SOFTWARE – PRACTICE AND EXPERIENCE
Softw. Pract. Exper. 2013; **43**:911–935
Published online 13 June 2012 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/spe.2132

crimsonHex: a learning objects repository for programming exercises‡

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

[1]*CRACS & DI-ESEIG/IPP, Porto, Portugal*
[2]*CRACS & DCC-FCUP, University of Porto, Portugal*

SUMMARY

A repository of learning objects is a system that stores electronic resources in a technology-mediated learning process. The need for this kind of repository is growing as more educators become eager to use digital educational contents and more of it becomes available. The sharing and use of these resources relies on the use of content and communication standards as a means to describe and exchange educational resources, commonly known as learning objects. This paper presents the design and implementation of a service-oriented repository of learning objects called crimsonHex. This repository supports new definitions of learning objects for specialized domains and we illustrate this feature with the definition of programming exercises as learning objects and its validation by the repository. The repository is also fully compliant with existing communication standards and we propose extensions by adding new functions, formalizing message interchange and providing a REST interface. To validate the interoperability features of the repository, we developed a repository plug-in for Moodle that is expected to be included in the next release of this popular learning management system. Copyright © 2012 John Wiley & Sons, Ltd.

Received 1 August 2011; Revised 4 May 2012; Accepted 10 May 2012

KEY WORDS: eLearning; repositories; SOA; standards; interoperability

1. INTRODUCTION

A learning object repository (LOR) is a type of software, similar to a digital library, which enables educators to share, manage and use educational resources. These resources, usually called learning objects (LOs), are defined as small, self-contained, reusable units of learning, usually tagged with metadata useful to catalogue and to search them. This concept has been promoted by the eLearning industry since 2001 and many repositories appeared as a way of storing and managing LOs [1]. Despite their relative success, they have also been target of criticism: the repository tools are too general and most of them do not implement metadata and communication standards making it difficult to integrate with other eLearning systems [2].

In this paper, we highlight the interoperability features of crimsonHex. This repository supports new definitions of learning objects for specialized domains and we illustrate this feature with the definition of programming exercises as learning objects and its validation by the repository. The repository also provides standard compliant repository services to a broad range of eLearning systems, exposing its functions using two alternative kinds of web services. For the sake of standards compliance these functions are based on the IMS Digital Repositories Interoperability (DRI) specification [3]. Our experience with using these recommendations led us to propose extensions to its

*Correspondence to: Ricardo Queirós, CRACS & DI-ESEIG/IPP, Porto, Portugal.
†E-mail: ricardo.queiros@eu.ipp.pt
‡Supporting information may be found in the online version of this article.

set of functions and to the XML binding that currently lacks a formal definition. To evaluate the proposed extensions to the IMS DRI specification and its implementation in the crimsonHex repository, we developed a crimsonHex plug-in for the 2.1 release of the popular Moodle learning management system (LMS). Moodle users will be able to download LOs from crimsonHex repositories because this LMS is expected to include the plug-in described in this paper in its distribution.

This research was carried out under a European project called EduJudge. The EduJudge project aims to integrate the 'UVA On-line Judge', an existing on-line programming trainer with an important number of programming exercises and users, into an effective educational environment [4] consisting of the e-learning platform Moodle and the competitive learning tool called QUESTOURnament. These systems will interact with a remote repository where the programming exercises will be stored.

Parts of the research described in this paper were presented at a conference on enterprise information systems [5,6], addressing the essential aspects of crimsonHex's development, and the use of this system from Moodle. This paper provides a comprehensive view of the crimsonHex system and details certain parts of its design, such as the definition of programming problems as learning objects and the Web interface. To provide a better understanding of the decisions taken when designing crimsonHex, this paper also adds a survey on learning object repositories and specifications related to learning objects.

The remainder of this paper is organized as follows: Section 2 includes a survey on repositories distinguishing two major types: learning objects repositories and digital libraries and presents the recent efforts made to foster the interoperability on repositories. In the following section we introduce the architecture of crimsonHex and its application interfaces. Then, we provide implementation details of a crimsonHex plug-in for Moodle 2.0 using the proposed IMS DRI extensions. Finally, we conclude with a summary of the main contributions of this work and a perspective of future research.

2. REPOSITORIES

A repository can be thought of as a storage area from which users can publish/retrieve resources. Most of these resources are described by metadata for discoverable purposes. The need for repositories has been growing in the last decade, because more educators are eager to create and use digital content and more of it is available. This growth led many to neglect interoperability issues that are fundamental to share educational resources and to (re)use them on different contexts.

In this section, we analyse the repository software distinguishing two types of repositories — digital libraries and learning objects repositories — and we provide examples of such types of systems. Then we take a careful look at recent efforts made to foster interoperability with repositories. These efforts are organized as two facets: content and communication. In the former, we focus on the metadata and package specifications used to describe and deploy learning content. In the latter, we highlight the publishing, searching and retrieval specifications used to enhance the communication between the repository and other e-Learning systems.

The evolution of the software and specification for repositories can be seen in Figure 1. In the next sections we detail both.

2.1. Software

There are several typologies to classify the repositories available in the literature. McGreal [7] categorises repositories into three types based on resource location: (i) those that house content primarily on site (e.g. MIT Open Courseware); (ii) those that provide metadata with links to resources housed at other sites (e.g. Merlot) — also called *referatories* [8]; and (iii) those that provide both content and links to external content (e.g. Ariadne). Ochoa and Duval [9] conducted a detailed quantitative study of the process of publication of learning objects in repositories and grouped repositories in five types: learning object repositories, learning object referatories, open courseware initiatives, LMS and institutional repositories. Other relevant studies [10–13] categorize repositories mainly by general characteristics such as language used, subject area, end users, fee type, quality control, etc.

DOI: 10.1002/spe

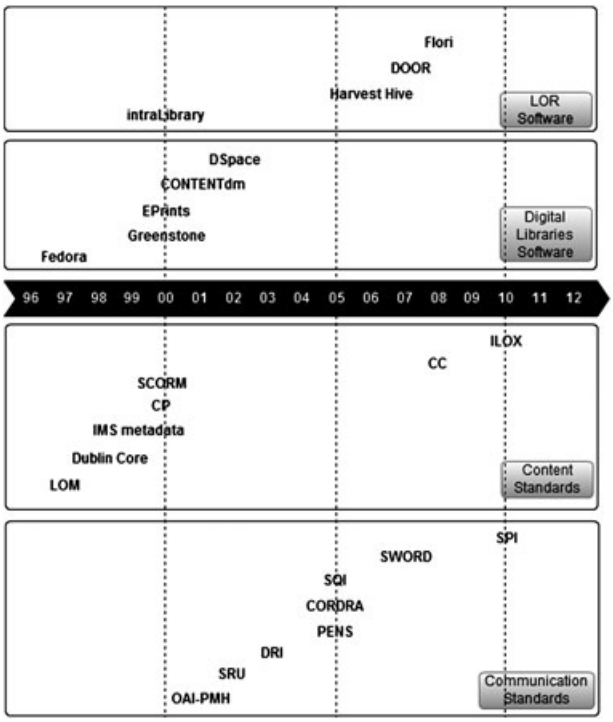

Figure 1. Evolution of software and specifications for repositories.

In this section, we categorize repositories to LORs and digital libraries (DLs) — two concepts that overlap to a certain extent. Also, we distinguish between the actual repositories and the software used to implement them, highlighting the differences in software features for both categories.

Digital libraries store documents in digital formats and metadata for those documents. Learning objects repositories store learning objects, which are objects containing educational content and metadata for those contents. At first sight a LOR seems to be basically a DL storing educational content, but there are also differences related to metadata and packaging. The metadata schemata used by DLs are generic (e.g. Dublin Core, DC and Metadata Encoding and Transmission Standard, METS), while those used by LORs are specific to eLearning (e.g. learning object metadata, LOM). Also, LORs package content and metadata in a single unit using eLearning standards (e.g. IMS Content Packaging (IMS CP), Sharable Content Object Reference Model (SCORM)), while DLs usually keep content and metadata separated.

In spite of these differences and similarities there are several references in the literature [14, 15] that attempt to implement LORs using DLs software. A reasonable explanation for this is the lack of software specifically for implementing LORs. In fact, unlike what happens with DLs, most implementations of LORs actually use home-grown software. Thus, the remainder of this section first analyzes the existing software for creating DLs before proceeding to the software for creating LORs.

2.1.1. Software for digital libraries. A nonexhaustive list of software to create DLs is presented in Table I [11]. This table reveals that these systems share many features, but they are far from equal because they have strengths and weaknesses in different functional areas.

According to the OpenDOAR (Directory of Open Access Repository) — an authoritative directory of academic open access repositories — and the Registry of Open Access Repositories data, in December 2011, the majority of repositories (in a sample of 2164 repositories) were built using the DSpace software (165 Washington Street, Suite #201 Winchester, MA 01890), as shown in Figure 2.

DOI: 10.1002/spe

Table I. Digital library software.

