

Defining Programming Problems as Learning Objects

José Paulo Leal¹ and Ricardo Queirós²

¹CRACS & DCC-FCUP, University of Porto, Portugal

zp@dcc.fc.up.pt

²CRACS & DI-ESEIG/IPP, Porto, Portugal

ricardo.queiros@eu.ipp.pt

Abstract — Standards for learning objects focus primarily on content presentation. They were already extended to support automatic evaluation but it is limited to exercises with a predefined set of answers. The existing standards lack the metadata required by specialized evaluators to handle types of exercises with an indefinite set of solutions. To address this issue existing learning object standards were extended to the particular requirements of a specialized domain. A definition of programming problems as learning objects, compatible both with Learning Management Systems and with systems performing automatic evaluation of programs, is presented in this paper. The proposed definition includes metadata that cannot be conveniently represented using existing standards, such as: the type of automatic evaluation; the requirements of the evaluation engine; and the roles of different assets - tests cases, program solutions, etc. The EduJudge project and its main services are also presented as a case study on the use of the proposed definition of programming problems as learning objects.

Keywords— Content Packaging, eLearning Services, Interoperability, Learning Objects.

I. INTRODUCTION

Learning Objects (LO) are units of instructional content that can be used, and most of all reused, on web based eLearning systems. The LO definition was targeted for Learning Management Systems (LMS) and thus they are specialized on content presentation. They 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 and for making effective use of their content in LMS. Standardize metadata plays an important role in keeping LO neutral to different vendors, both of LMS and of repositories.

Despite its success in the promotion of the standardization of eLearning content, the generic LO standards are inadequate to some domains. This fact led to the creation of application profiles – extensions to standards, policies and guidelines meeting the needs of specific communities. Those application profiles are still targeted mostly for general purpose systems,

such as LMS and repositories and do not cater for the needs of specialized eLearning systems such as automatic evaluators.

This paper focuses on a definition of programming problems as LO adequate to the interoperability of services in the area of programs automatic evaluation. This definition is a new application profile for learning objects based on Instructional Management Systems (IMS) specifications. It is being used in a European research project called EduJudge, which aims to integrate a collection of problems created for programming contests into an effective educational environment.

The remainder of this paper is organized as follows. Section 2 traces the evolution of LO standards and schema languages used for defining them. The following section starts with the definition of an evaluation model for programming problems and, based on it, a new application profile extending standard specifications and guidelines is presented, as well as the data model for representing metadata of programming problems. Then, a case study regarding the use of the new application profile in the EduJudge project is presented. Finally, a summary of the main contributions and a perspective of future research conclude this paper.

II. LEARNING OBJECT STANDARDS

The evolution of eLearning systems in the last two decades was impressive. In their first generation, eLearning systems were developed for a specific learning domain and had a monolithic architecture [1]. Gradually, these systems evolved and became domain-independent, featuring reusable tools that can be effectively used virtually in any eLearning course. The systems that reach this level of maturity usually follow a component-oriented architecture in order to facilitate tool integration. An example of this type of system is the LMS that integrates several types of tools for delivering content and for recreating a learning context (e.g. Moodle, Sakai).

The present generation values the interchange of learning objects and learners' information through the adoption of new standards that brought content sharing and interoperability to eLearning. Standards can be viewed as "documented

agreements containing technical specifications or other precise criteria to be used consistently as guidelines to ensure that materials and services are fit for their purpose" [2]. In the eLearning context, standards are generally developed with the purpose of ensuring interoperability and reusability in systems. In this context, several organizations [3]-[5] have developed specifications and standards in the last years [6]. These specifications define, among many others, standards for eLearning content [7]-[9] and interoperability [10], [11].

The most widely used standard for LO is the IMS Content Packaging (IMS CP). This content packaging specification uses an XML manifest file wrapped with other resources inside a zip file. The manifest includes the IEEE Learning Object Metadata (LOM) standard to describe the learning resources included in the package. This standard proposes a set of 77 elements, distributed among nine categories. Though all elements are optional, the standard is being used in several eLearning projects all over the world [12].

