Journal of Universal Computer Science, vol. 18, no. 11 (2012), 1454-1482
submitted: 30/11/11, accepted: 15/5/12, appeared: 1/6/12 © J.UCS

Orchestration of E-Learning Services for Automatic Evaluation of Programming Exercises

Ricardo Queirós
(CRACS & INESC-Porto LA & DI-ESEIG/IPP, Porto, Portugal
ricardo.queiros@eu.ipp.pt)

José Paulo Leal
(CRACS & INESC-Porto LA, Faculty of Sciences, University of Porto, Portugal
zp@dcc.fc.up.pt)

Abstract: Managing programming exercises require several heterogeneous systems such as evaluation engines, learning objects repositories and exercise resolution environments. The coordination of networks of such disparate systems is rather complex. These tools would be too specific to incorporate in an e-Learning platform. Even if they could be provided as pluggable components, the burden of maintaining them would be prohibitive to institutions with few courses in those domains. This work presents a standard based approach for the coordination of a network of e-Learning systems participating on the automatic evaluation of programming exercises. The proposed approach uses a pivot component to orchestrate the interaction among all the systems using communication standards. This approach was validated through its effective use on classroom and we present some preliminary results.

Keywords: e-Learning; Interoperability; Service Oriented Architectures.
Categories: D.3, L.1.2, L.3.0, L.3.6

1 Introduction

Learning programming involves more than the knowledge of programming language syntax and algorithms. It requires the development of skills in problem understanding, problem-solving, debugging strategies, and unit testing, among others. These skills are acquired by the resolution, on a regular basis, of programming exercises. However, programming exercises, like in any type of exercise, must be evaluated and students must receive feedback on their work. Otherwise students may consolidate false beliefs based on the exercises where they did not achieve the intended goal [Truong, 07], [Wang, 08].

Creating, managing, accessing, evaluating and proving feedback on a large number of exercises, covering all the points in the curricula of a programming course, in classes with large number of students, can be a daunting task without the appropriated tools working in unison. The best tools for this job cannot be found on a single system. They are found on the best-of-bread for each category; whether it is tool for developing programs or a tool to automatically evaluate them.

The architecture of e-Learning platforms is in fact moving away from centralised systems towards decentralised networks of heterogeneous systems [Leal, 10]. There are several types of systems participate in those networks ranging from existing e-

Learning systems to supporting services for specialized tasks. These systems and services may participate in various learning processes that can be easily reconfigured to meet changing requirements and demands. We are particularly interested in networks of e-Learning systems providing services related to the automatic evaluation of programming exercises. Networks of this kind include heterogeneous systems such as Learning Management Systems (LMS), Evaluation Engines (EE), Learning Objects Repositories (LOR) and Integrated Development Environments (IDE). These types of systems have a completely different nature. Some expose their functions as web services, such as LOR or EE. Others have their own web interfaces for students and teachers, such as LMS. Some, as is the case with IDE - that we use as an exercise resolution environment - were not even designed to interact in the e-Learning realm and must be extended for that purpose. Modelling a network with such heterogeneity is rather complex and, at the same time, challenging.

The main goal of the research described in this paper is to study the applicability of e-Learning specifications for designing and implementing an integration model for a network of services for programming languages learning. A network of this kind will help students to solve programming exercises and to receive automatic feedback on the quality of their solutions. The paper is based on previous work [Leal, 11] where we explore the possibility of embedding a pivot component in an LMS that acted as an exercise resolution environment and coordinated the communications between the LMS and a set of web services. In this work, the novel idea is the integration of an IDE in the e-Learning network, introducing the students to a programming tool that they will continue to use latter on their active life.

The remainder of this paper is organized as follows: section 2 analyses e-Learning interoperability on several facets such as frameworks and specifications. Section 3 presents the overall architecture of a network of e-Learning systems participating in the resolution and automatic evaluation of programming exercises. We highlight the interoperability features of the network based on content and communication specifications. In the former, we describe the approach used to define programming exercises as learning objects. In the latter we enumerate and explain the communication specifications used on the network to ensure a standard communication among all the network components. In section 4 we start by describe an early initiative that based our current work. Then, we present our current approach by identifying the systems used in the current network and how they relate to each other. In the end of this section we detail the effective use of this work on classroom and provide usage data to validate the feasibility of this network regarding the interoperability efforts made. Finally, the paper summarizes and point out the main contributions of this work while revealing some open challenges for research on this topic.

2 E-Learning interoperability

This section presents the state-of-the-art in e-Learning systems interoperability from different perspectives, starting with generic frameworks and proceeding with specific types of e-Learning systems involved in this research, such as automatic assessment systems, learning management systems and learning object repositories.

2.1 Frameworks

The principles of Service Oriented Architecture (SOA) are an established trend in software design and development and there are several initiatives to adapt them to e-Learning [Leal, 10]. These initiatives, commonly named eLearning frameworks, have a same common goal: to provide flexible learning environments for learners worldwide. While e-Learning frameworks are general approaches for e-Learning system integration, several authors proposed service oriented approaches specifically targeted to some of the previously mentioned types of e-Learning systems, such as the LMS and the LOR. Regarding the former, there are several references in the literature to middleware components for LMSs integration in SOA based e-Learning systems.

Apostolopoulos proposes a middleware component [Apostolopoulos, 03] to address the lack of integration of e-Learning services. In this approach the e-Learning components are implemented as agents maintained in a local management information base, and can communicate with the agent manager through the Simple Network Management Protocol (SNMP).

Casella developed an architecture based on a middleware component [Casella, 07] that uses Web Services to integrate different software components and to improve interoperability among different systems. The middleware component enables the student learning process traceability since it has been developed to be compliant with Sharable Content Object Reference Model (SCORM).

2.2 Assessment systems

Service-orientation has been proposed also specifically for assessment systems. Al-Smadi presents a SOA [Al-Smadi, 10] for a generic and flexible assessment system with cross-domain use cases. There are several other assessment systems that support integration with specific LMS by providing the evaluation results on the LMS grade book. AutoGrader, CTPraticals and EduComponents are integrated with specific LMS, respectively, CascadeLMS, Moodle and Plone. All these approaches have in common the need of a modification of LMS for each specific vendor, with the implementation of a new module or building block. Alstes developed an assessment system called Verkkoke [Alstes, 07] that does not depend on a specific LMS and can be integrated on any LMS that supports the SCORM specification. To the best of the authors' knowledge there are no more references in the literature to the use of common standards supported by the major LMS vendors as a means to integrate the LMS in a service oriented network of e-learning systems participating on automatic evaluation of programming exercises.

2.3 Learning Management Systems

SOA has been adopted in several e-Learning projects and from different perspectives within the e-Learning domain. An extensive review [Riad, 09] reveals that services based learning systems promotes integration, interoperability, and scalability and they are recommended in order to fulfil the demanding e-Learning systems architectural requirements. There have been several initiatives to join LMS to other applications. Those initiatives can be organized in three groups [Conde, 11]:

- LMS defined from scratch based on service-oriented architectures [Al-Smadi, 10] and [Casquero, 10];
- Inclusion of web services layers within the LMS infrastructure [Conde, 10], [de-la-Fuente-Valentín, 08] and [Severance, 08];
- Support to interoperability specifications [Leal, 11].

Although web services open the LMS to other applications, they do not provide a standard solution among the broader range of LMS in the Web. In order to provide a real integration approach, interoperability specifications (e.g IMS LTI, OKI) must be taken into account [Severance, 08]. This would be the case for applications that require graphical interaction in order to be used. In this scope, IMS launched in 2011 the IMS Learning Tools Interoperability (LTI) [IMS LTI, 11] aiming to provide an uniform standards-based extension point in LMS allowing remote tools and content to be integrated into LMSs. The main goal of the LTI is to standardize the process for building links between learning tools and the LMS. The LTI has 3 key concepts: the Tool Provider, the Tool Consumer and the Tool Profile. The *tool provider* is a learning application that runs in a container separate from the LMS. It publishes one or more tools through tool profiles. A *tool profile* is an XML document describing how a tool integrates with a tool consumer. It contains tool metadata, vendor information, resource and event handlers and menu links. The *tool consumer* publishes a Tool Consumer Profile (XML descriptor of the Tool Consumer's supported LTI functionality that is read by the Tool Provider during deployment), provides a Tool Proxy Runtime and exposes the LTI services. A LTI subset was launched in 2010 - called IMS Basic LTI – to expose a single (but limited) connection between the LMS and the tool provider. In this subset, there is no provision for accessing run-time services in the LMS and only one security policy (OAuth protocol) is supported.

