Design of a Repository of Programming Problems

José Paulo Leal *) and Ricardo Queirós **)

*) CRACS & DCC-FCUP / Portugal **) CRACS & DI-ESEIG / Portugal
zp@dcc.fc.up.pt ricardoqueiros@eseig.ipp.pt

The EduJudge project aims to open the repository of programming problems of the University of Valladolid *Online Judge (online-judge.uva.es) to pedagogical uses in secondary and higher education. This project integrates a repository of learning objects that will be used for managing collections of programming exercises and retrieving those suited to the profile of a particular student. This paper describes the conceptual architecture of the repository, based on a book library metaphor, and the definition of programming problems as learning objects. It describes also the repository's concrete architecture, starting with the definition of its use cases and its structure in components. We finish with an enumeration of the tasks ahead of us.*

1. Introduction

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

The architecture of the EduJudge system integrate three main components types: Learning Objects Repository (LOR), Evaluation Engine (EE), and Learning Management Systems (LMS). These components are autonomous in the sense that they will be deployed in different servers and each component will be able to connect to several others, creating a network as displayed in Figure 1. There will be no central node in the EduJudge network.

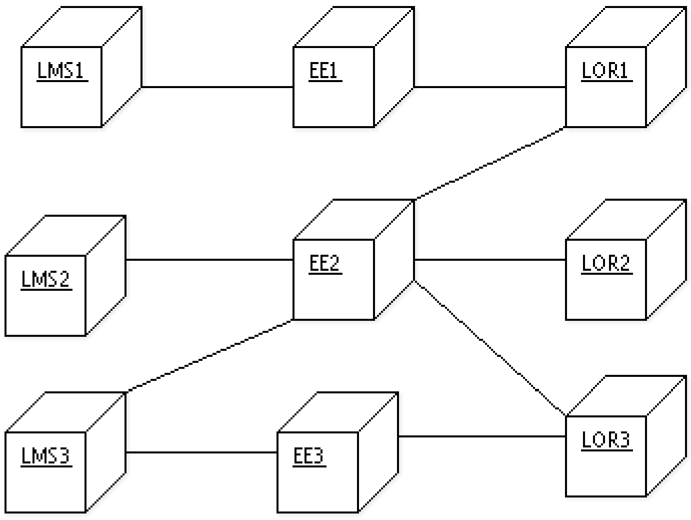

Figure 1 - EduJudge Deployment Diagram

The EduJudge projects foresees an integration with existing LMS, namely the Moodle and Quest systems. Nevertheless, our goal is to position this repository as a service provider for a larger set of systems with an automatic evaluation engine, such as a LMS or even programming contest management systems.

The remainder of this paper is organized as follows. Section 2 presents the features of existing repositories, the main expectations regarding repository systems and the features needed to make them interoperable and based in standards. Next, we present our view of the repository as a book library and the issues related with the representation of programming problems as learning objects. In section 4 we model our repository presenting its the use cases and main components. Finally, we conclude with perspective of future work.

2. State of the art

A repository of learning objects can be defined as "a system that stores electronic objects and meta-data about those objects" [ADL 2003-2]. The need for this kind of repositories is growing as the more educators are eager to use digital educational contents and more of it is available. Not surprisingly, a growing number of this type of systems appeared in recent years. The repositories currently available **[**ADL**-2003-1]** include many relevant features, such as:

- Reviews to certify quality of LO;
- Update notifications (Really Simple Syndication - RSS);
- Meta-data compliant with standards;
- Web based forms enforcing compliance with controlled vocabularies;
- Multiple forms of classification (by peer reviews, by collections, etc.).

A simple repository can be a web site listing learning objects and theirs characteristics, with features similar to those of a web search engine. It can be just a web site with information on the learning objects using Google, or any other search engine, to index its pages. Actually, the majority of the repositories are basically specialized web search engines, indexing on fields that describe in detail a learning object. One of the best examples is MERLOT (*Multimedia Educational Resource for Learning and On-Line Teaching*) repository, developed in 1997 by the *California State University Center for Distributed Learning*. It provides links to online learning materials, categorised in academic disciplines and includes a search engine.