Repository	License	Metadata	Communication
CONTENTdm	Commercial	DC, METS	OAI, Z39.50
DigiTool	Commercial	DC, MODS,METS	OAI, Z39.50
DSpace	Free	DC, MODS,METS	OAI, SWORD, OpenSearch, REST,SRU/SRW
EPrints	Free	DC, MODS,METS	OAI, SWORD
Equella	Commercial	LOM	OAI, SWORD
Fedora	Free	LOM	OAI, SWORD
Greenstone	Free	METS	OAI, Z39.50
Zentity	Free	DC, METS	OAI, SWORD, RDFS

DC: Dublin Core; METS: Metadata Encoding and Transmission Standard; MODS: Metadata Object Description Schema; LOM: learning object metadata; OAI: Open Archives Initiative; SWORD: Simple Web-service Offering Repository Deposit; REST: Representational State Transfer; SRU: Search/Retrieve via URL; SRW: Search/Retrieve Web service; RDFS: Resource Description Framework Schema.

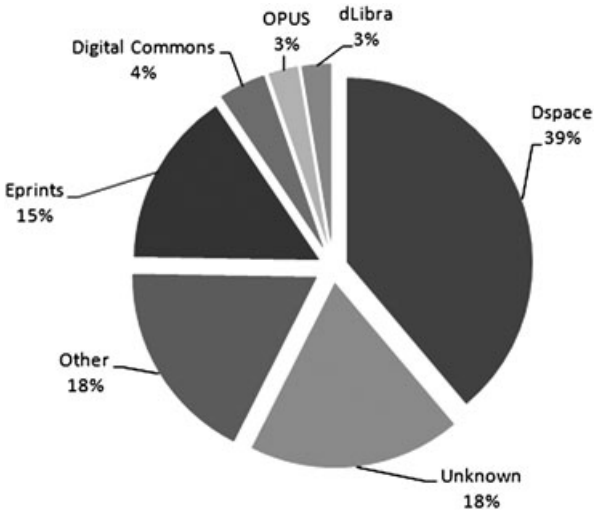

Figure 2. Usage of digital library software worldwide.

Software for DLs was designed for repositories of digital content in general not specifically for learning objects. Thus, this type of software in general lacks some of the features required by LORs such as the support for:

- eLearning metadata (e.g. LOM) and content packaging (e.g. SCORM)
- Federated searching (e.g. simple query interface (SQI), SRU/SRW, Open Knowledge Initiative (OKI) open service interface definitions (OSIDs))
- Classification using folksonomies (e.g. tags)
- Versioning, reviewing and evaluation
- Interoperability with eLearning systems (e.g. LMSs, authoring tools)

However, the newest versions of some of the systems listed in Table I recently added support for metadata export using learning standards such as LOM. These efforts led us to conclude that in the near future these types of repository will converge.

2.1.2. Software for learning objects repositories. A LOR is a repository that manages learning objects and their respective metadata. The majority of the LORs available nowadays are Web applications developed using nonrepository software because of the lack of support of these tools for the eLearning requirements as stated above. A nonexhaustive list of LORs is presented in Table II. All these LORs are free of charge provided that they are not used for commercial purposes.

DOI: 10.1002/spe

Table II. Learning objects repositories.

Repository	#LO	Type
Bepress	52678	Legal
BerkleeShares	123	Music
Connexions	19783	General
GEM	47321	General
LeMill	43734	General
LO.NET	302	General
LRE (European Schoolnet)	> 200K	General
Maricopa	1818	General
MERLOT	30398	General
Scriptorium	512	Humanities
Wisc-Online	2133	General

Most LORs store multidisciplinary content. Programming exercises are mostly enclosed in online judges. An example of an online judge is the UVa Online Judge (UVA OJ) — an automated judge for programming exercises created in 1995 with the aim of training users who participate in worldwide programming competitions. The judge is hosted by the University of Valladolid and its archive contains over 2700 exercises. The set of exercises is continuously being extended but it lacks interoperability features such as standardization of the exercises content as learning objects and implementation of communication specifications to improve accessibility for the educational community of teachers and students.

In regard to interoperability features for repositories the Jorum Team made a comprehensive survey [13] of the existing repositories and concluded that user expectations regarding standardization, content management and interoperability are not completely met by existing LORs. In fact, most of the LORs enumerated in the table above do not support content and communication standards.

Existing repositories usually store learning objects from several domains (referatories). They provide on-line catalogues through specific and tightly coupled Web-based interfaces. These interfaces provide tools for the management throughout the life cycle of learning objects, namely, submission, comment/review, browse/search and download. It has also been noticed that most of the existing repositories do not store actual learning objects. They just store meta-data describing LOs, including pointers to their locations on the Web, and sometimes these are dangling pointers. Although some 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 an LMS. A specialized system such as an evaluation engine (EE) to perform specific evaluations or an intelligent tutor system cannot query the repository and automatically import the LO it needs. In summary, current repositories provide specialized search engines for LOs and not adequate for feeding specialized services.

Although much useful work has been accomplished and considerable progress made, the existing software for creating LORs is still to a great extent confined to home-grown software. Table III presents a list of existing software for creating LORs with the licensing and supported standards.

Table III. Learning objects repositories software.

Repository	License	Metadata	Communication
ARIADNE	Free	DC,LOM,MLR	SPI,SWORD,PENS, SQI,SRU/SRW,OKI
Flori	Free	—	—
HarvestRoad Hive	Commercial	LOM/SCORM	OAI, OKI
IntraLibrary	Commercial	DC, LOM/SCORM	SWORD,SRU/SRW

SPI: Simple Publishing Interface; PENS: package exchange notification services; MLR: Metadata for Learning Resources.

DOI: 10.1002/spe

2.2. Interoperability

The corner stone of the interoperability of eLearning systems is the standard definition of learning objects. LOs are units of instructional content that can be used, and most of all reused, on Web-based eLearning systems. They usually encapsulate a collection of interdependent files (HTML files, images, web scripts, style sheets) with a manifest containing metadata. This metadata is important for classifying and searching LO in digital repositories. Standardized metadata plays an important role in keeping LO neutral to different repository vendors. Despite its success in the promotion of the standardization of eLearning content, current LO content specifications are quite generic and not adequate for specific domains [16]. However, these specifications were designed to be straightforward to extend, meeting the needs of a target user community through the creation of application profiles. When applied to metadata the term Application Profile generally refers to 'the adaptation, constraint, and/or augmentation of a metadata scheme to suit the needs of a particular community' [17]. The creation of application profiles is based on one or more of the following approaches:

- Selection of a core subset of elements and fields from the source schema;
- Addition of elements and/or fields (normally termed extensions) to the source schema, thus generating the derived schema;
- Substitution of a vocabulary with a new or extended vocabulary to reflect terms in common usage within the target community;
- Description of the semantics and common usage of the schema as they are to be applied across the community.

Beyond the standardization of content, the repositories need to interact with other systems that typically cohabit in the eLearning realm. Examples of these systems are authoring tools, learning management systems, harvesting systems, intelligent tutors, and evaluation engines. In fact, some surveys [11–13, 18] concluded that user expectations regarding standardization, content management and interoperability are not completely met by existing repositories.

In recent years, several organizations (e.g. Institute of Electrical and Electronics Engineers (IEEE) Learning Technology Standards Committee, IMS Global, OKI, International Federation for Learning, Education, and Training Systems Interoperability (LETSI), Advanced Distributed Learning (ADL), European Committee for Standardization (CEN)) have developed specifications and standards [16] to address these interoperability issues. For the sake of readability we detail only the most prominent [19] specifications organized into two facets: content and communication.

2.2.1. Content. A learning object is composed of one or more educational resources. These resources are described by metadata for discovery purposes and packaged for deployment and storage purposes.

One of the earliest international metadata standards is the DC. DC metadata is a set of vocabulary terms that can be used to describe generic resources for the purpose of discovery. It was developed in 1995 by a group of librarians and content experts. It was called 'Dublin Core' because it was created on a workshop in the city of Dublin (OH, USA). The Dublin Core Metadata Initiative is responsible for the development, standardization and promotion of the Dublin Core Metadata Elements Set, which includes two levels: Simple and Qualified. The Dublin Core Simple includes 15 elements (title, creator, subject, description, publisher, contributor, date, type, format, identifier, source, language, relation, coverage and rights). The Qualified includes three additional elements (audience, provenance and rights holder), and a group of element qualifiers that refine the semantics of the elements.

In 2002, the IEEE Standards Association published the IEEE 1484.12.1 – 2002 Standard for Learning Object Metadata as an open standard for the description of learning objects (LO) as units of instructional content that can be used, and most of all reused, on Web-based eLearning systems. Later, in 2005, IEEE published the official LOM XML binding, enabling syntactic interoperability. The purpose of LOM is to support the reusability of LOs, to aid discoverability, and to facilitate their interoperability, usually in the context of online learning management systems.

DOI: 10.1002/spe

Table IV. LOM data model categories.

Category	Description
General	Describe the learning object as a whole. This category includes elements such as identifier, title, language, keywords.
Lifecycle	Describe features related to the history and current state of the LO such as version, status, and contributors.
Metametadata	Group information about the metadata such as identifier, contributors and language used in the metadata.
Technical	Describe the technical requirements and characteristics of the LO such as MIME type, size, required software/hardware.
Educational	Describe educational and pedagogic characteristics of the LO such as interactivity type, learning resource type, interactivity level, semantic density, educational context, typical age range.
Rights	Describe the intellectual property rights and conditions of use for the LO (whether or not any cost is involved, and whether copyright and other restrictions apply).
Relation	Describe features that define the relationship between this LO and others ('based on', 'part of', etc.).
Annotation	Provide comments on the use of the LO and information on when and by whom the comments were created.
Classification	Describe where the LO can be classified within a particular classification system.

The binding data model is organized in nine categories. Table IV enumerates these categories based on the IEEE Learning Technology Standards Committee official Web site [20].

