Integration of E-Learning Systems With Repositories of Learning Objects

José Paulo Leal[1], Ricardo Queirós[2]
[1]Department of Computer Science, Faculty of Sciences University of Porto, Porto, Portugal
[2]Department of Informatics, ESEIG – Polytechnic Institute of Porto, Porto, Portugal
zp@dcc.fc.up.pt
ricardo.queiros@eu.ipp.pt

Abstract: This paper describes a communication model to integrate repositories of programming problems with other e-Learning software components. The motivation for this work comes from the EduJudge project that aims to connect an existing repository of programming problems to learning management systems. When trying to use the existing repositories of learning objects we realized that they are mainly specialized search engines and lack features for integration with other e-Learning systems. With this model we intend to clarify the main features of a programming problem repository, in order to enable the design and development of software components that use it. The two main points of this model are the definition of programming problems as learning objects and the definition of the core functions exposed by the repository. In both cases, this model follows the existing specifications of the IMS standard and proposes extensions to deal with the special requirements of automatic evaluation and grading of programming exercises. In the definition of programming problems as learning objects we introduced a new schema for meta-data. This schema is used to represent meta-data related to automatic evaluation that cannot be conveniently represented using the standard: the type of automatic evaluation; the requirements of the evaluation engine; or the roles of different assets - tests cases, program solutions, etc. In the definition of the core functions we used two different web services flavours - SOAP and REST - and described each function as an operation for each type of interface. We describe also the data types of the arguments of each operation. These data types consist mainly on learning objects and their identifications, but include also usage reports and queries using XQuery.

Keywords: e-Learning; learning objects; content packaging; repositories; web services.

1. Introduction

The University of Valladolid Online Judge (UVA Online Judge, n.d.) is being used for some years as a training tool, mostly for teams that participate in the International Collegiate Programming Contests (ICPC). In fact, the UVA repository includes problems from several ICPC contests, including all problem sets from regional and world finals of the last seven years.

The EduJudge project aims to open the UVA Online Judge's repository to pedagogical uses in secondary and higher education. This project integrates three main components types: Learning Management Systems (LMS), Learning Objects Repositories (LOR) and Evaluation Engines (EE). A communication model between these components must be defined in order to the LOR be used for managing the collections of programming exercises and retrieving those suited to the profile of a particular student. In this model, the LOR plays an important role, since it responds to the request for services from the other components. These operations include the submission, search and download of learning objects.

The majority of the repositories of Learning Objects (LO) existing nowadays were not designed to support automatic integration with e-Learning systems: they are meant just for interactive use. Human interaction is necessary to select LOs with both the appropriated instructional content and the format required by a particular e-Learning system. In fact, this task is difficult to automate since repositories store different types of LOs, ranging from simple HTML files to complex SCORM (2004) compliant objects.

A tighter connection between repositories and other e-Learning systems is justifiable only when there is a large number of LOs in a common format and in the same domain, as in the UVA Online Judge. In this case an e-Learning system can automatically select a LO based on its meta-data and even try to adjust it to a specific student's profile. To achieve this goal it is necessary to define a flexible and

platform independent communication service layer to connect repositories with other e-Learning components.
The remainder of this paper is organized as follows: Section 2 presents a general view of the repositories and the main requirements and recommendations regarding the interoperation with LO repositories. The following section presents the model of our repository, including: the definition of programming problems as learning objects; the overall architecture of the repository and the main operations it provides. Finally, we conclude with a perspective of future work.

2. State of the art

A repository of learning objects can be defined as a 'system that stores electronic objects and meta-data about those objects' (Holden, 2004:1). The need for this kind of repositories is growing as more educators are eager to use digital educational contents and more of it is available. The Jorum Team made a comprehensive survey (2006) of the existing repositories and noticed that most of these systems do not store actual learning objects. They just store meta-data describing LOs, including pointers to their locations on the Web, and sometimes these pointers are dangling. Although some of these repositories list a large number of pointers to LOs, they have few instances in any category, such as programming problems. Last but not least, the LOs listed in these repositories must be manually imported into a LMS. An evaluation engine cannot query the repository and automatically import the LO it needs. In summary, the current repositories are specialized search engines of LOs and not adequate for feeding an automatic evaluation engine.

