Extending the Learning Object definition to represent Programming Problems

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

[1] CRACS & DCC-FCUP, University of Porto, Portugal
zp@dcc.fc.up.pt

[2] CRACS & DI-ESEIG/IPP, Porto, Portugal
ricardo.queiros@eu.ipp.pt

Abstract. The present generation of eLearning platforms values the interchange of learning objects standards. Nevertheless, for specialized domains these standards are insufficient to fully describe all the assets, especially when they are used as input for other eLearning services. To address this issue we extended an existing learning objects standard to the particular requirements of a specialized domain, namely the automatic evaluation of programming problems. The focus of this paper is the definition of programming problems as learning objects. We introduce a new schema to represent metadata related to automatic evaluation that cannot be conveniently represented using existing standards, such as: the type of automatic evaluation; the requirements of the evaluation engine; or the roles of different assets - tests cases, program solutions, etc. This new schema is being used in an interoperable repository of learning objects, called crimsonHex.

Keywords: eLearning, Learning Objects, Content Packaging, Interoperability.

1 Introduction

The majority of the eLearning platforms available today follow a component-oriented architecture. These systems assemble a collection of generic tools - such as forums or multiple choice quizzes - that are considered to be useful for all learning areas. Despite their success, they have also been target of criticism: their tools are too general and they are difficult to integrate with other eLearning systems [1]. These issues led to recent initiatives to adapt Service Oriented Architecture (SOA) [2] to eLearning. Apart from the systems integration, other problems arise related with the standardization of eLearning content. The existing standards are too generic and not adequate to specific domains, such as the definition of programming problems.

This paper focuses on a definition of programming problems as learning objects (LO) adequate to the interoperability of services in the area of automatic evaluation of programming problems. This new definition represents also a new application profile for learning objects based on Instructional Management Systems (IMS) specifications and extended to accommodate domain specific issues. This definition 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 eLearning system resulting from the EduJudge project includes different types of services hence a precise definition of programming problems as learning objects is essential to ensure interoperability among them.

The remainder of this paper is organized as follows: Section 2 traces the evolution of LO standards with emphasis on schema languages. In the following section we present a new application profile based on IMS specifications to represent programming problems. In this section we also detail the combination of two different types of validation languages: grammar and rule-based schema languages. Then, we present a case study regarding the use of the new application profile in crimsonHex, a repository of specialized learning objects. Finally, we conclude with a summary of the main contributions of this work and a perspective of future research.

2 State of Art

The evolution of eLearning systems 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 Learning Management System (LMS) that integrate 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" [3]. In the eLearning context, standards are generally developed with the purpose of ensuring interoperability and reusability in systems. In this context, several organizations [4, 5, 6] have develop specifications and standards in the last years [7]. These specifications define, among many others, standards for eLearning content [8, 9, 10] and interoperability [11, 12].

As said before, current LO standards are quite generic and not adequate to specific domains, such as the definition of programming problems. The most widely used standard for LO is the IMS Content Packaging (IMS CP). This content packaging format 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. However, LOM was not specifically designed to accommodate the requirements of automatic evaluation of programming problems. For instance, there is no way to assert the role of specific resources, such as test cases or solutions. Fortunately, IMS CP was 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" [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 aims to meet the needs of the target user community, aid integration and enhance interoperability between tools and services of the community. The creation 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 [10]. 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 we consider that QTI is not adequate for automatic evaluation of programming exercises, although it may be supported for sake of compatibility with some LMS. Recently, IMS GLC proposed the IMS Common Cartridge [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 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 W3C XML Schema documents.

3 Application profile

Based on the previous approaches to create a new eLearning application profile, we defined programming problems as learning objects by extending the IMS CP specification. An IMS CP learning object assembles resources and metadata into a distribution medium, in our case a file archive in zip format, with its content described in a file named `imsmanifest.xml` in the root level. The manifest contains four sections: 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 our case, 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. 1.

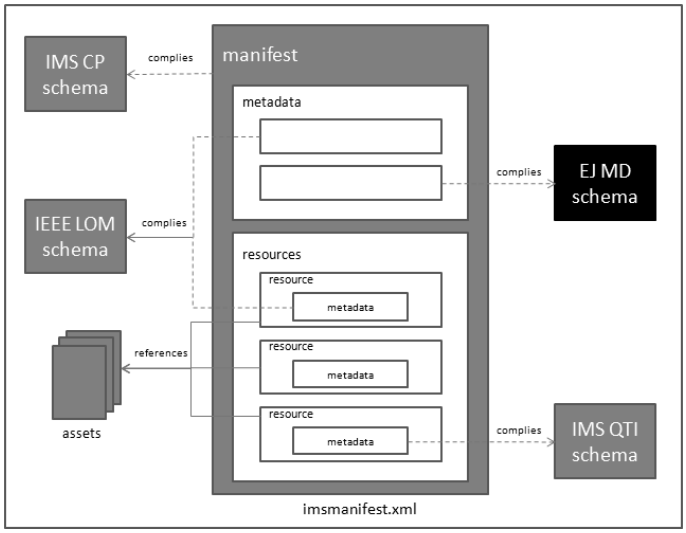

Fig. 1. Structure of a programming problem as a learning object.

