# CrimsonHex: a Service Oriented Repository of Specialised Learning Objects

José Paulo Leal[1] and Ricardo Queirós[2],

[1] CRACS & DCC-FCUP, University of Porto, Portugal
zp@dcc.fc.up.pt
[2] CRACS & DI-ESEIG/IPP, Porto, Portugal
ricardo.queiros@eu.ipp.pt

**Abstract.** The corner stone of the interoperability of eLearning systems is the standard definition of learning objects. Nevertheless, for some domains this standard is insufficient to fully describe all the assets, especially when they are used as input for other eLearning services. On the other hand, a standard definition of learning objects in not enough to ensure interoperability among eLearning systems; they must also use a standard API to exchange learning objects. This paper presents the design and implementation of a service oriented repository of learning objects called crimsonHex. This repository is fully compliant with the existing interoperability standards and supports new definitions of learning objects for specialized domains. We illustrate this feature with the definition of programming problems as learning objects and its validation by the repository. This repository is also prepared to store usage data on learning objects to tailor the presentation order and adapt it to learner profiles.

**Keywords:** eLearning, Repositories, SOA, Interoperability.

## 1 Introduction

Component oriented systems are predominant in most of eLearning platforms. Despite their success, they have also been target of criticism: their tools are too general and they are difficult to integrate with other eLearning systems [1]. These issues led to a new generation of service oriented eLearning platforms, easier to integrate with other systems. This paper focuses the design and implementation of crimsonHex, a service oriented repository of specialized learning objects (LO). It provides standard compliant repository services to a broad range of eLearning systems, exposing its functions using two alternative web services flavours. The definition of LOs can be customized to the requirements of these systems. To illustrate this customization we document the process of extending generic LOs to a specific learning domain – programming exercises.

The extended definition of LOs to programming problems is being used in a European research project called EduJudge. This project aims to integrate a collection

of problems created for programming contests into an effective educational environment. This project includes three types of services:

- Learning Objects Repository (LOR) to store the exercises and to retrieve those suited to a particular learner profile;
- Evaluation Engine (EE) to automatic evaluate and grade the students attempt to solve the exercises;
- Learning Management System (LMS) to manage the presentation of exercises to learners.

The remainder of this paper is organized as follows: Section 2 traces the evolution of eLearning systems with emphasis on the existing repositories. In the following section we extend the generic definition of a LO as a programming problem. Then, we present the architecture of the repository and highlight its components, functions and communication model. The next section, we focus on the main facets of its implementation: storage, validation, interface and security. In Section 6 we describe the tests and evaluation of the repository. Finally, we conclude with a summary of the main contributions of this work and a perspective of future research.

## 2   State of Art

The evolution of eLearning systems comprises the last two decades. In the "first generation", eLearning systems had a monolithic architecture and were used on a specific learning domain [1]. Gradually, these systems evolved and became independent from a particular domain, incorporating tools that can be effectively reused in several scenarios. Different kinds of component based eLearning systems targeted to a specific aspect of eLearning, such as student or course management. There are several acronyms trying to differentiate between these types of eLearning systems. Nevertheless, the trend in eLearning systems is integration therefore most of them evolved to the same set of standard features and many of these acronyms are used as synonyms. The most usual designation of such systems is the LMS (e.g. Moodle, Sakai, and WebCT).

This "second generation" allows the sharing of learning objects and learner information. In this phase, some standards emerge, namely, IMS Content Packaging (IMS CP), Sharable Content Object Reference Model (SCORM) and IEEE Learning Object Metadata (IEEE LOM) that brought interoperability and content sharing to eLearning. Despite the advantages of these systems and standards, some criticism arose for several reasons, such as: focus on content, lack of support to response to specific needs and difficult to integrate with other eLearning systems.

These issues triggered a new generation of eLearning platforms based on services that can be integrated in different scenarios. This new approach provides the basis for a Service Oriented Architecture (SOA) [2]. In the last few years there have been initiatives to adapt SOA to eLearning, such as the eLearning Framework (ELF) and the IMS Abstract Framework. These initiatives contributed with the identification service usage models and a categorisation of genres of services for eLearning [3]. Some of these services are related with a key system in an eLearning platform – the repository.

A repository of learning objects can be defined as a 'system that stores electronic objects and meta-data about those objects' [4]. 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. One of the best examples is the repository Merlot (Multimedia Educational Resource for Learning and Online Teaching). The repository provides pointers to online learning materials and includes a search engine. The Jorum Team made a comprehensive survey [5] 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, most of the current repositories are specialized search engines of LOs and not adequate for interact with other eLearning systems, such as, feeding an automatic evaluation engine.