The LOM standard has achieved a high degree of acceptance in learning communities. However a closer inspection reveals a low adoption rate of LOM elements [12]. Since LOM elements are optional and in some cases too generic, several projects that have adopted the standard usually define application profiles to meet the needs of specialized domains [12].

For instance, LOM was not specifically designed to accommodate the requirements of automatic evaluation of programming problems. There is no way to assert the role of specific resources, such as test cases or solutions. Fortunately, IMS CP was designed to be straightforward to extend, meeting the needs of a target user community through the creation of the already referred 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" [13]. A well know eLearning application profile is SCORM [14] that extends IMS CP with more sophisticated sequencing and Contents-to-LMS communication.

The creation of application profiles is based in one or more of the following approaches:

- Selection of a core sub-set 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.

Following this extension philosophy, the IMS Global Learning Consortium (GLC) upgraded the Question & Test Interoperability (QTI) specification [9]. QTI describes a data model for questions and test data and, from version 2, extends the LOM with its own metadata vocabulary. QTI was designed

for questions with a set of pre-defined answers, such as multiple choice, multiple response, fill-in-the-blanks and short text questions. It supports also long text answers but the specification of their evaluation is outside the scope of the QTI. Although long text answers could be used to write the program's source code, there is no way to specify how it should be compiled and executed, which test data should be used and how it should be graded. For these reasons QTI cannot be considered adequate for automatic evaluation of programming exercises, although it may be supported for sake of compatibility with some LMS. Recently, IMS GLC proposed the IMS Common Cartridge [15] that bundles the previous specifications and its main goal is to organize and distribute digital learning content.

All these standards are described by schema languages, most often using the XML Schema Definition language (XSD). This language overcame Document Type Definition (DTD) limitations and provided several advanced features, such as, the ability to build new types derived from basic ones, manage relationships between elements (similar to relational databases) and combine elements from several schemata.

In spite of its expressiveness, XSD lacks features to describe constraints on the XML document structure. For instance, there is no way to specify dependencies between attributes, or to select the content model based on the value of another element or attribute. To address these issues several schema languages were proposed, such as RELAX NG [16] (based on TREX [17] and RELAX [18]), DSD (Document Structure Description) [19] and Schematron [20]. The Schematron language provides a standard mechanism for making assertions about the validity of an XML document using XPath expressions and can be easily combined with XML Schema.

III. PROGRAMMING PROBLEMS AS LEARNING OBJECTS

A LO containing a programming problem must include metadata to allow its use by different types of specialized eLearning services, such as evaluation engines, 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.

Firstly, an evaluation model for the programming problems is identified. Secondly, a new application profile based on the IMS-CP and LOM is proposed, relating the several metadata schemata. Thirdly, the data model of the metadata associated to the resources that compose a programming problem is described.

A. Evaluation model

The goal of defining programming problems as learning objects is to use them in systems supporting automatic evaluation. The automatic evaluation of programming problems is more complex than the automatic evaluation of

exercises supported by other application profiles, such as QTI, where answers are selected from a small predefined set. To evaluate a programming problem the learner must submit a program in source code. The evaluation of this source code usually includes a static phase, where the source code is compiled or checked for syntactic errors, and a dynamic phase, where the program is executed and its behavior is analyzed.

There are several approaches to evaluate the behavior of a program. The most common is to compare its output and side effects with those of a standard solution. Another approach is to compare a set of programs from different learners and evaluate them competitively. In order to provide meaningful metadata to the evaluation engines, a programming problem definition must have an unambiguous evaluation model. Otherwise, authors could create programming problems that risked to be evaluated differently from what they intended.