Based in other surveys, Holden (2004: 15-18) shows that users are concerned with issues that are not completely addressed by the existing systems, such as interoperability. The communication model of the repository should be based on international standards, such as those proposed by the IMS Digital Repositories specification (2003). The IMS DRI provides recommendations for common repository functions, namely the submission, search and download of LOs. It recommends the use of web services to expose the repository functions. Moreover, these technologies simplify the discovery and consumption of the repository's services, thus providing the basis for a Service Oriented Architecture (SOA) (Girardi, 2004).

Two main protocols provide a communication layer between remote components, namely, the Simple Object Access Protocol (SOAP) (2007), defined by W3C, and Representational State Transfer (REST) (Fielding, 2000). SOAP web services are usually action oriented, specially when used in Remote Procedure Call (RPC) mode, while REST web services are object (resource) oriented. SOAP web services are normally implemented by an off the shelf SOAP engine such as Axis (2006). The web services based on the REST style are implemented directly over the HTTP protocol, using, for example, Java servlets, mostly to put and get resources, such as LOs and usage data.

Besides the features of the repository it's important to take other important decisions, such as the definition of programming problems as LO according to the existing standards. The most widely used standard for LO is the content packaging format defined by IMS Global Learning Consortium (IMS 2008). The IMS Content Packaging (2004) uses an XML manifest file wrapped with other resources inside a zip file. The manifest includes the IEEE Learning Object Metadata (IEEE LOM) standard (2002) to describe the learning resources included in the package. However, LOM was not specifically designed to accommodate the requirements of automatic evaluation of programming problems and, in our view, needs to be extended for that purpose. Friesen (2004) mentions four ways that have been used to extend the IEEE LOM model:

- combining the IEEE LOM elements with elements from other specifications (this approach can introduce new categories to the standard);
- defining extensions to the IEEE LOM elements while preserving its set of categories;
- simplifying LOM, reducing the number of LOM elements and the choices they present;
- extending and reducing simultaneously the number of LOM elements.

Following this extension philosophy, the IMS Global Learning Consortium upgraded the Question & Test Interoperability (2005) specification. QTI describes a data model for questions and test data and, unlike in its previous versions, extends the IEEE LOM with its own meta-data vocabulary. QTI was designed for questions with a set of pre-defined answers, such as multiple choice, multiple response, fill-in-the-blanks and short text questions. It supports also long text answers but the specification of their evaluation is outside the scope of the QTI. Although long text answers could be used to write the program's source code, there is no way to specify how it should be compiled and executed, which test data should be used and how it should be graded. For these reasons we consider that QTI is not adequate for automatic evaluation of programming exercises, although it may be supported for sake of compatibility with some LMS.

3. Communication model

The repository will play a main role in the overall architecture of the EduJudge project, since it will act as a service provider for the other e-Learning systems. The clients of the repository need to understand two key points of the communication model: the definition of *programming problems as learning objects*, based on the IMS CP specification; and the *core functions of the repository*, based on the IMS DRI specification.

3.1 Programming problems as learning objects

The corner stone of this definition of programming problems as learning objects is automatic evaluation. Learning objects should include all data relevant for their automatic evaluation. Consequently, this definition assumes the existence of a component responsible for evaluating learners attempts based on the learning object and producing a result. Moreover, it needs also to assume one (or more) evaluation model(s) to relate attempts, learning objects and results. After considering several possible alternatives we decided on a single and simple evaluation model.

1. The evaluator receives:
   - a reference to the learning object with a programming problem;
   - an attempt to solve it - a single file, a program or an archive containing files of different types (e.g. JAR, WAR);
   - a reference to the learner submitting the attempt.
2. The evaluator processes this data as follows:
   a) loads the learning object from a repository using its reference;
   b) uses the assets available in the LO (static tests, generated tests, unit tests, etc.) according to their role;
   c) produces a result (correction, classification and feedback) that may depend on the learner's reference;
   d) stores the result for future incremental feedback to the same learner (optional).
3. The evaluator returns the result immediately or with a short delay.