These categories cover many facets of a LO. However, LOM was designed for general LO and does not to meet the requirements of specialized domains. Fortunately, it was designed to be straightforward to extend. According to Al-Khalifa and Davis [21], an important feature of LOM is that it is simple to use and has an inherent extension capability. Next, we enumerate four ways that have been used [22] to extend the LOM data model:

- Combining the LOM elements with elements from other specifications;
- Defining extensions to LOM elements while preserving its set of categories;
- Simplifying LOM, reducing the number of LOM elements and its choices;
- Extending and simultaneously reducing the number of LOM elements.

A good example of a LOM-based metadata is the Canadian Core Metadata Application Profile. The Canadian Core standard is a LOM streamlined version that reduces the complexity and ambiguity of this specification.

Despite the wide use of both DC and IEEE LOM to describe learning resources, several semantic and interoperability issues are still not addressed. For instance, in DC the date element can be written in plain language and is ambiguous, it can be used for resource creation, update or publication time. In LOM, the cost element can only have a 'yes' or 'no' value [23]. To overcome these issues, the ISO/IEC 19788 MLR standard is intended to provide optimal compatibility with both DC and LOM and to specify metadata elements and their attributes for the description of learning resources [24].

Other standards for metadata such as METS, MODS, PREMIS and MIX are mostly related to digital libraries. The most prominent are the first two. The METS is an XML standard for describing metadata regarding objects within a digital library [25]. The standard is supported by the Network Development and MARC Standards Office of the Library of Congress. The MODS is an XML-based bibliographic description schema developed by the United States Library of Congress' Network Development and Standards Office. MODS was designed as a compromise between the complexity of the MARC format used by libraries and the simplicity of DC metadata [26].

Packaging of the learning resources complements content description and is crucial to facilitate the deployment, storage and reuse of learning resources. One of the earliest efforts was from the Aviation Industry Computer-Based Training Committee (AICC). The AICC association developed

DOI: 10.1002/spe

in 1998 a content package format called AICC HTTP-based AICC/CMI Protocol (HACP) consisting of four comma-separated American Standard Code for Information Interchange (ASCII) files that define details about the learning content including a universal resource locator (URL).

In 2000 IMS Global launched the IMS CP. An IMS CP learning object assembles resources and meta-data into a distribution medium, typically an archive in ZIP format, with its content described in a manifest file at the root level. The manifest file — named imsmanifest.xml — adheres to the IMS CP schema and contains the following sections: Metadata — describes the package as a whole; Organizations — describes the organization of the content within a manifest; Resources — contains references to resources (files) needed for the manifest and metadata describing these resources; and Submanifests — defines subpackages. The manifest uses the LOM standard to describe the learning resources included in the package. More recently (2008), the IMS Global Learning Consortium proposed the IMS Common Cartridge (CC) that adds support for several standards (e.g. IEEE LOM, IMS CP, IMS Question and Test Interoperability specification (QTI), IMS Authorization Web Service) and its main goal is to shape the future regarding the organization and distribution of digital learning content. The latest revised version [27] was released in May 2011.

The IMS CC manifest (Figure 3) includes references for two types of resources:

- Web Content Resources: static web resources that are supported on the Web such as HTML files, GIF/JPEG images, PDF documents, etc.
- Learning Application Objects (LAO): special resource types that require additional processing before they can be imported and represented within the target system. Physically, a LAO consists of a directory in the content package containing a descriptor file and optionally additional files used exclusively by that LAO. Examples of Learning Application Objects include QTI assessments, Discussion Forums, Web links, Basic LTI descriptors, etc.

Other well-known package format is the SCORM. SCORM was created by the ADL initiative with the first production version launched in 2001. It is an application profile for content packaging that extends the IMS CP specification with more sophisticated sequencing and Contents-to-LMS communication. It defines communications between client side content and a host system called the run-time environment, which is commonly supported by a learning management system.

The large number of package and metadata standards means that there may be several versions of a learning object (e.g. an English version and a French version) and available in different

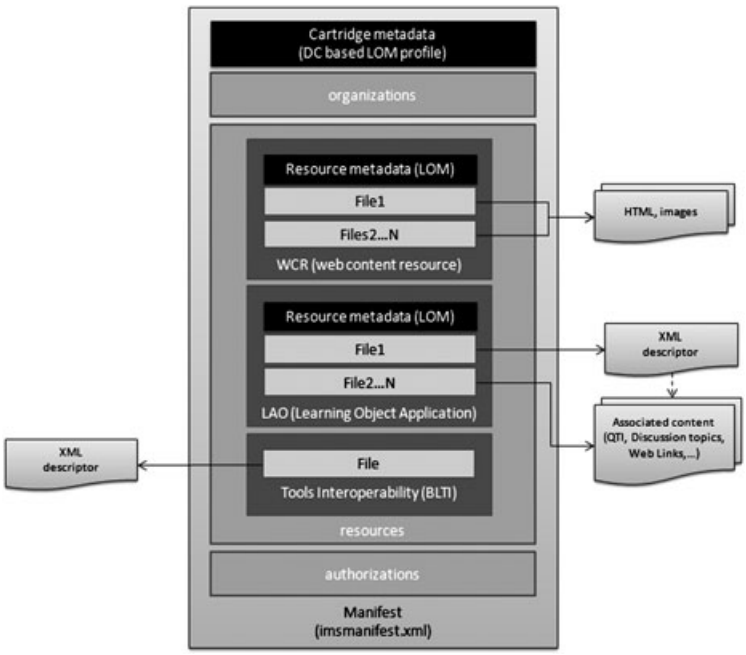

Figure 3. IMS Common Cartridge package.

DOI: 10.1002/spe

formats (e.g. as an IMS Common Cartridge or as an SCORM 2004 package) from several heterogeneous repositories. To overcome the potential data exchange issues, Massart *et al*. [28] created the Information for Learning Object eXchange framework. This framework organizes multiple metadata specifications in a container that can be handled as a whole. It was developed as part of the IMS Learning Object Discovery and Exchange (LODE) specification that aims to facilitate the discovery and retrieval of learning objects stored across more than one collection and over a federation of repositories.

2.2.2. Communication. The share and reuse of learning objects depends not only on the adoption of common formats to describe the content but also on standard mechanisms to publish data to repositories and to search and retrieve data from repositories. The basic functionalities provided by a repository boil down to **data push** (publication of data from a source into the repository) and **data pull** (searching/harvesting/gathering of data from the repository).

There are several protocols for **publishing** learning objects and/or their metadata to digital repositories. A learning object can be sent to a repository by value or by reference. In the former the publishing method embeds the learning object, after encoding, into the message that is sent to the repository. In the latter, the repository embeds a reference (e.g. URL) to the learning object that is being published.

The IMS DRI specification was created by the IMS Global Learning Consortium and provides a functional architecture and reference model for repository interoperability. The IMS DRI recommends common repository functions. One of these functions is the submit function for submitting LO to a DRI compliant repository through the transmission of an IMS-compliant content package using Simple Object Access Protocol (SOAP) Messages with attachments.

The Package Exchange Notification Services (PENS) protocol, developed by the AICC in 2005, supports a notification service for content packages. Using this service a source can announce the location of a package that is available for transport. When an application (e.g. LMS) receives a PENS notification, it can retrieve the package from the URL that is provided. The PENS specification contains an abstract data model and provides a binding to the Hypertext Transfer Protocol (HTTP).

The OKI created OSIDs to enable the submission of assets (learning object and metadata) to a digital repository. The repository OSID includes a JAVA Asset interface that offers methods for adding and deleting records.

The SRU [29] Record Update service supports the creation, replacement and deletion of metadata records. This specification can be implemented only on metadata resources.

The SWORD (Simple Web-service Offering Repository Deposit) standard allows digital repositories to accept the deposit of any content from multiple sources. SWORD is a profile of the Atom Publishing Protocol (AtomPub — a simple HTTP-based protocol for creating and updating Web resources) restricting the scope of depositing resources into scholarly systems.

The Simple Publishing Interface (SPI) specification [30], partly sponsored by the CEN Workshop on Learning Technologies, defines a protocol that aims to facilitate the communication between content producing tools and repositories that persistently manage learning resources and metadata. The SPI is an abstract model for publishing metadata and resources. A binding to a technology makes these methods more concrete and defines how applications can interoperate. SPI has been bound to the AtomPub, compatible with the SWORD profile, which is widely used by institutional repositories.

Learning objects are described by metadata stored in repositories. The latter should support different search/harvesting protocols to expose metadata to users and/or services. One of the earliest search protocols was the Z39.50. It is a client–server protocol for searching and retrieving information from remote libraries. Z39.50 is a pre-Web technology (work on the Z39.50 protocol began in the 1970s). Since then there have been several efforts to evolve the protocol under the designation ZING (Z39.50 International: Next Generation). One of the most important is the twin protocols SRU/SRW, which introduce a new communication protocol (HTTP) making the specification more lightweight. SRU is REST based and enables queries to be expressed in URL query strings; SRW service uses SOAP. Both expect search results to be returned as XML. Queries in SRU and SRW

DOI: 10.1002/spe

are expressed using the Contextual Query Language as a new query language that was based on the semantics of Z39.50. All these standards (SRW, SRU and Contextual Query Language) were promulgated by the United States Library of Congress.

