BabeLO—An Extensible Converter of Programming Exercises Formats

Ricardo Queirós and José Paulo Leal

Abstract—In the last two decades, there was a proliferation of programming exercise formats that hinders interoperability in automatic assessment. In the lack of a widely accepted standard, a pragmatic solution is to convert content among the existing formats. BabeLO is a programming exercise converter providing services to a network of heterogeneous e-learning systems such as contest management systems, programming exercise authoring tools, evaluation engines and repositories of learning objects. Its main feature is the use of a pivotal format to achieve greater extensibility. This approach simplifies the extension to other formats, just requiring the conversion to and from the pivotal format. This paper starts with an analysis of programming exercise formats representative of the existing diversity. This analysis sets the context for the proposed approach to exercise conversion and to the description of the pivotal data format. The abstract service definition is the basis for the design of BabeLO, its components and web service interface. This paper includes a report on the use of BabeLO in two concrete scenarios: to relocate exercises to a different repository, and to use an evaluation engine in a network of heterogeneous systems.

Index Terms—Interoperability, web services, REST, programming exercise formats, e-learning

1 Introduction

Assessment plays a vital role in learning. However, automatic assessment of exercises other than multiple choice can be a rather a complex task. This is certainly the case with assessment of computer programs in two distinct learning contexts: curricular and competitive learning.

Introductory programming courses are part of the curricula of many engineering and sciences programs. These courses rely on programming exercises, assignments, and practical examinations, to consolidate knowledge and evaluate students. The enrolment in these courses is usually very high, resulting in a great workload for the faculty and teaching assistants. In this context, the availability of many programming exercises from different sources is of great importance.

Competitive learning relies on the competitiveness of students to increase their programming skills. This is the common goal of several programming contests where students at different levels compete such as the International Olympiad in Informatics (IOI), for secondary school students; the ACM International Collegiate Programming Contests (ICPC), for university students; and the IEEE-Xtreme, for IEEE student members. Contest management systems and online judges where students train rely also on automatic assessment. In this context, there is also the need for new programming problems that may come from different systems. Also, competitive learning is usually a source or programming exercises for curricular learning that are recast as learning objects.

In both contexts, the lack of a standard—or at least a widely used format—creates a modern Babel tower made of learning objects, of assessment items that cannot be shared among automatic assessment systems. These systems whose interoperability is hindered by the lack of a common format include contest management systems, evaluation engines (EE), repositories of learning objects, and authoring tools. A pragmatical approach to remedy this problem is to create a service to convert among existing formats. A kind of translation service specialized in programming problems formats.

To convert programming exercises on-the-fly among the most used formats is the purpose of the BabeLO—a service to cope with the existing Babel of Learning Object formats for programming exercises. BabeLO was designed as a service to act as a middleware in a network of systems typically used in automatic assessment of programs. It provides support for multiple exercise formats and can be used by 1) evaluation engines to assess exercises regardless of its format; 2) repositories to import exercises from various sources; and 3) authoring systems to create exercises in multiple formats or based on exercises from other sources.

The remainder of this paper is organized as follows: Section 2 analyses several of existing formats to highlight both their differences and their similar features. Based on this analysis, Section 3 presents an approach to extensible format conversion and details the features of Programming Exercises Interoperability Language (PExIL)—the pivotal format in which the conversion is based—and the service functions. Section 4 provides details on the design and implementation of BabeLO, including the service API and the interfaces required to extend the conversion to a new format. To evaluate the effectiveness and efficiency of this approach, this paper reports on two actual uses of BabeLO:

- *R. Queirós is with the CRACS and INESC-Porto LA, Faculty of Sciences, University of Porto, and DI-ESEIG/IPP, Rua D. Sancho I. 981, Vila do Conde, 4480-876 Porto, Portugal. E-mail: ricardo.queiros@eu.ipp.pt.*
- *J.P. Leal is with the CRACS and INESC-Porto LA, Faculty of Sciences, University of Porto, DCC - R. do Campo Alegre, 1021/1055, 4169-007 Porto, Portugal. E-mail: zp@dcc.fc.up.pt.*

Manuscript received 7 May 2012; revised 25 July 2012; accepted 6 Oct 2012; published online 15 Oct. 2012.
For information on obtaining reprints of this article, please send e-mail to: lt@computer.org, and reference IEEECS Log Number TLT-2012-05-0066.
Digital Object Identifier no. 10.1109/TLT.2012.21.