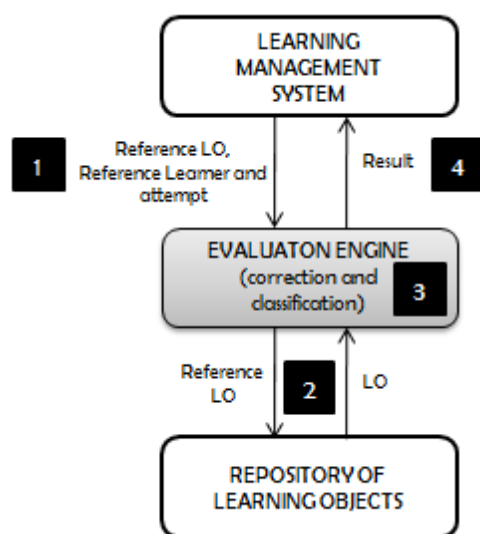


Fig. 1 Evaluation model

After considering several alternatives a single and simple four steps evaluation model was selected. This model is depicted in Fig. 1 and enumerated below.

1. The evaluator receives three pieces of data: a reference to the LO with a programming problem; an attempt to solve it - a single file, a program or an archive containing files of different types (e.g. JAR, WAR); and a reference to the learner submitting the attempt.

2. The evaluator loads the LO from a repository using the reference and uses the assets available in the LO (static tests, generated tests, unit tests, etc.) according to their role.

3. The evaluator produces an evaluation report with a classification and possibly also with a correction and feedback. The feedback that may depend on the learner's reference and may be stored for future incremental feedback to the same learner.

4. The evaluator returns the evaluation report immediately or makes it available within a short delay.

The learning object metadata assigns a role to each asset assuming this simple model. It is the responsibility of the evaluation component to use each asset appropriately

according to its role.

More specialized evaluation models were considered. For instance, unit tests can be used to perform program evaluation instead of test cases. Unit testing seems a reasonable candidate for its own specialized evaluation model, requiring a source code for a particular unit testing framework, for instance Junit. However, a similar result can be achieved without a unit testing framework but with boilerplate code linked with the learner's attempt. In this case it may help (or not) to use test files, that would be associated with a "standard" evaluation model. On the other hand, unit testing using a framework fits the general evaluation model described above, removing the need for a specialized model.

For every considered specialized model, requiring some features and excluding others, ways to combine it with assets from other evaluation models would come up. This fact led to a simple and maximal evaluation model with several optional extension points, where a specific resource (such as a test case generator or a special corrector) can be inserted.

It should be noticed that, although this evaluation model is maximal, it excludes some kinds of programming problems. For instance, it excludes programming problems where several programs from different learners are evaluated simultaneously in a competitive fashion. This case was considered for a second evaluation model. However, since this kind of programming problem is relatively rare, especially in the eLearning context, that decision was postponed to a next version of this definition.

B. Application profile

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

Metadata information in the manifest file usually follows the IEEE LOM schema, although other schemata can be used. These metadata elements can be inserted in any section of the IMS CP manifest. In this definition, the metadata that cannot be conveniently represented using LOM is encoded in elements of a new schema - EduJudge 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 and lacking in the IEEE LOM. 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 a LO, is depicted in Fig. 2.