The IMS DRI also provides a recommendation for a search function. The Search reference model defines the searching of the metadata associated with content exposed by repositories. It suggests two query languages: XQuery for searching IMS (XML) metadata format and Z39.50 for searching library information.

The IMS LODE specification aims to facilitate the discovery and retrieval of learning objects stored across more than one collection. LODE is based on the following assumptions: (i) LOs are described by metadata such as LOM or DC; (ii) multiple metadata instances might be necessary to adequately describe all the aspect of a LO and to create searchable catalogues of LO using the Information for Learning Object eXchange framework; (iii) repositories can be searched programmatically using SQI or SRU; (iv) large catalogues can be created by harvesting (i.e. mirroring) metadata stored in repositories using protocols such as OAI Protocol for Metadata Harvesting (PMH).

The SQI specification, supported by CEN, presents an API for querying learning object repositories. SQI is neutral in terms of results format and query languages, thus it makes no assumptions about the query language or results format [31].

OpenSearch — created by A9.com (an Amazon.com company) — is a collection of simple formats for the sharing of search results. The OpenSearch description document format can be used to describe a search engine so that it can be used by search client applications. The OpenSearch response elements can be used to extend existing syndication formats, such as Really Simple Syndication (RSS) and Atom, with the extra metadata needed to return search results.

The ProLearn Query Language (PLQL), developed by the PROLEARN 'Network of Excellence', is a query language for repositories of learning objects. PLQL is primarily a query interchange format, used by source applications (or PLQL clients) for querying repositories (or PLQL servers). PLQL has been designed with the goal of effectively supporting search over LOM, DC and MPEG-7 metadata. However, PLQL does not assume or require these metadata standards.

Harvesting protocols enable pulling learning objects and metadata from a repository. An example of a metadata harvesting protocol is the OAI-PMH. Large catalogues can be created by harvesting (i.e. mirroring) metadata stored in repositories using OAI-PMH. To obtain content, the OAI-PMH can be used in combination with a protocol for obtaining content, such as the National Information Standards Organization's (NISO's) OpenURL.

3. crimsonHex

In this section, we introduce the crimsonHex repository, its architecture, data model and main components. We also present its API used both internally and externally. Internally the API links the main components of the repository. Externally the API exposes the functions of the repository to third party systems. To promote the integration with other eLearning systems, the API of the repository adheres to the IMS DRI specification. The IMS DRI specifies a set of core functions and an XML binding for these functions. In the definition of API of crimsonHex we needed to create new functions and to extend the XML binding with a Response Specification language. The complete set of functions of the API and the extension to the XML binding are also both detailed in this section.

3.1. Architecture

The architecture of the crimsonHex repository can be summarized by the UML components diagram in Figure 4.

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.

DOI: 10.1002/spe

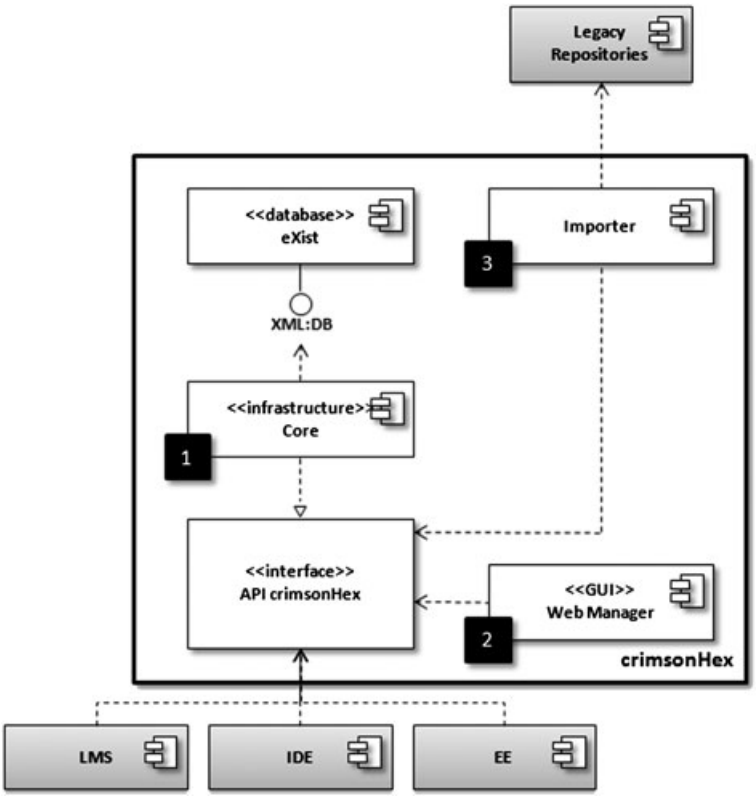

Figure 4. Components diagram of the repository.

Using the **crimsonHex API**, the repository exposes a set of functions implemented by a core component that was designed for efficiency and reliability. 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 also using this API.

Thus, the architecture of crimsonHex repository is divided into three main components:

(1) 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;
(2) The Web Manager allows the searching, previewing, uploading and downloading of LOs and related usage data;
(3) The Importer populates the repository with content from existing legacy repositories, while converting it to LOs.

In the remainder we focus on the Core component, more precisely, its functions, communication model and implementation.

3.2. Data model

An LO containing a programming problem must include metadata to allow its use by different types of specialized eLearning services, such as evaluation engines and programming problem repositories, among others. The existing LO standards are insufficient for that purpose, which led us to the development of a new application profile based on existing standards and guidelines. This section details the definition of programming problems as LO by extending the LOM metadata schema with new elements to support programming problems and their automatic evaluation.

The new metadata information is included in an IMS CP manifest file that usually follows the IEEE LOM schema, although other schemata can be used. In this definition, the metadata that cannot be conveniently represented using LOM is encoded in elements of a new schema — EduJudge

DOI: 10.1002/spe

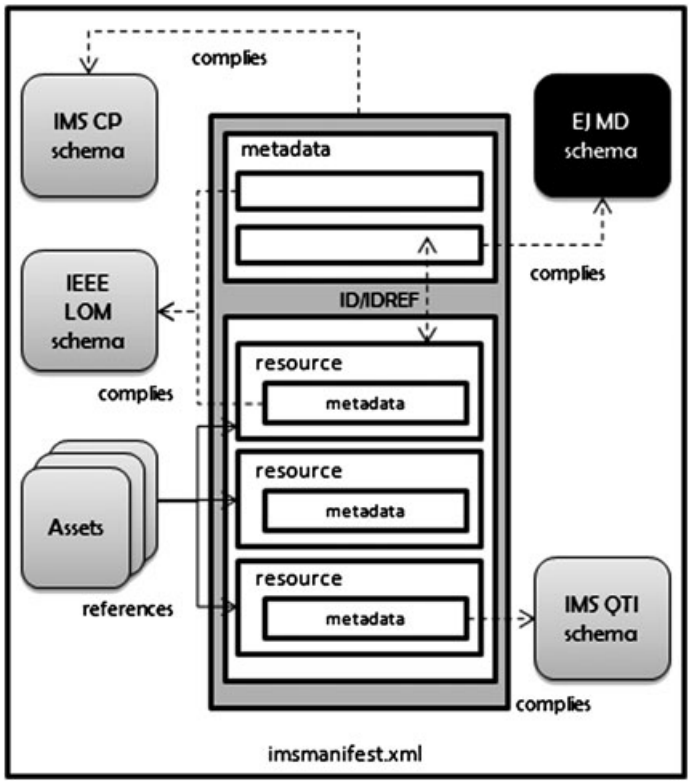

Figure 5. Structure of a programming exercise as an LO.

Meta-Data (EJ MD) — and included *only* in the metadata section of the IMS CP. This section is the proper place to describe relationships among resources, as those needed for automatic evaluation. The compound schema can be viewed as a new application profile that combines metadata elements selected from several schemata. The structure of the archive, acting as distribution medium and containing the programming problem as an LO, is depicted in Figure 5.

The archive contains several files represented in the diagram as gray rectangles. The manifest is an XML file and each element's 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; 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 are represented by solid arrows.

The resources section of the IMS CP provides a suite of resource elements where each one is composed of several files. To link the EJ MD domain metadata, it is necessary to create a reference mechanism to link it with the related resources. This mechanism takes the ID/IDREF types of the XML Schema specification to link the EJ MD metadata element with the identifier attribute of the resource element.

The core of the proposed application profile is the EduJudge schema that introduces new elements for resources specific to programming problems. This section presents its data model, represented schematically in Figure 6.

The domain metadata is a hierarchy of elements whose leaves are resources.

The basic Resource type is an asset in the distribution medium, referenced by a relative filename.

The ProgramResource is a specialized type of resource that refers to a source code program file. This type of resource requires as attributes all the information to compile and execute the program, including the language name and version, and compilation and execution command lines.