1) to relocate exercises to a different repository and 2) to use an evaluation engine in a network of heterogeneous systems. Section 5 summarizes the main contributions of this research and plans futures developments of this service.

2 Programming Exercises Formats

The increasing popularity of programming contests worldwide resulted in the creation of several contest management systems. At the same time, Computer Science courses use programming exercises to encourage the practice on programming. The interoperability between these type of systems is becoming a topic of interest in the scientific community. In order to address these interoperability issues several problem formats were developed. The next sections details four formats: CATS, FreeProblemSet (FPS), Mooshak Exchange Format (MEF) and Peach Exchange Format (PEF). Then, their features are synthesize based on a specific exercises format expressiveness model.

2.1 CATS

CATS[1] is a format for programming assignments [1]. The format encoded in XML describes the conditions of the problem and a set of files with additional tests, test programs, etc. All these files are wrapped up in a ZIP file to facilitate deployment. A typical XML file contains the description of

1. the problem statement codified in the Simple Text Markup Language (STML) format—a simplified subset of HTML,
2. the format of input and output files and samples,
3. the restrictions on the input format, and
4. references for external resources such as tests generators, special checkers, plug-ins and solution programs.

2.2 FreeProblemSet

Freeproblemset[2] is a transport file format for the storage of all information about problems. It aims to provide free problems sets for managers of ACM/ICPC Online Jugdes by transporting data from one judge to another. The format uses XML to formalize the description of a programming problem. It includes information on problem, test data, special judger(optional) and answer(optional). Currently, the FPS format is supported by several online judge systems including HUSTOJ,[3] ACM-Server,[4] and Woj-land (OJ from Wuhan University).[5]

2.3 Mooshak Exchange Format

Mooshak[6] is a web-based competitive learning system originally developed for managing programming contests over the Internet [2]. Recently, it was upgraded to expose the evaluation functions as services. These services are expected to integrate in heterogeneous networks of e-learning types of systems, including the learning management systems (LMSs), integrated development environments (IDEs), and learning objects repositories (LORs) [3]. Despite the context where it is used, Mooshak has its own internal format to describe problems called Mooshak Exchange Format. MEF includes a XML manifest file referring several types of resources such as problem statements (e.g., PDF, HTML), image files, input/output test files, correctors (static and dynamic) and solution programs. The manifest also allows the inclusion of feedback and points associated to each test.

Currently, Mooshak is being used in several Universities worldwide to support learning activity. In the competitive context, it was used as the official evaluation system for the IEEE programming contests for some years.

1. http://imcs.dvgu.ru/cats/docs/format.html.
2. http://code.google.com/p/freeproblemset/.
3. http://code.google.com/p/hustoj/.
4. http://code.google.com/p/acm-server/.
5. http://code.google.com/p/woj-land/.
6. http://mooshak.dcc.fc.up.pt/.

TABLE 1
Textual Information Facet

Feature	CATS	FPS	MEF	PEF
Multilingual	-	-	X	X
HTML format	X	X	X	X
$\LaTeX$format	-	-	X	-
Image	X	X	X	X
Attach files	X	-	-	X
Description	X	X	X	X
Grading	-	-	-	-
Samples	-	-	-	-

2.4 Peach Exchange Format

Peach[7] is a system for the presentation, collection, storage, management, and evaluation (automated and/or manual) of assignments. The Peach exchange format [4] is a specific format for programming task packages used in Peach. Peach task packages are stored in a directory tree with a predefined structure. Currently, Peach is being used by the Eindhoven University of Technology.[8]

2.5 Synthesis

Several approaches can be found in literature [4], [1], [5] to evaluate the expressiveness of programming assignments formats. This section synthesizes the formats described previously according to the model proposed by Verhoeff. This model describes conceptually the notion of a task package as an unit for collecting, storing, archiving, and exchanging all information concerning with a programming task. The choice of the Verhoeff model over the alternatives is due to its more comprehensive coverage of the required features. This model organizes the programming exercise data in five facets:

1. Textual information—programming task human-readable texts,
2. Data files—source files and test data,
3. Configuration and recommendation parameters—resource limits,
4. Tools—generic and task-specific tools, and
5. Metadata—data to foster the exercises discovery among systems.