2.4 Learning Object Repositories

Some interoperability efforts [IMS DRI, 03], [Eap, 04], [McCallum, 06], [Ternier, 10] address the interoperability issues on a specific component: the repository. A repository of learning objects can be defined as a 'system that stores electronic objects and meta-data about those objects' [Holden, 04]. The need for this kind of repositories is growing as more educators are eager to use digital educational contents and more of it is available. Several surveys on repositories [RSP, 10][Holden, 04] [Neven, 02] showed that the interoperability is a major issue that is not completely addressed by the existing systems. The IMS DRI specification appeared to overcome this issue. It was created by the IMS Global Learning Consortium (IMS GLC) and provides a functional architecture and reference model for repository interoperability. The IMS DRI provides recommendations for common repository functions, namely the submission, search and download of LO. It recommends the use of web services to expose the repository functions based on the Simple Object Access Protocol (SOAP) protocol, defined by W3C. Despite the SOAP recommendation, other web service interfaces can be used, such as, Representational State Transfer (REST) [Fielding, 05].

3 Network architecture

In this section we present the overall architecture of a network of e-Learning systems participating in the resolution and automatic evaluation of programming exercises. The architecture [Fig. 1] is composed by the following components:

Learning Objects Repository (LOR) to store and retrieve exercises;
Evaluation Engine (EE) to evaluate students' exercises;
Learning Management System (LMS) to present the exercises to students;
Integrated Development Environment (IDE) to code the exercises.
Pivot Component to mediate the communication among all components.

In order to fulfil this goal, the integration of the pivot component with the other systems must rely on content and communication standards. Using content and communication standards we can abstract the use of specific systems for each type of system. For instance, we can use on this network any repository as long it supports the IMS CC specification to formalize the description of programming exercises and it implements the IMS DRI specification for communication with other services.

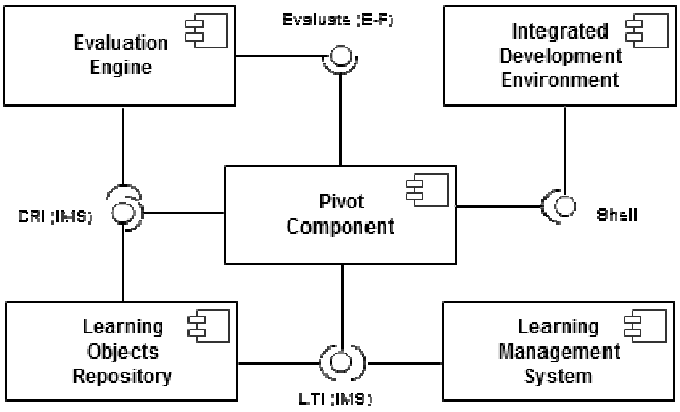

Figure 1: UML components diagram of the repository.

3.1 Pivot component

The pivot component coordinates the communication among all the components of the network from the LMS where students start the instruction until the IDE where students solve the exercises. This pivot component orchestrates all the communications within a single instance of the network. Since it is distributed over each network use (they are replicated on the users' machines for each instance of the network) this approach prevents any single-point-of-failure issues that might occur.

This pivot component will typically interface with two types of user profiles: teachers and students. In order to author and deploy a programming exercise using the pivot component teachers must perform the following three tasks:

- Create programming exercises;
- Deploy programming exercises in a repository;
- Configure programming activity in LMS.

In order to solve programming exercises using the pivot component students performs the following two tasks:

- Select an activity in the LMS;
- Execute the activity using the IDE and the interface offered by the pivot component.

In this scenario, 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 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. This last task will provide data for future adaptability services that will adjust the presentation order in accordance with the effective difficulty of programming exercises (not the difficulty stated on the LO) and the needs of a particular student.

This architecture can be reused in other contexts where automatic evaluation is needed to assess sutdents performence, where a place to store exercises is required and even where a tool to track students and gather their grades is mandatory. In this case, the pivot component needs to be reimplemented while preserving the interoperability specifications guidelines detailed in this paper.

Regarding the evaluation of languages, this approach is not confined strictly to programming languages, and other languages can be used such as query languages (e.g. SQL), modeling languages (e.g. UML) and user interfaces (e.g. HTML). However, this architecture is not specific for the programming domain. This model of combining specialized services can be extended to competitive learning in other domains such as business training, for instance. In this domain teachers use business simulation games to improve the strategic thinking and decision making skills students in particular areas (e.g. finances, logistics, and production). Through these simulations students compete among them, as they would in a real world companies. A business simulation service fulfils a role similar to that of the EE in programming exercises and it also requires a repository containing specialized LO describing simulations.

3.2 Data Model

The concept of Learning Object (LO) is crucial for the standardization on e-Learning. The latest standard for LO is the IMS Common Cartridge (IMS CC). The IMS Common Cartridge specification defines an open format for the distribution of rich web-based content. Its main goal is to organize and distribute digital learning content and to ensure the interchange of content across any Common Cartridge conformant

tools. In its latest version (1.2) released in October 2011 [IMS CC, 11], the IMS CC package organizes and describes a learning object based on two levels of interoperability: content and communication. At the content level, the IMS CC distinguishes two types of resources:

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

In the communication level the cartridge describes how the target tool of the cartridge (usually a LMS) should communicate with other remote web applications using the IMS Basic LTI specification. Both levels enhance the interoperability of the cartridge among a network of eLearning systems. In this scope a new IMS CC specification feature is introduced to support authorization at two levels: either the whole cartridge can be protected or individual resources can be protected. In the following subsections we detail the most important elements of the CC content hierarchy.

A view of the CC manifest is depicted in [Fig. 2]. The manifest is composed by four sections: metadata, organizations, resources and authorizations. The Metadata section is used to store the cartridge metadata restricted to a loose binding of IEEE Learning Object Metadata (LOM) elements based on the Dublin Core (DC) specification. The Organization section will be used to represent the Common Cartridge Folder content type as a structural approach to organize content. The Resources section will be used to refer assets included in the cartridge. The authorizations section extends the manifest to protect the learning object (or a specific resource) as a whole.

In order to describe a programming exercise as a LO we developed an XML dialect called PExIL [Queirós, 11], standing for Programming Exercises Interoperability Language. The aim of PExIL is to consolidate in a single document all the data required in the programming exercise life-cycle, from when it is created to when it is graded, covering also the resolution, the evaluation and the feedback. PExIL documents can be used for authoring LOs containing programming exercises. The generation of an LO is based on a valid PExIL instance. The Generator tool uses as input a solution file and produces automatically several resources (e.g. exercise description, test cases and feedback files) described by a valid IMS CC manifest and wrapped up inside a ZIP file.

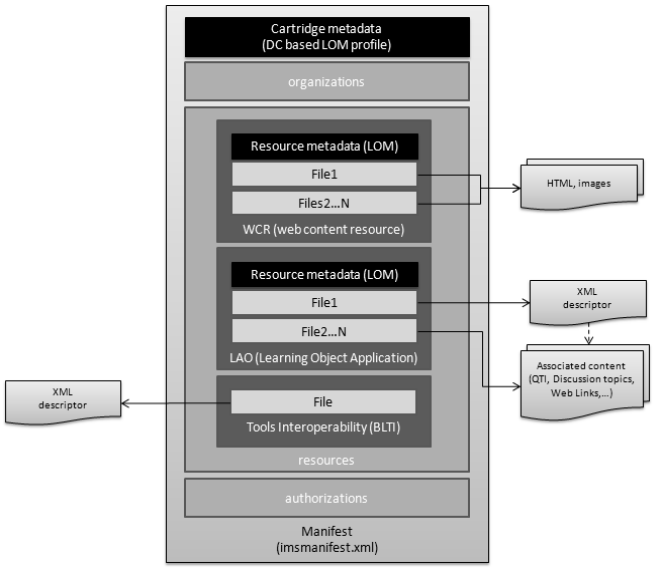

Figure 2: The structure of an IMS CC package.

3.3 Digital Repositories Interoperability (DRI)

A learning objects repository (LOR) is a system that stores electronic educational resources in a technology-mediated learning process. The number of LOR is growing rapidly [Neven, 02] as more educators are eager to use digital educational contents and more of it is available. At the same time users become more demanding and concerned with issues that are not completely addressed by the existing systems, such as interoperability. Moreover, for some specialized domains, such as computer programming, the existing standards are insufficient and need to be extended. In this sub-subsection we detail the communication features of crimsonHex, a LOR developed to support the requirements of evaluating engines.