DOI: 10.1002/spe

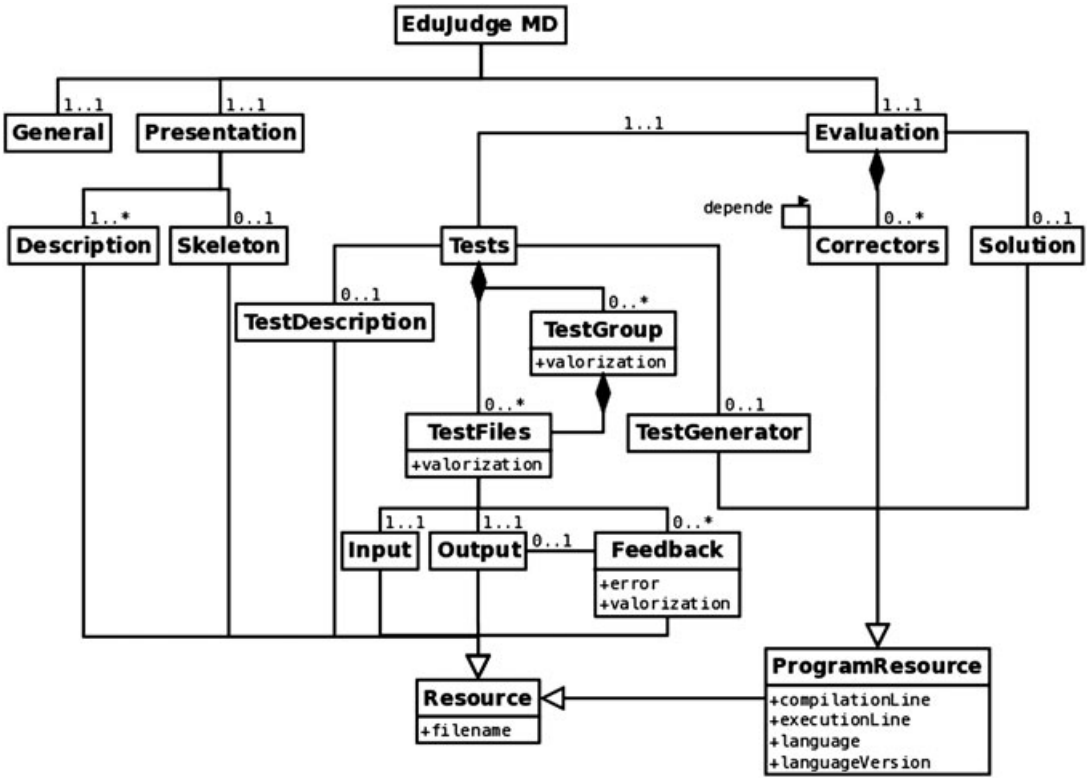

Figure 6. The EduJudge Metadata model.

The metadata type hierarchy has three main categories in the first level: the General category describes generic metadata and recommendations; the Presentation category describes metadata on resources that are presented to the learner (e.g. description and skeleton resources); the Evaluation category describes the metadata for resources used to evaluate the learner's attempts and provide feedback.

The elements of the Evaluation type define all the resources needed to judge a programming problem. It has attributes to identify the problem's evaluation module and its version and three elements pointing to different types of evaluation resources: tests, correctors and solutions.

The elements of type Tests describe resources supplied to evaluate the submitted program.

This definition supports several testing methodologies, each with a specific element type, including among others:

- TestFiles contains a pair of input and output files;
- TestGroup contains a collection of test files and associated valorization;
- TestDescription identifies tests encoded in a language describing test cases;
- TestGeneration identifies a program that generates input files for test cases.

The TestFiles element supports the simplest type of evaluation and is expected to be the most commonly used. This element must contain references to input and output files, and may have a valorization and feedback. An element of this type corresponds to a single test case, thus it can be repeated to create a comprehensive set of tests. In this case the learner's program is executed once for each TestFile element, receiving as input the content of the file referenced by the corresponding element, and/or from the arguments attribute. The resulting output is compared with the expected output contained in the TestFile element.

The TestFiles element can also be used for grading and correcting programs. This element may include a *valorization* attribute, in which case the grade of the program is the sum of the valorizations of successful executions.

DOI: 10.1002/spe

Corrections to the program may be indicated using the optional Feedback element. These elements provide, for each test case, a feedback message associated with a particular error condition (e.g. 'Wrong Answer', 'Time Limit Exceed', 'Execution Error') or invalid output. The *showAfterNumberAttempts* attribute controls when the feedback message should be sent to the learner based on the actual number of attempts. The *valorization* attribute of the feedback element enables partial grading for predefined errors.

The TestGroup element is a container of TestFile elements and is used to create different test sets, with an optional valorization for the complete set.

The TestDescription element refers to a file describing test cases. This file is meant as input for a test case generation tool. The test description is an asset of the LO but the test generation tool must be available to the evaluation engine.

Alternatively, the TestGenerator element refers to a program that when executed generates tests for this particular programming exercise.

The Correctors element is optional and refers to custom programs that change the general evaluation pattern for a given problem. There are two types of correctors: Static — invoked immediately after compilation, before any execution, to compute software metrics on the source code, judging the quality of source code, performing unit testing on the program or checking the structure of the program's source code; and Dynamic — invoked after each execution with a test case to deal with nondeterminism (e.g. the solution is a set of unordered values, in this case the corrector normalizes the outputs before comparing them). A single programming problem may use an arbitrary number of correctors. The order in which they are executed is defined by the *depends* attribute.

Finally, optional elements of type Solution refer to files containing the problem solution.

The following code excerpt shows an example of the `evaluation` element.

```
<ejmd:evaluation
       ejmd:evaluationModel="ICPC"
       ejmd:evaluationModelVersion="1">
  <ejmd:tests>
    <ejmd:testFiles ejmd:arguments="" ejmd:valorization="1">
      <ejmd:input ejmd:resource="INPUT-00"/>
      <ejmd:output ejmd:resource="OUTPUT-00"/>
    </ejmd:testFiles>
    <ejmd:testFiles ejmd:arguments="" ejmd:valorization="1">
      <ejmd:input ejmd:resource="INPUT-01"/>
      <ejmd:output ejmd:resource="OUTPUT-01"/>
    </ejmd:testFiles>
  </ejmd:tests>
  <ejmd:correctors>
    <ejmd:corrector
       ejmd:resource="CORRECTOR"
       ejmd:compilationLine="gcc -lm -lcrypt -O2 -pipe -DONLINE_JUDGE -
       lstdc++ -w p11524.judge.cpp -o p11524.judge"
       ejmd:executionLine="./p11524.judge"
       ejmd:language="C++"
       ejmd:languageVersion="4.2.2"/>
  </ejmd:correctors>
  <ejmd:solution
       ejmd:resource="SOLUTION"
       ejmd:compilationLine="gcc -lm -lcrypt -O2 -pipe -DONLINE_JUDGE -
       lstdc++ -w p11524.cpp -o p11524"
       ejmd:executionLine="./p11524"
       ejmd:language="C++"
       ejmd:languageVersion="4.3.2"/>
</ejmd:evaluation>
```

3.3. 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.

DOI: 10.1002/spe

The Register/Reserve function requests a unique ID from the repository. We separated this function from Submit/Store to allow the inclusion of the ID in the metadata of the LO itself. This ID is an URL that must be used for submitting an 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 an 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 conformance 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. 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 Report/Store function associates a usage report with an existing LO. This function is invoked by the LMS to submit a final report, summarizing the use of an 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 an RSS feed. With this option a user can have up-to-date information through a feed reader.

3.4. Communication model

The communication model of the repository defines the interaction between the repository and the other eLearning systems. The model is composed from a set of core functions exposed in the previous section. Figure 7 shows an UML diagram to illustrate the sequence of core functions invocations from these eLearning systems to repositories.

The life cycle of a LO starts with choosing an identification and the submission of the 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 at the problem solution to the EE. On the basis of the feedback the learner could repeat the process. At 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.

4. IMPLEMENTATION

In this section we detail the design and implementation of the Core component of crimsonHex on the Tomcat servlet container. The following subsections detail the development of the four main facets of the Core: storage, validation, interface and security.

4.1. Storage

Searching LOs in the repository is based on queries on their XML manifests. Because manifests are XML documents with complex schemata we paid particular attention to database 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 internally store data in XML format hence they need extra processing to support querying using XQuery. Because queries in this standard are a DRI recommendation, this type of storage is not the best option. In contrast, NXD uses the XML document as a fundamental unit of (logical)

DOI: 10.1002/spe

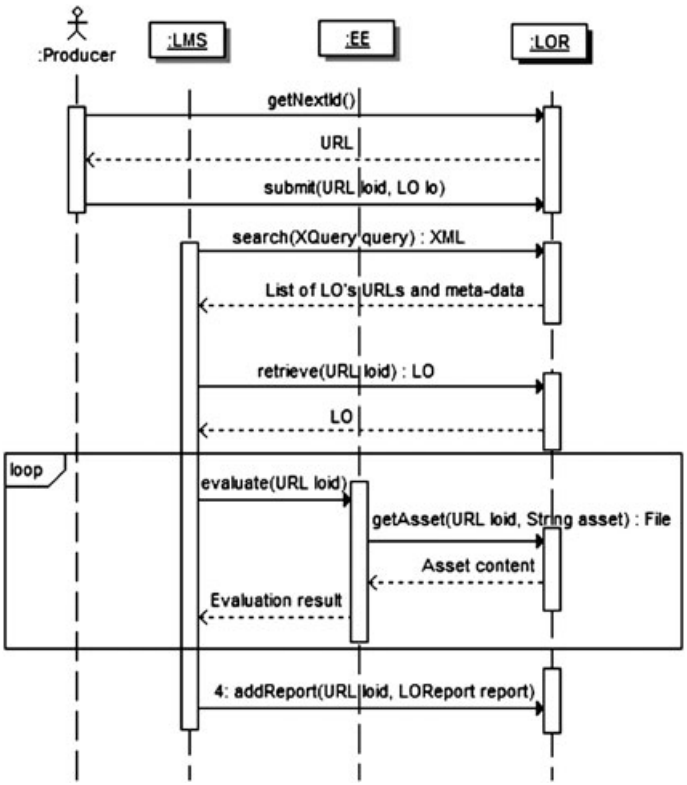

Figure 7. Communication between the repository and the other eLearning systems.

