Case Studies on the Support of Computer Managed Instruction Functionalities in e-Learning Systems

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

The term Computer Managed Instruction (CMI) often refers to a set of functionalities which allow Learning Objects to be launched in the Learning Management System and to exchange data with it. A framework for the support of CMI Functionalities in Learning Management Systems, named CMIFramework, has been developed at the University of Salerno. In this paper we present two case studies concerning the adoption of CMI functionalities, using CMIFramework, in different e-learning contexts. Our present work is aimed at demonstrating the ease in using the framework and its power in solving several problems connected to the adoption of CMI functionalities.

1. Introduction

In the past years, a big effort has been made to define standards, reference models and guidelines for e-learning. This effort is aimed at obtaining a stronger interoperability among *Learning Management Systems (LMS)*. In the context of these systems, the term *interoperability* refers to the sharing of *Learning Objects (LO)*, and, consequently, their re-use, with remarkable time and resource saving for the content developers.

Among the specifications produced, some, such as *Learning Object Metadata* and *Content Packaging*, have reached quite a good maturity level and have been adopted in software systems. Some others have not reached the same success, probably due to their intrinsic difficulty in being understood adequately and implemented properly. This is the case for the set of specifications named *Computer Managed Instruction (CMI)*, consisting of a set of functionalities which allow *LOs* to be launched in the *LMS* and to exchange data with it. A complete reference on the way in which the launch and the communication take place has been proposed in several specification documents, issued by the producers of the main standards and guidelines for e-learning, such as *AICC* [1], *SCORM* [2] and *IEEE* [3]. Even though the proposed basic model is the same, several differences are present among the documents issued by different producers and often among different versions of the same specification. These differences have resulted in an incompatibility problem: *LO* developed in compliance with a specific document cannot be launched in environments designed for different ones. This problem, added to the difficulty of adopting a specification not easy to understand, has resulted in an insufficient adoption of a specification whose importance is attested by the attention of the three main producers of standards and of several software vendors. Furthermore, many *LOs* compliant with specifications no longer in use, must be up-graded, wasting time and opportunity to re-use material.

A framework, named *CMIFramework*, has been developed at University of Salerno with the aim of facilitating the adoption of *CMI* functionalities in *LMSs* [4]. The goal of this paper is to show that the proposed framework effectively allows developers to make their *LMS* compliant to most of the specification documents produced so far, equipping them with an environment in which *LOs*, compliant with different specifications or different versions of a specification, can be launched without incurring incompatibility problems, thus avoiding the effort necessary to up-grade the older contents. To this aim, we present two case studies connected to the adoption of *CMI* functionalities. The software solutions presented have been developed instantiating *CMIFramework*. In the first case study, presented in section 4, we show how a system for on-line testing and assessment can be enhanced with a no-standard module which tracks students' client-side interactions. This information can be used to enrich the data available to data mining systems for catching cheats among the students. A more comprehensive case-study, with which we provide an *LMS* with a module to launch *SCORM* compliant *LOs*, is presented in section 5. The next section contains a brief explanation of the architecture specified in *CMI*

specifications. For a more comprehensive explanation, please refer to [1, 2, 3]. In section 3 we outline some of the main features and the architecture of *CMIFramework*. Final remarks and future work conclude the paper.

2. The CMI Functionalities

The *CMI* specifications propose a standard environment in which the *LOs* could be launched and can exchange data with the *LMS*. The way in which the communication works is shown in figure 1, which depicts a Web-based scenario, where a *LO* has already been launched in a Web browser window and the *LMS* runs within a Web Server. The *LO* must be equipped with 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 *LO*, through which it can invoke methods to handle the communication with the *LMS* server and to exchange data with it. The *API Instance* must handle error conditions which can occur during the communication. Another important part of the specifications defines the *Data Model*, which is the schema of the data on which the communication between the *LO* and the *LMS* must be based. Some rules on how to launch LOs are established as well.

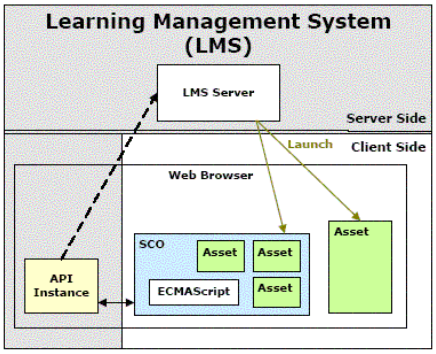

Figure 1 - CMI architecture (SCORM RTE)

As mentioned before, there are some differences among the specifications produced so far, and several changes have been made over time. The most significant of them have regarded the definition of the *API Interface* and the structure of the *Data Model*. *API Interfaces* have been changed in methods signatures (e.g., from *SCORM* 1.2 to *SCORM 2004*, the *LMSInitialize()* method has become *initialize()*, *LMSFinish()* has become *terminate()* and so on) and in the error handling system (error names and codes have changed, new errors have been introduced, etc.). The *Data Model* has changed above all in regards of the element names.