The crimsonHex API exposes the functions of the repository to third party systems by adhering to the IMS DRI specification [IMS DRI, 03]. The IMS DRI specifies a set of core functions and an XML binding for these functions. In the definition of API of crimsonHex we needed to create new functions and to extend the XML binding with a Response Specification language. The complete set of functions of the API and the extension to the XML binding are also both detailed in this section.

To comply with standards, the IMS DRI recommends the implementation of core functions as web services. We chose to implement two distinct flavours of web services: SOAP and REST. SOAP web services are usually action oriented, mainly when used in Remote Procedure Call (RPC) mode and implemented by an off-the-shelf SOAP engine such as Axis. The web services based on the REST style are object (resource) oriented and implemented directly over the HTTP protocol, mostly to put and get resources (such as LOs and usage data). The reason to implement two distinct web service flavours is to promote the use of the repository by adjusting to

different architectural styles. The repository functions [Leal, 09] are summarized in [Tab. 1].

Function	SOAP	REST
Reserve	*URL getNextId()*	*GET /nextId > URL*
Submit	submit(URL loid, LO lo)	PUT URL < LO
Request	LO retrieve(URL loid)	GET URL > LO
Search	XML search(XQuery query)	POST /query < XQUERY > XML
Report	*Report(URL loid,LOReport rep)*	*PUT URL/report < LOREPORT*
Alert	RSS getUpdates()	GET /rss > RSS
Create	*XML Create(URL collection)*	*PUT URL*
Remove	*XML Remove(URL collection)*	*DELETE URL*
Status	*XML getStatus()*	*GET URL?status > XML*

Table 1: Core functions of the repository.

Each function is associated with the corresponding operations in both SOAP and REST web services interfaces. The lines formatted in italics correspond to the new functions added to the DRI specification, to improve the repository communication with other eLearning systems.

To describe the responses generated by the repository we defined a **Response Specification** as a new XML document type formalized in XML Schema. The advantage of this approach is to enable client systems to achieve more information from the server and be able to standardize the parsing and validation of the HTTP responses. [Fig. 3] depicts the elements of the new language and their types.

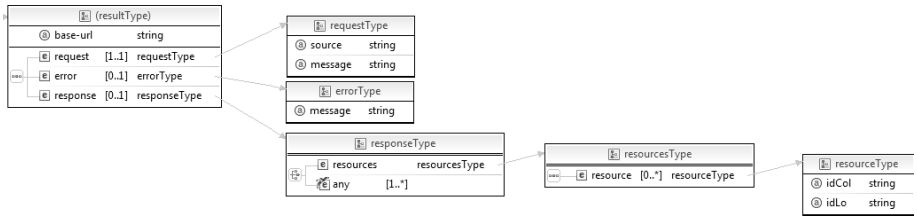

Figure 3: The Response specification schema.

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

- `base-url`– defines a base URL for the relative URLs in the response;
- `request` – contains the full request URL and an human readable request message;
- `error` - contains an error message - client systems will search for this element to verify the existence of errors;
- `response` – describes a successful execution of the function – it is composed by an human readable response message and, for some functions, by a

`resources` element that groups a set of resources defined individually in `resource` elements. A `resource` element contains an identification of the collection absolute path (attribute `idCol`) and an identification of the LO itself (attribute `idLo`).

In the remainder of this section we enumerate the Core functions of the repository, describing both the request and response data. For sake of simplicity we illustrate the requests using the REST interface since these can be used as command lines in a Linux system shell.

The **Register/Reserve** function requests a unique ID from the repository. We separated this function from Submit/Store in order to allow the inclusion of the ID in the meta-data of the LO itself. This ID is an URL that must be used for submitting or retrieving an LO. The producer may use this URL as an ID with the guarantee of its uniqueness and with the advantage of being a network location from where the LO can be downloaded. This action is performed, for instance, by sending a GET HTTP request to the server, as in the following example.

```
GET http://server/ch/lo?nextId > URL
```

The HTTP response includes an XML file complying with the Response Specification and containing all the details of the response generated by the Core. Nevertheless, in this particular function and for convenience of programmers using REST, the HTTP *Location* header contains the URL returned by the server.

```
Location: http://server/ch/lo/3
```

The **Submit/Store** function uploads an LO to a repository and makes it available for future access. This operation receives as argument an IMS CP compliant file and an URL generated by the Reserve function. This operation validates the LO conformity to the IMS Package Conformance and stores the LO in the internal database. To send the LO to the server we could use, in the REST flavour, the PUT or the POST HTTP methods. An example using the POST syntax is the following.

```
POST http://server/ch/lo/3 < LO
```

The repository responds with submission status data compliant with the Response Specification.

The **Search/Expose** function enables the eLearning systems to query the repository using the XQuery language, as recommended by the IMS DRI. This approach gives more flexibility to the client systems to perform any queries supported by the repository's data. After creating the XQuery file you can use the following POST request.

```
POST http://server/ch/lo < XQUERY
```

Alternatively, you can use a GET request with the searched fields and respective values as part of the URL query string, as in the following example.

```
GET http://server/ch/lo?author=Manzoor
```

Queries using the GET method are convenient for simple cases but for complex queries the programmer must resort to the use of XQuery and the POST method. In both approaches the result is a valid XML document such as the following.

```
<result base-url="http://server/ch/lo/">
 <request
  source="http://server/ch/lo/"
  message="Querying repository" />
 <response message="3 LOs found...">
<resources>
   <resource idCol="" idLo="5">
     Hashmat the Brave Warrior
   </resource>
   <resource idCol="" idLo="123">
        Summation of Four Primes
      </resource>
   <resource idCol="graphs/" idLo="2">
        InCircle
      </resource>
</resources>
 </response>
</result>
```

The **Report/Store** function associates a usage report to an existing LO. This function is invoked by the LMS to submit a final report, summarizing the use of an LO by a single student. This report includes both general data on the student's attempt to solve the programming exercise (e.g. data, number of evaluations, success); particular data on the student's characteristics (e.g. gender, age, instructional level). With this data, the LMS will be able to dynamically generate presentation orders based on previous uses of LO, instead of fixed presentation orders. This function is an extension of the IMS DRI.

The **Alert/Expose** function notifies users of changes in the state of the repository using a RSS feed. With this option a user can have up-to-date information through a feed reader.

Next, we present an example of a GET HTTP request.

```
GET http://server/ch/lo?alert+seconds > RSS
```

The repository responds with an RSS document.

The **Create** function adds new collections to the repository. To invoke this function in the REST interface the programmer must use the PUT request method of HTTP. The only parameter is the URL of the collection.

```
PUT http://server/ch/lo/newCol
```

The following is an example of the repository response to a create function.

```
<result base-url="http://server/ch/lo/" ...>
  <request
    source="http://server/ch/lo/newCol"
    message="Creating new collection" />
  <response message="Collection created">
 <resource idCol="newCol" idLo=""/>
  </response>
</result>
```

The **Remove** function removes an existent collection or learning object. This function uses the DELETE request method of HTTP. The only parameter is an URL identifying the collection or LO, as in the following example.

```
DELETE http://server/ch/lo/123
```

The following is an example of the repository response to a remove function.

```
<result base-url="http://server/ch/lo/" ...>
  <request source="http://server/ch/lo/123" message="Deleting a LO" />
  <response message="LO deleted">
 <resource idCol="" idLo="123" />
  </response>
</result>
```

The **Status** function returns a general status of the repository, including versions of the components, their capabilities and statistics. This function uses the GET request method of HTTP, as in the following example.

```
GET http://server/ch/lo?status
```

The repository responds with status data compliant with the Response Schema Specification.

3.4 Learning Tools Interoperability

An LMS is a software application for the administration, documentation, tracking, and reporting of training programs, classroom and online events [Ellis, 09]. In the majority of the cases an LMS integrates with other systems based in one of three levels:

Data Level - is the simplest and most popular form of integration in content management. This type of integration uses the import/export features of both systems and relies on the support of common formats

API level - allows client applications to use directly the functions of an eLearning system. These APIs foster client application development through data encapsulation and behavioural reuse.

Tools level - a uniform standards-based extension point in LMS allowing remote tools and content to be integrated into LMSs.

