ECP-2008-EDU-428037

LiLa – Library of Labs

D2.1 – Documentation of the software interfaces of the virtual portal

Deliverable number/name	*D-2.1 – Documentation of the software interfaces of the virtual portal*
Dissemination level	*Public*
Delivery date	*30 November 2009*
Status	*Final*
Author(s)	*L. Bellido, P. Debicki, A. Gallardo, V. Mateos, T. Richter, V. Villagrá*

e*Content*plus

This project is funded under the *e*Content*plus* programme[1],
a multiannual Community programme to make digital content in Europe more accessible, usable and exploitable.

[1] OJ L 79, 24.3.2005, p. 1.

Table of contents

1 Introduction

The objective of the Lila project is to combine virtual laboratories and remote experiments (i.e. simulated experiments and experiments which are controlled remotely by computers) spread out over Europe, making them reachable in an environment with central retrieval and access facilitating synchronous collaboration and user generated production. The goal of this project is not only to integrate experiments and laboratories into a software infrastructure, but also to build a virtual portal in which experiments and laboratories are provided, including on-line course system which guides users through experiments. From a technical point of view, the project will:

- create a repository of laboratories on a central server, equipping the labs with meta-data to integrate them in library search engines and link-resolver technology,
- exploit them with a well defined access control (partially technical, partially organisational),
- lead users with a tutoring system,
- connect the labs to the 3D-engine Wonderland as collaboration environment for students, teachers and researchers, and
- refer users to an organisational framework for online collaboration, for the transfer of virtual laboratories and connected support-services as well as for access-opportunities to remote experiments.

These objectives are addressed with a 4 Tier approach:

- Already available content in the form of virtual and remote experiments and traditional media
- Technical components for managing and accessing the content, including access control technologies, content & service discovery with content metadata, authorization and scheduling of the access to the experiments, etc.
- A tutoring system which can orchestrate the relevant resources, combine different information sources in order to provide the most effective learning experience adapted to the student’s needs in terms of background and curricula.
- A Virtual Portal that will act as a platform integrating all the rest of modules and providing access to all the LiLa functionalities.

In the scope of this project, Work Package 2 is responsible of designing and integrating the needed components in order to provide the Virtual Portal functionality which will integrate the media and tutoring systems defined by WP3 and WP4, including also additional components as collaborative tools for scheduling access to experiments and federated access control mechanisms for a single sign-on authentication of the students.

This deliverable D21 is the first document produced in the scope of this workpackage and is intended to provide a description of the initial Virtual Portal architecture that has been designed in the project. This architecture has been defined with the following components:

- A virtual portal technology based on portlets that will host all the LiLa specific components. The virtual portal will include an elementary SCORM compliant run-time system, for accessing and interfacing the SCORM objects with the provided contents. These SCORM object can also be downloaded and executed in any SCORM-compliant Learning Management Systems (LMS).
- Initial design of components for central authentication and resources reservation and scheduling.
- Interfaces for the access to the provided contents as SCORM objects.
- The integration of the 3D-engine Wonderland will be based on specific individual experiments rather on the whole LiLa portal, due to the scalability problems found and described in section 2.1

In this document, section 2 will provide a description of the main technologies that are being used in the LiLa Work Package 2, including portlets technologies, SCORM standards and the most used Learning Management Systems. Section 3 is describing the overall architecture of the LiLa project and the initial design of the components related to access control and access scheduling and reservation. Section 4 describes the interfaces to the experiments with an encapsulation of the experiments as SCORM objects in what is called a LLO: LiLa Learning Object. Finally Section 5 describes the relationship of the contents of this document with the rest of Work packages.

2 Technology overview

2.1 *Wonderland technology*

Project Wonderland is the name of an open source project providing a virtual 3D world very much in the spirit of "Second Life". Wonderland has been originally developed by Sun Microsystems as a platform for connecting home-working engineers with their colleagues, to allow chatting, project meetings and presentations even for remote participants. For LiLa, Wonderland is also envisioned as a virtual meeting place for students running experiments, then remotely controlled by their avatars in the virtual 3D world, and to this end the original portal design proposed Wonderland as the entry portal to gain access to all of the LiLa content.