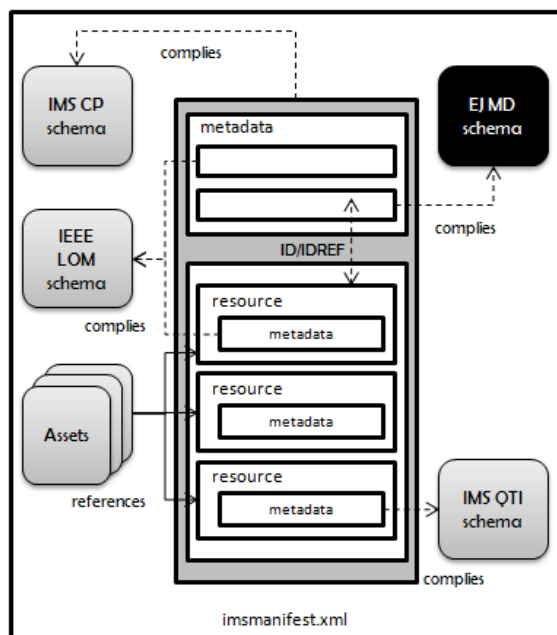


Fig. 2 Structure of a programming problem as a LO

The archive contains several files represented in the diagram as gray rectangles. The manifest is an XML file and its elements' structure is represented by white rectangles. Different elements of the manifest comply with different schemata packaged in the same archive, as represented by the dashed arrows: the manifest root element complies with the IMS CP schema; elements in the metadata section may comply either with IEEE LOM or with EJ MD; 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 composed each one by several files. In order 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 IMS CP specification is defined by a W3C XML Schema Definition (XSD). The schema describes which elements may exist in the document manifest and how those elements may be structured. Unfortunately, not all constraints of EJ MD can be expressed in XML Schema. For instance, the XSD cannot check if the EJ MD elements are included in the proper place of the manifest. Thus Schematron rules embedded in the XSD of EJ MD are also used. The XSD can be preprocessed using a XSLT; the resulting Schematron schema is further processed as a second order transformation to validate the manifest.

This application profile uses elements from several schemata and namespaces were used to avoid name clashes. In the EJ MD specification, the namespaces, filenames and namespace prefixes of XML instances are as follows:

TABLE I
SCHEMATA IN THE NEW APPLICATION PROFILE

Spec.	Namespace	Filename
IMSCP	http://www.imsglobal.org/xsd/imscp_v1p1	imscp_v1p1.xsd
LOM	http://www.imsglobal.org/xsd/imsmd_v1p2	imsmd_v1p2.xsd
QTI	http://www.imsglobal.org/xsd/imsqti_v1p1	imsqti_v1p1.xsd
EJMD	http://www.edujudge.eu/ejmd_v2	ejmd_v2.xsd

These references will be used for online validation, to conform to IMS CP Best Practice Document - to prefer online references on the IMS website, rather than static XSD files in the LO package, as they will be the most up-to-date specifications.

To represent programming problems as learning objects, able to be evaluated according to model just described, the metadata of the IMS CP was extended to assign a role to each asset. Metadata can be inserted in several points of the manifest. The placement of different types of metadata related to assets was assigned to the available extension points:

- **Domain metadata** (EJ MD), related to the automatic evaluation, in IMS CP manifest/metadata element;
- **Resource metadata** (IEEE LOM), independent from their use in automatic evaluation, within the IMS CP manifest/resource/file/metadata elements (without any domain metadata) and linked by the domain data through IDREF attributes.

C. Data Model

The core of the proposed application profile is the EduJudge schema that introduces new elements for resources specific to programming problems. This subsection presents its data model, represented schematically in Fig. 3.

The domain metadata is a hierarchy of elements whose leaves are resources. The basic **Resource** type is an asset in the distribution medium, referred 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.

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 on resources used to evaluate the learner's attempts and provide feedback.