The three levels were analysed in [Queirós, 11:2] and the last two prove to be the most efficient. Tool integration is arguably the best choice in general since it provides a good balance between implementation effort and coupling and security. This is especially true if only unidirectional communication is required and Basic LTI is

used. This tool integration flavour is simple to implement and is already supported by most LMS vendors. If bidirectional communication is required then full LTI is needed but in this case the implementation is harder and few LMS vendors support this flavour of the specification. In both cases, tool integration has the added value of providing some basic security features based on the OAuth protocol aiming to secure the message interactions between the Tool Consumer and the Tool Provider.

Based on our previous research [Queirós, 11:2] we decide to use the LTI specification for the integration of the LMS with the pivot component. The workflow for using Basic LTI starts when the Teacher (or LMS administrator) configures the pivot component as a basic LTI tool in the LMS Control panel. Then, the Teacher must add the pivot into the course structure as a Basic LTI tool. This requires setting a resource link URL, a secret, and key as metadata for the resource link. Later on, when a student select the tool, the LMS uses the URL, secret, and key information to launch the pivot component in the student's running environment. The pivot component receives a launch request that includes user identity, course information, role information, and the key and signature. The launch information is sent using an HTTP form generated in the user's browser with the Basic LTI data elements in hidden form fields and automatically submitted to the external tool using JavaScript.

The following is a subset of the information that the LMS (Tool Consumer) sends to the pivot (Tool Provider):

resource_link_id=1 //An unique identifier for the resource in the LMS.
resource_link_title= My First Exercise //Title of the resource
resource_link_description= Description... //Description of the resource.
user_id=2 // User identifier
user_image=myPhoto.gif // Profile picture.
roles= Instructor,Administrator //List of one or more user roles.
context_title=Course Fullname 101 //A title of the context (e.g. course information).

All these data items are included on the POST data when a Basic LTI launch is performed. These data items can be used, for instance, to personalize the frontend of the tool provider.

The pivot component is launched as a Java Web Start (JAWS) application, a framework that allows users to start JAVA application software directly from the Internet using a web browser. The Java Network Launching Protocol (JNLP), defined with an XML schema, specifies how to launch JAWS applications. This approach facilitates the pivot component to interact with the students IDE by using commands over the shell (file system) of the students' runtime environment.

Since the LTI specification supports only web-based Tool Providers we need to inject the LTI variables on the pivot component. The approach used was the creation of an Adapter (coded as a JAVA servlet) to dynamically generate a JNLP file "on-the-fly" based on the LTI variables. After the generation, the Adapter redirects the request to the generated JNLP file in order to launch the JAWS program (Core component) as depicted in [Fig. 4].

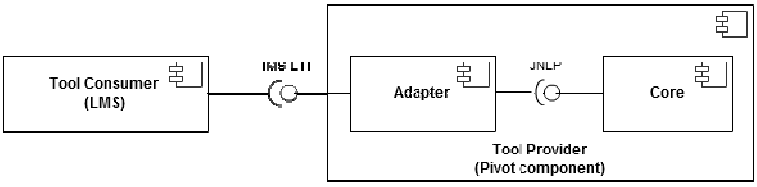

Figure 4: LTI workflow.

The JNLP file is composed by a root element called `jnlp`. This element has a `codebase` attribute that specifies an URL base for all the relative URLs specified in `href` attributes in the JNLP file. The value of the attribute is generated according to the current system path of the pivot component Adapter. The root element has four sub-elements: `information`, `security`, `resources` and `application-desc`.

The `information` element holds the name of the application (`name` element), the vendor of the application (`vendor` element), the home page for the application (`homepage` element), a reference for an image file to appear during launch when Java Web Start presents the application to the user (`icon` element) and a short statement of the application (`description` element). The `security` element can be used to request unrestricted access. Each application runs (by default) in a restricted execution environment, similar to the Applet sandbox. The inclusion of the `all-permissions` element guarantees that the application will have full access to the client machine and local network. If an application requests full access, then all JAR files must be signed. After that the user will be prompted to accept the certificate the first time the application is launched. The following code shows an excerpt of the generated JNLP file:

```
<?xml version="1.0" encoding="utf-8"?>
<jnlp spec="1.0+" codebase="$codeBase">
       <information>
              <title>Pivot Component</title>
              <vendor>eclipse.org</vendor>
              <homepage href="$codeBase" />
              <icon href="resources/icon.png" />
              <icon href="resources/logo.png" kind="splash"/>
              <description>
                     Application for assist teachers and students
                     to practice programming exercises solving.
              </description>
       </information>
       <security>
              <all-permissions />
       </security>
       <resources>
              <j2se version="1.5+" />
              <jar href="resources/Pivot_Component.jar" />
              ...
       </resources>
```

```
        <application-desc main-class="Pivot">
                <argument>$colId</argument>
                <argument>$studentFirstName</argument>
                <argument>$courseName</argument>
                ...
        </application-desc>
</jnlp>
```

The `resources` element is used to specify all the resources, such as Java class files and native libraries which are part of the application. A `jar` element specifies a JAR file that is part of the application's classpath. The jar file is loaded into the JVM using a `ClassLoader` object. The jar file contains Java classes and other resources, such as icons and configuration files (available through the `getResource` mechanism). The `j2se` element specifies what Java 2 SE Runtime Environment (JRE) versions are supported by the application.

The `application-desc` element indicates that the JNLP file is launching an application (as opposed to an applet). The application element has an optional attribute, `main-class`, which can be used to specify the name of the application's main class, i.e., the class that contains the `public static void main(String argv[])` method where execution must begin. Arguments can be specified to the application by including one or more nested `argument` elements.

3.5 Evaluation service

The purpose of a programming exercise evaluator is to mark and grade exercises in computer programming courses and in programming contests. By exposing its functions as services, an evaluator of this kind is able to participate in business processes integrating different system types. To formalize the definition of this service we used a particular e-Learning framework, the E-Framework. The new service - **Evaluate Programming Exercise** - models the evaluation of an attempt to solve a programming exercise defined as a LO and produces a detailed report [Leal, 10:2]. This evaluation report includes information to support exercise assessment and grading by client systems. The three types of request handled by this service are:

ListCapabilities - provides the client systems with the evaluator capabilities;
EvaluateSubmission - allows the request for a programming exercise evaluation;
GetReport - allows a requester to get an evaluation report using a ticket.

Their syntax as SOAP and REST web services is summarized in [Tab. 2].

Function	WS	Syntax
ListCapabilities	SOAP	ERL ListCapabilities()
	REST	GET /evaluate/ > ERL
Evaluate	SOAP	ERL Evaluate (LO, Attempt ,Capability, Language)
	REST	POST /evaluate/$CID?id=LOID&lang=LANG < PROGRAM > ERL
GetReport	SOAP	ERL GetReport(Ticket)
	REST	GET $Ticket > ERL

Table 2: Core functions of the Evaluation Engine.

All these functions respond with an XML document complying with the Evaluation Response Language (ERL) [Leal, 10:2]. The ERL is formalized in XML Schema and covers the definition of the response messages for the three evaluator functions. The specification includes two main elements: `request` and `reply`. The former echoes the request function and its parameters as received by the evaluation service. It contains a different sub-element according to the function type. The `reply` element depicted in [Fig. 5] contains the output to that request.

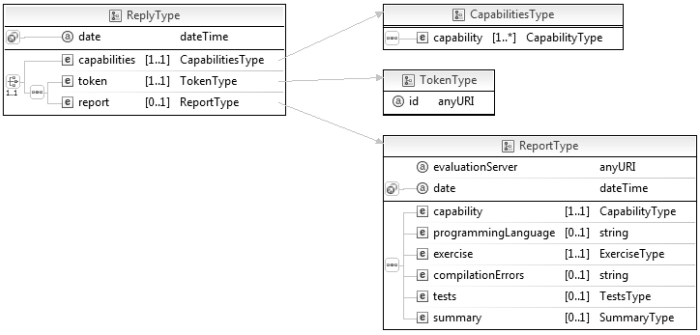

Figure 5: The reply element.

The `reply` element includes the possible responses of the service, more precisely, the `capabilities` element for the `ListCapabilities` function and the `token` and `report` elements for the `Evaluate` function. The former has several `capability` sub-elements each with several `feature` elements to describe it. The latter element includes a `token` element which holds a ticket to recover a report on a later date and the optional `report` element with the effective evaluation data.

Next, we detail the three functions of the Evaluate service.