Early discussions with Sun in San Jose in July 2009, however, revealed that Wonderland is not yet quite as mature as it should be and has currently a scaling problem: Due to the backend server structure, the system usability severely suffers if more than fifteen to twenty participants are online at the same time - a user number much too low for our envisioned LiLa user counts. Instead, we now re-designed the portal architecture to use a traditional 2D web portal interface that scales much better, and which grants access to individual experiments hosted at partner institutions. An experiment might now be a flat 2D "traditional" experiment, or a visionary 3D experiment which is then, however, run on a server hosted by the corresponding LiLa partner. As each experiment is now hosted by its individual server in this new portal design, the scaling problem no longer exists since student groups are, also for didactical purposes, rarely larger than four or five members.

At the time of writing, LiLa does not yet contain any 3D content, but a pilot project to be created by MathCore in Linköping is under construction and will likely become available in Q1 or Q2 of 2010.

Because only limited experience with Wonderland exists at this time, its integration into the portal and the software interfaces required to connect to Wonderland are limited to the absolute minimum. As defined in section 4.2, a Wonderland experiment is, at present, only a java webstart link for the given portal architecture, which is sufficient to run a Wonderland application from the portal and thus to test the applicability of virtual 3D worlds in education. However, some functionality remains undefined at this moment, namely authentication and authorization of users, i.e. the current infrastructure does not allow a single-sign-on process for Wonderland experiments, but rather requires users to log in a second time into the virtual world. This functionality would require an extension of the Wonderland technology which would pass parameters like the user name and the URI of the booking system into Wonderland upon its initialization, and Wonderland would contact this booking server then to test for the availability of the 3D world.

2.2 Portlet technology

The virtual portal of the LiLa consortium is based on the Portlet Standard JSR 168 [1] and its successor, JSR 286 [2]. The portlet technology allows maintainers to easily modify and adapt the contents of the portal, and furthermore allows flexible integration of content on demand.

Even though the content is uploaded to the LiLa project in the form of SCORM packages, a standard coming from the world of Learning Management Systems, we decided not using exclusively a Learning Management System like Moodle or Ilias as virtual portal. Instead, we will design an elementary SCORM compliant run-time system, specified below, sufficient for SCORM packages to render correctly at the user side while allowing their integration into a JSR 168 compliant system. The following reasons were important in this decision:

- SCORM provides only very limited possibilities to realize the LiLa booking system. If at all, such a booking system is part of the Learning Management System, but as such components are not in the main focus of the development teams, they often lack the functionality we need. For example, we would prefer a booking system that grants higher priority to institutions providing and maintaining the experiments, and we would also prefer a booking system that operates independent from the portal, as several access portals to the same content may exist.

- Portlets provide much higher flexibility than SCORM and javascript based technology. Desired functionality, as for example the ability to rate content, or user-based content already exists in the form of portlets. Missing functionality can be simply added to the system by providing a corresponding portlet; for moodle or Ilias, such functionality could only be added by modifying the core code of the LMS itself. Even though possible as both LMS are open source, this approach is problematic as any patches to the core would require updating on each revision of the LMS.

The Portlet technology is an extension of the Java Servlet technology in which a Java application server interprets Java code, here being part of the Portlet, to render dynamically created HTML code. Portlets extend servlets by defining a standardized package format for individual components from which a web page is constructed, it defines an archive format by which such components are deployed, and a standardized set of interfaces allowing maintainers and users of a portal to configure and edit the portlets, forming the contents of the portal, as required. Portlets are close analogues to windows of a standard desktop system: Their contents are rendered independently, and they can be freely moved or resized on the portal page. The decoration of portlets, i.e. their look and feel, can be freely configured by the means of the portal itself, and a standardized interface allows to reconfigure the internals of portlets.

Technically, a portlet consists of a Java ".war" archive, which is just a ".jar" package with standardized contents; it consists of the HTML code itself, still containing snippets of Java code to be executed by the server when rendering the portlet, XML files describing its contents, and compiled java classes that provide additional information required to render the HTML contents. Typically, these java classes keep the internal states of the portlet and represent them as Java Beans, which again are accessed within the HTML code by Java code snippets. Details on this mechanism are found in the JSR 168 and JSR 286 documentation.

Even though the LiLa Portal does use the portlet technology, its contents are not represented by portlets; this is because existing Learning Management Systems (LMS) predate the servlet/portlet technology and use an older mechanism to bundle its contents. The predominant standard for Learning Objects (LOMs) is SCORM (see section 2.3), which represents HTML pages in a ".zip" container and uses client-based javascript instead of server-based java.

The LiLa Portal will thus accept contents – media and experiments – in the form of simplified SCORM objects (LiLa-Objects) and will provide the necessary wrapper to render them in a Portlet-based technology.