The archive contains several files represented in the diagram as grey rectangles. The manifest is an XML file and its elements' structure is represented by white rectangles. Different elements of the manifest comply with different schemata packaged in the same archive, as represented by the dashed arrows: the manifest root element complies with the IMS CP schema; elements in the metadata section may comply either with IEEE LOM or with EJ MD; 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 in solid arrows.

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. Instance documents can be validated using this XSD schema. In our application profile we used 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 1. Schemata in the new Application Profile.

Specification	Namespace	Filename
IMS CP	http://www.imsglobal.org/xsd/imscp_v1p1	imscp_v1p1.xsd
IEEE LOM	http://www.imsglobal.org/xsd/imsmd_v1p2	imsmd_v1p2.xsd
IMS QTI	http://www.imsglobal.org/xsd/imsqti_v1p1	imsqti_v1p1.xsd
EJ MD	http://www.edujudge.eu/ejmd_v2	ejmd_v2.xsd

These references will be used for on-line 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.

3.1 The EduJudge schema

The corner stone of this definition of programming problems as learning objects is automatic evaluation. Consequently, this definition assumes the existence of a component responsible for evaluating learners' attempts based on the learning object and producing a result. Moreover, it needs also to assume an evaluation model supported by the evaluator. After considering several possible alternatives we decided on a single and simple evaluation model following three steps:

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

Assuming this simple model, the learning object metadata simply assigns a role to each asset. It is the responsibility of the evaluation component to use each asset appropriately according to its role.

To represent programming problems as learning objects, able to be evaluated according to model we just described, we extended the metadata of the IMS CP, as foreseen in this specification. New metadata can be inserted in several points of the manifest. Based on the available choices we decided to place different types of metadata in the following 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.

The domain metadata shown in Figure 2 is divided in three categories: 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.

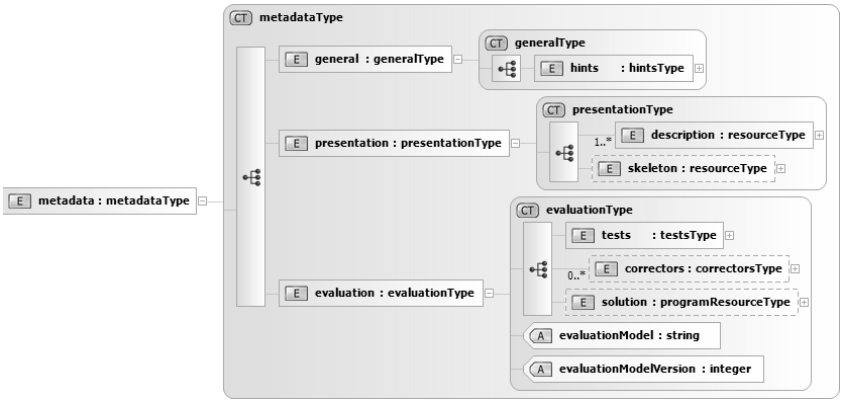

Fig. 2. The domain metadata of the EJ MD specification

The IMS CP resources section is a collection of resource elements, each one grouping several files. In order to link the EJ MD domain metadata described above, with the related resources, we used the IDREF XML Schema type in the domain metadata to reference the `resource` elements, more precisely, the IMS CP `identifier` attribute, as represented in Fig. 3.

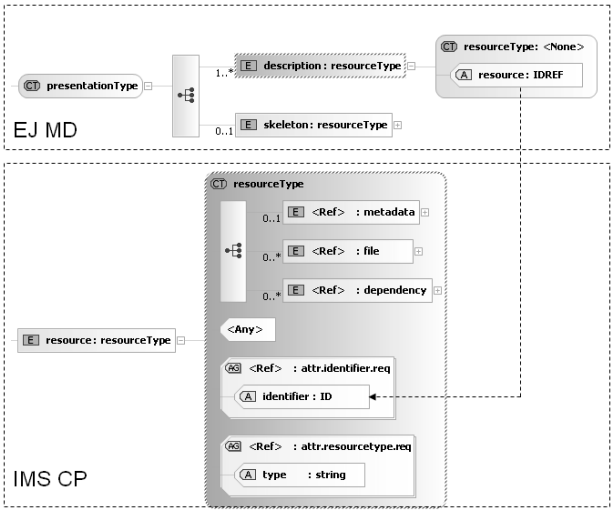

Fig. 3. The binding model of EJ MD domain metadata and the IMS CP resources

3.2 Schematron processing

Despite the expressiveness of XML Schema, there are several situations where it is not possible to validate a document with only this language. For example, the following cases cannot be validating using an XML Schema:

- general type is included only in the `manifest/metadata` element (because of the multiple extension points provided by the IMS CP);
- IDREF type points to a file resource with the appropriate type;
- value of the `imsmd:minimumversion` element is less than the value of the `imsmd:maximumversion` element.

To check this type of constraints we cannot use XML Schema. There are, at least, three options: combine with others schema languages; write code in a programming language to express the additional constraints; use an XSLT/XPath stylesheet. We will use the former, because we want to maintain the solutions based in XML technologies and, if possible, in a single schema document. There are several alternative schema languages, such as RELAX, TREX and Schematron. In this case, we need to use a rule-based validation language in order to find certain patterns in the XML document. A good candidate to this “second level of validation” is Schematron. The last constraint enumerated could be validated with this rule as a separate file:

```
<schema xmlns="http://www.ascc.net/xml/schematron" >
  <pattern name="version validation">
    <rule context="//imsmd:requirement">
      <assert test="imsmd:minimumversion &gt; imsmd:maximumversion">
        ERROR
      </assert>
    </rule>
  </pattern>
</schema>
```

Schematron validation can be used in conjunction with a XML schema validation using two approaches:

- as separate files (using pipeline validation languages [21,22]);
- as a unique file (embedding Schematron rules in the XML Schema).

To simplify the file version management we decided for the second option and used Schematron rules embedded within the `appinfo` elements in the XSD document. However, a W3C XML Schema processor does not validate constraints expressed by the embedded Schematron rules. They need to be extracted from the source schema and concatenated into a new Schematron document. To address this issue we created a stylesheet (`Schematron-Generator.xsl`) to extract embedded Schematron rules from a W3C XML Schema document and merge them into a complete schema. This approach was used in Robertsson work [23] and can be summarized by Fig. 4.

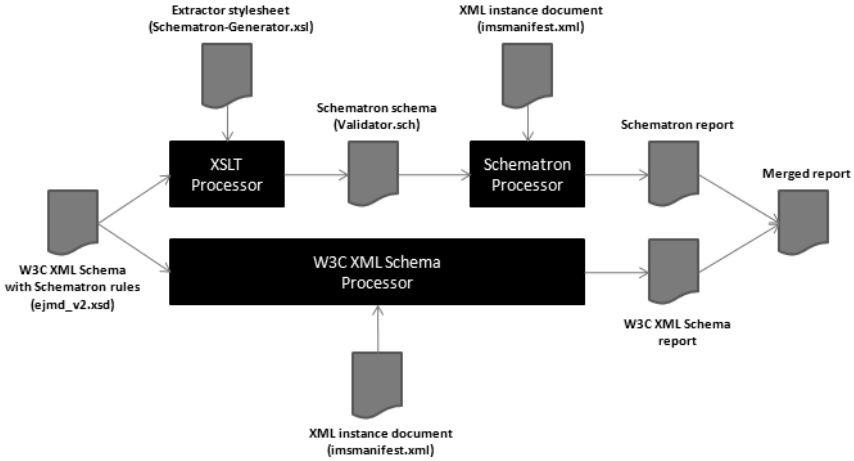

Fig. 4. Validation of XML files by a W3C XML Schema with Schematron rules.

Since Schematron rules are built using XPath and XSLT functions, the Schematron processor depicted in the previous figure is based on a XSLT processor.

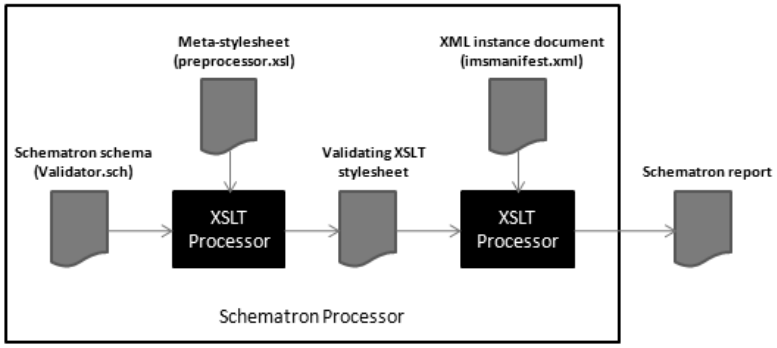

Fig. 5. Schematron processing.