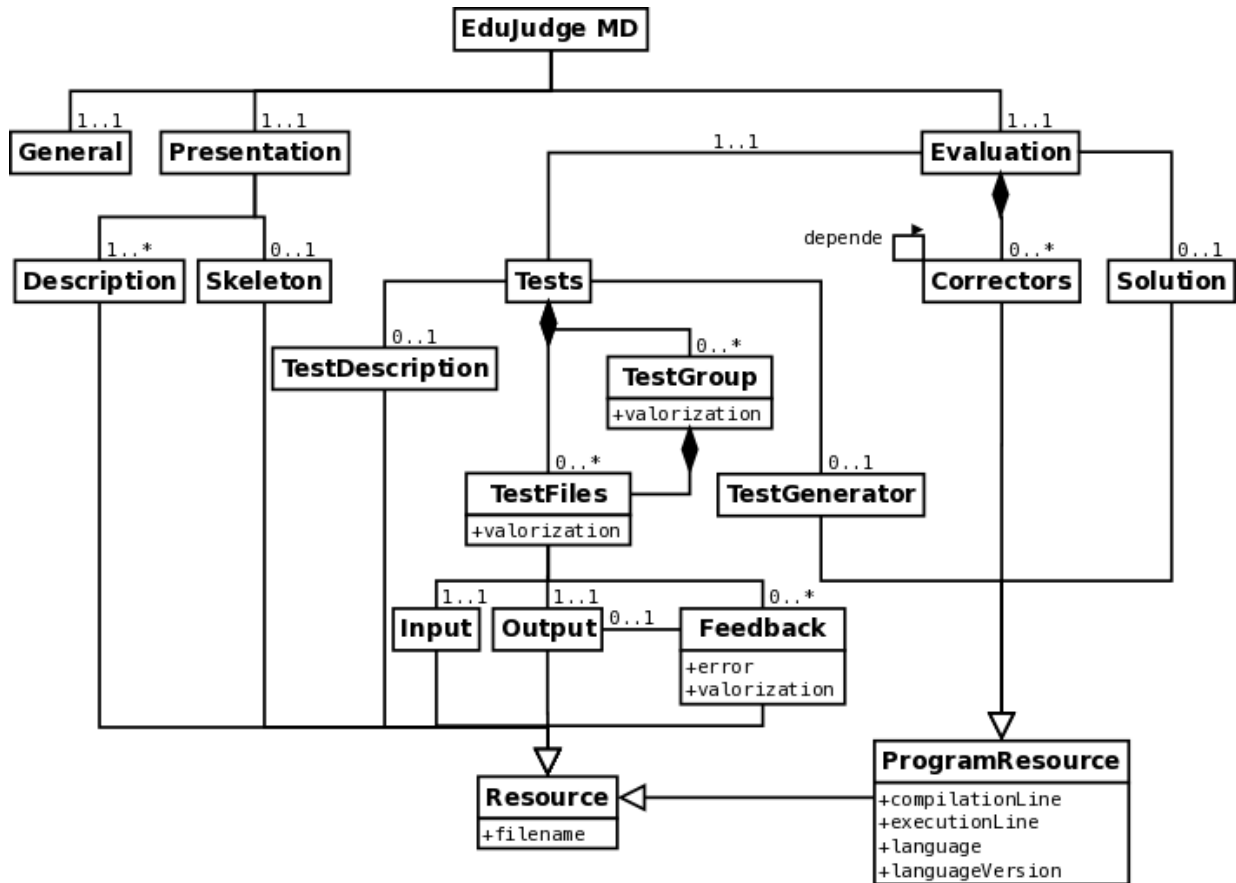


Fig. 3 The EduJudge data model

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:

1. **TestFiles** contains a pair of input and output files;
2. **TestGroup** contains an unbound collection of test files and an associated valorization;
3. **TestDescription** identifies a test file encoded in a language that describes test cases;
4. **TestGeneration** identifies a program that will generate 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 to 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. To correct the program is used 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 in 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 generate tests to 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. Can be used to: compute software metrics on the source code, judging the quality of source code; perform unit testing on the program; check the structure of the program's source code.
- **Dynamic:** invoked after each execution with a test case. Deals with non-determinism (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.

IV. CASE STUDY

The purpose of a LO is to make a particular piece of instructional content available to multiple eLearning systems, especially LMS.

A LO containing a programming problem, with adequate metadata for a well-defined evaluation model, can also be used by specialized eLearning systems and promote their interoperability. These features are part of the requirements of the EduJudge project that was used as a case study for the proposed definition of programming problems as LO.

This section starts with a general description of the EduJudge project and proceeds with a brief explanation of its components. For each component is succinctly described its architecture and highlighted the impact of the programming problem definition presented on the previous section.

A. The EduJudge project

The European research project EduJudge [21] aims to open the Valladolid online judge (<http://uva.onlinejudge.org/>) to secondary and higher education, benefiting from its considerable collection of programming problems from international and worldwide ACM-ICPC [22] competitions. The vision of the EduJudge project is of an eLearning system that integrates systems already in use, such as LMS, with programming problems that are already available from programming competitions.

To fulfill this vision the architecture of the EduJudge system adheres to service oriented principles [23]. This architectural model is based on services that are able to participate on different reconfigurable processes. Services reside on a physical location, act on their own resources and are loosely coupled to other services. The EduJudge project includes three types of such services:

Learning Objects Repository (LOR), to store programming problems and to retrieve those suited to a particular learner profile;

Evaluation Engine (EE), to automatically evaluate and grade the students' attempts to solve the problems;

Learning Management System (LMS), to manage the presentation of problems to learners.