2.3 SCORM technology

Sharable Content Object Reference Model (SCORM) [3] is a set of technical standards, specifications and guidelines designed to meet the functional requirements of Advanced Distributed Learning (ADL) initiative. These standards are aimed at e-learning products like Learning Management Systems (LMS). SCORM is exclusively a technical standard, not pedagogical.

SCORM stands for "Sharable Content Object Reference Model":

- "Sharable Content Object": SCORM defines how to create "sharable content objects" or "SCOs" that can be reused in different systems and contexts.
- "Reference Model": SCORM isn't actually a standard. ADL noticed that the industry already had many standards that solved *part* of the problem. SCORM simply references these existing standards and tells developers how to properly use them together.

Functional Requirements

SCORM was designed to satisfy a set of functional requirements:

- Accessibility: ability to locate and access instructional components from multiple locations and deliver them to other locations.
- Durability: ability to withstand technology evolution and/or changes without costly redesign, reconfiguration, or recoding.
- Interoperability: ability to take instructional components developed in one system and use them in another system.
- Reusability: flexibility to incorporate instructional components in multiple applications and contexts.

These requirements make the SCORM technology very useful to the LiLa scenario, because one of the objectives of LiLa is developing a technical, integrated and organizational framework for the mutual exchange of experiments across Europe, enhancing the learning experience of science students.

Technical specifications

SCORM specifies that content should be packaged in a ZIP file and described in a XML file, named imsmanifest.xml (the "manifest file"). The XML file contains all the information the LMS needs to deliver the content, like information about how to launch each SCO and (optionally) metadata that describes the course and its parts.

On the other hand, the run-time specification of SCORM, states that the LMS should launch content in a web browser. The communication with the LMS is via JavaScript, primarily via ECMAScript API, that is provided by the LMS. This API, known in the context of SCORM as the "API Adapter", has functions that permit the exchange of data with the LMS.

Finally, the sequencing specification allows the content author to govern how the student can navigate between parts of the course (SCOs). It is defined by a set of rules and attributes written in XML. These rules allow the content author to determine which navigational controls the LMS should provide to the user, to specify what activities must be completed before others (prerequisites), or to create optional sections, among others.

SCORM versions

SCORM has evolved through the years. There are currently three different implementable versions of SCORM and the latest version (SCORM 2004) has several different editions.

SCORM 1.1 was the first real and implementable version of SCORM, released on January 2001. It was an implementable specification and commercial vendors began to adopt it. it used a Course Structure Format XML file based on the AICC specifications to describe content structure, but lacked a robust packaging manifest and support for metadata.

SCORM 1.2, released on October 2001, was the first version that was widely used. SCORM 1.2 incorporated all of the lessons learned from the early adoptions of SCORM 1.1 to create a robust and implementable specification. It is still widely used and is supported by most LMS today.

SCORM 2004 is the current version. This version has four different editions, but the 3rd Edition is the most used. This version solves many ambiguities of previous versions. It specifies adaptive sequencing of activities that use the content objects. It includes the ability to share and use information about success status for multiple learning objectives. A more robust test suite helps ensure good interoperability.

The table below summarizes the comparison of each standard:

	Widely used	Run-Time	Packaging	Metadata	Sequencing
SCORM 1.1	No	Yes	Yes (weak)	Yes (weak)	No
SCORM 1.2	Yes	Yes	Yes (robust)	Yes (robust)	No
SCORM 2004 3rd Edition	Yes	Yes	Yes (robust)	Yes (robust)	Yes

Section 4 will specify the level of support for SCO that the LiLa Portal will provide.

2.4 Learning Management Systems

A Learning Management System can be described as a software application designed to deliver, track, report on and manage learning content, student progress and student interaction. The most popular LMSs are web-based, in order to facilitate access to learning content and administrations. In LiLa, the work is focused on the standards and technologies related to Web-based LMSs, specifically the SCORM standard to create learning content that can be re-used by different LMSs. Virtual experiments and remote laboratories access will be contained in learning objects based on SCORM, and it will be possible to use these objects in external LMSs, such as ILIAS or Moodle. Note that the LiLa portal will not make use of a LMS, but will rather provide a run-time system sufficient to render LLOs (LiLa Learning Objects), which are a restricted form of SCOs. Neither ILIAS nor Moodle form an integral part of the LiLa portal, though LLOs are sufficiently compatible to SCORM to allow their deployment in such environments.