Based in other surveys [4] the users are concerned with issues that are not completely addressed by the existing systems, such as interoperability. Some major interoperability efforts [6] were made in eLearning, such as, NSDL, POOL, ELENA/Edutella, EduSource and IMS Digital Repositories (IMS DRI). The IMS DRI specification was created by the IMS Global Learning Consortium (IMS GLC) and provides a functional architecture and reference model for repository interoperability. 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 based on the Simple Object Access Protocol (SOAP) protocol, defined by W3C. Despite the SOAP recommendation, other web service interfaces could be used, such as, Representational State Transfer (REST) [7].

Besides the interoperability features of the repository its necessary to look to the current standards that describes learning objects. As we said before, the actual standards are quite generic and not adequate to specific domains, such as the definition of programming problems. The most widely used standard for LO is the IMS CP. This content packaging format uses an XML manifest file wrapped with other resources inside a zip file. The manifest includes the IEEE LOM standard 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. For instance, there is no way to assert the role of specific resources, such as test cases or solutions. Fortunately, LOM was designed to be straightforward to extend it. Next, we enumerate four ways that have been used [8] to extend the LOM model:

- combining the LOM elements with elements from other specifications;
- defining extensions to the 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 GLC upgraded the Question & Test Interoperability (QTI) specification. QTI describes a data model for questions and test

data and, unlike in its previous versions, extends the 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. Recently, IMS GLC proposed the IMS Common Cartridge that bundles the previous specifications and its main goal is to organize and distribute digital learning content.

## 3   Specialised Learning Objects

We defined programming problems as learning objects based on the IMS CP. 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.
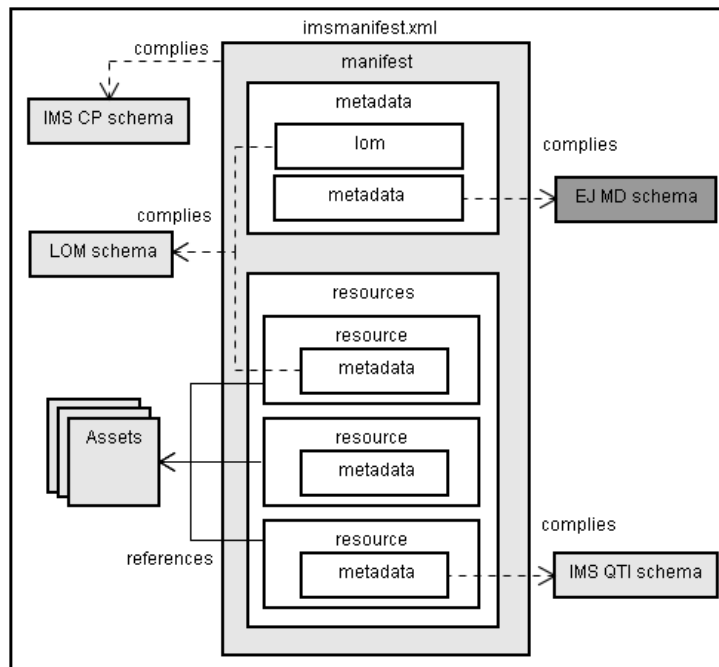


**Fig. 1.** Structure of a programming problem as a learning object.

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 - 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. 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 structure of the archive, 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 in solid arrows.

## 4 Architecture

In this section, we present the architecture of the crimsonHex repository described by the UML component diagram shown in Figure 2. Using the **API crimsonHex**, the repository exposes a core set of functions that can be efficiently implemented by a simple and stable component. All other features are relegated to auxiliary components, connected to the central component using this API. Other eLearning systems can be plugged into the repository using also this API.

### 4.1 Components

In the design of crimsonHex we set some initial requirements, in particular, to be simple and efficient. Simplicity is the best way to promote the reliability and efficiency of the repository. In fact, the core operations of the repository are uploading and downloading LO - ZIP archives - which are inherently simple operations that can be implemented almost directly over the transport protocol. Other features may need a more elaborate implementation but do not require the same reliability and efficiency of the core features. The architecture of crimsonHex repository is divided in three main components:

- **the Core** exposes the main features of the repository, both to external services, such as the LMS and the EE, and to internal components - the Web Manager and the Importer;