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 [32] worth mentioning: support for collections, to structure the database in groups of related documents and automatic indexes to speed up the database access.

4.2. Validation

CrimsonHex is a repository of specialized learning objects. To support this multityped content the repository must have a flexible LO metadata 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: LOs 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 its structure and content. This standard also requires XSD validation of their manifests. For particular domains it is possible to configure specialized validations in crimsonHex by supplying a

DOI: 10.1002/spe

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 for each 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 bad 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 h. Outdated schemata are reloaded from their original Internet location using a conditional HTTP request that downloads it only if it has effectively changed.

4.3. 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 wWeb services: SOAP and REST [33]. SOAP Web services are usually action oriented, mainly when used in remote procedure call mode and implemented by an off-the-shelf SOAP engine such as Axis. 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 V. Each function is associated with the corresponding operations in both SOAP and REST Web services interfaces. The lines formatted in italics correspond to the new functions added to the DRI specification, to improve the repository communication with other eLearning systems.

To describe the responses generated by the repository we defined a Response Specification as a new XML document type formalized in XML Schema (XML Schema, 2004). The advantage of this approach is that client systems can obtain more information from the server and be able to standardize the parsing and validation of the HTTP responses. Figure 8 depicts the elements of the new language and their types.

Table V. 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
Create	XML Create(URL collection)	PUT URL
Remove	XML Remove(URL collection)	DELETE URL
Status	XML getStatus()	GET URL?status > XML

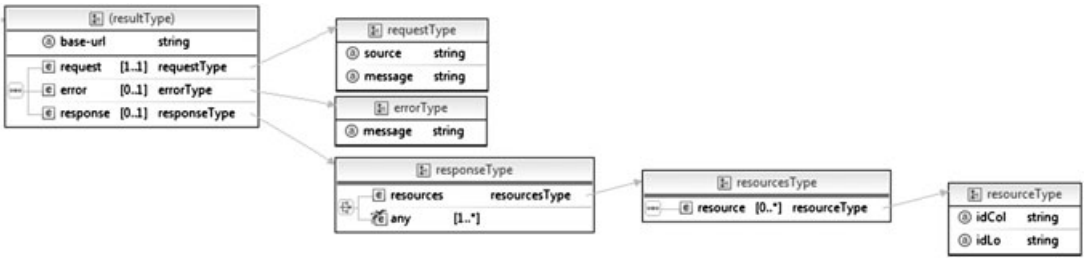

Figure 8. Response specification schema.

DOI: 10.1002/spe

The schema defines two top level elements: `result` and `rss`. The former will be used by all the functions except the Alert function that returns a feed compliant with the RSS (2003) 2.0 specification. The `result` element contains the following child components:

- `base-url` attribute, defining a base URL for the relative URLs in the response;
- `request` element, containing the full request URL and an human readable request message;
- `error` element, containing an error message — client systems will search for this element to verify the existence of errors;

`response` element, describing a successful execution of the function — it is composed of a human readable response message and, for some functions, of a `resources` element that groups a set of resources defined individually in `resource` elements. A `resource` element contains an identification of the collection absolute path (attribute `idCol`) and an identification of the LO itself (attribute `idLo`).

In the remainder of this section we detail the Core functions of the repository.

The Register/Reserve function requests a unique ID from the repository. We separated this function from Submit/Store 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 or retrieving an LO. The producer may use this URL as an ID with the guarantee of its uniqueness and with the advantage of being a network location from where the LO can be downloaded.

The Submit/Store function uploads an LO to a repository and makes it available for future access. This operation receives an argument providing an IMS CP compliant file and an URL generated by the Reserve function. This operation validates the LO conformity to the IMS Package Conformance and stores the LO in the internal database.

The Search/Expose function enables the eLearning systems to query the repository using the XQuery language (XQuery, 2007), 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 LO manifest and its 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 subfolders. From the XQuery point of view the database is a collection of manifest files.

The Report/Store function associates a usage report with an existing LO. This function is invoked by the LMS to submit a final report, summarizing the use of an 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 these data, the LMS will be able to dynamically generate presentation orders based on previous uses of LO, instead of 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 an RSS feed. With this option a user can have up-to-date information through a feed reader.

The Create function adds new collections to the repository and the Remove function removes an existent collection or learning object.

The Status function returns a general status of the repository, including versions of the components, their capabilities and statistics.

4.4. 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 because we can control these features in the communication layer. Because both Web services flavours use HTTP as transport protocol we secure the channel using Secure Sockets Layer (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 Secure Sockets Layer. In practice, to implement this approach, we just needed to configure the servlet container

DOI: 10.1002/spe

(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. These functions are used for managing and signing client certificates and their implementation is based on the Java Security APIs.

4.5. Web manager

In this section, we present the crimsonHex user interface. We start by presenting our strategy to design the user interface, followed by detailed descriptions of its main tasks, namely browsing, authoring and searching.

To design this user interface (Figure 9) we started with the identification of task and usage profiles, task objects and task actions. We identified the following task profiles:

1. *Archivist* - a person responsible for a set of activities related with the collection management, such as: creation of collections, assigning of learners and reviewers to collections;
2. *Author* - a person that develops and submits LO to the repository. The submission of LO will be enforced to comply with controlled vocabularies defined in metadata standards (IEEE LOM) and possible extensions. This class of users will contribute with new learning objects and receive peer reviews from specialists;
3. *Reviewer* - a person that controls the quality of the repository by validating the submitted LO;
4. *Consumer* - a person that browses (part) of the repository and has limited access to its content (LO, usage reports, reviews, comments).

We assume that users will have different usage profiles. On one hand, many will be novice or first-time users, especially among authors and consumers. On the other hand, we expect some users, especially reviewers and archivists, to use it frequently, tending to become experts in its use.

After the identification of users and usage profiles, we proceeded to identify the tasks they need to perform on this interface. We clearly identified LO and collections of LO as our task objects, each

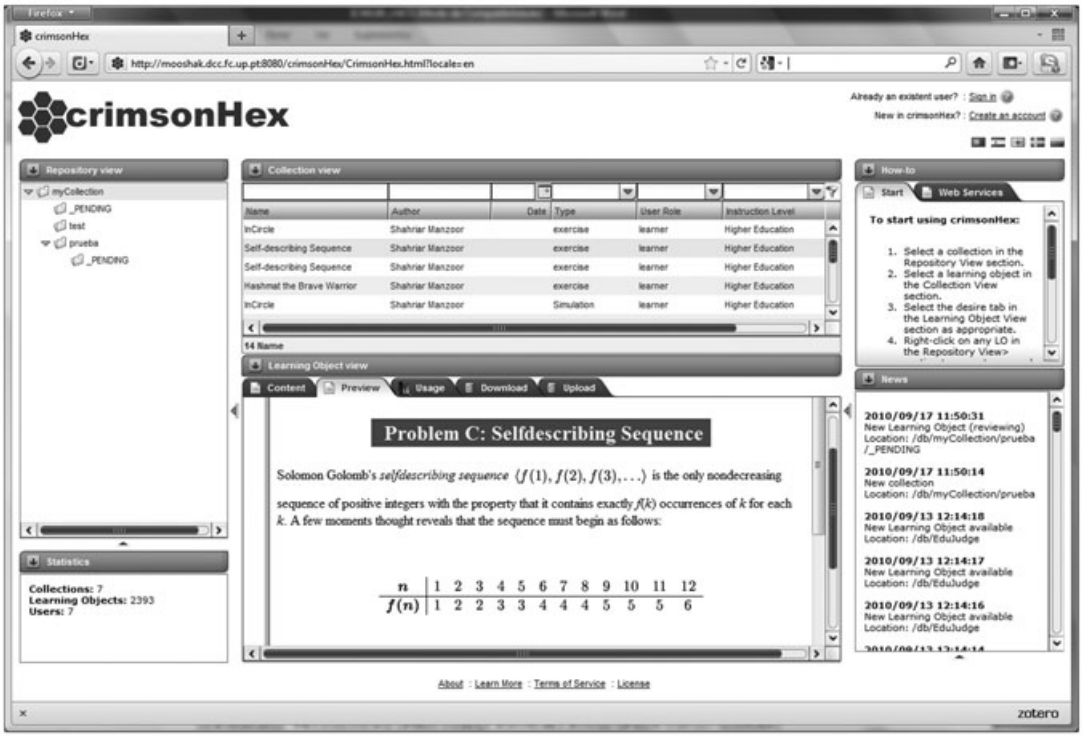

Figure 9. crimsonHex WebManager.

DOI: 10.1002/spe

with a number of associated task actions, depending on user profiles. Task actions over LO include: viewing, reviewing, downloading, and commenting. Task actions on collections include creating, removing and authoring/uploading (LO to that collection).

On the basis of the previous identifications we defined a screen layout — a single screen (Figure 9) with specific areas for task object selection and task actions. Task object selection is needed by all users, although the selectable content depends on the user's profile, thus it can be implemented by a common tree-based control. Different task actions require specific forms or panels that also share a common control on the user interface. Because the number of task actions is comparatively small we choose a tabbed control to aggregate them. The tab configuration shown to users depend both on their profile and on the current task object selection.