The *extual information facet* (Table 1) accommodates all the information (or pointers to such information) that the exercise author wants to offer to students or contestants

7. http://peach3.nl.
8. http://peach.win.tue.nl/.

TABLE 2
Data Files Facet

Feature	CATS	FPS	MEF	PEF
Solution	X	X	X	X
Skeleton	-	-	-	-
Multi-language	X	X	-	X
Tests	X	X	X	X
Test groups	X	-	-	X
Sample tests	-	X	-	-
Grading	X	-	X	X
Feedback	-	-	X	-

TABLE 3
Configuration and Recommendation Parameters Facet

Feature	CATS	FPS	MEF	PEF
Compiler	-	-	-	-
Executer	-	-	-	-
Memory limit	X	X	-	X
Size limit	-	-	-	-
Time limit	X	X	-	-
Code lines	-	-	-	-

TABLE 4
Tools Facet

Feature	CATS	FPS	MEF	PEF
Compiler	-	-	-	X
Test gen.	X	-	-	-
Feedback gen.	-	-	X	-
Skeleton gen.	-	-	-	-
Checker	X	-	-	X
Corrector	-	-	X	-
Library	X	-	X	X

TABLE 5
Metadata Facet

Feature	CATS	FPS	MEF	PEF
exercise	X	X	X	X
author	X	-	-	X
event	-	X	-	X
keywords	-	-	X	X
license	-	-	-	-
platform	-	-	-	X
management	-	-	-	X

about the exercise. It includes, for instance, the exercise challenge, background information, grading information and input/output samples. These texts may be available in several languages and formatted in plain text or other open format standards such as HTML or LATEX. Since this data are encoded in flexible text data formats, they can be transformed into various (open) presentation formats such as PDF. Other texts (e.g., grading information and samples) should ideally be generated from the evaluation data.

Besides human-readable texts, a programming exercise can also include several other files, in both text or binary format. The *data files facet* (Table 2) covers the following files: program and skeleton source files (ideally with support for multiple programming languages), input/output test data (ideally with support for grouping and multiple visibility mode), feedback files associated with a specific test case and others files.

The description of a programming exercise can also include parameters related with the submission, compilation, and execution of the student attempts to solve the exercise. These parameters can be organized in terms of *configuration and recommendation* (Table 3). The former deals with the configuration of compiler and linkers such as the compilation line of source files and associated parameters. The latter includes recommendations usually expressed in terms of resource limits such as the size and number of lines of a submission, compilation and execution time and memory limits. These recommendations depend on the actual platform used for evaluation runs. Thus, platform information should be associated for a more accuracy control.

When creating, solving, or evaluating an exercise, several software tools are needed such as editors, compilers, libraries, linkers, test, and feedback generators, input/output format checkers, evaluators, etc. The **tools facet** (Table 4) covers the support of the formats either by referencing these tools or by formalizing data that can be used as input for these tools.

The *metadata facet* (Table 5) comprises all the data that provide useful information on the exercise for classification and discovery purposes. There are several types of metadata that can be included in a programming exercise, such as exercise metadata (exercise type, keywords, difficulty level), authors metadata (name, contact), event metadata (name, type, local, date, number of participants, etc.), solver metadata (platform, operating system, etc.) and management metadata (status of development, version information, etc.).

This study confirms the disparity of programming exercise formats highlighting both their differences and their similar features. This heterogeneity hinders the interoperability among the typical systems found on the automatic evaluation of exercises. Rather than attempting to harmonize the various specifications, a pragmatic solution is to provide a service for exercises formats conversion. The next section presents an approach to extensible format conversion and details the features of PExIL—the pivotal format in which the conversion is based—and of an abstract view of the conversion service functions.

3 Exercise Format Conversion

Data conversion is the conversion of computer data from one format to another. Data conversion can typically occur based on two approaches: *direct* or *pivotal* [6].

In the direct conversion, the converter receives the input format and apply transformations according to the output format. One apt example is the transcoder[9] developed by JISC Centre for Educational Technology and Interoperability Standards (JISC CETIS) that enables the conversion between e-learning content packages. It addresses conversions between the most common e-learning content formats in use such as IMS Content Packaging (IMS CP), Sharable Content Object Reference Model (SCORM), and IMS Common Cartridge (IMS CC).