Assuming this simple model, the learning object meta-data simply assigns a role to each asset. It is the responsibility of the evaluation component to use each asset appropriately according to its role. We considered defining more specialized evaluation models. For instance, the LO may include unit tests to perform evaluation instead of using test cases. Unit testing seems like a reasonable candidate for its own specialized evaluation model, requiring a source program for evaluation and replacing test data. However, the same thing can be done without a unit testing framework (say JUnit) but with some boilerplate code linked with the learner's attempt. In this case it may help (or not) to use a test data, that would be associated with a "standard" evaluation model. In every specialized model we considered, requiring some features and excluding others, we could come up with ways to combine it with assets from other evaluation models. In the end we had this simple and maximal evaluation model with several optional extension points (either the resource is available or not).

Although maximal, it should be notice that some kinds of programming problems are excluded from this evaluation model. For instance, programming problems where the evaluator aggregates programs submitted by two or more learners are excluded from this model. We considered also including this case as a second evaluation model. However, this type of programming problem is absent from the UVA repository and we know too little about the assets it requires, and thus we decided to postpone that decision to a next version of LO definition.

As mentioned before, we defined programming problems as learning objects based on the IMS CP specification. This standard was defined for LO in general, not specifically for programming problems. In particular, the IMS CP schemata (including the IEEE LOM) lack features for describing all the resources required to perform the automatic evaluation of programming problems. For instance, there is no way to assert the role of specific resources, such as test cases or solutions. Fortunately, IMS CP was designed to be straightforward to extend it and thus we were able to use this standard for our purpose of defining programming problems as learning objects.

An IMS CP learning object assembles resources and meta-data into a distribution medium, in our case a file archive in zip format, with its content described in a file named `imsmanifest.xml` in the root level. The manifest contains four sections: meta-data, organizations, resources and sub-manifests. The main sections are meta-data, which includes a description of the package, and resources, containing a list of references to other files in the archive (resources) and dependency among them.

Meta-data information in the manifest file usually follows the IEEE LOM schema, although other schemata can be used. These meta-data elements can be inserted in any section of the IMS CP manifest. In our case, the meta-data that cannot be conveniently represented using LOM is encoded in elements of a new schema - the EduJudge Meta-data Specification (EJ MD) - and included only in the meta-data section of the IMS CP. This section is the proper place to describe relationships among resources, as those needed for automatic evaluation and lacking in the IEEE LOM. To relate this meta-data with the corresponding resources we use an IDREF attribute on the EJ MD meta-data elements pointing to an ID attribute on the IMS CP `resource` element. The compound schema can be viewed as a new application profile that combines meta-data elements selected from several schemata. This approach is similar to the SCORM 1.2 application profile that extends IMS CP with more sophisticated sequencing and Contents-to-LMS communication. The elements of EJ MD schema are embedded in an IMS CP manifest file using an XML namespace. The URI of the current version of this namespace is http://www.edujudge.eu/ejmd_v1. This extension complies with the IMS Package Conformance Level 1: the package includes a manifest file (imsmanifest.xml) that contains additional namespace extensions, described using a schema, also included within the package.

The structure of the archive file, acting as distribution medium and containing the programming problem as a LO, is depicted in Figure 1. The archive contains several files represented in the diagram as grey rectangles. The manifest is an XML file and its elements' structure is represented by white rectangles. Different elements of the manifest comply with different schemata packaged in the same archive, as represented by the dashed arrows: the manifest root element complies with the IMS CP schema; elements in the metadata section may comply either with IEEE LOM or with EJ MD schemas; metadata elements within resources may comply either with IEEE LOM or IMS QTI. Resource elements in the manifest file reference assets packaged in the archive, as represented by the solid arrows.