As a rule, all available task actions have an associated tab, thus helping novice users to recognise which are the available actions. However, some of these task actions can be executed directly over selected task objects, without requiring additional data. In general, these task actions are meant for frequent users and will be bound to contextual menus on the tree-control, and to accelerator keys.

Figure 9 shows the user interface layout of the repository with two main areas: selection on the left side and action in the middle. In the selection area the user navigates through the repository structure to select task objects. In the action area the user executes task actions on the selected task objects. Secondary areas in this layout are the header, used for authentication and registration, and the right side, used for news and statistics. The remainder of this section details the design of task actions available in the main areas.

Selected LO can be viewed in different perspectives, such as, Content, Usage and Review. As would be expected, each perspective is assigned to a different tab on the action area.

The Content tab shows the resources and metadata of the selected LO using XSLT transformations on its manifest file. Different stylesheets can be selected to configure content presentation, including:

- *Resources* — lists resources and their metadata with support for viewing/downloading individual resources;
- *Meta-data* — shows all global metadata, including LOM and extensions;
- *LOM* — show LOM metadata grouped in main categories (General, Lifecycle, Technical, Educational, Rights);
- *Extension* — shows metadata related with the extension schemata.

The Usage tab presents statistics on the use of a selected LO. This feature is related to the report function that associates usage reports to an existing LO. This function is invoked by a consumer of the repository services (typically an LMS), summarising an episode of using a LO with a particular student. The aim of this function is to provide the LMS with the ability to dynamically generate presentation orders based on previous uses of LO, instead of using fixed presentation orders. 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 Review tab assists in the review process of an LO. Before validation the LO is not available to general users of the repository. The availability of the LO depends on this validation. If the LO is not accepted the reviewer could justify the rejection and/or supply comments to the author of the LO. These comments may lead to new versions that must be submitted as a new LO.

The Web Manager component was developed using an Ajax framework to enable the implementation of the single screen design resulting from the last section. We selected the Google Web Toolkit (GWT), an open source Java software development framework that allows a rapid development of AJAX applications in Java. When the application is deployed, the GWT cross-compiler translates Java classes of the GUI to JavaScript files and guarantees cross-browser portability. The framework also supports asynchronous remote procedure calls. This way, tasks that require significant computational resources (e.g. complex searching within the repository) can be triggered asynchronously, increasing the user interface's responsiveness. The complex controls required by the selection and action areas are provided by SmartGWT, a GWT API for SmartClient, a Rich Internet Application system.

DOI: 10.1002/spe

Table VI. 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

The Web Manager component is organised into two main packages: the back-end (server) and the front-end (client). The back-end includes all the service implementations triggered by the user interface. These implementations rely on the gateway class for managing the communication with the Core of the repository. A single class implementing the Gateway design pattern concentrates the interaction with the core component. To interact with other DRI compliant repositories only this class will have to be reimplemented.

4.6. Tests

Reliability is one of our main concerns regarding the Core component of crimsonHex. We adopted JUnit as our automated unit testing framework because crimsonHex is implemented in Java and this tool is supported by Eclipse, the integrated development environment 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 twofold: 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 them 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. For the experiment we use the same PC for client and server purposes. The PC was an Asus M70VSeries (ASUSTeK Computer Inc., Beitou District, Taipei, Taiwan) with Windows Vista Home Premium (32 bits) (Microsoft Corporation, Albuquerque, New Mexico, United States), Intel(R) Core(TM)2 Duo P8400 @2.26 GHz and 4 GB RAM. We used 100 LOs on the experiment ranging in storage size from 2 MB to 5 MB. Each function has been repeated 10 times. Average function execution times for the set of functions are shown in Table VI.

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

Scalability has 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 arbitrarily large depending on relevant file system limits, for example, the max size of a file in the file system. To test the scalability of eXist, some queries were made with ever increasing data volumes. The experiment shows linear scalability of eXist's indexing, storage and querying architecture.

5. CASE STUDY

To evaluate the interoperability features of the crimsonHex repository we integrated it with Moodle, arguably the most popular LMS nowadays. In this section we present the new APIs supported by Moodle (since version 2.0) for the creation of plug-ins and we provide implementation details of a plug-in for crimsonHex repositories. The development of this plug-in was straightforward. In terms of programming effort we spent half a day to produce approximately 100 new lines of code. This quick and simple integration benefited from the new interoperability features of the repository.

DOI: 10.1002/spe

Currently, Moodle 2.2 includes support for different types of repositories. Several APIs are available to enable the development of plugins by third parties, including:

- File API for managing internal repositories;
- Repository API for browsing and retrieving files from external repositories;
- Portfolio API for exporting Moodle content to external repositories.

We chose the Repository API for testing the integration features of the crimsonHex repository in Moodle. The goal of this particular API is to support the development of plug-ins to import content from external repositories. The Repository API is organized in two parts: Administration, for administrators to configure their repositories and File picker, for teachers to interact with the available repositories.

To create a plug-in for Moodle using the Repository API, one must implement a set of related files. For instance, the steps to create the crimsonHex plug-in for Moodle are the following:

1. To create a folder for the plug-in (*moodle/repository/crimsonHex*);
2. To add to the plug-in folder the files
   a. *repository.class.php* — subclassing a standard API class and overriding its default methods;
   b. *icon.png* — providing the icon displayed in the file picker;
3. To create the language file *repository_crimsonHex.php* and add it to the folder *moodle/repository/lang/en_utf8/.*

The *repository.class.php* is responsible for handling the communication between Moodle and all repository servers of that type. In this case the repository type is crimsonHex but other types are being developed for other types of repository, such as Merlot, YouTube, Flickr and DSpace. For Moodle, each repository is just a hierarchy of nodes. This allows Moodle to construct a standard browser interface. The repository server must provide: a uniform resource identifier (URI) to download each node (e.g. a LO) and a list of the nodes (e.g. LO and collections) under a given node (e.g. collection). In addition to these requirements, a repository can optionally support authentication, provide additional metadata for each node (mime type, size, dates, related files, etc.), describe a search facility or even provide copyright and usage rules.

As explained before, the Repository API has two parts — Administration and File Picker — each with its own GUI. In Figure 10 we show the file picker GUI of the crimsonHex plug-in that will be used by the teacher to pick up the suitable exercises for the class. In the left panel are listed the available repositories as defined by the administrator. Two crimsonHex repository instances are marked

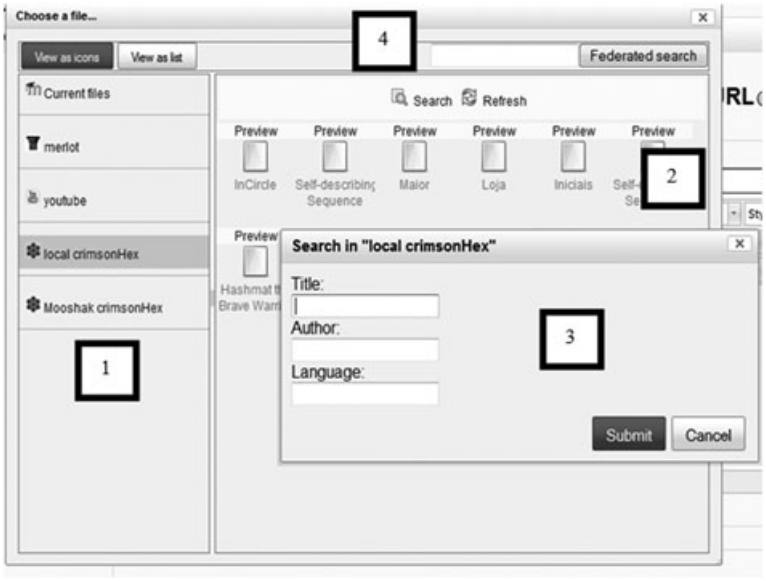

Figure 10. crimsonHex plugin interface.

DOI: 10.1002/spe

Table VII. Features of crimsonHex and Moodle Repository APIs for implementing File Picker.

Moodle Repository API	crimsonHex API
get_file	retrieve
Preview	retrieve
Search	search
global_search	search

with label 1. Label 2 marks the default listing of the selected repository. Pressing the 'Preview' link presents a preview of the respective LO. Pressing the 'Search' link pops-up a simple search form, marked as 3 in Figure 10. Federated search in all available crimsonHex repositories uses the text box marked as 4. In Table VII we indicate the features in Moodle's Repository API and in crimsonHex's API, used to implement the features we marked on the file picker's GUI.

Not all features of crimsonHex API were needed for this plug-in but all features of the Moodle's API were covered. The remaining features of the crimsonHex API are useful for uploading LOs and managing the structure of the repository and thus are out of the scope of this plug-in. They would be matching the needs of a plug-in implementing the Moodle's Portfolio API. Each feature of the plug-in is implemented by a method in the *repository.class.php* file.

A typical method includes: a repository invocation (SOAP or REST), the parsing of its response (using the PHP *simplexml_load_string* function to parse the XML data), a selection of the pertinent data (using XPath) and an iteration over the new results (for instance, populating an array with the relevant data). The next example shows an excerpt of the overridden search function.

```
private function _search($queryString) {
   $list  = array();
   $c = new curl();
   $content=$c->get($this->options['url'] . $queryString);
   $xml = simplexml_load_string($content);
   $result = $xml->xpath("//resource");
   foreach ($result as $entry) {
       $attr = $entry->attributes();
      $list[] = array(
       'title'=>(string)$entry,
       'thumbnail'=>$OUTPUT->icon_url(path),
       'date'=>'',
       'size'=>'',
      'source'=>$attr['url'].$attr['idCol']
      .$attr['idLo']);
   } return $list; }
```

6. CONCLUSIONS AND FUTURE WORK

In this paper, we describe the design and implementation of a repository of specialized learning objects called crimsonHex focusing on two main parts: the definition of specialised LOs, where programming problems are given as a concrete example, and the design and implementation details of the repository, more precisely, its components and functions.