The communication among these components complies to the IMS DRI specification and is depicted schematically in Fig. 3 as an UML sequence diagram. The concept of programming problem as a LO is central to this communication model.

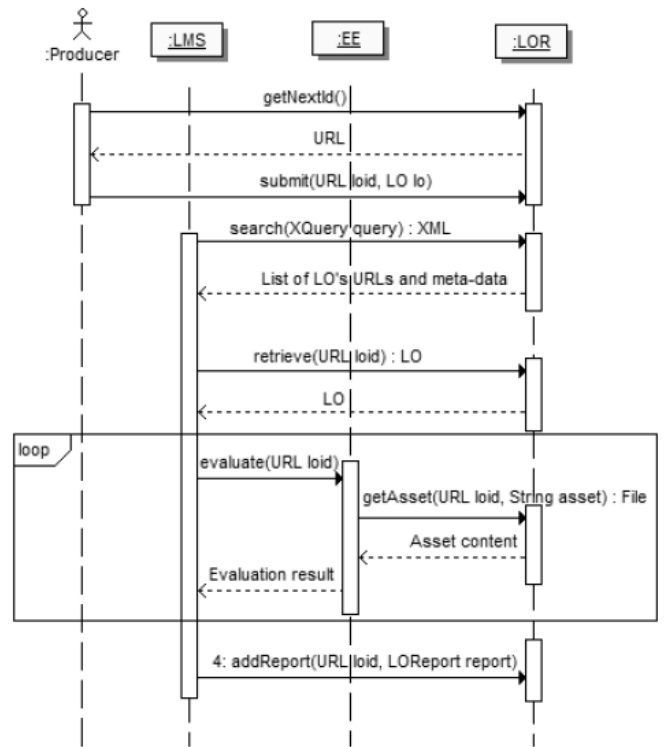


Fig. 4 Communication model among EduJudge components

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

B. Learning object repository

The repository of specialized LO of EduJudge is named crimsonHex. It was developed as part of the EduJudge project to act as a programming problem repository service to the EE and the LMS. This subsection highlights the architecture of crimsonHex and its relation to the programming problem definition presented in the previous section. Details on the

implementation of crimsonHex can be found elsewhere [24].

The architecture of crimsonHex repository is divided in three main components:

Core, to expose 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;

Web Manager, to allow the creation, revision, uploading/downloading of LO and related metadata, enforcing compliance with controlled vocabularies;

Importer, to populate the repository with existing legacy repositories.

Searching LO in the repository is based on queries on their XML manifests. Since manifests are XML documents with complex schemata, particular attention was paid to databases systems with XML support: XML enabled relational databases and Native XML Databases (NXD), such as eXist and Sedna.

XML enabled relational databases are traditional databases with XML import/export features. They do not store internally data in XML format hence they do not support querying using XQuery. Since queries in this standard are a DRI recommendation this type of storage is not a valid option. In contrast, NXD uses the XML document as fundamental unit of (logical) storage, making it more suitable for data schemata difficult to fit in the relational model. Finally, eXist [25] NXD was chosen since it supports all the required XML standards and it has a strong user community.

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

C. Evaluation engine

The evaluation engine of the EduJudge project is an improvement and optimization of the Online Judge evaluation engine [26]. To process an evaluation request the engine receives a program in source code and a programming problem reference. This reference is an URL that is used for downloading the LO from the repository. The metadata from the EJ MD schema is used for identifying the relevant assets in the LO, in particular test files, valorizations and feedback.

The evaluation engine has three main components:

Submission handler, responsible for receiving evaluations requests from different sources, (web services, web forms, email messages) and feeding them to the judge daemon's queue; it returns a ticket that is used by the service client, typically an LMS, to retrieve the evaluation report;

Judge daemon, processes a queue of evaluation requests

and, for each request, fetches the programming problem definition, compiles the submitted source and executes it against the provided test cases; it is also responsible for grading and correcting using the metadata provided by the LO;

Web front-end, for configuring the service and submitting programs to test and debug the evaluator.