These changes have been enough to prevent the launch of *LOs* developed in compliance with a specific document in environments designed for different ones.

3. CMIFramework

The *CMIFramework* is an Object-Oriented Java framework which can be instanced in order to alleviate the work of *LMS* developers in adopting *CMI* functionalities in their systems, thanks to the software re-use principle. It also solves the incompatibility problems mentioned before, allowing the launch of *LOs* compliant with any *CMI* specification in the same environment. Presently, *CMIFramework* can support most of the specifications produced so far, avoiding the time-consuming task of up-grading all the *LO* compliant with older versions of the specifications. Nevertheless, it has been conceived flexible enough to address several future changes in the specifications. Furthermore, it goes beyond the standard functionalities, allowing the developers to define customized solutions, not necessarily adhering to them strictly.

More in detail, among the features of *CMIFramework*, we can find the support for user-defined *API Interfaces* with the related error handling system and for user-defined data models. This user-defined solutions can be combined to standard ones, providing all these functionalities in a unique environment. Other interesting features of the *CMIFramework* is the caching of *LO-LMS* communication and the server-side persistence of the *Run-Time* data.

CMIFramework has an innovative architecture: on the client-side, based on inter-applet communication, it allows the deploying of any number of *API* Interfaces. This is simply done editing the XML- based configuration and coding the Interface. The configuration also allows the developer to completely define the elements of the *Data Models*.

On the server-side, a small amount of code must be written in order to customize the *LMS* behavior. The server-side component consists mainly of a *Java Servlet*, which have to be extended by the *LMS* developer. In order to accomplish this task, three methods can be overridden: *onInitialize()*, *onCommit()* and *onTerminate()*. The former method allows us to initialize the *Run-Time* data, before the communication starts, with *LMS*-specific settings. The latter two methods allow the *LMS* to manipulate the *Run-Time*

data upon the commit and termination of the communication.

4. Case Study I: Learner's Interaction Tracking in On Line Testing

As proven by several studies in the education field, many learners cheat at exams, when they can [5,6]. Cheating detecting in assessment tests is not an easy task: most of the techniques employed so far have been based on the comparison of the results obtained in the tests [7]. These techniques cannot give the certainty of the guilt, since a high similarity of two tests can be due to coincidence. Furthermore, as in all fraud detection systems, the task is complicated by several technological and methodological problems [8]. It could be useful to gain information on the learners' behavior during the test. Analysis on these data can be integrated to results comparison in order to have a more comprehensive data set as input for a data mining technique to detect cheating. This is possible when the tracking of the client-side interactions of the learner with the test during its execution is performed. For example, let's consider the following situation: during the test, *learner A* answers *true* to a question and *learner B*, who is seated behind the former, answers the same few instants later. The tracking of this information could be useful to prove that the learner has cheated, looking on the screen of his classmate. Due to the static nature of HTML pages, there are several challenges in performing the interaction tracking. Our case study illustrates how *CMIFramework* has been instanced to obtain a software module able to generate a complete log of the interactions of the learner during the execution of the test. This module has been added to an advanced *Computer Aided Assessment* system, developed at the University of Salerno, named *eWorkbook* [9]. This system can be used for evaluating learner's knowledge by creating (the tutor) and taking (the learner) on-line tests based on multiple choice, multiple response and true/false question types.

eWorkbook, which is fully accessible with a Web browser, launches the tests from its main window in a child window. The question items are shown one at a time, and a simple button bar in the bottom of the page can be used to browse the test: there is one button to submit the test and two more, labeled respectively with *next* and *previous*, to browse the items sequentially. The submission of the test can occur voluntarily from the learner when the test is finished or forced by the system on the expiry of a timer. Once the test is submitted, the child window is closed.