For the first part we detailed the steps to define LOs from a domain that is not covered by existing standards in a way that can be reproduced in similar contexts. The main contribution of this part was 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.

For the second part we described the design and implementation of a repository of specialized LOs. We adopted the IMS DRI and proposed extensions to its recommendations. More precisely, we included new functions and a formal definition of a response specification for the complete

DOI: 10.1002/spe

function set. To evaluate the proposed extensions, we implemented a plug-in for the 2.1 release of Moodle that uses the new interoperability features of crimsonHex.

The improved interoperability of crimsonHex is expected to support the development of new eLearning tools requiring greater integration with repositories. The repository plug-in will facilitate the use of crimsonHex by Moodle users. In its current status crimsonHex is available for test and download at the site of the project at `http://crimshonhex.dcc.fc.up.pt`.

Adding authoring features to crimsonHex is the next step in this research. Creating LOs with metadata of good quality is a challenge because the typical author of eLearning content usually lacks the knowledge of metadata standards. This is also an interoperability issue because the LMS is where eLearning content is tested and used in the first place, but repositories are the appropriate place to promote content reuse as LOs. We plan to continue using Moodle's repository APIs for that purpose, in particular the Portfolio API. A plug-in using this API will enable the content author to upload learning content to crimsonHex and create a new LO with the essential metadata. Then, using the authoring features of crimsonHex, the content author will be assisted in refining the LO metadata.

Other future work is to support the new package specification from IMS called Common Cartridge. This new specification defines a new model for packaging content with LOM metadata (based on simple Dublin Core) and assessment (QTI). Other features in IMS-CC relevant for this line of work are discussion topics, authorization for protected content and the support for Basic LTI, a subset of the full LTI specification.

REFERENCES

1. Nash SS. Learning objects, learning object repositories, and learning theory: Preliminary best practices for online courses. *Interdisciplinary Journal of Knowledge and Learning Objects* 2005; **1**:217–228. Available from: http://ijklo.org/VolumeI/v1p217228Nash.pdf [21 July 2011].
2. Dagger D, O'Connor A, Lawless S, Walsh E, Wade V. Service Oriented eLearning Platforms: From Monolithic Systems to Flexible Services. *IEEE Internet Computing Special Issue on Distance Learning* 2007; **11**(3):28–35.
3. IMS DRI - IMS Digital Repositories Interoperability, Core Functions Information Model, v. 1.0, 2003. Available from: http://www.imsglobal.org/digitalrepositories/driv1p0/imsdri_infov1p0.html [21 July 2011].
4. Verdú E, Regueras LM, Verdú MJ, Leal JP, Castro JP, Queirós R. A Distributed System for Learning Programming On-line. *Journal Computers & Education* 2012; **58**(1):1–10.
5. Leal JP, Queirós R. CrimsonHex: a service oriented repository of specialised learning objects. In *Proceedings of ICEIS'09: 11th International Conference on Enterprise Information Systems*, Vol. 24, Filipe J, Cordeiro J (eds). Springer: Milan, Italy, May 2009; 102–113. ISBN: 978-3-642-01346-1. DOI: 10.1007/978-3-642-01347-8_9.
6. Leal JP, Queirós R. Integration of repositories in elearning systems. In *ICEIS 10 - 12th International Conference on Enterprise Information Systems*, Madeira (Portugal), 2010.
7. McGreal R. A typology of learning object repositories. In *Handbook on Information Technologies for Education and Training, International Handbooks on Information Systems*, Adelsberger HH, Kinshuk, Pawlowski JM, Sampson DG (eds). Springer: Berlin Heidelberg, 2008; 5–28.
8. Rogers SA. Developing an institutional knowledge bank at Ohio State University: From concept to action plan. *Portal: Libraries and the Academy* 2003; **3**(1):125–136.
9. Ochoa X, Duval E. Quantitative Analysis of Learning Object Repositories. *IEEE Transactions on Learning Technologies* Sept 2009 July; **2**(3):226–238. DOI: 10.1109/TLT.2009.28.
10. Ternier S. Standards based Interoperability for Searching in and Publishing to Learning Object Repositories. *PhD Thesis*, Katholieke Universiteit Leuven, 2008.
11. Repository software survey, in Repositories Support Project, November, 2010. Available from: http://www.rsp.ac.uk/start/software-survey/results-2010/ [21 July 2011].
12. Fay E. *Repository, Software Comparison*, Building Digital Library Infrastructure at LSE, in Ariadne Issue 64. Web Magazine for Information Professionals, 2010. Available from: http://www.ariadne.ac.uk/issue64/fay.
13. JORUM team. E-Learning Repository Systems Research Watch. *Technical report*, JISC (University of Bristol), 2006.
14. Leslie S. Challenges to Implementing DSpace as a LOR. In *COPPUL Distance Education Group*, 2006. Available from: http://www.slideshare.net/sleslie/using-dspace-as-a-lor.
15. Wolpert A. CWSpace - Archiving MIT OpenCourseWare to MIT DSpace, MIT Libraries, 2009. Available from: http://cwspace.mit.edu/ [21 July 2011].
16. Friesen N. Interoperability and Learning Objects: An Overview of E-Learning Standardization, *Interdisciplinary Journal of Knowledge and Learning Objects*. Informing Science Institute: 131 Brookhill Court, Santa Rosa, CA, 95409, USA, 2005; **1**(1): 23–31.
17. IMS Application Profile Guidelines Overview, Part 1 – Management Overview, Version 1.0, 2005. Available from: http://www.imsglobal.org/ap/apv1p0/imsap_oviewv1p0.html.

DOI: 10.1002/spe

18. Tzikopoulos A, Manouselis N, Vuorikari R. An Overview of Learning Object Repositories. In *Learning Objects for Instruction: Design and Evaluation*. IGI Global, 2007; 29–55. DOI:10.4018/978-1-59904-334-0.ch003.
19. Queirós R, Leal JPA. comparative study on LMS interoperability. In *Higher Education Institutions and Learning Management Systems: Adoption and Standardization*, IGI Global, 2011; 142–161.
20. IEEE Learning Technology Standards Committee (LTSC) official Web site. http://www.ieeeltsc.org [21 July 2011].
21. Al-Khalifa H, Davis H. The evolution of metadata from standards to semantics in E-learning applications. In *Proceedings of the 17th ACM Conference on Hypertext and Hypermedia*, Denmark, 22-25 Aug 2006; 69–72.
22. Friesen N. Semantic and Syntactic Interoperability for Learning Object Metadata. In *Metadata in Practice*, Hillman D (ed.). ALA Editions: Chicago, 2004.
23. Pons D, Hilera J, Pagés C. E-Learning metadata standards. In *Learning Technology Newsletter of IEEE Computer Societys Technical Committee on Learning Technology (TCLT)*, Vol. 13 Issue 3, Graf S, Karagiannidis C (eds), July 2011. ISSN 1438-0625.
24. ISO/IEC 19788-1:2011. Information technology - Learning, education and training - Metadata for learning resources. Part 1: Framework. First edition, 07 January 2011. Available from: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=50772.
25. Cantara L. *Metadata Encoding and Transmission Standard: Primer and Reference Manual.* Imprint, 40 (September), 2007. Available from: http://www.informaworld.com/openurl?genre=article&doi=10.1300/J104v40n03_11&magic=crossref.
26. MODS: Uses and Features. Library of Congress, October 18, 2010. Available from: http://www.loc.gov/standards/mods/mods-overview.html.
27. IMS Common Cartridge Profile: overview, Version 1.1 Final Specification. http://www.imsglobal.org/cc/ccv1p1/imscc_profilev1p1-Overview.html [21 July 2011].
28. Massart D, *et al*. Taming the Metadata Beast: ILOX , D-Lib Magazine. *D-Lib Magazine* November/December 2010; **16**(11/12). DOI: 10.1045/november2010-massart.
29. McCallum SH. A look at new information retrieval protocols: SRU, OpenSearch/a9, CQL, and XQuery. *In World Library and Information Congress: 72nd IFLA General Conference and Council*, IFLA, IFLA, 2006.
30. Ternier S, Massart D, Totschnig M, Klerkx J, Duval E. The Simple Publishing Interface (SPI). *D-Lib Magazine* 2010; **16**:957–962.
31. Simon B, Massart D, Van Assche F, Ternier S, Duval E, Brantner S, Olmedilla D, Miklós Z. A simple query interface for interoperable learning repositories. *Proceedings of the 1st Workshop on Interoperability of Web-based Educational Systems*, Chiba, Japan, 2005; 11–18.
32. Meier W. eXist: An Open Source Native XML Database. In: *NODe 2002 Web and Database-Related Workshops*, Erfurt, Germany, 2002; 169–183.
33. Fielding RT, Taylor RN. Principled Design of the Modern Web Architecture. *ACM Transactions on Internet Technology (TOIT)* May 2002; **2**(2):115–150. DOI: 10.1145/514183.514185, ISSN 1533-5399. New York: Association for Computing Machinery.