The **ListCapabilities function** informs the client systems of the capabilities of a particular evaluator. In a computer programming evaluator the capabilities are related with the programming language compiler or interpreter. Each capability has a number of features to describe it and for a programming language they may be the language name (e.g. Java) its version (e.g. 1.5) and vendor (e.g. JDK). In this function, the request doesn't accept any parameter and the response returns a list of all capabilities of the evaluator. Each capability is described by a list of features, with a name and a value. Using the REST API this operation is performed by sending a GET HTTP request to the evaluator, as in the following example.

```
GET http://eval.domain.org/evaluate > ERL
```

The following document is an example of the HTTP response complying with the ERL specification.

```
<message date="2001-12-31T12:00:03">
 <request date="2001-12-31T12:00:03"/>
```

```
    <reply date="2001-12-31T12:00:05" >
      <capabilities>
       <capability id="MyService.C">
        <feature name="Compiler" value="gcc"/>
        <feature name="Compile" value="/usr/bin/gcc -lm $file"/>
        ...
       </capability>
      </capabilities>
    </reply>
  </message>
```

The **EvaluateSubmission** function allows the request of an evaluation for a specific exercise. The request includes a reference to an exercise represented as a LO held in a repository and a single attempt to solve a particular exercise. The request also includes a specific evaluator capability necessary for a proper evaluation of the attempt. The request may also include a specific ISO 639-1 language for the evaluation report. The response returns a ticket for a later report request and may return also a circumstantial report about the respective evaluation of the requester attempt. A sequence diagram of this function is shown in [Fig. 6].

The service endpoint provides the interfaces for the requests and responses for the evaluation functionality. Internally the service implementation may include several features (indexing, queuing, transforming, flow control, etc.) needed to provide the defined functionality and a connection with a remote data source holding the objects such as a LOR. The evaluator returns an evaluation report, if completed within a predefined time frame.

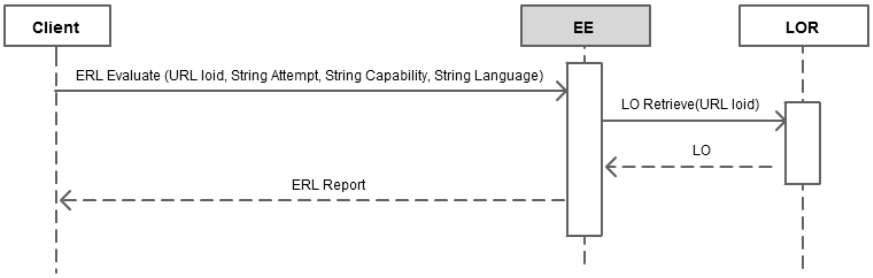

Figure 6: Sequence diagram of the Evaluate function.

Using the REST API this operation is performed by sending a POST HTTP request to the server, as in the following example.

```
POST http://eval.domain.org/evaluate/java1.6?
       id=http://lor.domain.org/lo/123&lang=pt < PROGRAM > ERL
```

The HTTP parameter `id` is a reference to a LO with the programming exercise. The `lang` attribute defines the language for the report. The `PROGRAM` is an attempt to solve it. The `ERL` is the content of the HTTP response to the above request. It includes a ticket and may include an evaluation report. The following is the respective response including only a ticket to later recover of the evaluation report.

```
<message date="2001-12-31T12:00:03" >
  <request date="2001-12-31T12:00:03">
    <capability=”MyService.C”/>
    <learningObject=http://lor.domain.org/lo/123/>
    <program><![CDATA[  … program code here …  ]]></program>
  </request>
  <reply date="2001-12-31T12:00:05">
    <token id=”https://eval.domain.org/report/123/xpto”/>
  </reply>
</message>
```

The `id` attribute of the `token` element can be used to recover the report on a later date. The **GetReport function** allows a requester to get a report for a specific evaluation using a previous ticket. The report included in this response may be transformed in the client side based on a XML stylesheet. This way the client will be able to filter out parts of the report and to calculate a classification based on its data. The request of this function includes a ticket sent previously by the service in response to an evaluation. The response returns a report about an evaluation. The evaluation report does not compute a grade, points or classification, nor produces a feedback for any particular scenario. Using the REST API this operation is performed by sending a GET HTTP request to the evaluator, as in the following example.

```
GET https://eval.domain.org/report/123/xpto > ERL
```

The URL is the ticket obtained in the last request. The following is an example of the HTTP response.

```
<message date="2001-12-31T12:00:00">
 <request date="2001-12-31T12:00:00">
  <token id=”https://eval.domain.org/report/123/xpto”/>
 </request>
 <reply date="2001-12-31T12:00:00">
   <token id=”https://eval.domain.org/report/123/xpto”/>
   <report
       date="2001-12-31T12:00:00"
       evaluationServer=”https://eval.domain.org/”>
       <capability id="MyService.C"/>
       <exercise href="http://lor.domain.org/lo/123”>
             A very simple Problem
       </exercise>
       <compilationErrors/>
       <tests>
         <test executionTime="100" mode="program">
           <input><![CDATA[/home/…/R43/tests/T1/in-1.txt]]></input>
           <expectedOutput>4</expectedOutput>
           <obtainedOutput>4</obtainedOutput>
           <outputDifferances></outputDifferences>
           <classify>Memory Limit Exceeded</classify>
           <mark>0</mark>
           <feedback/>
           <environmentValues>
              <environmentValue name="memory" value="12kb" />
```

```
          </environmentValues>
        </test>
      </tests>
      <summary>...</summary>
    </report>
  </reply>
</message>
```

The response contains a detailed evaluation report but should not reach to any conclusion.

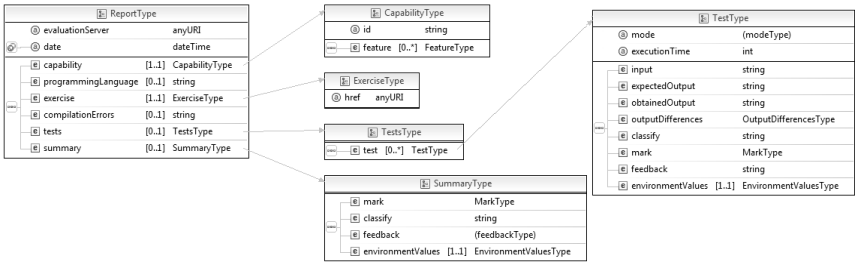

Figure 7: The report type on the ERL specification.

The element `report` depicted in [Fig. 7] from the ERL specification contains the raw data sent to the client and can be used as input for other systems (e.g. classification systems, feedback systems). It has a single mandatory `evaluationServer` attribute representing the URL of the evaluator. This element also includes the following sequence of sub-elements:

- `capability`: a specific evaluator capability used to evaluate this attempt;
- `programmingLanguage` – the language used to code the solution;
- `exercise`: a reference to the Learning Object and the title of the exercise;
- `compilationErrors` – compilation error messages of the user's code;
- `tests`: contains a set of tests for the evaluation of the submitted attempt. Each test element represents a test case describing resources supplied to evaluate the submitted program;
- `summary` – the synthesis of the assessment;

As shown in Fig. 6, each test corresponds to a single test case that can be repeated to create a test set. The submitted program is executed once for each test element, receiving as input the content of the input element. The resulting output, stored in the `obtainedOutput` element, is compared to the expected output contained in the `expectedOutput` element. The `outputDifferences` element describes the differences between the two previous elements using the syntax of the Unix diff command. The `test` element contains also data for grading and correcting programs. This element includes a `mark` element to assign a mark for a successful execution. The client may compute a grade for the submission as the sum of the marks of successful executions. The optional `feedback` element contains detailed feedback for an unsuccessful execution. The environment values are a list of property-value pairs

that may be supplied by the execution environment. For instance, if the execution environment is able to report the memory usage of a program execution then this data is recorded in this element

3.6 Integrated Development Environment

Experimenting environments – environments for practicing on a learning subject to consolidate learning – are an important type of specialized services to be integrated in learning processes. These environments need a user interface to interact with learners and application interfaces to be integrated on the learning process. In some cases they will have to be developed for specific domain, while in other they can be adapted from existing systems. In the computer language programming domain an IDE [Nourie, 05] is arguably the best place for a student to practice by solving programming exercises, definitely better than any tool available on a LMS. Unfortunately, the available IDEs lack the features to communicate with other specialized services. In extensible IDEs (e.g. Eclipse[1], Visual Studio Express[2]) this shortcoming may be overcome using plug-ins. However a plug-in for a certain IDE vendor will not work on an IDE from a different vendor, which raises an interoperability issue. For this reason we opted for a somehow easier but more general solution: to interact with the IDE via shell commands.

