Integrating IMS Learning Design and ADL SCORM using CopperCore Service Integration

Paul Sharples¹, David Griffiths¹, Colin Tattersall²

1 CETIS, The University of Bolton, Deane Road, Bolton, BL3 5AB, UK
Email: [p.sharples, d.e.griffiths] @bolton.ac.uk

2 Open University of the Netherlands

Abstract: The services available in IMS-LD runtime are limited, and the integration of additional services is complex. The TENCompetence project seeks to resolve this problem. As a first a system was developed which creates a SCORM service and then integrates it with an IMS Learning Design player. This enables a Unit of Learning to contain SCORM material. The new runtime system is able to launch a SCORM 1.2 Sharable Content Object (SCO) object found within a Unit of Learning, and to use the properties generated by that object to guide the learning flow of the Learning Design. Thus the learning Design runtime system is able to correctly detect and then process SCORM content. The system was successfully trialled using a UOL which demonstrated this functionality.

Keywords: IMS Learning Design; ADL SCORM

1 Introduction

The motivation which informed the development of Educational Modelling Language (EML) (Koper & Manderveld, 2004) by OUNL in the early years of this decade, and its successor IMS-LD (IMS Global Learning, 2003), was to provide a framework which formally describes the usage of learning objects (Koper & Es, 2003). As Koper states “The key principle in learning design is that it represents the learning activities and the support activities that are performed by different persons (learners, teachers) in the context of a Unit of Learning. These activities can refer to learning objects … and … services (e.g. forums, chats, wiki’s) that are used to collaborate and to communicate in the teaching/learning process” (Koper, 2005). The infrastructure available for running Units of Learning (UOLs) provides an effective solution for running activities which refer to learning objects, but the implementation of activities in the context of a range of services remains problematic.

In this paper we describe the problems which are presented by the orchestration of such services in IMS-LD, and the way in which the TENCompetence project (TENCompetence, 2006) seeks to resolving them.

Three factors in the development of the IMS-LD specification contribute to this problem.
IMS (and other) specifications have been developed to resolve specific interoperability problems, with no integrating framework.

- IMS-LD was developed by adapting EML. In this process a number of EML services were removed because they were already covered by other IMS specifications (e.g. assessment).

- Only a small number of simple services were included in IMS-LD (send mail, conference, index search, monitor), in order to make implementation (and hence adoption) simpler.

These historical factors mean that it is hard to integrate IMS-LD with the services such as forums, chats and wikis mentioned by Koper, and in order to resolve this, the TENCompetence project, is working towards the establishment of a generalisable connector protocol to enable IMS-LD run-time engines to connect to external communication and collaboration services.

Moreover, since the publication of the IMS-LD specification there has been a trend towards distributed service oriented architectures, which is manifested in eLearning in initiatives such as the eFramework (Olivier, Roberts, & Blinco, 2005), the Personal Learning Environment approach (Wilson et al., 2006). These tendencies are undermining the dominance of the monolithic Virtual Learning Environments which were assumed as a given when EML and IMS-LD were developed, and make the resolution of the services problem still more urgent.

We note that within IMS there is related work underway towards the integration of services. The IMS Tools Interoperability Specification (IMS Global Learning, 2006b) seeks to facilitate the integration of third party tools with core LMS platforms, and IMS Shareable State Persistence describes a way of storing and sharing state information in runtime systems (IMS Global Learning, 2006a). In many ways Business Process Execution Language for Web Services version 1.1 (BPEL) (IBM, BEA Systems, Microsoft, & Siebel Systems, 2005) is also addressing similar problems in a different domain, and might be a source of future solutions. None of these initiatives, however, provides a usable solution to the specific problems of orchestrating services in IMS-LD (although they may be able to contribute to a future solution).

TENCompetence is taking a dual approach to integrating services into IMS-LD. Desk research is being carried out on the requirements and possible approaches to developing a connector protocol, but in parallel practical implementations are being developed, not only to provide much needed functionality, but also to ensure that the connector protocol which is produced will be rooted in real examples of integrated services. It is this latter aspect of the work which is reported here.

2 The service implemented

As described by Olivier Chapter 2 in the Springer Learning Design book by Bill Olivier
“Learning services are likely to come in two varieties: those ... which are set up as part of a local environment; and those that are set up as remote web services”. These varieties are also termed closely and loosely linked scenarios. In the work reported here we have focused on the closely linked scenario, in order to avoid the additional complications of Single Sign On or account replication (see the presentation by (Wilson, 2005) for an outline of these and other relevant issues).