- **the Web Manager** allows the creation, revision, versioning, uploading/downloading of LOs and related meta-data, enforcing compliance with controlled vocabularies;
- **the Importer** populates the repository with existing legacy repositories. In the remainder we focus on the Core component, more precisely, its functions, communication model and implementation.
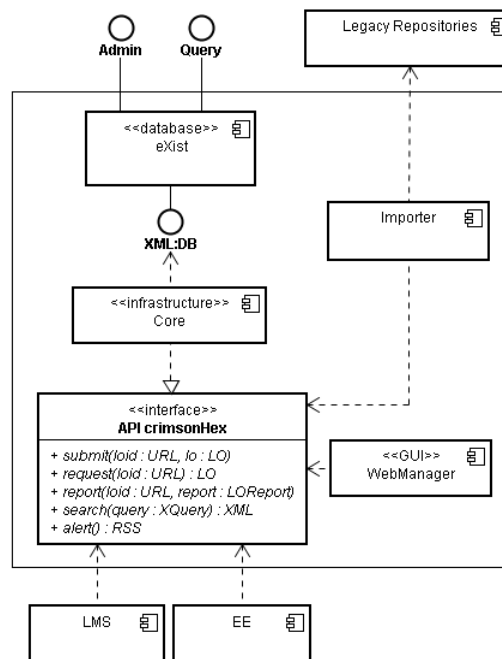


**Fig. 2.** Components diagram of the repository.

### 4.2 Functions

The Core component of the crimsonHex repository provides a minimal set of operations exposed as web services and based in the IMS DRI specification. The main functions are the following.

The **Register/Reserve** function requests a 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 uniqueness 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 Package Conformance and stores the package in the internal database;

The **Search/Expose** function enables the eLearning 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. The client developer 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.

```
declare namespace imsmd = "http://...";
for     $p in //imsmd:lom
where   contains($p//imsmd:difficulty,easy)
return  $p//imsmd:title//text()
```

The previous example displays a FLWOR ("For, Let, Where, Order by, Return") expression based in XQuery language to locate all such elements. This approach is used in SOAP requests. For REST requests we can simple write in a browser the URL: `http://host/crimsonHex?difficulty=easy`. In both approaches 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 XSLT (Extensible Stylesheet Language Transformation) file. For frequent queries it's possible to compile and cache them as XQuery procedures.

The **Report/Store** function associates a 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). 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. This function is an extension of the IMS DRI.

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


### 4.3 Communication Model

The communication model of the repository defines the interaction between the repository and the other eLearning systems. The model is composed by a set of core functions, most of them, exposed in the previous section. The figure 3 shows an UML diagram to illustrate the sequence of core functions invocations from these eLearning systems to repositories.
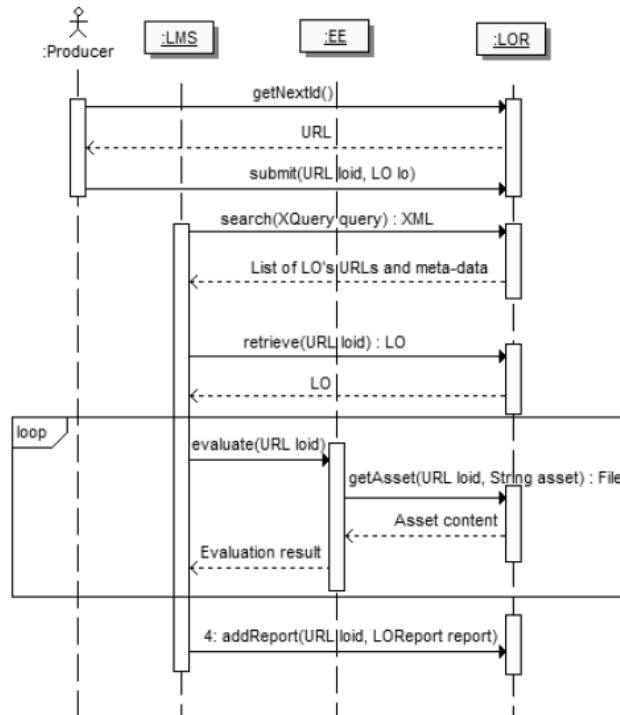
**Fig. 3.** Communication between the repository and the other eLearning systems.

The life cycle of a LO starts with the reserve of an identification and the submission of a LO to the repository. Next, the LO is available for searching and delivering to other eLearning systems. Then, the learner in the LMS could use the LO and submit it sending an attempt of the problem solution to the EE. Based in the feedback the learner could repeat the process. In the end, the LMS sends a report of the LO usage data back to the repository. This DRI extension will be, in our view, the basis for a next generation of LMS with the capability to adjust the order of presentation of the programming exercises in accordance with the needs of a particular student.