All components use a shared Structured Query Language (SQL) database as primary means of communication among them.

The new evaluation engine is planned to support several evaluation models including, among others: 1) single input-output test files; 2) multiple input-output test files; 3) interactive server problems and 4) interactive user problems. The first two models overlap the evaluation model underlying the proposed definition of programming problems as LO. Moreover, all the problems in the UVA Online Judge correspond to the first model. The second model is very important from a pedagogical point-of-view since it allows better grading and feedback. Part of the effort of populating the EduJudge repository was the automatic conversion between these two models. The last two models are not yet covered by the definition but they are seldom used in eLearning and they are absent from the UVA collection of programming problems.

D. Learning management system

Moodle [27] is the reference LMS selected for the EduJudge system. The integration of Moodle in the EduJudge network is achieved through a set of plugins and modules. These include a user interface for configuration of remote services (LOR and EE) and to select competitive learning strategies implemented locally that complement the services provided by the evaluation engine. Moodle provides several extension mechanisms, two of which were used in EduJudge to implement these central components:

Activity Module, an evolution of a contest-driven learning activity module (QUESTOURnament) [26] that incorporates competitive and collaborative contests involving both programming problems and general purpose questions;

Question-Type plugin, managing question-types for remote evaluation (provided by an EE) and remote storage (in a LOR). With this plugin Moodle is able to delegate to external services the evaluation of some kinds of exercises.

The Question-Type plugin provides also a centralized questionnaires management system for the QUESTOURnament module. Each challenge can be defined as a complete questionnaire made up of a set of questions from the database. The plug-in was implemented on top of the Question Engine and the Question Bank of Moodle.

The Question-Type plug-in interacts with the repository in order to populate the Question Bank and uses both general metadata provided by the LOM schema, such as name and author, and also specific metadata provided by the EJ MD

schema, such as problem descriptions and source code skeletons.

V. CONCLUSION

This paper presents a definition of programming problems as learning objects. The main contribution of this work is the extension of an IMS standard to the particular requirements of a specialized domain - the automatic evaluation of programming problems. The described approach can be adapted to other learning domains, in particular those with other forms of non-trivial automatic evaluation.

The definition of programming problems as learning objects is framed by an evaluation model that allows us to assign specialized roles to different assets. Based on this model a scope for the new metadata, and how it interplays with existing specifications and guidelines, was defined. For this new application profile a data model for the metadata that characterizes assets of LO containing programming problems was defined.

The result of this research work is being used in EduJudge project to promote interoperability among its services. The experience with EduJudge is presented as a case study of the applicability of the proposed definition. A short description of the project and of the services that are most affected by this definition was included

In its current status the EduJudge Metadata (EJ MD) is available for test and download [28]. The future work includes the adaptation of the schema to support new evaluation models, for instance, programming problems where the evaluator aggregates programs submitted by two or more learners.