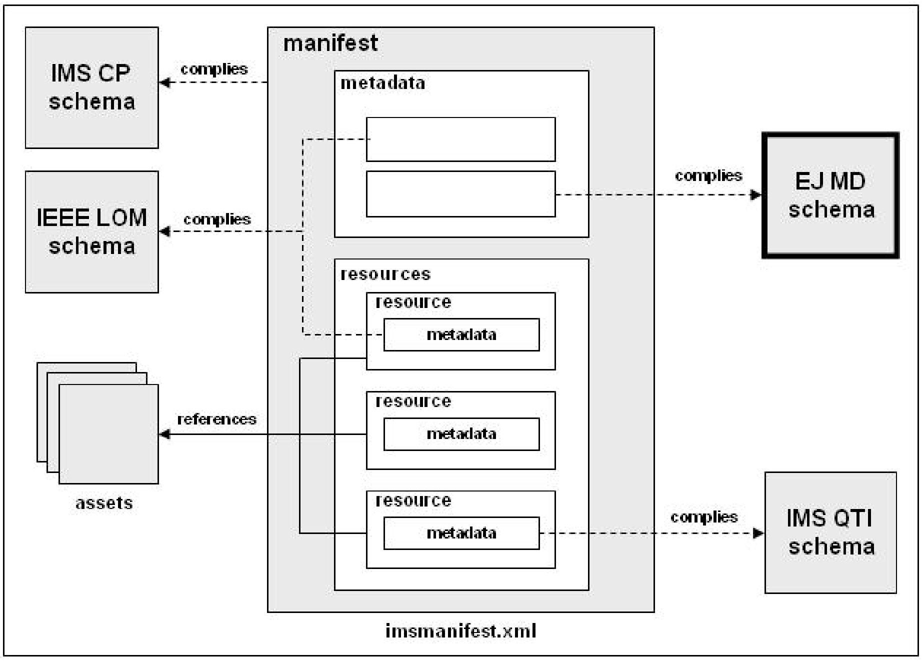

Figure 1: The structure of a programming exercise as a learning object

Another challenge we faced was to distinguish between LO identification and LO location. URL's are a convenient way to locate LOs since they can be used to download them. Within a repository an URL can also be used to identify a LO. However, being a resource location it cannot identify multiple copies of the same LO in several repositories. If a LO is replicated to another repository, the new URL loses the reference to the original. Hence, an URL cannot be seen as an identification of a LO.

The standard way to deal with this problem proposed by the IMS DRI is to use resolution services. These services enable a single name to be used persistently to manage the object, even its location changes. Examples of resolution systems for finding the appropriate copy, or copies of an item stored in multiple locations, are DOI (2006), OpenURL (2006) and PURL (2006).

Our model is concerned only with the communication between e-Learning systems and the repository. Hence, the location of a LO is sufficient for the purpose of identifying it within this point-to-point communication. Therefore, in the core functions of the repository we make extensive use of URL to identify the LO. It should be noted that this use of URLs to locate/identify LO does not preclude with the use of other identifications recorded in the meta-data of the LO itself, using (or not) any of the resolution services mentioned before.

3.2 Core functions of the repository

In this sub-section we identified a set of core functions that the repository must expose. The life cycle of a LO starts with the reserve of an identification and the submission to the repository. Following that, the LO is available for searching and delivering from other e-Learning systems. Figure 2 shows an UML diagram to illustrate the sequence of core functions invocations from these e-Learning systems to the Learning Objects Repository (LOR).