The Jorum Team made a comprehensive survey of the existing repositories **[**JOR **2006]** and noticed that most of these systems don't store actual learning objects. They just store meta-data describing LO, including pointers to their locations on the Web. Arguably, these systems shouldn't be considered repositories at all. Since these repositories are just catalogues of pointers to LO, there is no guarantee that the listed LO are actually available. The same thing happens with the listings of any search engine, but they are not repositories of web pages, although Google will actually let you access a cached version of the page it used for indexing it.

The Jorum Team **[**JOR **2006]** also noticed that even the few repositories that actually store learning objects don't use a packaging standard. This fact precludes these repositories from exposing their functionalities to other software components. Consequently, they lack the features needed to connect with LMS and evaluation engines, since these components need to automatically query and download LOs. One of the most relevant packaging standards is the SCORM (*Sharable Content Object Reference Model*) [SCO 2004] specification (data IMS CP and meta-data IEEE LOM). The IEEE-LOM [IEEE, 2004], was not specifically designed to contemplate the requirements of automatic evaluation of programming problems. Friesen [FRI-2004] mentions forth ways that have been used to extend the IEEE-LOM model:

- Combining the IEEE-LOM elements with elements from other specifications. This approach can introduce new categories to the standard;
- Defining extensions to the IEEE-LOM elements while preserving its set of categories;
- Simplifying LOM, reducing the number of LOM elements and the choices they present;
- Extending and reducing simultaneously the number of LOM elements.

The expectations regarding repository systems vary widely and are increasing. Some surveys **[**ADL **2004]** revealed that users are concerned with issues that are not completely addressed by the existing systems , such as:

- Interoperability;
- Meta-data creation and management;
- Content versioning;
- Digital rights management.

However, the current trend in repositories is federation **[**JOR **2006]**. Using standard querying protocols and a central registry, a federation of repositories can be searched in parallel. Federation is expected to improve the reuse of LOs, although it can be argued that it also introduces new problems: since most repositories only record pointers to LOs, and each LO can be listed by several repositories, federated search has to eliminate duplicated references. The solution is trivial assuming that a LO has a unique id - for instance its URL - but it can be complex if different repositories point to multiple copies of the same LO, each with its own URL.

To our knowledge, there is currently no repository focused on LO for automatic evaluation, not to mention programming problems. Most repositories list all kinds of digital objects, ranging from PDF files to LMS files. This is understandable due to the comparatively small number of LOs available and to the lack of standards for interoperability with EE. In practice, it would be very difficult to create activities for a programming course using any of the existing repositories. In summary, we consider that the existing repositories lack the features needed to interact with the LMS and an automatic evaluation engine, such as:

- Availability of catalogued LOs, not just pointers to them;
- Interoperability with other software components;
- Compliance to existing LO packaging standards;
- Relevant number of programming problems.

3. Conceptual Architecture

We modeled our repository using a book library as metaphor: Learning Objects (LO) may be seen as books that the learner (the reader) accesses trough the LMS. As in a book library, the repository has a catalog that provides efficient search. After selecting a LO the learner will receive an actual copy of it, as supplied by its author, not just a pointer to another service. As LOs are a kind of e-books the repository will have an unlimited number of copies to lent, unlike in a physical book library.

This approach stresses the object in learning objects: they are a unit that should be stored and catalogued as such. Just like a book is a not teared apart, and its pages kept in multiples shelves, the learning object should not be stored in a different tables of a relational database, or transformed in any way to fit the requirements of a particular persistence mechanism. As a digital object, the LO should be stored and catalogued as was submitted by the producer, and retrieved unchanged.

On the other hand, a repository of LOs cannot accept just any data BLOB (Binary Large OBject). Going back to our metaphor, a book is a specific type of object: its a collection of pages with normalized dimensions, bound together by a cover. Library shelves are planned for books with these characteristics. Moreover, librarians expect to be able to reads at least parts of the book in order to get meta-data to catalogue them. By the same token, LOs must a have common set of features to be recorded in a repository. They can be a collection of files of several formats, bound together in a common archiving format, and with a manifest stating the content of the LO.