Since both the pivot component and the IDE run on the machine of the student it is easy for the pivot to interact with the IDE using shell commands. These commands are used, for instance, to create a skeleton of the project or to execute locally the student program. Due to the nature of this we formalized the interface between the pivot and the IDE using the programming language in which the pivot is implemented. Since it is implemented in Java we abstracted in a Java interface the type of operations that must be executed on the IDE. Table 3 enumerates the four methods that need to be implemented.

Method	Description
makeProject(path:String, language:String)	Creates a project on the IDE. The *path* argument includes the location and the name of the project. The *language* argument is the programming language for the project.
getWorkspace()	Returns a string with the workspace location of the project.
getProjectName()	Returns a string with the name of the project.
getName()	Returns a string with the IDE name.

Table 3: ProjectCreator *interface methods.*

For each IDE vendor there is an implementation of this interface. This implementation executes shell commands to modify the file system, execute programs or activate IDE functions. Currently, there are implementations for Eclipse and for Visual Studio.

[1] http://www.eclipse.org/
[2] http://www.microsoft.com/express

4 Case Study

In this section we start by making a brief description of an early initiative that based our current work. Then, we present a learning scenario where our current approach may fit including the identification of the systems used and how they relate to each other. Finally we detail the effective use of this work on classroom and provide usage data to validate the feasibility of this network regarding the interoperability efforts made.

4.1 PERE

As a proof of concept of the pivot component we developed a component called Programming Exercises Resolution Environment (PERE) [Leal, 11]. The task of this component is twofold: to coordinate the communications between the LMS and a set of web services; and to act as an exercise resolution environment where students solve their programming exercises assignments, in replacement of the IDE. The PERE component is organised in two main packages: the back-end (used by the teacher) and the front-end (used by the student). In the back-end the teacher configures an assignment by searching for programming exercises in the repository and associating the most relevant. In the front-end, the student reads the exercise description, solves it in PERE and gets the evaluation report that will help him to refine the exercise and, if necessary, resubmit it. Despite the success of this approach some issues raised regarding the use of an *ad-hoc* environment for the resolution of the exercises. We consider the IDE as as far better environment for a student to practice computer programming and solving programming exercises. This issue led to the integration of the IDE on this network to enrich the environment of the student.

4.2 PETCHA

One of the distinctive features of this new approach was the integration of an IDE in the current architecture. This integration was a huge challenge since IDEs were not designed to interact in the e-Learning realm and must be extended for that purpose. In order to enhance interoperability we decided to abstract the use of the IDE for solving programming exercises, rather than creating a plug-in for a specific IDE. To accomplish this task we changed the deployment of the pivot component from a web based solution running on the LMS side to a local JAVA application (JWS) running on the user's environment side. Using this approach the pivot component – now called Programming Exercises Teaching Assistant (PETCHA) - is launched by the LMS and runs locally (on the user's environment) to facilitate the communication over the shell with a supported IDE.

In this context, the pivot component acts as an automatic TA with two main tasks: to assist teachers in the authoring exercises and to help students in solving them. Although complementary, these two tasks share a number of requirements. Both teacher and student need to: code and test programs in an IDE; send and retrieve learning objects from a Learning Objects Repository (LOR); check program code against test cases. Thus, although the graphical user interface of both user profiles shown in [Fig. 8] is apparently very different, they actually share many internal functions of the pivot component.

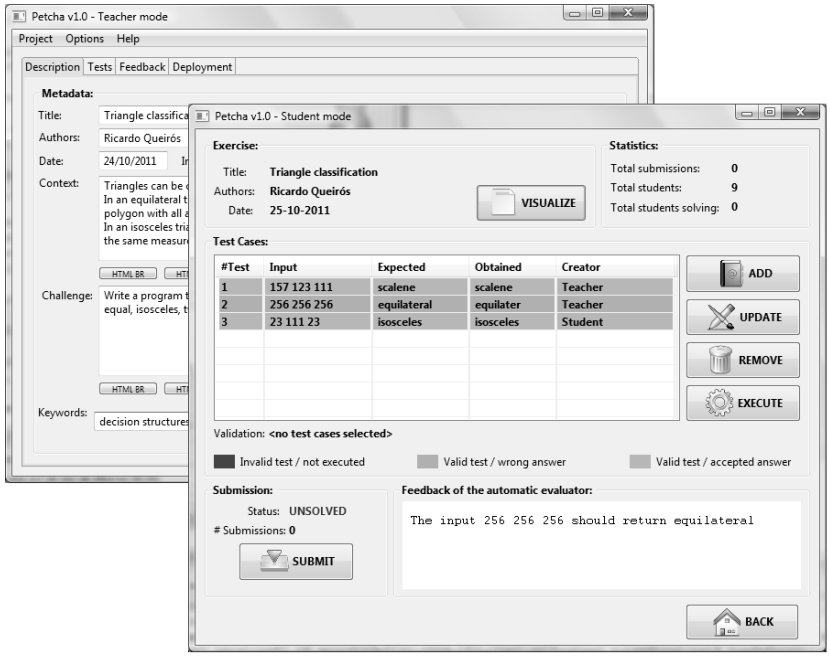

Figure 8: The GUI of the pivot component with teacher and student modes.

One of the most important tasks of both user profiles is the exercise coding: the teacher needs to code the program solution and the student needs to code an attempt to solve it. Both require the use of an IDE. Thus, Petcha uses the *ProjectCreator* interface [Tab. 3] to abstract the use of specific IDEs on Petcha. After the definition of this interface class, developers should implement it in order to provide support for other IDEs.

For instance, when a student starts solving an exercise, the Petcha component automatically creates a project on the IDE of the student invoking the *makeProject* method. Currently Petcha supports "out of the box" the creation of JAVA and C# projects in Eclipse and Visual Studio Express. Beyond this built-in support, developers can extend Petcha to other IDEs and languages. For this extension one must require attention for the *makeProject* method. A project contains source code and related files for building a program in a specific programming language. Thus, a set of predefined files need to be generated for the project creation. These files are related with the IDE foundations and the chosen programming language.

After the automatic creation of the project the student reads the exercise description in Petcha's GUI and solves it on the IDE. The student should test the code locally by executing the teachers' test cases and is encouraged to create new ones. If new test cases are created, a validation step is performed to verify that they meet the specification defined by the teacher in the authoring phase. The right window on Figure 8 shows an example where the student's code did not pass all the local tests (two provided by the teacher and one new test created by the student). Even so, the

student decided to submit the code to the evaluator and received a feedback message indicating an input data that generated a wrong answer. After testing, the student should submit the solution to the Evaluation Engine (EE) where the submission is checked against the complete test set provided by the teacher. The report on the evaluation returned by the EE is presented to the student. The student may submit repeatedly, integrating the feedback received from the EE. In the end of this cycle, Petcha reports the exercise usage data back to the repository.

Another important point was the choice of the systems that comprise the current network. Since we made several efforts to address interoperability issues, the selection of the tools was straightforward as shown in [Tab. 4].

System	Version	Type	Supports	Requirements
Moodle	1.9	LMS	Basic LTI 1.0	Windows/Linux + XAMP 1.7.7
CrimsonHex	0.8	LOR	CC 1.1 & DRI 1.0	Windows/Linux + XAMP 1.7.7
Mooshak	1.6a2	EE	Evaluate Service	Linux + Apache + TCL
Eclipse	3.7.1	IDE	-	Windows/Linux

Table 4: Network selected systems.

On the LMS side we choose Moodle since it is a popular and open source LMS [Davis, 09], arguably the most popular LMS nowadays. We used the version 1.9 that supports the Basic LTI specification with the further installation of an IMS bLTI consumer[3]. Currently, the version 2.2 supports the IMS LTI 1.1 (a merge version of basic and full LTI) and import IMS CC packages. The exportation of CC packages will come in version 2.3. We successfully tested also the Sakai LMS on this network evidencing the interoperable characteristics of the proposed approach.

For the LOR selection, we had more difficulties to find a system that supports the defined content and communication specifications respectively the IMS CC and IMS DRI specifications. The final choice fell on a home-made system called CrimsonHex - a repository of programming exercises described as learning objects and complying with the IMS CC specification. The repository also adheres to the IMS DRI specification to communicate with other systems.