The particular service which we have chosen to implement is a player for SCORM 1.2 Sharable Content Objects (see (Advanced Distributed Learning) for an introduction to SCORM). There are two principal reasons for this. Firstly, there are many SCORM objects available, and it would be valuable to learning designers to be able to use them in UOLs which can provide a much wider range of pedagogic contexts. Secondly, because IMS-LD redresses the pedagogic limitations of SCORM (see, for example (Hummel, Koper, & Tattersall, 2005)) there is a mistaken perception that use of the two specifications together is not useful. We hope to correct this.

The analysis carried out in (Tattersall, Burgos, Vogten, Martens, & Koper) provides a starting point for our implementation. This paper proposes that SCORM Sharable Content Objects (SCOs) could appear within the Environment section of an IMS-LD activity, and describes two approaches to implementation.

The first approach involves the SCO being physically located within a separate SCORM aware VLE. The Learning Object within the UOL simply references the web address of the SCO running on the remote VLE. The advantage is that the IMS-LD Runtime Environment does not have to know how to handle the SCOs runtime calls to the API, but the Learning Design Environment does not have access to the data model that the SCO interacted with, so there is little or no interaction between the SCO and the executing Learning Design that referenced it.

The second proposed method involves the SCO being physically part of the Learning Design package, and directly imported into the IMS-LD runtime environment. When the SCO needs to be launched within the IMS-LD Player, it is passed to a dispatcher. This acts as an interface to the Learning Design runtime environment and in this case, the SCORM runtime environment. The dispatcher acts as the SCORM aware LMS, provides the environment for the SCO to execute in, and has access to the data model which the SCO interacts with. Subsequently the changes in the SCOs data model can be used to update properties and conditions within the Learning Design, and so provide true interaction between the SCO and the rest of the Learning Design. This was considered to be required functionality, and so this second method was adopted.

3 The software developed

Two existing open source systems were used, the CopperCore Learning Design Engine (Martens & Vogten, 2005), and the Reload SCORM player (Reload). A dispatcher framework was available in the CopperCore Runtime Environment, the CCSI (CopperCore Service Integration framework) (Vogten et al., 2006). Some related work had already been carried out to integrate CopperCore and IMS-QTI, using the dispatcher
method to integrate the APIS QTI service. However, whereas IMS-QTI content essentially consists of an XML file which needs to be processed and rendered, a SCO presents a different set of problems. Typically a SCO will consist of a single HTML page with embedded javascript. The HTML page needs to be able to access an API adapter object within the same page (or frameset hierarchy) as the user interacts with the learning material. The API adapter will communicate user changes made within the SCO to the LMS. It is the responsibility of the LMS to provide the APIAdapter and it allows the SCO to set and get values from a defined data model. The data model holds various values that would allow an LMS to track details on how the user interacted with the SCO. Without an APIAdapter, the SCO cannot set and get these values and so is really no different to ordinary web content.

We now briefly indicate the different parts of the implementation work carried out.
3.1 The Initial structure of the Reload SCORM runtime software

![Diagram of SCORM runtime software structure]

The first task was to uncouple the constituent parts mentioned above and isolate the engine and user interface parts. It was also necessary to identify exactly what information the engine would have to process and return in order for it to become an effective SCORM 1.2 SCO service.

A section of the Reload SCORM engine code was adapted to hold the state information of a particular SCO, once it has been initialised. The SCO state is held in a structure which makes it easy to manipulate from the CopperCore Service Integration framework. When playing a single SCO, once an appropriate property is updated within the Learning Design the SCO data model is no longer needed by the Learning Design runtime environment. Consequently a design decision was made to hold the SCO data model in memory until the user had finished working with it, and then dispose of it.

3.2 Changes to the IMS-LD Manifest

For a Learning Design runtime environment to be able to process a particular learning object as special SCORM content it must be able to identify it within a IMS Learning Design manifest. Following the convention established for QTI we used the identifier “adl_sco_v1.2”.

3.3 Mapping a Learning Design Property to a SCO data model element.

It was then necessary to define the interaction of the UOL with the SCO data model, which has a specified format of values that can be accessed by the SCO at runtime. A number of these elements could be useful within a UOL. For example the value “cmi.core.session_time” can be used to hold the value of how long a user spent viewing...
a page. The approach taken was to design a UOL which has IMS-LD Level B properties that are named to correspond with values in the SCO data model.