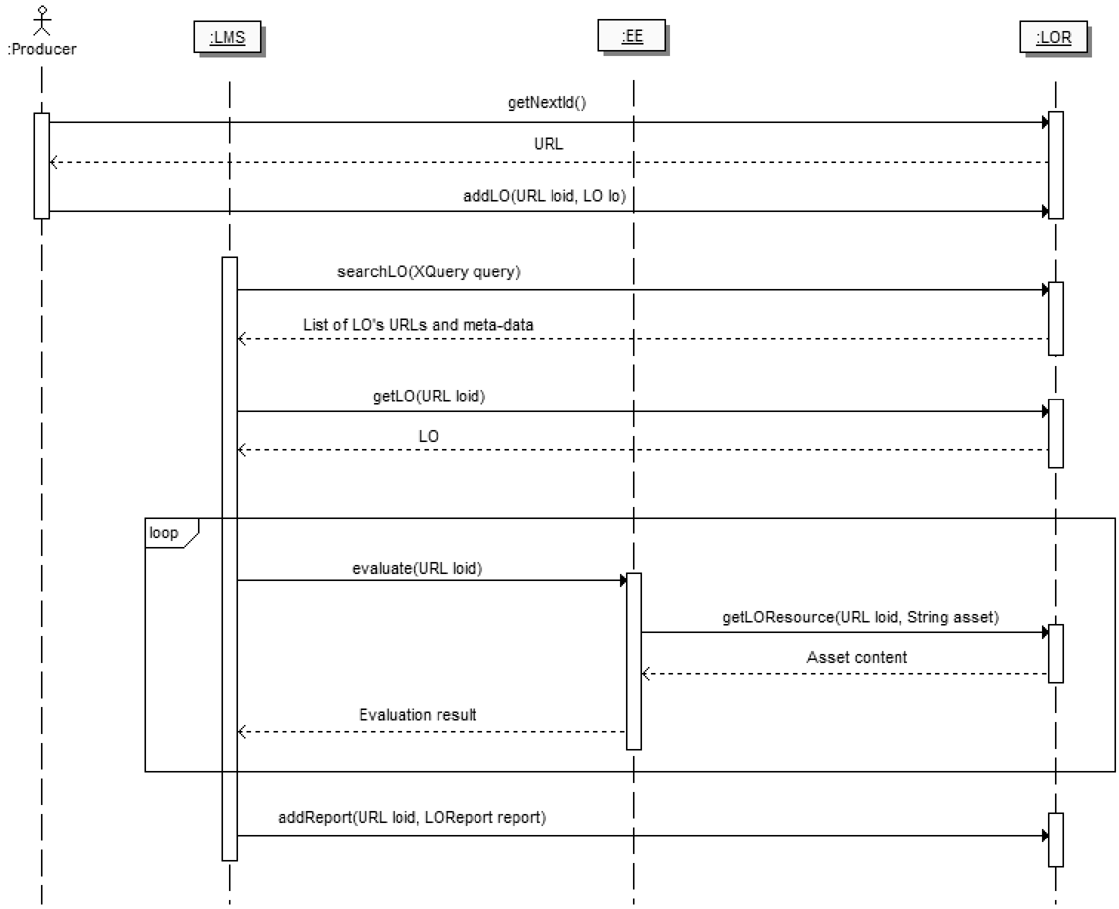

Figure 2: Repository's sequence diagram

We distinguish two types of systems: Learning Management systems (LMS) that present the programming exercise to the student and the Evaluation Engine (EE) responsible for the automatic evaluation and grading of the students attempt to solve it.

To comply with standards, the IMS DRI recommends the implementation of core functions as web services. We choose to implement two distinct flavours of web services: SOAP and REST. The reason to implement two distinct web service flavours is to promote the use of the repository by adjusting to different architectural styles.

The core functions of the repository are summarized in Table 1. Expect for the reserve/register and the report/store, all functions belong to the DRI specification. The reserve/register function provides the generation of identifiers for the LOs to submit; the report/store function provides reporting of LOs usage data. Each function is associated with the corresponding operations in both SOAP and REST web services interfaces. The SOAP interface exposes a method using RPC and we present the method's signature. For the REST interface is shown the HTTP method (GET, POST, or PUT), the requested URL and its input and output, following the Unix syntax of redirection operators. Strings in italic are replaced by values of that type.

Table 1: Core functions of the repository

Function	SOAP	REST
reserve/register	URL getNextId()	GET /nextId > URL
submit/store	addLO(URL loid, LO lo)	PUT URL < *LO*
request/deliver	LO getLO(URL loid)	GET *URL* > *LO*
report/store	addReport(URL loid, LOReport report)	PUT *URL*/report < *LOREPORT*
search/expose	XML searchLO(XQuery query)	POST /query < *XQUERY* > *XML*
alert/expose	RSS getUpdates()	GET /rss > RS

In the SOAP interface, complex data - such as LO packages, XQuery (2007) files or LO usage reports – is upload as attachments in their original formats using the SwA specification (2000), instead of serialized in a binary type such as xsd:base64Binary or xsd:hexBinary, as recommended by the IMS DRI. In the remainder we detail the operations behind more complex core functions.

The *Register/Reserve function* requests an unique ID from the repository. We separated this function from Submit/Store in order to allow the inclusion of the ID in the meta-data of the LO itself. This ID is an URL that must be used for submitting a LO. The producer may use this URL as an ID with the guarantee of its unicity and the advantage of being a network location from where the LO can be downloaded.