The EE system selected was Mooshak. Mooshak is an open source system for managing programming contests on the Web including automatic judging of submitted programs. The current version (1.6a2) supports the Evaluate service (E-F).

On the IDE side we selected Eclipse. Eclipse is a free and open source multi-language software development environment comprising an integrated development environment (IDE) and an extensible plug-in system. We tested also the Visual Studio Express IDE on this network with success for C# assignments. In this case we need to install Mono to run .NET applications on the Mooshak server. Mono is a free and open source project to create a standard compliant .NET-compatible set of tools including a Common Language Runtime, C# compiler and others.

The requirements to recreate the proposed network are few. For the LMS and LOR installation it was necessary to install the XAMPP package that comprises a Web server (Apache), a servlet container (Tomcat), a database (MySQL) and a

[3] http://code.google.com/p/basiclti4moodle/

server-side scripting language (PHP). For the Mooshak instalation we use a Linux distribution (Ubuntu) with a Apache version 1.2 (or better) and Tcl version 8.3 (or better).

The diagram shown in [Fig. 9] could be applied to a typical pedagogical learning process such as a classroom assignment in a Computer Science course.

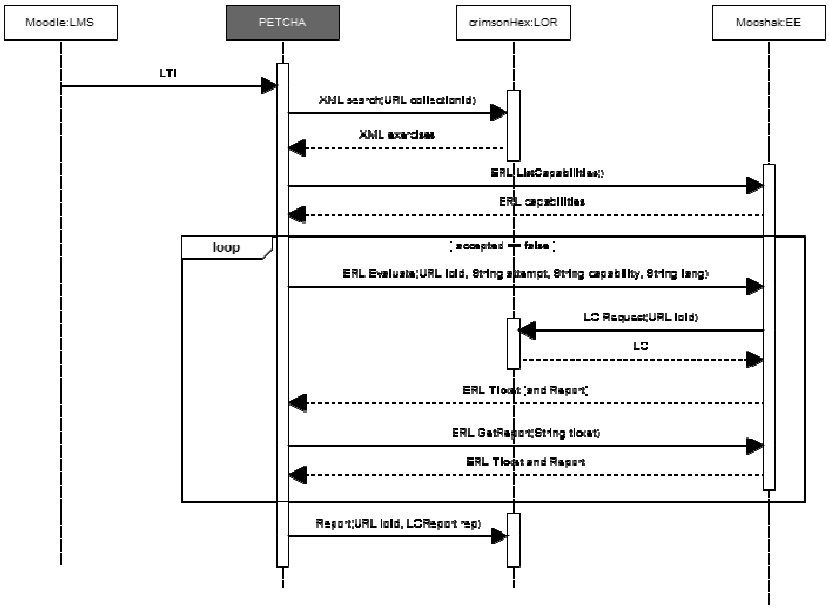

Figure 9: Sequence diagram.

4.3 Experiment and usage data

In order to validate the feasibility of this network we conducted an experiment at ESEIG - a school of the Polytechnic Institute of Porto. Students from the first-year of the course Algorithmic and Programming (degree in Mechanical Engineering) participated in a two-month experiment (6 classes). The course aims to widen the students' programming skills using the C# programming language. The course has an average enrolment of 40 students per year organized in two classes and delivered through two lectures of one hour each and one lab session of 4 hours per week. The final grade has two components: 30% for course work (e.g. exercises) carried out during the year, and 70% for one interdisciplinary work. The experiment methodology was the following: only one class used the system while the other class kept the traditional learning approach. In the end of the experiment we compared both classes regarding, for instance, the number of solved exercises, the coverage of the syllabus by the exercises or the feedback level. We conducted also a survey to collect the opinion of students and teachers involved in this experiment. From this data, we present here only what is meaningful to evaluate the interoperation of the systems in the network. [Tab. 5] summarizes the communication between pairs of systems. These results were gathered from 6 lab sessions. In each session a class of 21 students

had 3 exercises to solve. Based on these figures we computed the following expected values: the expected number of accesses to the system that is given by multiplying the number of students with the number of assignments (21 * 6 = 126); and the expected number of accesses to the exercises by all the students that is given by multiplying the number of students with the number of assignments and with the number of exercises by assignment (21 * 6 * 3 = 378).

Observation point	Expected	Real
# accesses to the system (LMS→PETCHA)	126	135
# exercises requested to the repository (PETCHA→LOR)	378	345
# exercises that students try to solve (PETCHA→IDE)	378	342
# submissions (PETCHA→EE)	378	819
# exercises requested to the repository (EE→LOR)	18	18
# exercises in which the students got feedback (EE→PETCHA)	378	810

Table 5: Statistical data on interoperability of the network components.

The first line of the table indicates that the system worked well since only nine extra sessions were used. These extra values were mainly due to the accidental closing of the application by students.

The number of exercises requested by PETCHA measures the number of times that students got an exercise statement from the repository. This action triggers an automatic request from PETCHA to the repository. From the collected data, we can observe that not all exercises were actually read by all the students. There are two possible justifications: either the students did not have time to read all the available statements or some students may have given up after reading the exercise title.

The following line is the number of exercises that students tried to solve. We considered that a student tried to solve an exercise when (s) he ran locally a set of tests to validate his/her code. The real number is less than the expected and less than the number of exercise statement readings. Most probably some student read an exercise statement but did not have time to code a solution or run the tests, or just gave up solving it.

The number of submissions is the number of requests for evaluation that the EE receives. The obtained values show an average of approximately two submissions per exercise for each student.

The number of exercises requested by EE to the LOR reflects the need of the Evaluation Engine to obtained the fully LO from to repository given its reference. Since the EE has a cache mechanism the expected and real values are identical thus showing that the EE cache feature is working as expected and is accelerating the evaluation process.

The number of exercises in which the students got feedback should be similar to those of the number of submissions. Since they are almost identical (error margin of 9) we can conclude that the communication among the two systems (PETCHA and EE) works well. The value in the expected column is due to the fact that in a optimal scenario students will submit their solutions and they will got accepted at the first attempt.

The figure presented in Table 4 shows that the proposed approach for e-Learning system orchestration is useful in practice. The figures collected during the experiment

are within reasonable bound from the expected values. With these results we conclude that the network is stable enough to handle a much larger number of students and exercises and to be used in a more demanding setting.

The results and analysis of the experiment (using a quasi-experimental design) regarding the use of the system from a qualitative point of view can be read in [Queirós, 12].

5 Conclusion

In this paper we presented an approach for a standard based integration of several systems involved in the automatic assessment of programming exercises. The design and implementation of the integration model was based on a pivot component that acts as an orchestrator in the network and implements several communication standards. In order to validate the feasibility of this approach we tested the system on a actual classroom and this paper presents results of that experiment related to system interoperability.

The main contribution of this paper is twofold: the creation of an environment that fosters the practice on solving programming exercises and the design of an integration model using communication standards that can be used or reused in similar scenarios.

This network is currently being used in the practical classes of an undergraduate programming course. The experiments designed to assess the impact of this tool at an interoperability level were presented in this paper. These experiments showed that the interoperability efforts made in the design and implementation phase of the network were rewarded. Firstly, the reliability of the network was validated taking into account the number of launches of the pivot component from the LMS. Then, systematic tests were made to guarantee the effectiveness of the communication between the network components. For systematic accesses to components (e.g. evaluator) cache features were implemented lowering the number of accesses and accelerating the evaluation process. Based on these figures one can conclude that the network is stable enough to handle a much larger number of students and exercises and to be used in a more demanding setting.

The proposed architecture can be used in other problem-based e-learning scenarios. As further work the challenge is to use the proposed architecture in other domains. One interesting domain is serious games applied to management courses where students can train management processes through simulation. In this domain teachers use business simulation games to improve the strategic thinking and decision making skills students in particular areas (e.g. finances, logistics, and production). Through these simulations students compete among them, as they would in a real world companies. A business simulation service fulfils a role similar to that of the assessment systems in programming exercises and it also requires a repository containing specialized LO describing simulations. Thus, this specific domain poses challenges not only in the development of the network pivot component , but also in the refinement of other systems (e.g. repository, assessment system) to meet the new evaluation domains requirements.

Other future work includes the integration of a sequencing and adaptation component that controls a specific learning workflow by guiding the student on a set of expository and evaluation resources; a plagiarism detection component to prevent

inappropriate coding practices; and also increase the range of assessment scenarios for the evaluation of new languages such as query languages (e.g. SQL), modeling languages (e.g. UML) and user interfaces (e.g. GUI).