The client component (the *API Instance*) of the framework has been integrated in the main window, exactly in the page that launches the test, which is open for the duration of the test. The test itself has been modified in order to incorporate the *ECMAScript* module, which has the duty of dialog with the client component of the framework. For the *APIInstance*, the standard API of the *SCORM 2004* has been chosen, since it is quite easy to find a free implementation of an *ECMAScript* module for this combination of producer and version. A very simple data model has been defined to record the interactions, as the reader can see by looking at the configuration code, shown in figure 2.

```
<datamodels>
  <datamodel id="eworkbook">
    <element id="eworkbook.start_time"
                 type="date"  privilege="WO" />
    <element id="eworkbook.delivery_type"
                 type="string"  privilege="WO" >
        <value set="volunteer,forced"/>
    </element>
    <element id="eworkbook.interactions.*.time"
                 type="date" privilege="WO" />
    <element id="eworkbook.interactions.*.type"
                 type="string" privilege="WO">
        <value
set="set,reset,next,previous,submit"/>
    </element>
    <element id="eworkbook.interactions.*.item_id"
                 type="string" privilege="WO"/>
    <element id="eworkbook.interactions.*.response"
                 type="string"   privilege="WO" />
    <derived-element
         id="eworkbook.interactions._count"
         type="int"
         class="org.l3.RTEFramework.client.error.util
         ity.CountManager" privilege="RO"/>
  </datamodel>
</datamodels>
```

Figure 2 - The definition of the *Data Model* for the Learner's Tracking Module in *eWorkbook*

For each interaction of the learner with a question item, the information recorded is as follows: timestamp, type, id of the item and the response given by the learner. As a type of interaction, the following values are admitted: *set*, for response given to an item; *reset*, for response aborted; *next*, *previous*, for browsing items and *submit*, for the submission of the test. Finally, the timestamp of the start of the test and the information whether the test submission was voluntary or forced are recorded.

The library *CMIServer.jar* has been imported in the system. The main server side programming activity has consisted in sub-classing the *CMIServlet* in order to tailor the server side behavior to our simple requirements. To elaborate, *onTerminate*() has been overridden, in order to output the results of the tracking and to save them in the relational database of the system through the use of simple *JDBC* code.

On the client-side, the module *CMIClient.jar* has been deployed. The *APIInstance* has been easily inserted in the JSP pages using the *TagLib* provided by the

framework. The interactivity level required for our purposes has made necessary a modification to the *JSP* code which generates the pages of the tests. In particular, an *ECMAScript* module, freely downloaded from the Internet, has been added to the test pages. To correctly plug it, some event-handling code has been added to the page of the test, for example, the *onClick* events of the input elements of the test page (radiobuttons or checkboxes for the options, buttons for the navigation controls) have been handled.

5. Case Study II: A SCORM Module for Sakai

The *Sakai Project* [10] is a community source software development effort to design, build and deploy a new *Collaboration and Learning Environment (CLE)* for higher education. The *Sakai* application framework has been customized by our developers to obtain a learning environment called *Running Platform (RP)*, which has been used at our department for the management of the courses, in a blended learning style.

A prototype for a new tool for *SCORM RTE* has been developed in order to test the effectiveness of our framework in creating an environment in which *LOs*, compliant with different version of the *SCORM* specification could have been launched. The module, originally designed as a stand-alone application, was later integrated into the *RP*. The stand-alone application is a minimal system, able to launch only pre-loaded *LOs* conformant to the versions *1.2* and *2004* of the *SCORM*.

The framework has been configured declaring the *API Interfaces* and the data models for both the *1.2* and *2004* versions of the *SCORM*. An extract from the *apis.xml* configuration file is shown in figure 6. The figure shows how the two *API Interfaces* are declared with the different names of *API* and *API_1484_11*, as required respectively by the *1.2* and *2004* versions. For space reasons, the configuration of the error handling system is shown only for the *setValue()* method of the *2004* interface.

```
<APIs>
   <APIset id="API"

class="org/l3/CMIFramework/client/SCORM1_2API">
      ...
   </APIset>

   <APIset id="API_1484_11"

class="org/l3/CMIFramework/client/SCORM2004API">
      <errors method="initialize" return="false">
         ...
      </errors>
      <errors method="setValue">
         <error property="apiState"
             check="not_initialized" code="132"/>
         <error property="apiState"
             check="not_terminated" code="133"/>
         <error property="param1" check="required"
             code="401"/>
         <error property="param1" check="defined"
             code="401"/>
         <error property="param1"
check="implemented"
             code="402"/>
         <error property="param1" check="read_only"
             code="404"/>
         <error property="param2" check="type_match"
             code="406"/>
         <error property="param2" check="range"
             code="407"/>
      </errors>

      ...
   </APIset>
</APIs>
```