The pivotal conversion is based on an intermediate format allowing any source format to be converted to its target. Compared with the previous approach, this pivotal encoding approach provides several advantages such as

9. http://purl.oclc.org/NET/transcoder.

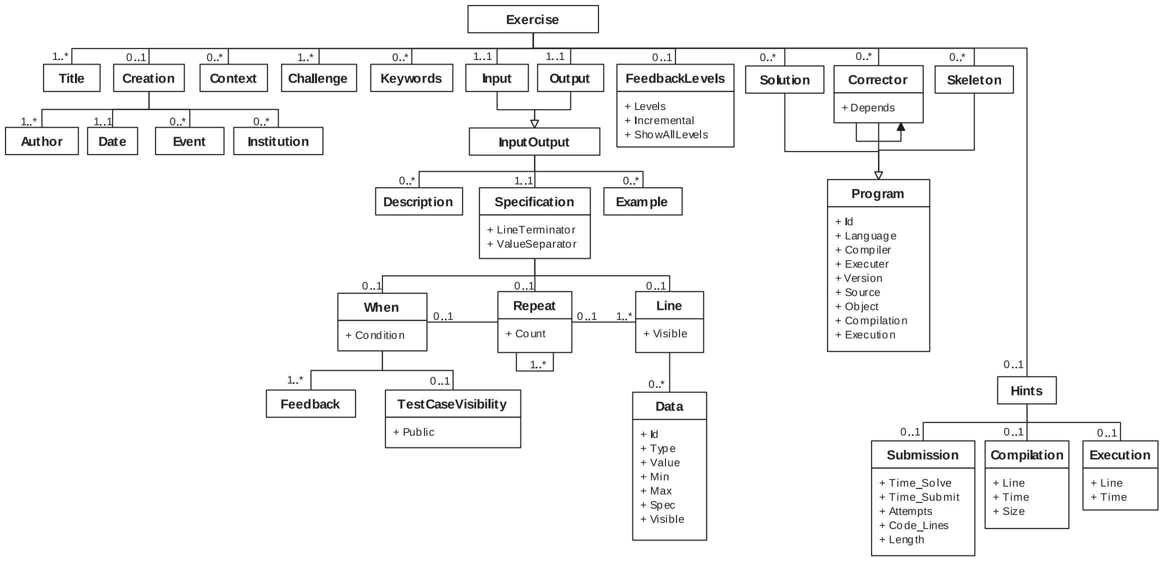

Fig. 1. PExIL data model.

manageability. A data format converter would have to support a huge number of mappings for all the permutations of the data formats supported. Using a intermediate format scales down this number because only one mapping is needed for each format supported. Pivotal conversion is often used in several areas. For instance, Office applications use the OpenDocument file format as a pivot for the conversion between office file formats. Despite its use, the pivotal approach is also subject of criticism [6] such as the augment of noise due to the propagation of the translation errors and inaccuracy or lost of data due to the conversion between formats that are conceptually different.

3.1 Approach

Based on the current conversion approaches, it was decided to use the pivotal approach regarding the exercise formats conversion.

Using a pivot format reduces drastically, the number of permutations needed from $n \times (n-1)$ to $2 \times n$, where n is the number of formats supported.

Since one of the design requirements of the converter is its extensibility, it is important to simplify the support for new formats.

3.2 PExIL—a Pivot Exercise Format

The PExIL [7] is a XML dialect that aims to consolidate all the data required in the programming exercise life cycle.

The life cycle comprises several phases. In the *creation phase* the content author should have the means to automatically create some of the resources (assets) related with the programming exercise such as the exercise description and test cases and the possibility to package and distribute them in a standard format across all the compatible systems (e.g., learning management systems, learning objects repositories. In the *selection phase,* the teacher must be able to search for a programming exercise based on its metadata from a repository of learning objects and store a reference to it in a learning management system. In the *presentation phase,* the student must be able to choose the exercise description in its native language and a proper format (e.g., HTML, PDF). In the *solving phase*, the learner should have the possibility to use test cases to test his attempt to solve the exercise and the possibility to automatically generate new ones. In the *evaluation phase*, the evaluation engine should receive specialized metadata to properly evaluate the learner's attempt and return enlightening feedback.