## 5 Implementation

In this section we detail the design and implementation of the Core component of crimsonHex on the Tomcat servlet container.

Reliability and efficiency were our main concern when designing the Core. The best way to achieve them is through the simplicity. These are the main design goal that guided us in the development of the four main facets of the Core - storage, validation, interface and security - analysed in the following subsections.

## 5.1 Storage

Searching LOs in the repository is based on queries on their XML manifests. Since manifests are XML documents with complex schemata we paid particular attention to databases systems with XML support: XML enabled relational databases and Native XML Databases (NXD).

XML enabled relational databases are traditional databases with XML import/export features. They do not store internally data in XML format hence they do not support querying using XQuery. Since queries in this standard are a DRI recommendation this type of storage is not a valid option. In contrast, NXD uses the XML document as fundamental unit of (logical) storage, making it more suitable for data schemata difficult to fit in the relational model. Moreover, using XML documents as storage units enables the following standards:

- XPath for simple queries on document or collections of documents;
- XQuery for queries requiring transformational scaffolding;
- SOAP, REST, WebDAV, XmlRpc and Atom for application interface;
- XML:DB API (or XAPI) as a standard interface to access XML datastores.
- XSLT to transform documents or query-results retrieved from the database.

We analysed several open source NXD, including SEDNA, OZONE, XIndice and eXist, Only eXist implements the complete list of the features enumerated above, which led us to select it as the storage component of crimsonHex. It has also two important features [9] worth mentioning: support for collections, to structure the database in groups of related documents and automatic indexes to speed up the database access**.**

## 5.2 Validation

The crimsonHex is a repository of specialized learning objects. To support this multi typed content the repository must have a flexible LO validation feature. The eXist NXD supports implicit validation on insertion of XML documents in the database but this feature could not be used for several reasons: LO are not XML documents (are ZIP files containing an XML manifest); manifest validation may involve many XML Schema Definition (XSD) files that are not efficiently handled by eXist; and manifest validation may combine XSD and Schematron validation and this last is not fully supported by eXist.

All LOs stored in crimsonHex must comply with the IMS Package Conformance that specifies it structure and content. This standard also requires the XSD validation of their manifests. For particular domains it is possible to configure specialized validations in crimsonHex by supplying a Java class implementing a specific interface. These validations extend those of the IMS Package Conformance and may introduce new schemata, even using different type definition languages, such as Schematron.

Validations are configured per collection of documents. Thus, different types of specialized LO may coexist in a single instance of crimsonHex. As mentioned before, IMS CP main schema imports many other schemata (more than 30) that according to the IMS Package Conformance must be downloaded from the Internet. This

requirement has a huge impact on the performance of the submit function. To accelerate this function we implemented a cache. A newly stored schema has a time to live of 1 hour. Outdated schemata are reloaded from their original Internet location using a conditional HTTP request that downloads it only if it has effectively changed.

### 5.3 Interface

To comply with standards, the IMS DRI recommends the implementation of core functions as web services. We chose to implement two distinct flavours of web services: SOAP and REST. SOAP web services are usually action oriented, mainly when used in Remote Procedure Call (RPC) mode and implemented by an off the shelf SOAP engine such as Axis.

**Table 1.** Core functions of the repository.

| Function | SOAP | REST |
|---|---|---|
| Reserve | URL getNextId() | GET /nextId > URL |
| Submit | submit(URL loid, LO lo) | PUT URL < LO |
| Request | LO retrieve(URL loid) | GET *URL* > *LO* |
| Search | XML search(XQuery query) | POST /query < *XQUERY* > *XML* |
| Report | Report(URL loid,LOReport rep) | PUT *URL*/report < *LOREPORT* |
| Alert | RSS getUpdates() | GET /rss > RSS |

The web services based on the REST style are object (resource) oriented and implemented directly over the HTTP protocol, using, for example, Java servlets, mostly to put and get resources, such as LOs and usage data. The reason to implement two distinct web service flavours is to promote the use of the repository by adjusting to different architectural styles. The repository functions are summarized in Table 1. Each function is associated with the corresponding operations in both SOAP and REST web services interfaces.

### 5.4 Security

Following the design principles of simplicity and efficiency we decided to avoid the management of users and access control in the Core. This decision does not preclude the security of this component since we can control these features in the communication layer. Since both web services flavours use HTTP as transport protocol we secure the channel using Secure Sockets Layer (SSL) (i.e. HTTPS). This ensures the integrity and confidentiality of assets in LO. To achieve authentication and authorization we rely on the verification of client certificates provided by SSL. In practice, to implement this approach we just needed to configure the servlet container (e.g. Tomcat) to support HTTPS requests with authorized certificates. Nevertheless, managing certificates is a comparatively complex procedure thus we provide a set of auxiliary functions in the core that act as a mini Certificate Authority (CA). These functions are used for managing and signing client certificates and their implementation is based on the Java Security APIs.