Figure 3 – Extract from *RP apis.xml* config file

```
<datamodels>
   <datamodel id="SCORM1.2">
            ...
   </datamodel>

   <datamodel id="SCORM2004">
      <element id="cmi._version" type="string"
                privilege="RO" >
         <value init="1.0"/>
      </element>
         ...
      <derived-element
         id="cmi.comment_from_learner._count"
         type="int"
         class="org.l3.CMIFramework.DerivedCount"/>
      <element
         id="cmi.comments_from_learner.{n}.comment"
         type="string"  privilege="RW"  />
        ...
      <element id="cmi.comment_from_lms._children"
         type="string" privilege="RO" >
         <value init="comment,location,timestamp"/>
      </element>
        ...
      <element id="cmi.completion_status"
         type="string" privilege="RW">
         <value set="complete, incomplete,
             not_attempted, unknown" init="unknown"/>
         <depends idRefs="cmi.completion_threshold,
                cmi.progress_measure"/>
      </element>
        ...
   </datamodel>
</datamodels>
```

Figure 4 – Extract from RP *datamodels.xml* config file

As for the *API Interfaces*, even for the data models, additional work was requested and performed on the *datamodels.xml* file, in order to define the data models for both the versions of the *SCORM*.

As for case-study I, the *CMIServlet* has been subclassed in order to customize the server side behavior of the application. In this case, both the methods *onInitialize*() and *onTerminate*() have been implemented. In the former, the data model used for the communication has been initialized with the data to pass from the *LMS* to the *LO*. The latter has been used for the opposite purpose. In both cases, simple *JDBC* code has been added to these methods. The server-side persistence of *run-time* data, provided by the framework, has been used to share the data model

instances across multiple sessions of the same learner on the same *LO*.

Sakai offers a suitable container for tools and associated services. Its architecture is quite flexible to allow different levels of integration for the tools. The most loosely coupled integration level allows the developer to integrate stand-alone applications. At the scope, two main rules must be followed:

1. The request must be intercepted and dispatched to the application by a module called *Sakai Web-App Gateway*
2. Basis services, such as authentication and authorization management, must be provided by an interface called *Sakai API Gateway*.

In light of these arguments, the main integration programming activity has consisted in the modification of the application in order to dialog with the *Sakai APIs*. To plug the *Sakai WebApp Gateway* in the application, actually, there was no need to modify the application: we just needed to develop a *servlet filter* for the requests and the responses. A *filter* entry was added to the *deployment descriptor* (the *web.xml* file) of the application. The work necessary to plug the *Sakai API Gateway* in the application has been slightly more complicated: the handling of the user accounts, based on *JDBC*, of the stand-alone version have been substituted with some calls to the *Sakai API Gateway*. This has been done in every part of the application dealing with the user authentication. Additionally, some user accounts and information have been imported from the application database to the *RP* one.

6. Conclusion

In this paper we have presented two case studies useful in proving the ease of use and the power of *CMIFramework*, developed at the University of Salerno. In the first one we have added a learner's interactions tracking module, whose log can be useful to detect cheating in on-line exams. The description of the data mining process which will allow us to accomplish this task goes beyond the scope of this paper and is postponed to future work.

In the second case study, the development of a module to launch *SCORM* compliant *LOs* in *Sakai* has been shown. Through this module, we can launch content compliant with the most recent versions of the *SCORM* in the same environment without incurring incompatibility problems, avoiding the effort necessary to up-grade the older content to support newer versions of the specifications, thus allowing for a better re-use of the authored content.

Being written in Java, CMIFramework has a restriction: only Java-based *LMSs* can benefit from it. A solution based on Web Services, aimed at overcoming this limitation, is now in the development phase.

7. References

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

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

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

[4] Costagliola, G., Ferrucci, F., Fuccella, V., "A Framework for the Support of the SCORM Run-Time Environment", Proc. of the 2006 Int. Conf. on SCORM 2004, Taiwan, 21-26

[5] Dick, M., Sheard, J., Bareiss, C., Carter, J., Joyce, D., Harding, T., Laxer, C. "Addressing student cheating: definitions and solutions", Proc. of ITiCSE 2002, 172-184

[6] Harding, T.S.; Carpenter, D.D.; Montgomery, S.M.; Steneck, N.H.; "The current state of research on academic dishonesty among engineering students", Proc. of FIE '01, vol. 3, F4A 13-18

[7] Mulvenon, S. W., Turner, R. C., & Thomas, S. "Techniques for detection of cheating on standardized tests using SAS". Proc. of the 26th Annual SAS Users Group Int. Conf., Miami, FL, 1 – 6.

[8] Shao, H.; Zhao, H.; Chang, G. R.; "Applying data mining to detect fraud behavior in customs declaration", Proc. of ICMLC'02, 1241-1244 vol.3

[9] Costagliola, G., Ferrucci, F., Fuccella, V., Gioviale, F., "A Web Based Tool for Assessment and Self-Assessment", Proc. of ITRE '04, 131-135

[10] Sakai Project, http://www.sakaiproject.org/