The PExIL definition is formalized through the creation of a XML Schema depicted in Fig. 1. The schema comprises three groups of elements:

- *Textual*—elements with general information about the exercise to be presented to the learner (e.g., title, date, challenge).
- *Specification*—elements with a set of restrictions that can be used for generating specialized resources (e.g., test cases, feedback).
- *Programs*—elements with references to programs as external resources (e.g., solution program, correctors) and metadata about those resources (e.g., compilation, execution line, hints).

The expressiveness of PExIL was validated according to the multifacet model proposed by Verhoeff. Table 6 shows the PExIL elements coverage.

Textual elements group (G1) in PExIL can be used in several phases of the programming exercise life cycle: in the selection phase as exercise metadata to aid discoverability and to facilitate the interoperability among systems (e.g., LMS, IDE); in the presentation phase as content to be present to the learner (e.g., exercise description); in the

TABLE 6
PExIL Coverage on the Verhoeff Model

Verhoeff / PExIL	G1	G2	G3
Text	X	X	-
Data files	X	X	X
Parameters	-	-	X
Tools	-	-	X
Metadata	X	-	X

solving phase, as skeleton code to be included in the student's project solution.

Specification elements group (G2) in PExIL can be used in several phases of the programming exercise life cycle by 1) the content author to automatically generate an input and output test example to be included on the exercise description for presentation purposes, 2) the learner to automatically generate new test cases to validate his attempt, and 3) the Evaluation Engine to evaluate a submission using the test cases.

Program elements group (G3) contains references to program source files as external resources (e.g., solution program, correctors) and metadata about those resources (e.g., compilation, execution line, hints). These resources are used mostly in the evaluation phase of the programming exercise life cycle to allow the Evaluation Engine to produce an evaluation report of a students' attempt to solve an exercise.

This analysis asserts the total coverage of PExIL elements based on the Verhoeff model and guarantees PExIL as a good candidate to act as a pivot format for a conversion service of programming exercises formats. More details about PExIL can be found in literature [7].

3.3 Abstract Functions

This section describes the generic capabilities of a Converter service expressed in terms of their behaviors, without prescribing how to make them operational. A service of this genre is responsible for the conversion of programming exercises formats. It supports five functions: `GetFormats`, `Convert`, `ConvertSets`, `ConvertFromSet`, and `ConvertToSet`. The following section details the five service functions.

3.3.1 The `GetFormats` Function

The `GetFormats` function provides the requester with a list of all the formats supported by the service. In order to support a format, the service must implement the format conversion from and to the pivotal format. In this function, the request may omit the parameter or have one representing the input format. In the former, the response returns a list of all formats of the converter. In the latter, the response includes only the formats that can be converted from the input format given as a parameter. In the response, each format is described by its name and a list of formats that can be used as outputs and its corresponding URL paths. This will allow client systems to automate the conversion request based on the available formats returned by the GetFormats function.

3.3.2 The `Convert` Function

The `Convert` function performs the conversion of a given programming exercise from an input format to an output format. The function includes three parameters: 1) the format of the exercise to convert, 2) the conversion output format, and 3) a reference (URL based) of the exercise to convert. The function returns an archive with its contents complying with the output format.

3.3.3 The `ConvertSets` Function

The `ConvertSets` function converts a set of programming exercises from an input format to an output format. This function is useful with formats that deal only with exercise sets, or when they are convenient for a particular task. In a competitive setting, a contest manager may want to feed a new contest based on a problem set of a existing programming contest. In a more pedagogical setting, a teacher may want to use in the classroom a set of problems from an external source. In both cases, the request parameters are similar to the previous function. The function returns an archive with a set of exercises complying the given output format.