3 LiLa Portal

3.1 *Portal software Architecture*

The portal architecture used by the LiLa Portal is based on a combination of Sun Web Space Server (Community Edition 10.1) [9] and Glassfish V2.1 [10]. Through this approach we carry forward adopted features like the availability for the upcoming Glassfish V3 and other useful OpenSource Java Technologies. (See next section) The Portal is the main entry point for LiLa. The user authenticates at the portal and get access to the virtual laboratories presented through one or more portlets. These portlets access the SCORM run time system and enable the user to schedule and view her virtual experiments. All scheduling maintenance at server side is handled by a designated scheduling server through REST. Through this approach the whole architecture remains scalable and can be clustered.

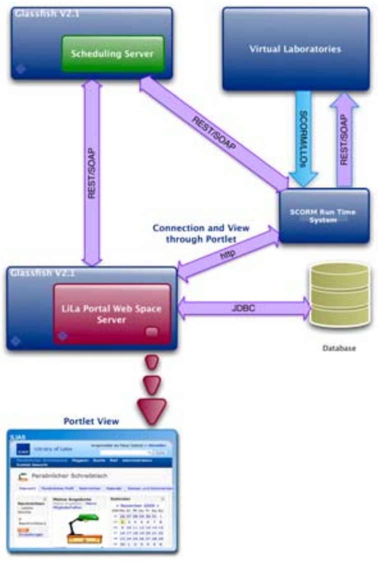


Fig. 1: LiLa Portal Architecture

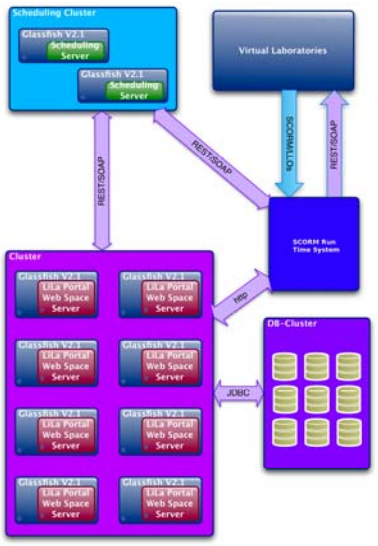
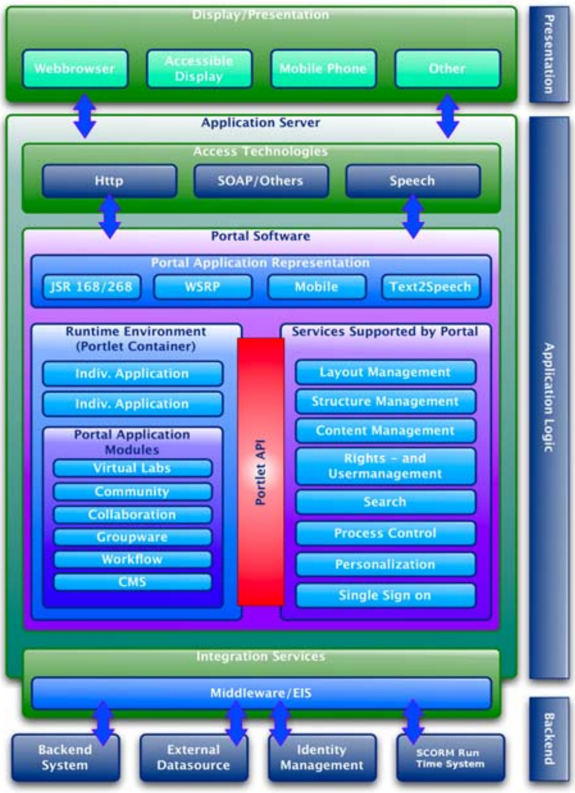