Libraries also keep records of requisitions. It is therefore very simple for a librarian to know which are the most popular books, those that were never requested and the books that readers take longer to read. In our view, this kind of information will be interesting for the next generation of LMS. Nowadays most LMS present LOs in a predefined order. Based on information about their previous use they will able to change the presentation order dynamically. For instance, a particular LO may be catalogued as easy but its usage data reveal that it was not solved by most students on an introductory level; therefore, it should only be given to students in advanced levels. To accommodate these features our repository will give to the LMS the option to record information on the use of a LO and will provide statistics on this data.

Our first challenge was the definition of programming problems as LO. They must contain every information needed either by programming contest management system or by LMS with automatic evaluation of programming problems. The LO must be a package including assets and meta-data describing them. In our case assets include problems descriptions, test vectors, solution programs, etc. Our LO definition provides standard meta-data - who wrote it, when, versions, languages, etc - as well as domain specific - problem classification, role of assets, etc. This meta-data will be recorded in a Extensible Markup Language (XML) [XML 2006] file with references to the assets in the archive. Instead of following any of the possibilities for extending the LOM proposed by Friesen [FRI-2004] mentioned in the previous section, we are currently defining our own language for describing this meta-data. To keep this approach compatible with LOM we plan to provide a transformation from our language to this standard.

Another challenge we faced was LO identification. URL's are a convenient way to identify LOs since they can be used to download it. However, being a location on a repository, an URL is not a unique LO identifier. If a LOs is replicated to another repository, the new URL loses the reference to the original. To deal with this problem we defined a structure for URLs locating LO and introduced the concept of original repository where the LO was initially stored. Hence, when copied to a diferent repository LO will preserve the original repository host in their path, immediatly after the initial "lo" directory.The only requirement made to and URL location a LO is that its path starts in the directory "lo" since we reserve the initial directory to identify the type of object downloaded from the repository. Also, we assume a single repository installation per host. For instance, a LO with identified with the URL **http://***original.repository.host***/lo/***path/to/lo* when copied to a ropository in a different host would have the following URL

http://*new.repository.host***/lo/***original.repository.host***/***path/to/lo*

4. Concrete architecture

To guide the design and implementation of the EduJudge repository, we defined a set of design principles. These principles are based not only in what we see as requirements of a repository for the EduJudge system but also as what we see as requirements for repository feeding any system with an automatic evaluation. As general principles we think that the repository must be:

- reusable in other contexts;
- complying with existing standards;
- reliable and efficient.

Based in these principles we defined the use cases for the repository and divided it in major components, as presented in the following sub-sections.

4.1 Use Cases

This repository is designed for the EduJudge System but our ambition is to implement a repository that may be **reused** in other contexts and with other systems. When defining the use cases of this repository we had in mind this possibility. We started by identifying all actors - classes of persons and external components - that need use the repository, namely:

- **Administrator**: a person responsible for **managing** the repository;

- **Producer** - a person that develops and **submits** LOs to the repository;
- **Reviewer -** a person responsible for controlling the quality of the repository by **validating** the submitted LOs;
- **Evaluation Engine (EE)** - a component that evaluates programs submitted by a student (**Learner**);
- **Learning Management System (LMS)** - a component responsible for presenting contents to the student (**Learner**);

We identified four cases in which these actors will need to use the repository. These use cases are the following:

- **Manage System** - This use case includes a set of activities related with the LOR management, such as: user access control, configuration, etc.
- **Submit LOs** - The submission of LOs, enforcing the compliance with controlled vocabularies defined in meta-data standards (IEEE LOM) and possible extensions;
- **Validate LOs** - The review process of submitted LOs, performed in order to certify the quality to the content of the repository;
- **Use LOs** - This use case is composed by several activities, including:
    - Browsing the repository - Enables the user to list, in a filtered fashion, the LOs;
    - Downloading the LO - Allows the user to obtain the LO or part of it. This activity could be done by the LMS or the EE;
    - Recording usage data - Enables the LMS/EE to post usage data back to the repository to update statistics that will change the LO profile.

In the UML use case diagram [UML] shown in Figure 2, these actors are associated to the use cases of the repository. The diagram depicts also the dependencies between use cases: LO must have been submitted by a producer before being validated by reviewers; the learner will use them, through the Evaluation Engine or the LMS, after validation. It should be noted that each of these use cases will be composed of several activities that are not *per se* use cases. For instance, the use of LO includes browsing the repository, downloading the LO and recording usage data.