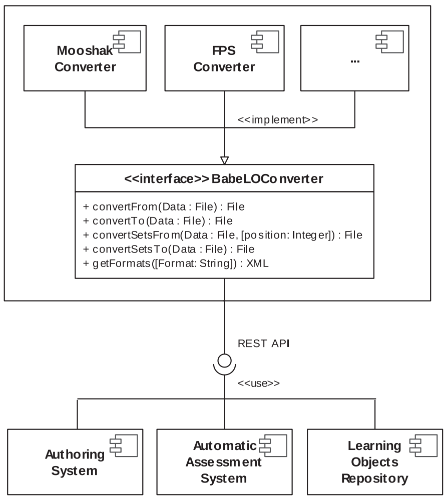
Mooshak Converter
FPS Converter
...
<<implement>>
<<interface>> BabeLOConverter
+ convertFrom(Data : File) : File
+ convertTo(Data : File) : File
+ convertSetsFrom(Data : File, [position: Integer]) : File
+ convertSetsTo(Data : File) : File
+ getFormats([Format: String]) : XML
REST API
<<use>>
Authoring System
Automatic Assessment System
Learning Objects Repository

Fig. 2. BabeLO architecture.

3.3.4 The `ConvertFromSet` Function

The `ConvertFromSet` function converts a single programming exercise from a set of exercises described with the input format to a single exercise in the output format. The exercise to convert is given by its position within the set as an additional parameter. The function returns an archive with a single exercise complying the given output format.

3.3.5 The `ConvertToSet` Function

The `ConvertToSet` function converts a single programming exercise complying with the input format to a single collection in the output format. The function returns an archive containing a collection with only one exercise in the given output format.

4 BabeLO

This section details the design and implementation of BabeLO on the Tomcat servlet container. Fig. 2 shows the architecture of the BabeLO service described by an UML component diagram.

In order to allow client systems to use the conversion features of BabeLO, its core functions are exposed as services using a REST web service interface. The REST implementation uses Jersey[10]—the reference implementation of JAX-RS (the Java API for RESTful web services). In Table 7, each function is associated with the corresponding operation in the REST flavor.

10. http://jersey.java.net.

TABLE 7
BabeLO REST API

Functions	REST API
GetFormats	convert[/in]
Convert	convert/in/out/ex
ConvertSets	convertsets/in/out/ex
ConvertToSet	converttoset/in/out/ex
ConvertFromSet	convertfromset/in/out/pos/ex

The **GetFormats function** returns a list of the formats supported by the service. The syntax of the GET HTTP request is

$$\texttt{GET/convert}[\texttt{/inputFormat}] > \texttt{BRL}.$$

The response is formalized using the BabeLO Response Language (BRL). The *Convert function* converts a given programming exercise from an input format to an output format. The syntax of the GET HTTP request is

$$\texttt{GET/convert/in/out/ex} > \texttt{Archive}.$$

The function includes three parameters: in—the format of the exercise to convert, out—the conversion output format, ex—reference (URL based) of the exercise to convert. The function returns an archive with its contents complying the output format.

Fig. 3 shows a typical conversion between two formats: Mooshak and FPS. BabeLO uses the `convertFrom()` method of `MooshakConverter` to transform the programming exercise from mooshak format to the PExIL format. Then, it is uses the `convertTo()` method of the `FPSConverter` to transform the exercise from the PExiL format to the FPS format.

Being an extensible converter, BabeLO can be augmented with other classes implementing a specific Java interface defining the `convertFrom()` and `convertTo()` methods mentioned above, as well as other similar methods to handle collections.

5 Evaluation Results

This section focus the evaluation the effectiveness and efficiency of the proposed approach. It is based on two actual uses of BabeLO: 1) to relocate exercises to a different repository and 2) to use an evaluation engine in a network of heterogeneous systems.

5.1 Case Study 1: Repositories Exchange

In the first case study, the BabeLO service is used to create a repository collection based on a problem set of an existing programming contest. Fig. 4 depicts the interconnection of BabeLO with two other components to achieve this purpose.

In order to measure the efficiency of the BabeLO Service, two repositories were used: an installation of crimsonHex

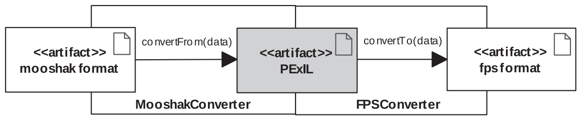

Fig. 3. The convert function.

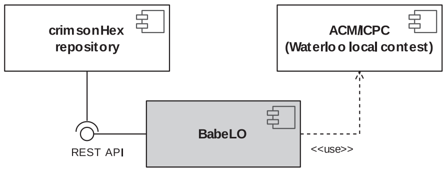

Fig. 4. Case study 1—exchanging exercises between repositories.