The *Submit/Store function* copies a LO to a repository and makes it available for future access. This operation receives as argument an IMS CP with the EJ MD extension and an URL generated by the Register/Reserve function with a location/identification in the repository. This operation validates the LO conformity to the IMS CP level 1 and stores the package in the internal database;

The *Report/Store function* associates an usage report to an existing LO. This function is invoked by the LMS to submit a final report, summarizing the use of a LO by a single student. This report includes both general data on the student's attempt to solve the programming exercise (e.g. data, number of evaluations, success) and particular data on the student's characteristics (e.g. gender, age, instructional level). The former is represented as a fixed set of attributes and includes the following data enumerated in Table 2.

Table 2: Student's attempt general data

Attribute	Content	Description
lo-id	URL	reference to LO
data	timestamp	data/time of usage
time	integer (seconds)	resolution time
attempts	integer	number of attempts
success	boolean	success in solving problem

We decided to create a meta-model for representing data to characterize students. This meta-model must be abstract enough to accommodate any unexpected requirements and simple enough to provide an efficient implementation of the search features of the repository. Having this in mind we decided to represent a student as a collection of attribute-values pairs, without enforcing the use of any attributes in particular. The attributes for characterizing students will not be fixed by the repository and cannot be assumed to be present (or absent). Nevertheless, a standardization of attribute names describing students will enable an LMS component to reuse information recorded by another LMS, even from a different vendor. The Table 3 shows some of these attributes.

Table 3: Student's characteristics particular data

Attribute	Content	Description
gender	`male female`	gender of student
age	integer	age of student when (solving to problem)
country	iso-code of country	student's country of residence
language	iso-code of language	student's native language
level	integer	instruction level

With this data, the LMS will be able to dynamically generate presentation orders based on previous uses of LO, instead of using fixed presentation orders.

The *Search/Expose function* enables the e-Learning systems to query the repository using the XQuery language, as recommended by the IMS DRI. This approach gives more flexibility to the client systems to perform any queries supported by the repository's data.

To write queries in XQuery the programmers of the client systems need to know the repository's database schema. These queries are based on both the content of the LO manifest and the LOs' usage reports, and can combine the two document types. As mentioned in the previous sub-section, the LO manifest schema complies with the IMS CP schema. The schema of LOs usage reports was also introduced above.

The programmer needs also to know that the database is structured in collections. A collection is a kind of a folder containing several resources and also other folders. From the XQuery point of view the database is a collection of manifest files. For each manifest file there is a nested collection containing the usage reports.
As an example of a simple search, suppose we want to find all title elements in the LO collection with an easy difficulty level. The following XQuery locates all such elements.

```
declare namespace imsmd="http://www.imsglobal.org/xsd/imsmd_v1p2";
for $p in //imsmd:lom
where contains
($p/imsmd:educational/imsmd:difficulty/imsmd:value/imsmd:langstring,
"easy")
return $p/imsmd:general/imsmd:title/imsmd:langstring/text()
```

In the above example the result is a set of strings. Alternatively, it can be a XML document. In this case it is possible to format the result using an Extensible Language Transformation (1999) stylesheet. Frequent queries can be compiled and cached as XQuery procedures.

The *Alert/Expose function* notifies users of changes in the state of the repository using an RSS feed. With this option a user can have up-to-date information through a feed reader.

4. Conclusion and future work

In this paper we described a communication model to integrate a repository of programming exercises with other e-Learning systems. The main contribution of this work is the extension of the existing specifications based on the IMS standard to the particular requirements of automatic evaluation. We focused mainly on two parts:

- the definition of programming problems as LO;
- the core functions of the repository.

For the first part we defined an evaluation model to base the programming problems and extended the IMS CP specification with a schema for representing meta-data related to automatic evaluation.

We detail the actions needed to define LOs from a domain that is not covered by the IEEE LOM in a way that can be reproduced in similar contexts.

For the second part we proposed two distinct flavours of web services and defined the operation for each function in both flavours. We also explained each data type used in these operations, based on the IMS DRI specification. We extended also this specification to separate registering LO from submitting them, in order to use LO locations as their IDs, and to submit reports on the LO's usage. This last feature, the ability to record usage reports of a LO, will be the basis to support a next generation of LMS with the ability to tailor the presentation order of programming exercises to the needs of a particular learner.