Fig. 2: LiLa Portal Clustered Architecture

Sun Web Space Server supports both Portlet APIs 1.0 and 2.0 and the corresponding JSRs 168 and 268, and the user/developer can even extend the lifecyle through hooks. It is based on the opensource OpenPortlet Container. The JSR 168 defines a portlet as: “A portlet is a Java technology based web component, managed by a portlet container, that processes requests and generates dynamic content. Portlets are used by portals as pluggable user interface components that provide a presentation layer to Information Systems. The content generated by a portlet is also called a fragment. A fragment is a piece of markup (e.g. HTML, XHTML, WML) adhering to certain rules and can be aggregated with other fragments to form a complete document. The content of a portlet is normally aggregated with the content of other portlets to form the portal page. The lifecycle of a portlet is managed by the portlet container. A portlet container runs portlets and provides them with the required runtime environment. A portlet container contains portlets and manages their lifecycle. It also provides persistent storage for portlet preferences. A portlet container receives requests from the portal to execute requests on the portlets hosted by it. The portlet container is not responsible for aggregating the content produced by the portlets. It is the responsibility of the portal to handle the aggregation. Web clients interact with portlets via a request/response paradigm implemented by the portal. Normally, users interact with content produced by portlets, for example by following links or submitting forms, resulting in portlet actions being received by the portal, which are forwarded by it to the portlets targeted by the user's interactions. The content generated by a portlet may vary from one user to another depending on the user configuration for the portlet." Portal federation is supported through Web Service Remote Portlets 1.0 and 2.0 (WSRP 1.0/2.0). Through this feature it is even possible to share the LiLa-Portlets with



other Portlet-Consumers on other Portals than the local.

Fig. 3: LiLa Portal Internal Architecture

Fig. 3 shows the internal architecture of the LiLa-Portal. The portal runs in an application server, here Sun Glassfish V2.1, but every other open source application server is possible, as Sun WebSpace Server supports everyone of them. The application server interacts through standardized protocols like http or xml with the presentation layer (webbrowser, mobile phone, etc.). For the backend connection Glassfish offers a bunch of integration services (e.g. JDBC, LDAP, etc). WebSpace Server is an application which exists inside the application server thus it can take full advantage of all the connection and integration services offered by Glassfish. Inside the portal we have a runtime environment for the portlets called portlet container, which is an extension to the servlet container offered and run by Glassfish (see [1]).

It offers possibilities like persistence, injection and other useful techniques, and manages the whole portlet lifecycle (Init, Render, Action, Destroy). The available portlets in WebSpace Server will provide functionalities, such as collaboration, content management, groupware management, etc. Section 3.2 discusses the functionalities available for the LiLa Portal.

3.1.1 LiLa learning objects (LLOs)

In the LiLa Portal, the SCORM standard is the basis for the definition of LiLa learning objects (LLOs). LLOs will be learning objects that will include all the necessary elements to access to a remote laboratory or to perform a virtual experiment. A LLO is a SCO (SCORM learning object) with additional constraints and extended by additional metadata. Specifically, LLOs should obey the SCORM standard for learning objects, and are packaged in ZIP containers similar to regular SCOs. Typically, a LLO will embed an applet in order to execute a virtual experiment or access a remote laboratory (see section 4.2). Fig. 4 shows the internal components of an LLO and the relations between an LLO, the LiLa Portal and a remote laboratory.

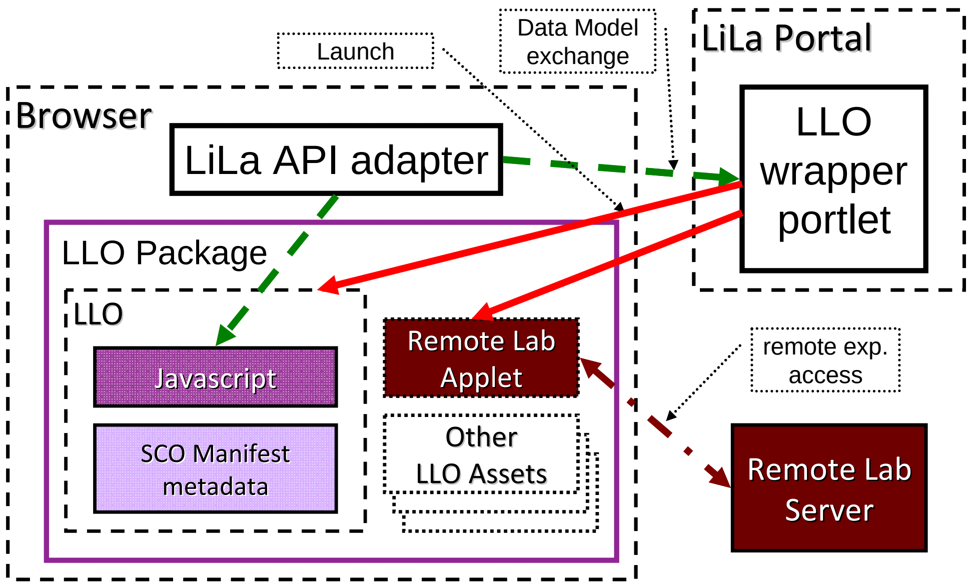


Fig. 4: LiLa Learning Object Components

By an appropriate wrapper, the LiLa Portal can display and include LLOs directly in the portal. To this end, the portal includes an upload functionality for LLOs that allows users to upload LLOs (and not only portlets) directly to the portal. A wrapper portlet, automatically created upon upload, then implements the interfaces the LLO depends on, such as setting javascript variables defining the user name, user ID and the course. Only a subset of the variables defined by the SCORM specification will be available to the LLO, see the specification in section 4 below. In addition, the LLO will use the LiLa metadata set defined in WP4 to identify and classify the content (see Section 5).