[8]—a repository of programming exercises created in the scope of an European project called Edujudge [9]—and the ACM/ICPC contest system of the University Waterloo. The exercises were copied from the Waterloo repository to crimsonHex installation. The former supports the IMS Common Cartridge (IMS CC) format for describing the exercises. The later uses the FPS format to describe the contest problem sets.

The time to convert collections with different numbers of exercises was measured and compared with a standard download, i.e., using the time of a HTTP GET as benchmark. The average size of the exercises used in this experiment was 12 KB and time was measured in milliseconds. Table 8 summarizes the results and presents the overheads introduced by BabeLO.

The main conclusion is that BabeLO introduces an overhead of a factor of 10 when converting an exercise. This overhead increases with the size of collections when taking as benchmark the direct download of a collection. This fact is understandable since BabeLO has to process each exercise, one-by-one, and takes little advantage of collections. Analyzing the columns for direct download one notices that, although the time for downloading a collection increases with the size of the collection, the average time per exercise reduces significantly. This is due to the fact that a collection download avoids the time of establishing a connection to the repository for each exercise, which is a significant part of the download. Analyzing the columns for the BabeLO download, one notices something similar; the time per exercise is reduced but it is not as expressive as in the direct download. This is due to the fact that the conversion time is a larger share of the overall time and saving the time of establishing connection has less impact.

At first sight, an overhead factor of 10 may seem impracticable for on-the-fly format conversion. However, it should be noted that exercises only need to be downloaded from a remote site for the first time they are used. As in any HTTP based system, a cache can and should be used to improve overall efficiency [10], as described in the next section.

5.2 Case Study 2: Automatic Assessment

In this section, the effectiveness of BabeLO is validated with an evaluation engine in a network of heterogeneous systems.

TABLE 8
Download Time (ms) and Overhead of BabeLO

#	Direct		BabeLO		Over
ex.	Coll.	P.Ex.	Coll.	P.Ex.	head
1	7	7	78	78	10.1
10	22	2.2	693	69.3	30.5
100	145	1.45	5,970	59.7	40.2
1000	1,395	1.395	56,348	56.3	39.4

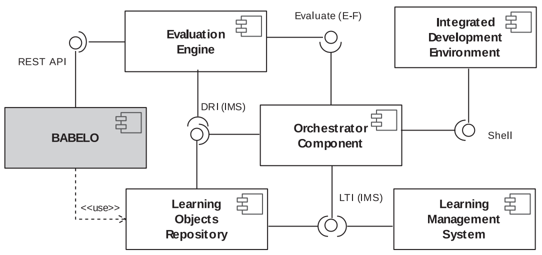

Fig. 5. Case study 2—automatic assessment.

The architecture depicted by UML component diagram in Fig. 5 is composed by the following systems and tools:

1. Learning Objects Repository to store/retrieve exercises.
2. Evaluation Engine to evaluate students' exercises.
3. Learning Management System to present the exercises to students.
4. Integrated Development Environment to code the exercises.

To start using this network, the teacher sets a number of activities (exercises) in the LMS by selecting a set of relevant programming exercises from the LOR. Then, the learner tries to solve the exercises assigned by the teacher using the pivot component (launched by the LMS). The pivot component recovers the exercise description from the LOR and shows it to the student. After coding the program in the IDE, the student uses the pivot component to send an attempt to the EE. The first time that the user send an attempt to the EE, the engine uses the BabeLO service to convert the exercise from its original format to the Mooshak internal format and then it caches the BabeLO response for further use. After that, the EE evaluates the student's attempt and returns an evaluation report. The student may submit repeatedly, integrating the feedback received from the EE. In the end, the EE sends a grade to the pivot component and reports the LO usage data back to the LOR.

In order to validate the efficiency of this network, an experiment was conducted at ESEIG—a school of the Polytechnic Institute of Porto. Students from the first of the Algorithmic and Programming course participated in a two-month experiment (six classes) with several assignments comprising 18 exercises. From a extensive list of results, it should be stressed that the number of times that the EE accessed to BabeLO (18) comparing it with the number of attempts submitted by students to solve the exercises (819). These results show two things: 1) the EE cache mechanism is working as expected and 2) the BabeLO service made successfully conversions since no more requests were made.