### 3.4 The Runtime Learning Design Player

The player needed to handle the new SCO 1.2 content. To do this it had to provide the SCO with the correct environment in which to communicate with the API adapter. It had to be then able set/get the correct properties which would subsequently update the unit of learning. The CopperCore Player is based around a Java 2 Enterprise Edition Servlet. This servlet is responsible for orchestrating the content to be delivered back to the web browser. From within the browser, the user clicks a navigation tree item which sends a call back to the servlet, with a request for content. The servlet realises that the content is of type “adl_sco_v1p2”, creates a link to a new SCO 1.2 service within the CCSI and creates a new data model for the given SCO. This data model is initialised with certain values, such as the student name for example under “cmi.core.student_name”. The CopperCore servlet takes this model and builds an XML document which it returns to the browser. This XML document contains the initialised data model and the URL to where the SCO resides.

### 3.5 The browser transform

The browser receives the XML from the servlet and then applies an XSLT stylesheet. The stylesheet builds the correct environment for the SCO within the browser. It first creates a link to the client APIAdapter. Next it uses the information within the XML to recreate a client side version of the data model. Once this has been initialised, it finally looks up the URL for the actual SCO and embeds this page using a layer or <DIV> HTML tag. The SCO can now be taken. Once the SCO is finished, or the user navigates away from the page, the updated data model is committed back to the server. This is done using the Ajax technology (Garrett, Jesse James. 2005).

### 3.6 The Updated API Adapter

In the original Reload SCORM Player, the API Adapter was written purely in javascript. All of the API calls and rules were implemented as a set of objects in javascript. An additional hidden frame was used within the frameset, which contained a web form. When the data model for the SCO needed to be saved the model was passed as a series of name/value pairs to the hidden frame. The form in the hidden frame was then submitted to the Java servlet backend which could commit the model to disk. The frame in which the SCO resided could not reload itself, because it would lose its current state information. This is why a hidden frame was used. When the Reload SCORM Player was originally written, the technology known as “Ajax” was not widely available. An Ajax call within a web page can call the backend web server for new content without having to reload the entire page. The new APIAdapter now uses an Ajax call to submit the data model to the server.
3.7 The new SCO 1.2 Service maps information to IMS-LD Level B properties

![Diagram of SCO communication]

The API adapter submits the results of the SCO via an Ajax call back to the player servlet. The servlet then gets the current handle to the SCO service and updates the changes made by the user as s/he took the SCO. As part of this, each of the data model elements are also propagated over to the LDengine service. If there is a property defined within the UOL which corresponds to the same name as a SCO data model name, then that property is updated. In effect the SCO 1.2 service broadcasts any changes in the SCO data model to the LDengine via CCSI.

4 Testing the prototype with a real example

In order to test the system a UOL was developed in which the learner’s interactions with a SCO determine the learning flow. The UOL contains six acts, and the SCO includes a test with five questions. At the start of the Unit of Learning the user is only able to access the first Learning Activity. From here the user has access to the SCO which is located within the Environment section of the Learning Activity. The user clicks on the link which loads the SCO and then s/he posts answers to the questions. Depending on which questions the user answered correctly, the structure of the Unit of Learning changes (although a browser refresh may be needed to reflect this). New learning activities then appear in the following acts which are designed to help the learner answer the questions previously answered incorrectly.
5 Conclusions and further work

During the process of designing, implementation and testing of the functionality, we have documented some of the problems and complexity posed by connecting a new service to an IMS-LD runtime system. The CopperCore Service Interface layer provides the necessary framework, but actually implementing a new service requires knowledge of this specific API, and not a defined and agreed standard. The framework also is quite open, allowing the developer to write his/her own calls between LD Engine and new service. While this allows a large amount of freedom to the developer, it is also very abstract. This experience provides the basis for defining the requirements for a connection protocol which can generalise the solution we have implemented.

In terms of the functionality of the demonstrator developed, further work could involve the use of full ADL SCORM 1.2 (or 2004) packages as learning objects found within an IMS Learning Design package, rather than using SCOs pre-loaded into a SCORM runtime system.

Acknowledgement

This work has been partially sponsored by the TENCompetence Integrated Project that is funded by the European Commission's 6th Framework Programme, priority IST/Technology Enhanced Learning. Contract 027087 (http://www.tencompetence.org)

References