ACKNOWLEDGMENT

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

REFERENCES

- [1] Dagger, D., O'Connor, A., Lawless, S., Walsh, E., Wade, V.: Service Oriented eLearning Platforms: From Monolithic Systems to Flexible Services (2007)
- [2] Bryden, A.: Open and Global Standards for Achieving an Inclusive Information Society.
- [3] IMS Global Learning Consortium. URL: <http://www.imsglobal.org>
- [4] IEEE Learning Technology Standards Committee. URL: <http://ieeeltsc.org>
- [5] ISO/IEC- International Organization for Standardization. URL: <http://standards.iso.org/ittf/PubliclyAvailableStandards/index.html>
- [6] Friesen, N.: Interoperability and Learning Objects: An Overview of E-Learning Standardization". *Interdisciplinary Journal of Knowledge and Learning Objects*. 2005.
- [7] IMS-CP – IMS Content Packaging, Information Model, Best Practice and Implementation Guide, Version 1.1.3 Final Specification IMS Global Learning Consortium Inc., URL: <http://www.imsglobal.org/content/packaging>.
- [8] IMS-Metadata - IMS MetaData. Information Model, Best Practice and Implementation Guide, Version 1.2.1 Final Specification IMS Global Learning Consortium Inc., URL: <http://www.imsglobal.org/metadata>.
- [9] IMS-QTI - IMS Question and Test Interoperability. Information Model, Best Practice and Implementation Guide, Version 1.2.1 Final Specification IMS Global Learning Consortium Inc., URL: <http://www.imsglobal.org/question/index.html>.
- [10] IMS DRI - IMS Digital Repositories Interoperability - Core Functions Information Model, URL: http://www.imsglobal.org/digitalrepositories/driv1p0/imsdri_infov1p0.html.
- [11] Simon, B., Massart, D., van Assche, F., Ternier, S., Duval, E., Brantner, S., Olmedilla, D., & Miklos, Z. (2005). A Simple Query Interface for Interoperable Learning Repositories. In Proceedings of the WWW 2005 Conference, retrieved March 16, 2006 from <http://nm.wu-wien.ac.at/e-learning/interoperability/www2005-workshop-sqi-2005-04-14.pdf>
- [12] Godby, C.J.: What Do Application Profiles Reveal about the Learning Object Metadata Standard? *Ariadne Article in eLearning Standards*, 2004.
- [13] IMS Application Profile Guidelines Overview, Part 1 - Management Overview, Version 1.0. URL: http://www.imsglobal.org/ap/apv1p0/imsap_oviewv1p0.html.
- [14] ADL SCORM URL: <http://www.adlnet.gov/Technologies/scorm>
- [15] IMS Common Cartridge Profile, Version 1.0 Final Specification. URL: http://www.imsglobal.org/cc/ccv1p0/imscc_profilev1p0.html
- [16] Clark, J, Murata, M.: RELAX NG Specification, OASIS Committee Specification, December 2001, <http://relaxng.org/spec-20011203.html>
- [17] Clark, J.: TREX - Tree Regular Expressions for XML. Thai Open Source Software Center, 2001, <http://www.thaiopensource.com/trex/>.
- [18] Murata, M.: RELAX (Regular Language description for XML). INSTAC (Information Technology Research and Standardization Center), 2001, <http://www.xml.gr.jp/relax/>.
- [19] Moller, A.: Document Structure Description 2.0, BRICS, 2002, <http://www.brics.dk/DSD/dsd2.html>.
- [20] The Schematron, An XML Structure Validation Language using Patterns in Trees, <http://www.ascc.net/xml/resource/schematron/schematron.html>.
- [21] EduJudge project – Integrating On-line Judge into Effective E-learning. URL: <http://www.edujudge.eu>
- [22] ACM ICPC – International Collegiate Programming Contest. URL: <http://icpc.baylor.edu/icpc/>
- [23] Krafzig, D., Banke, K., Slama, D. Enterprise SOA: Service-Oriented Architecture Best Practices. 1.ed. Estados Unidos da América: Prentice Hall, 2004. ISBN 0131465759
- [24] Leal, J.P., Queirós, R.: CrimsonHex: a Service Oriented Repository of Specialised Learning Objects. In: ICEIS 2009: 11th International Conference on Enterprise Information Systems, Milan (2009)
- [25] Meier, W.: eXist: An Open Source Native XML Database. In: NODE 2002 Web and Database-Related Workshops, (2002)
- [26] Regueras, L.M., Verdú, E., Castro, J.P., Pérez, M.A., Verdú, M.J. Design of a Distributed and Asynchronous System for Remote Evaluation of Students' Submissions in Competitive E-learning. In: ICEE 2008: International Conference on Engineering Education, Budapest (2008).
- [27] Cole, J., Foster, H.: Using Moodle - Teaching with the Popular Open Source Course Management System, O'Reilly – Community Press.
- [28] EduJudge MetaData (EJ MD) specification (version 2.0). URL: http://mooshak.dcc.fc.up.pt/~edujudge/schemaDoc/examples/ejmd/ejmd_v2.xsd