6 Conclusions

The goal of the research presented in this paper is to promote the interoperability among systems that participate in the automatic assessment of programs. The heterogeneity of theses systems is the least of the obstacles to interoperability. The major problem is the lack of a standard for assessment items, i.e., learning objects, understood by all the systems involved in assessment. The proposed approach is a service to convert programming exercises formats on-the-fly.

The contribution of this research is twofold, the service abstract definition and its actual implementation. The abstract definition comprehends the modular design of the conversion service based on a pivot format, the data model of this pivotal format and the definition of the service functions. The actual implementation of BabeLO is a web service implemented in a Java *servlet* container supporting a few formats and with the ability to be extended to new formats.

The effectiveness and efficiency of the proposed approach were evaluated with two case studies presented in this paper.

The BabeLO service is an open-source project and was already deployed in networks of e-learning systems that require on-the-fly conversion among learning object formats.

Currently, BabeLO supports only the conversion of programming exercises from and to Mooshak and FPS formats and this is its main drawback. As future work, the authors will extend the BabeLO compliance to other exercise formats. Also rather than maintaining a single installation of the conversion service, the authors intend to distribute the source code of BabeLO and promote installations of this service at each site. With this approach, they expect both to reduce the fear of unauthorized copy of learning objects, and to encourage users of other formats to contribute with code to convert to and from the pivotal format.

References

[1] A. Klenin, "Common Problem Description Format: Requirements," *Proc. ACM-ICPC World Final Competitive Learning Inst. of Symp.*, 2011.
[2] J.P. Leal and F.M.A. Silva, "Mooshak: A Web-Based Multi-Site Programming Contest System," *Software—Practice & Experience*, vol. 33, no. 6, pp. 567-581, 2003.
[3] J.P. Leal and R. Queirós, "A Programming Exercise Evaluation Service for Mooshak," *Proc. ACM-ICPC World Final Competitive Learning Inst. of Symp.*, 2011.
[4] T. Verhoeff, "Programming Task Packages: Peach Exchange Format," *Int'l J. Olympiads Informatics*, vol. 2, pp. 192-207, 2008.
[5] S.H. Edwards, J. Börstler, L.N. Cassel, M.S. Hall, and J. Hollingsworth, "Developing a Common Format for Sharing Programming Assignments," *SIGCSE Bull.*, vol. 40, no. 4, http://dx.doi.org/10.1145/1473195.1473240, pp. 167-182, 2008.
[6] T. Tsunakawa, "Pivotal Approach for Lexical Translation," PhD dissertation, Univ. of Tokyo, 2010.
[7] R. Queirós and J.P. Leal, "PExIL: Programming Exercises Interoperability Language," *XML: Aplicacoes e Tecnologias Associadas (XATA)*, 2011.
[8] J.P. Leal and R. Queirós, "CrimsonHex: A Service Oriented Repository of Specialised Learning Objects," *Proc. Int'l Conf. Enterprise Information Systems*, pp. 102-113, May 2009.
[9] E. Verdú, L.M. Regueras, M.J. Verdú, J.P. Leal, J.P. de Castro, and R. Queirós, "A Distributed System for Learning Programming On-Line," *Computers and Education*, vol. 58, pp. 1-10, Jan. 2012.
[10] R. Fielding and R. Taylor, "Principled Design of the Modern Web Architecture," *Proc. Int'l Conf. Software Eng.*, pp. 407-416, 2000.

Ricardo Queirós is currently working toward the PhD degree in computer sciences in the Faculty of Sciences of the University of Porto. He is an assistant professor at the School of Industrial Studies and Management in Vila do Conde. His scientific activity is related with e-learning standards and interoperability. He is an associated member of the Center for Research in Advanced Computing Systems.

José Paulo Leal is an assistant professor at the Department of Computer Science of the Faculty of Sciences of the University of Porto and an associate researcher of the Center for Research in Advanced Computing Systems. His main research interests are e-learning system implementation, structured document processing, and software engineering. He has a special interest on automatic exercise evaluation, in particular in the evaluation of programming exercises, on the semantic web, and on web adaptability.