To perform this validation we used an implementation of a Java API for the Schematron language [24] that organizes the Schematron processing in two steps, as shown in Fig. 5:

- The Schematron schema is transformed into a validating XSLT stylesheet by a meta-stylesheet provided by the API.
- The validating stylesheet is then used on the XML instance document and the result will be a report based on rules/assertions of the Schematron schema.

With this approach we can benefit from the combination of these two powerful validation languages and many of the constraints that previously had to be checked in the application code can now be abstracted to the schema. However it should be noticed that in time critical applications the overhead of processing the embedded Schematron rules may be unaffordable.

4 Case Study

In this section we describe the integration of the proposed programming problem definition in a specialized and interoperable repository of LO named crimsonHex. The crimsonHex repository is being used in a European research project called EduJudge that aims to open Valladolid on-line judge (http://uva.onlinejudge.org/) to secondary and higher education, benefiting from its considerable collection of programming problems from international and worldwide ACM-ICPC competitions.

The integration of the EduJudge schema in crimsonHex and the feedback from other partners in the project were crucial to evaluate the usefulness of the proposed IMS application profile. In the remainder of this section we make a succinct description of the repository with emphasis on XML storage and validation. Details on the implementation of crimsonHex can be found elsewhere [25].

4.1 Repository components

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

- **the Core** exposes the main features of the repository, both to external services, such as the LMS and the Evaluator Engine, and to internal components - the Web Manager and the Importer;
- **the Web Manager** allows the creation, revision, , uploading/downloading of LOs and related metadata, enforcing compliance with controlled vocabularies;
- **the Importer** populates the repository with existing legacy repositories.

4.2 XML Storage

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

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

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

We analysed several open source NXD, including SEDNA, OZONE, XIndice and eXist, Only eXist implements the complete list of the features enumerated above, which led us to select it as the storage component of crimsonHex. It has also two important features [26] worth mentioning: support for collections, to structure the database in groups of related documents and automatic indexes to speed up the database access**.**

4.3 Validation levels

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.

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

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

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

5 Conclusions

In this paper we described the 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. Although we focused on the automatic evaluation of programming problems, we think that the described approach can be adapted to other learning domains. This new application profile is being used in crimsonHex, an interoperable repository of learning objects.

In its current status the EduJudge Metadata is available for test and download at the following URL `http://www.dcc.fc.up.pt/schemaDoc`. Our future work will be to adapt the schema to support new evaluation models, for instance, programming problems where the evaluator aggregates programs submitted by two or more learners.

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

References

1. Dagger, D., O'Connor, A., Lawless, S., Walsh, E., Wade, V.: Service Oriented eLearning Platforms: From Monolithic Systems to Flexible Services (2007)
2. Krafzig, D., Banke, K., Slama, D. Enterprise SOA: Service-Oriented Architecture Best Practices. 1.ed. Estados Unidos da América: Prentice Hall, 2004. ISBN 0131465759
3. Bryden, A.: Open and Global Standards for Achieving an Inclusive Information Society.
4. IMS Global Learning Consortium. URL: http://www.imsglobal.org
5. IEEE Learning Technology Standards Committee. URL: http://ieeeltsc.org
6. ISO/IEC- International Organization for Standardization. URL: http://standards.iso.org/ittf/PubliclyAvailableStandards/index.html
7. Friesen, N.: Interoperability and Learning Objects: An Overview of E-Learning Standardization". Interdisciplinary Journal of Knowledge and Learning Objects. 2005.
8. 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.

9. 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.
10. 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.
11. IMS DRI - IMS Digital Repositories Interoperability - Core Functions Information Model, URL: http://www.imsglobal.org/digitalrepositories/driv1p0/imsdri_infov1p0.html.
12. 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.
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 Overview. 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. Document Schema Definition Languages (DSDL), Working Draft – ISO/IEC 2004. URL: http://www.dsdl.org/.
22. Jelliffe, R.: Schemachine: A framework for modular validation of XML documents, 2002.
23. Robertsson, E.: Combining Schematron with other XML Schema languages, 2002.
24. Hovhannisian, A., Becker, O.: JAVA API for Schematron. URL: http://www2.informatik.hu-berlin.de/~obecker/SchematronAPI/, 2001.
25. 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)
26. Meier, W.: eXist: An Open Source Native XML Database. In: NODe 2002 Web and Database-Related Workshops, (2002)