References

[Al-Smadi, 10] Al-Smadi & Gütl: SOA-based Architecture for a Generic and Flexible E-assessment System. In EDUCON'10, 2010.

[Alstes, 07] Alstes, A. and Lindqvist, J.: VERKKOKE: learning routing and network programming online, In Proceedings of The Twelfth Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE'07: University of Dundee, Scotland, 25-27 June 2007.

[Apostolopoulos, 03] Apostolopoulos, T. K., & Kefala, A.: An e-learning service management architecture. In Proceedings of the 3rd IEEE International Conference on Advanced Learning Technologies (pp. 140-144). Athens, Greece, 2003.

[Casella, 07] Casella, G., Costagliola, G.,Ferrucci, F., Polese, G., Scanniello, G.: A SCORM Thin Client Architecture for e-learning Systems based on Web Services. In International Journal of Distance Education Technologies, Vol. 5, No. 1, January-March, IDEA Group Publishing, pp.: 13-30, 2007.

[Casquero, 10] Casquero, Oskar, Javier Portillo, Ramón Ovelar, Manuel Benito, and Jesús Romo. 2010. iPLE Network: an integrated eLearning 2.0 architecture from University's perspective. Interactive Learning Environments 18 (3):293-308, 2010.

[Conde, 10] Conde, M. Á., García-Peñalvo, F. J., Casany, M. J., & Alier, M. (2010). Applying Web Services to define Open Learning Environments. Paper presented at the Twenty-First International Workshops on Database and Expert Systems Applications – DEXA 2010. Third International Workshop on Social and Personal Computing for Web-Supported Learning Communities – SPeL 2010, Bilbao, Spain, 30 August - 3 September 2010.

[Conde, 11] Conde, Miguel A., García, Francisco J., Alier, M. and Casany, María J. (2011) Merging Learning Management Systems and Personal Learning Environments. In: Proceedings of the The PLE Conference 2011 , July 2011, Southampton, UK.

[Davis, 09] Davis, B., Carmean, C. and Wagner, E.D. The Evolution of the LMS: From Management to Learning - Deep Analysis of Trends Shaping the Future of eLearning, Sage Road Solutions, LLC, 2009.

[de-la-Fuente-Valentín, 08] de-la-Fuente-Valentín, L., Leony, D., Pardo, A., & Kloos, C. D. (2008). Mashups in Learning Design: pushing the flexibility envelope. Paper presented at the Mash-Up Personal Learning Environments - 1st Workshop MUPPLE'08, Maastricht, The Netherlands.

[Eap, 04] Ty Mey Eap, Marek Hatala, and Griff Richards. Digital repository interoperability: design, implementation and deployment of the ecl protocol and connecting middleware. In Proceedings of the 13th international World Wide Web conference on Alternate track papers \& posters (WWW Alt. '04). ACM, New York, NY, USA, 376-377, 2004.

[Fielding, 05] Fielding, R.T. and Taylor, R.N.: Principled Design of the Modern Web Architecture, ACM Transactions on Internet Technology (TOIT) (New York: Association for Computing Machinery) 2 (2): pages: 115–150, doi:10.1145/514183.514185, ISSN 1533-5399, 2005.

[Holden, 04] Holden, C. and Academic A. D. L. Staff: What we mean when we say "repositories": User expectations of repository systems. Technical report, The Academic ADL Co-Lab, Madison, WI, July 2004.

[IMS CC, 11] Common Cartridge Overview v1.2 Final Specification, 2011, http://www.imsglobal.org/cc/ccv1p2/imscc_profilev1p2-Overview.html.

[IMS DRI, 03] IMS Digital Repositories Interoperability. [On-line]. Available at: http://www.imsglobal.org/digitalrepositories/driv1p0/imsdri_bestv1p0.html

[IMS LTI, 11] Learning Tools Interoperability v1.1 Public Draft Implementation Guide, 2011, http://www.imsglobal.org/lti/v1p1pd/ltiIMGv1p1pd.html.

[Leal, 09] José Paulo Leal, Ricardo Queirós, CrimsonHex: a Service Oriented Repository of Specialised Learning Objects, in Joaquim Filipe and José Cordeiro (Eds.) Proceedings of ICEIS'09: 11th International Conference on Enterprise Information Systems, pages 102-113, Milan, Italy, May 2009, ISBN: 978-3-642-01346-1, DOI: 10.1007/978-3-642-01347-8_9

[Leal, 10] Leal, J.P. and Queirós, R.: eLearning Frameworks: a survey. Proceedings of International Technology, Education and Development Conference - Valencia, Spain, 2010.

[Leal, 10:2] Leal, J.P., Queirós, R. and Ferreira, D.: Specifying a programming exercises evaluation service on the e-Framework, in Xiangfeng Luo, Marc Spaniol, Lizhe Wang, Qing Li, Wolfgang Nejdl and Wu Zhang (Eds), Advances in Web-Based Learning - ICWL 2010 - 9th Internation Conference, Shanghai, China, December, 2010, LNCS 6483, pp. 141-150, ISBN 978-3-642-17406-3, DOI.

[Leal, 11] Leal, José Paulo and Queirós, R.: Using the Learning Tools Interoperability Framework for LMS Integration in Service Oriented Architectures, in Technology Enhanced Learning, TECH-EDUCATION'11, Springer Verlag, May 2011.

[McCallum, 06] S. H. McCallum. A look at new information retrieval protocols: Sru, opensearch/a9, cql, and xquery. In In World Library and Information Congress: 72nd IFLA General Conference and Council - IFLA, 2006. http://archive.ifla.org/IV/ifla72/papers/102-McCallum-en.pdf.

[Neven, 02] Neven, F. and Duval, E.: Reusable learning objects: a survey of LOM-based repositories. In MULTIMEDIA '02: Proceedings of thetenth ACM international conference on Multimedia, pages 291–294, New York, NY, USA, 2002, ACM Press.

[Nourie, 05] Nourie, Dana. "Getting Started with an Integrated Development Environment". Sun Microsystems, Inc. 2005. Retrieved 9 September 2008.

[Riad, 09] Riad, A.M. et al. Review of e-Learning Systems Convergence from Traditional Systems to Services based Adaptive and Intelligent Systems. Journal of Convergence Information Technology (JCIT), Advanced Institute of Information Technology, Volume 4, Number 2, June 2009

[RSP, 10] Repositories support project: Repository Software Survey, Reviewed 11 November 2010, http://www.rsp.ac.uk/documents/Repository-Software-Survey-2010-11.pdf.

[Queirós, 11] Queirós, R. and Leal, J.P.: PExIL: Programming Exercises Interoperability Language", in Alberto Simões and da Cruz Daniela and Ramalho José Carlos (eds) "XATA 2011 –- 9ª Conferência Nacional em XML, Aplicações e Tecnologias Aplicadas", Junho 2011.

[Queirós, 11:2] Queirós, R., Oliveira, L., Leal, J.P. and Moreira, F.: Integration of ePortfolios in Learning Management Systems, in B. Murgante, O. Gervasi, A. Iglesias, D. Taniar, B. Apduhan (Eds.) Computational Science and Its Applications - ICCSA 2011, 11th International

Conference on Computational Science and Its Applications June 20-23, Santander, LNCS 6786/2011, pp 500-510, DOI: 10.1007/978-3-642-21934-4.

[Queirós, 12] Ricardo Queirós and José Paulo Leal, PETCHA - A Programming Exercises Teaching Assistant, ITiCSE 2012 - ACM SIGCSE 17th Anual Conference on Innovation and Technology in Computer Science Education, Haifa, Israel, 3-5 July 2012

[Severance, 08] Severance, C., Hardin, J., & Whyte, A. The coming functionality mash-up in Personal Learning Environments. Interactive Learning Environments, 16(1), 47-62. doi: 2134561, 2008.

[Ternier, 10] S. Ternier, D. Massart, M. Totschnig, J. Klerkx, and E. Duval. The simple publishing interface (spi). D-Lib Magazine, 16(9/10), 2010. http://www.dlib.org/dlib/september10/ternier/09ternier.html.

[Truong, 07] Truong, N. A web-based programming environment for novice programmers. PhD thesis. Queensland: University of Technology, 2007.

[Wang, 08] Wang, F. L., & Wong, T. Designing programming exercises with computer assisted instruction. Lecture Notes in Computer Science, 5169, 283–293, 2008.