To ease the dissemination of LiLa content, the LLO itself is also available for download, and then can be plugged into learning management systems (LMS) at institutions that do not want to link to the LiLa Portal, but rather prefer to run their own eLearning system.

An alternative upload mechanism will allow users to upload the applet code itself, and will provide an input mask for classification of the meta-data; the portal will, from that, create a corresponding LLO that will also be available for download by users (lecturers) of the portal. Finally, this LLO is again wrapped into a portlet and rendered on the portal.

3.1.2 Scheduling

For the purpose of the booking system, a LLO can be classified into two categories:

- Reservation Required LLO (RR-LLO): LLO that requires a previous booking through the portal system, and a way to authenticate and authorize the access. RR-LLOs are relevant for remote experiments that depend on a scheduling system.
- Reservation Free LLO (RF-LLO): LLO that is available immediately without booking. RF-LLOs provide easier access for virtual laboratories/simulation environments that do not require a reservation system.

For RF-LLOs giving access to simulations and virtual laboratories, the wrapper portlet renders the LLO directly whenever accessed. For RR-LLOs, a separate second wrapper portlet first checks the access rights of the user logged into the system, and only delivers the LLO contents to the user if authenticated correctly. Otherwise, an “access denied” page is delivered. It shall be noted that this “first barrier” access control cannot control whether the experiment is accessed in some other way, bypassing the portal. This happens if, for example, the LLO is downloaded from the portal and deployed directly in an LMS, or if the experiment is available on a separate URL anyhow. If desired, a second level security system, to be added to the experiment/laboratory server itself is required, that checks the access rights itself.

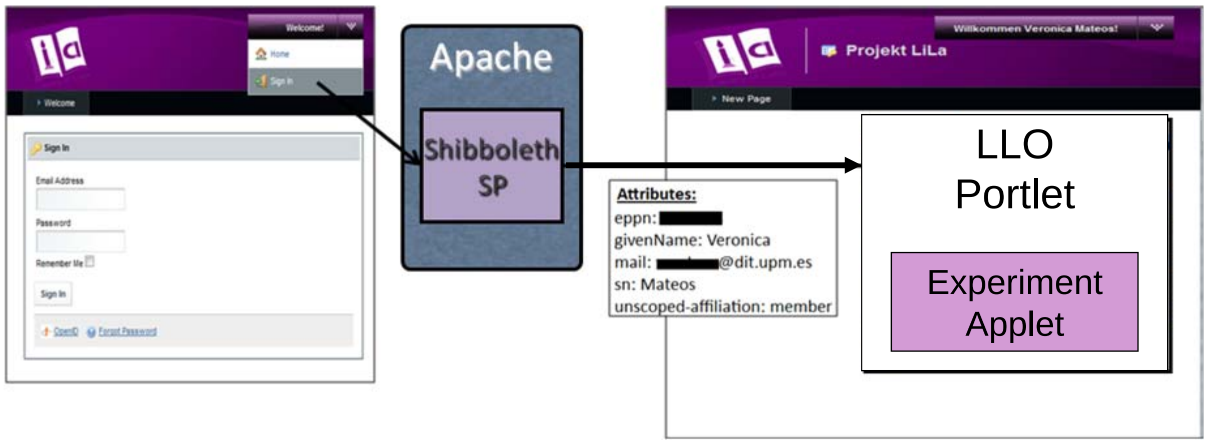
The whole management for scheduling and dispatching of time tickets for the virtual experiments is handled through a Scheduling Server, which is represented through a simple web service (RESTful) on a designated application server. This web service is accessed by the portlet before rendering an “access controlled” experiment and can also be accessed by the experiment core, being part of the LLO, directly to check for proper authentication. With the Portlet API 2.0 and the corresponding JSR 286 it is even possible for two different portlets to communicate with one another through the standard portlet life cycle, which means for example that the user can control the behaviour of different portlets through one central.

