of a collection. A derived element calculated by a user defined class can be defined as well, specifying the right value for the *class* attribute. Finally, a dependency is shown in line 23: it states that the element *cmi.completion_status* must be set after the elements *cmi.completion_threshold* and *cmi.progress_measure* are set. This element is initialized to *unknown*, afterwards it can take one of the values specified by the attribute *set* in line 21, that is, one of the following values: *complete*, *incomplete*, *not_attempted*, *unknown*.

6. Conclusion

I this paper we presented a framework which allows the extension of *LMS* systems with the support of *CMI* functionalities. We have been showing its main functional and architectural features and an application which witnesses its validity. The framework has shown itself to be flexible in supporting all the specifications produced so far, allowing the developers to overcome the incompatibility problems among them. The *MyLMS* application development has allowed us to show the simplicity of the use of our framework: the developer only has to write small pieces of code, concentrated in a few key points. Most of the merit should be given to the choice of making the majority of the functionalities configurable.

Providing the functionalities to obtain an environment in which LOs, which are compliant with several issues and versions of the specifications, can be launched without incurring incompatibility problems, our framework allows the avoiding of the time waste to update all of the existing *LOs*.

The framework is complete in its functionalities and it can be an excellent tool for small and medium enterprises which want to easily support *CMI* functionalities in their projects for *LMS* development. Future work is aimed at designing further applications based on the framework. An ambit of application where a project is already started is *Computer Aided Assessment* (*CAA*): we are trying to extend our *CAA* system *eWorkbook* [16] with the support of *CMI* functionalities.

The *CMIFramework* project is distributed under *GPL* license and it is at present hosted by *Sourceforge* platform. It is possible to visit its home page at the *URL* *http://www.sourceforge.net/projects/cmiframework*.

7. References

[1] Advanced Distributed Learning, http://www.adlnet.org/

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

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

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

[5] H. A. Schmid (1997), "Systematic Framework Design by Generalization", *Communications of ACM*, 40, 10, pp. 48-51

[6] O. Bohl, J. Schellhase, R. Sengler, U. Winand (2002), "The Sharable Content Object Reference Model (SCORM) – A Critical Review", *Int. Conf on Computers in Education*, pp. 950-951

[7] J.M. Santos, L. Anido, M. Llamas (2003), "On the Use of E-learning Standards in Adaptive Learning Systems", *3rd IEEE Int. Conf. on Advanced Learning Technologies*, p. 480

[8] N.H. Lin, T.K. Shih, H. Hui-huang, H. P. Chang, H. B. Chang, W. C. Ko; L.J. Lin (2004), "Pocket SCORM", *24th Int. Conf. on Distributed Computing Systems Workshops*, pp. 274-279

[9] C. Qu, W. Nejdl (2002), "Towards Interoperability and Reusability of Learning Resource: a SCORM-conformant Courseware for Computer Science Education", *Proceedings 2nd IEEE Int. Conf. on Advanced Learning Technologies*

[10] T.K. Shih, W. C. Chang, N.H. Lin, L.H. Lin, H. H. Hsu, C. T. Hsieh (2003), "Using SOAP and .NET web service to build SCORM RTE and LMS", *17th IEEE Int. Conf. on Advanced Information Networking and Applications*, pp. 408-413

[11] K. Nakabayashi, Y. Kubota, H. Yoshida, T. Shinohara (2001), "Design and Implementation of WBT System Components and Test Tools for WBT content standards", *1st IEEE Int. Conf. on Advanced Learning Technologies*, pp. 213-214

[12] C.P. Chu, C.P. Chang, C.W. Yeh, Y.F. Yeh (2004), "A Web-Service Oriented Framework for building SCORM Compatible Learning Management Systems", *Int. Conf. on Information Technology: Coding and Computing*, pp. 156-161

[13] J. Redol, D. Simões, A. Carvalho, H. Páscoa, J. Coelho, P. Grave, R. Luís, N. Horta (2003), "VIANET – A New Web Framework for Distance Learning", *3rd IEEE Int. Conf. on Advanced Learning Technologies*, pp. 258-259

[14] J. M. Su, S. S. Tseng, J. F. Weng, K. T. Chen; Y. L. Liu, Y. T. Tsai (2005), "An object based authoring tool for creating SCORM compliant course", 19th Int. Conf. on Advanced Information Networking and Applications, pp. 209-214 vol.1

[15] The Apache Jakarta Project – Commons - Validator, http://jakarta.apache.org/commons/validator/

[16] G. Costagliola, F. Ferrucci, V. Fuccella, F. Gioviale (2004), "A Web Based Tool for Assessment and Self-Assessment", *2nd Int. Conf. on Information Technology: Research and Education*