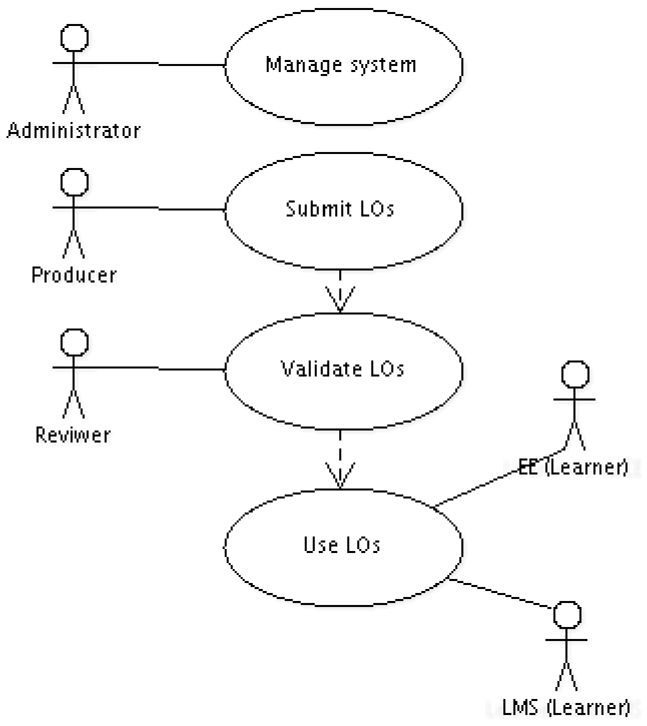

Figure 2 - Use cases of the repository

The diagram in Figure 2 also stresses the fact that we have two types of actors: persons (located on the left) and software components (located on the right). In fact, LO are expected to used mainly by software components, while other use cases require users interfaces for human users. Even so, the features required by all use cases will be available as an application programming interface (API) to enable further reuse.

Following the principle of complying to existing standards, these features will be exposed using **web services** [WS 2002]. The repository will implement two distinct flavors of web services: Simple Object Access Protocol (SOAP) [SOAP 2007] and Representational State Transfer (REST) [REST 2000]. The first is be based on the SOAP protocol, as defined by the W3C, and used to allow remote invocation (RPC) of the repository functionalities. These web services will be supported by an off the shelf SOAP engine such as Axis. The web services based on the REST style will be implemented directly over the HTTP protocol, using Java servlets, mostly to post and get resources, such as LO and usage data. The reason to implement two distinct web service flavors is also to promote reuse, by adjusting to different architectural styles.

4.2 Components

We want to keep the repository **reliable 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 - a ZIP archive - which are inherently simple operations that can be implemented almost directly over the transport protocol. Nevertheless, there are a number of features that require more elaborate implementation, in particular those associated with human actors in Figure 2 that requires a graphical users interface. However, it should be noted that these features do not require the same reliability and efficiency of the core features.

The differences between core and complementary features lead us to the design shown in Figure 3. This UML component diagram [UML] shows the repository divided in 3 components. The **Core** component exposes the main features of the repository, both to external components, such as the LMS and the EE, and to internal components - the web manager and the importer. The **Web Manager** will allow the creation, versioning, exporting, uploading of LOs and related meta-data, enforcing compliance with controlled vocabularies. The management covers also features like export to others standards, such as Dublin Core and IMS Metadata. The **Importer** component will be used to populate the repository with existing programming problems repositories, in particular the UVA on-line judge.