The work defined in this paper corresponds to an initial stage in the EduJudge project. This project aims to open the UVA repository of programming problems to LMS used in secondary and higher education. Although we do not anticipate major changes in the model described in this paper, we expect some challenges posed by the actual implementation of the repository and the needs of the other components of the EduJudge system. In particular, we anticipate the need to revise the schema that extends IMS CP and the schema that defines usage reports.

Acknowledgements

This work is part of the project entitled "Integrating Online Judge into effective e-learning", with project number 135221-LLP-1-2007-1-ES-KA3-KA3MP. This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

References

AXIS (2006) *Apache AXIS 1.4 Final*, [Online], Available: http://ws.apache.org/axis/ [3 Jul 2008].

DOI (2006) *Digital Object Identifier System,* [Online], Available: http://www.doi.org/ [3 Jul 2008].

Fielding, R. (2000) *Architectural Styles and the Design of Network-based Software Architectures (Phd dissertation)*, [Online], Available: http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm [10 Jul 2008].

Friesen, N. (2004) *Semantic and Syntactic Interoperability for Learning Object Metadata. In: Hillman, D. (ed.) Metadata in Practice. Chicago, ALA Editions*, [Online], Available: http://www.cancore.ca/semantic_and_syntactic_interoperability.html [11 Jun 2008].

Girardi, R. (2004) *Framework para coordenação e mediação de Web Services modelados como Learning Objects para ambientes de aprendizado na Web,* [Online], Available: http://www2.dbd.puc-rio.br/pergamum/tesesabertas/0220942_04_pretextual.pdf [12 Jan 2008].

Holden, C. (2004) *What We Mean When We Say "Repositories" User Expectations of Repository Systems*, Academic ADL Co-Lab, [Online], Available: http://www.hewlett.org/NR/rdonlyres/158FC043-A56F-43C6-ABA7-EB9A62656FCB/0/RepoSurvey2004-1.pdf [10 Jul 2008].

IEEE LOM (2002) *IEEE Standard for Learning Object Metadata IEEE 1484.12.1-2002* [Online], Available: http://www.ieeeltsc.org/standards/1484-12-1-2002/ [3 Jul 2008].

IMS CP (2004) *IMS Content Packaging v1.1.4 Final specification,* [Online], Available: http://www.imsglobal.org/content/packaging/index.html [10 Jul 2008].

IMS DRI (2003), *IMS Digital Repositories v1.0 Final specification*, [Online], Available: http://www.imsglobal.org/digitalrepositories/index.html [10 Jul 2008].

IMS QTI (2005) *IMS Question and Test Interoperability v2 Final specification,* [Online], Available: http://www.imsglobal.org/question/index.html [3 Jul 2008].

JORUM team (2006) *E-Learning Repository Systems Research Watch,* [Online], Available: http://www.jorum.ac.uk/docs/pdf/Repository_Watch_final_05012006.pdf [10 Jul 2008].

OPENURL (2006) *OpenURL Standard*, [Online], Available: http://www.oclc.org/research/projects/openurl/default.htm [3 Jul 2008].

PURL (2006) *Persistent Uniform Resource Locator*, [Online], Available: http://www.purl.org [3 Jul 2008].

SCORM (2004) *Scorm 2004 3rd Edition*, [Online], Available: http://www.adlnet.gov/scorm/index.aspx [10 Jul 2008].

SOAP (2007) *Version 1.2 Part 0: Primer (Second Edition)*, [Online], Available: http://www.w3.org/TR/soap12-part0/ [10 Jul 2008].

SwA (2000) *SOAP Messages with Attachments (W3C Note)*, [Online], Available: http://www.w3.org/TR/SOAP-attachments [10 Jul 2008].

UVA Online Judge, [Online], Available: http://icpcres.ecs.baylor.edu/onlinejudge [10 Jul 2008].

XQuery (2007) *Extensible Markup Language (XML) 1.1 (Second Edition),* [Online], Available: http://www.w3.org/TR/xquery/ [10 Jul 2008].

XSLT (1999) *Extensible Markup Language (XML) 1.1 (Second Edition),* [Online], Available: http://www.w3.org/TR/xslt [10 Jul 2008].