# 6 Tests and Evaluation

Reliability is one of our main concerns regarding the Core component of crimsonHex. We adopted JUnit as our automated unit testing framework since crimsonHex is implemented in Java and this tool is support by Eclipse, the Integrated Development Environment (IDE) used in this project. Apart from the unit tests, we created a tool for automatic generation of random requests to the repository, following the communication model summarized in Figure 3. The goal of this tool is two folded: to look for bugs in unpredicted sequences of requests and to stress-test the repository. The tool generates a random sequence of Core functions' invocations and records then in the Core's log file (through a Java-based logging utility called log4j). Errors generated by these request sequences are recorded by the Core in the same log files. After each test the log file is manually inspected looking for function sequences that originated errors. This approach was essential to discover errors that otherwise would only be detected in production. Efficiency and scalability are two other main concerns in the development of crimsonHex. To test performance we used the test tool to compare execution times of the main functions in the two supported web services interfaces: SOAP and REST. Each function has been repeated 10 times. Average function execution times for the set of functions are shown in Table 2.

**Table 2.** Average function execution times per interface (in seconds).

|      | submit | retrieve | Search |
|------|--------|----------|--------|
| SOAP | 4,53   | 1,57     | 2,23   |
| REST | 2,11   | 0,44     | 0,93   |

These figures show that our DRI extension, based on REST, twice as efficient as the standard SOAP interface. These results were expectable since the REST interface does not have to marshal request messages. In both interfaces submit times are significantly higher than the other functions due to weight of the validation process.

The scalability his other important issue. Scalability is bound by the database limits. The eXist NXD supports a maximum of $2^{31}$ documents and theoretically, documents can be arbitrary large depending on file system limits, e.g. the max size of a file in the file system, which have an influence. To test the scalability of eXist some queries were made [9] with increasing data volumes. The experiment shows linear scalability of eXist's indexing, storage and querying architecture.

# 7 Conclusions

In this paper we described the architecture, design and implementation of a repository of specialized learning objects called crimsonHex. The main contribution of this work is the extension of the existing specifications based on the IMS standard to the particular requirements of a specialized domain, such as, the automatic evaluation of programming problems. We focused mainly on two parts:

- the specialization of the definition of LO, where programming problems are given as a concrete example;
- the design of the repository, more precisely, its components, functions and details of its implementation.

For the first part 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 describe the design and implementation of a repository of specialized LOs. We adopt the IMS DRI and propose extensions to its recommendations, namely on the web service interfaces and on the standard functions. The new function 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.

In its current status crimsonHex Core can be deployed to a service oriented eLearning platform and is available for test and download at the following URL `http://mooshak.dcc.fc.up.pt:8080/crimsonHex/releases.jsp`. Our future work in this project includes developing a management and authoring tool; populating the repository with problem sets from existing sources, while classifying then and controlling their quality.

# References

1. Dagger, D., O'Connor, A., Lawless, S., Walsh, E., Wade, V.: Service Oriented eLearning Platforms: From Monolithic Systems to Flexible Services (2007)
2. Girardi, R.: Framework para coordenação e mediação de Web Services modelados como Learning Objects para ambientes de aprendizado na Web (2004)
3. Wilson, S., Blinco, K., Rehak, D.: An e-Learning Framework. Paper prepared on behalf of DEST (Australia). In: JISC-CETIS (UK) , Canada, (2004)
4. Holden, C.: What We Mean When We Say "Repositories" User Expectations of Repository Systems. In: Academic ADL Co-Lab, (2004)
5. JORUM team: E-Learning Repository Systems Research Watch. Technical report (2006)
6. Hatala, M., Richards, G., Eap, T., Willms, J.: The EduSource Communication Language: Implementing Open Network for Learning Repositories and Services. In: ACM symposium on Applied computing (2004)
7. Fielding, R.: Architectural Styles and the Design of Network-based Software Architectures - Phd dissertation, (2000)
8. Friesen, N.: Semantic and Syntactic Interoperability for Learning Object Metadata. In: Hillman, D. (ed.) Metadata in Practice. Chicago, ALA Editions, (2004)
9. Meier, W.: eXist: An Open Source Native XML Database. In: NODe 2002 Web and Database-Related Workshops, (2002)