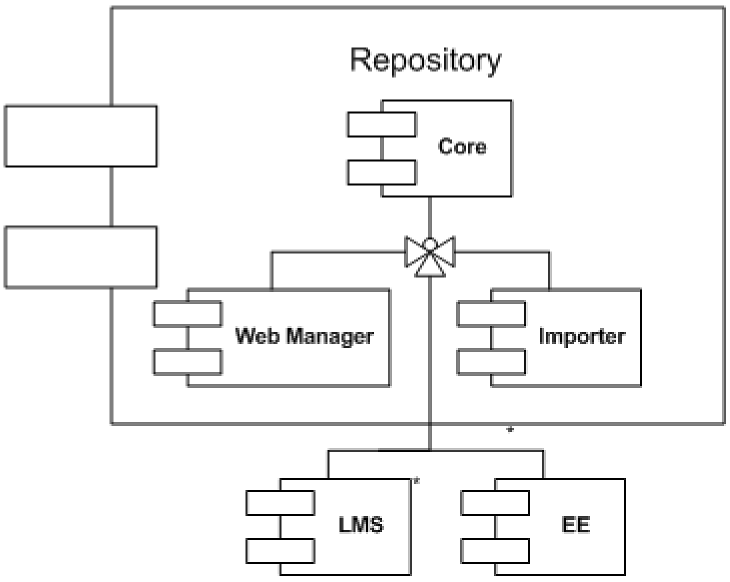

Figure 3 - Repository components

With this design, the core component can be efficiently implemented in a simple an stable component. Complementary features are delegated to helper applications that connect to the core component. Other components could be integrated in the core, enforcing the plug-able nature of the repository system.

5. Future work

This paper describes our initial design of the repository for the EduJudge system. Naturally, most of the work is still ahead of us. In this section we outline the next phases of this work-package and define our priorities that are the following:

1. **Complete the definition of LO** - We defined our LO as ZIP archives with an XML manifest but the exact definition of this manifest structure is not complete yet. We need to record data to support different types of automatic evaluation.
2. **Definition of usage data** - The LMS will post usage data regarding the LO it downloaded from the repository. This usage data will be stored in the repository, but not on the LO itself. Examples of usage data are: number of user attempts, final grade, time consumed, etc. We need to identify all the usage data that may be interesting for a LMS in order to define the this part of the repository schema.
3. **Definition of web services -** We will use web services to expose LOR features, promoting the service-oriented architecture of the repository. In the SOAP part, will define a set of generic actions related to the LO management. The REST approach, based in HTTP actions, will provide a more direct and hierarchical access to the LOR resources.
4. **Design and implementation of components -** The design of the repository components - core, web manager and importer - based on the design principles mentioned in section 4 and respective implementation.

6. References

[ADL 2003-1] Academic ADL Co-Lab 'Learning Repositories included in Learning Repository Investigation', October 2003.
(Online at http://www.academiccolab.org/resources/DraftRepositoriesList.pdf)

[ADL 2003-2] Academic ADL Co-Lab 'From Local Challenges to a Global Community:Learning Repositories and the Global Learning Repositories Summit', November 2003.
(Online at http://www.academiccolab.org/resources/FinalSummitReport.pdf)

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

[FRI-2004] Friesen, N. (2004) Semantic and Syntactic Interoperability for Learning Object Metada. In: Hillman, D. (ed.) Metadata in Practice. Chicago, ALA Editions.
(Online at: http://www.cancore.ca/semantic_and_syntactic_interoperability.html)

[IEEE-2004] IEEE Learning Technology Standards Committee (LTSC) (2002). Standard for Learning Object Metada IEEE 1484.
(Online at: http://www.ieeeltsc.org/standards/1484-12-1-2002/)

[JOR 2006] Jorum Team: E-Learning Repository Systems Research Watch. JISC / Jorum, 2006.
(Online at: http://www.jorum.ac.uk/docs/pdf/Repository_Watch_final_05012006.pdf)

[REST 2000] Roy Fielding, Representational State Transfer (Phd dissertation)
(Online at: http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm)

[SCO 2004] Scorm 2004 3rd Edition
(Online at: http://www.adlnet.gov/scorm/index.aspx)

[SOAP 2007] SOAP Version 1.2 Part 0: Primer (Second Edition)
(Online at: http://www.w3.org/TR/soap12-part0/)

[UML] The UML User Guide, Grady Booch, James Rumbaugh, Ivar Jacobson, Addison Wesley. ISBN: 0-201-57269-4

[WS 2002] Web Services Activity
(Online at: http://www.w3.org/2006/ws/)

[XML 2006] Extensible Markup Language (XML) 1.1 (Second Edition)
(Online at: http://www.w3.org/TR/2006/REC-xml11-20060816/)