By using Public Render Parameters the portlet is able to set global parameters which can enable this behaviour. The Resource Serving mechanism of the Portlet API 2.0 enables the portlet to directly serve multimedia content like movies or experiments even as applets. This approach is more flexible for the user because the main work is done on the server side and presented as web content even with Web 2.0 features like AJAX (like in a portlet, only a part from a web page is updated and not the whole). Unlike the applet approach, the user does not have to be afraid of some bad security settings for the applet sandbox. But for later usage a mix of both approaches might be desirable, because of some multimedia intense experiments which might overload the server even in cluster mode.

3.1.3 Authentication and Authorization Architecture

As explained above, students access remote experiments and virtual laboratories through a special type of SCORM objects called LLOs. Shibboleth [5] is the authentication and authorization technology used in LiLa to control the access by registered users to the portal as well as to control the access to experiments and remote laboratories.

As regards authentication and access control, the general portal software architecture is shown in figure below.



Attributes:
eppn: [redacted]
givenName: Veronica
mail: [redacted]@dit.upm.es
sn: Mateos
unscoped-affiliation: member

Fig. 5: Shibboleth Authentication in the LiLa Portal

The general process is the following:

- A user goes to the home page of LiLa Portal. In this page the user can see some portlets, which are described in the next section.
- To access the LMS and experiments, the user has to log in the portal and go to LiLa Project page. *Sign in* functionality allows user to log in.
- User authentication is done using Shibboleth. As a result of the authentication process, the Shibboleth's Service Provider (SP) obtains a set of session attributes. These attributes are used to allow or deny user to access to protected resources. Also, the attributes could include information for authentication and authorization related to the LLOs which are rendered by using a portlet (see *Section 4.4*).
- If a user is shibboleth authenticated and she has the required permissions, she will go to the LiLa Project page, where she will see the portlets giving access to experiments and will be able to access them.

3.2 Portal functionality

LiLa can take advantage of the different portlets available in WebSpace Server, providing functionalities such as collaboration, content management, groupware management, etc. This enables community leaders or users with the privileged rights to compose their for example faculty own portals including other web content. The user can also install her own portlet application as it is planned with our upcoming virtual labs portlet. Through the portlet API we can take full advantage of the services supported by the portal. So the whole authentication at the LMS (ILIAS) can be handled through built in functions of the portal software (Rights-and User Management or Single-Sign On). The user can personalize and individualize her portal layout and structure as granted through the role management, including setting up the virtual labs portlet according to his personal feeling.

By using portlets for views the portal also offers a wealth of Web 2.0 technologies for less user browser computation time and bandwidth usage. AJAX for example, mainly used for the portlet view as only little parts of the view are rerendered, reduces the whole bandwith. Only these elements of the view, which changed, are sent from the server to the users browser for rerender. The whole portlet API also enables the portlet to be portable from one portal software to another supporting JSR 168(/268). Web Services for Remote Portlets enable the portlet user to take advantage of remotely provided portlets (as the LiLa-Portlet could be).

The general use case of the LiLa-Portal is described as followed:

1. The user enters the portal site and logs in (or is automatically logged in through a Single Sign-On service).
2. The user enters the desired page.
3. The user enters the virtual labs portlet (also called LiLa-Portlet) and is
    a. Automatically logged in through the users right management of the portal.
    b. Redirected to a login-screen view for the desired LMS (here ILIAS).
4. After login the user is provided with the actual experiment offerings, scheduling possibilities and other supported features (see LLOs in section 4).
5. After choosing some actual available experiments the user enters the virtual lab and
    a. Is provided with some media or interactive content which could be displayed in the browser.
    b. Is provided with an applet which represents the actual virtual lab choice.
6. The virtual labs session is over and the user logs out.
7. The session parameters persist inside the portlet for the next visit.

The LiLa Portal contains the following elementary services:

- Rendering of content. Content is uploaded in the form of LLOs (LiLa-Learning Objects, see below), which is then, on upload, unpacked and wrapped again by auxiliary Java classes to form a valid portlet. The advantage of this approach is that LLOs can be directly deployed in all SCORM compliant LMS.
- Uploading of content: Content in the form of LLOs can be uploaded at the portal and then enters the LiLa database. Metadata contained in the database is used to classify the content accordingly. An additional upload mechanism for raw applets will be created which collects the necessary metadata to create a LLO from the applet internally.
- Downloading of content: Teachers interested in LiLa content may simply download the uploaded LLOs and deploy them directly in LMS. Depending on the content, some functionality might then be lost, e.g. the ability to book experiments.
- Searching of content: LLOs contain additional metadata that is used upon upload of a LLO to generate entries in a database. This database allows then users to search for content they need. The metadata set is here defined by WP 4, but specifically contains information on the scientific field of the experiment, its audience, usage conditions and origin. This meta-data will be denoted as “static metadata” in the following as it does not change over time.
- Annotation of content: Users, i.e. students and teachers, will be enabled to annotate content, let it be to enrich content by additional data how to use and deploy it, to embed it into a pedagogical design targeted at a specific audience, or to simply rate it for usability or quality. Such metadata will typically not leave the portal itself, and is not part of the LLOs.
- Authentication of users: Even though LiLa aims at providing the majority of experiments for free without authentication, some experiments may require scheduling through the booking system or authentication of users. To this end, the portal must provide at least two different types of LLO-to-portlet wrappers: An unrestricted simple wrapper, and a wrapper that checks for proper scheduling and authentication of users, and does not deliver the LLO contents otherwise.
- Authentication of accesses to experiments: Even though the portal checks itself whether an experiment has been scheduled for a specific user in a specific time-span, experiments downloaded as SCOs and installed elsewhere bypass this mechanism and would, without further activity, become accessible directly. To address this issue, the portal provides an additional restful service (i.e. based on the REST paradigm) that allows experiments to check whether they are currently booked. Experiments that absolutely require an additional security layer then need to contact the portal themselves and check for their availability. This step is otherwise hidden in the portlet built automatically around the LLO.
- Pedagogical embedding of content: That is, the portal must provide a framework that allows to design a didactically valuable course around the “raw” experiment on

the portal. A content page as delivered by the portal should hence contain more than just the experiment, but also the motivation, the background, use-cases and recommendations for optimal deployment.

4 LiLa learning objects (LLOs)

4.1 General description

A LiLa Learning Object (LLO) is a SCO (SCORM learning object) with additional constraints extended by additional metadata. Specifically, LLOs should obey the SCORM standard for learning objects, and are packaged in ZIP containers similar to regular SCOs. Note that a SCO package contains **at least** the manifest file, called imsmanifest.xml which again contains references to all further resources. Specifically, it may, and in case of LLOs, it must contain a HTML file that renders the content of the laboratory(see below). The following additional requirements and restrictions apply:

- May only render a single web page, i.e. may contain only a single SCO with only a single href (html-file), i.e. no complete courses.
- The SCO shall be complete, i.e. shall include all assets to run the experiment. For example, it should contain the html page rendering the content, the applet .jar file rendering the content and potentially additional metadata required by the applet.
- Shall contain the WP4 metadata set in the metadata set annotating the LLO, i.e. this metadata shall go between the <resource> tags of the single LLO contained in the package.
- Only a limited set of run-time functions and run-time variables are accessible for the LLO:
    - LMSInitialize() and LMSFinish()
    - LMSGetValue() with only a limited set of variables available:
        - cmi.core.student_id: an ID to uniquely identify the learner
        - cmi.core.student_name: a natural language name of the learner
        - cmi.learner_preference.language: the preferred language of the learner
        - cmi.interactions.n.timestamp: point in time at which the experiment was launched (for the booking system)
        - cmi.suspend_data: Opaque data stored by the portal/LMS
        - lila.scheduling.url: If existing, points to the URL of the booking system to check for availability of the experiment (extension)
    - LMSSetValue(), though variables set by it are ignored except for
        - cmi.suspend_data: Opaque persistent data stored at the portal
- In addition to the metadata developed in WP4, the following two meta-data elements shall also be present (to be discussed with WP4):
    - A metadata element that annotates whether a booking system is required

- A metadata element that identifies the URL of the booking system to be used

If no booking system is required, these metadata elements may be missing.

4.2 Applets in LLOs

The typical access path to an experiment is to embed an applet into a html page and deliver this as a SCO to the portal. A LLO object may, in addition, also contain "traditional" passive content in the form of pdf files, video clips or audio streams. The type of the data is then identified by the metadata describing the SCO.

In case the content consists of an applet providing access to the laboratory, the following options shall hold:

- First, the `imsmanifest.xml` file shall contain a reference to a resource consisting of an html page. To simplify the portal integration, this html file shall be named `applet.html`.
- The applet.html file shall be well-formed html containing a <`applet`> tag. This applet tag shall be complete and well-formed and shall contain all the parameters necessary to run the applet. Note that LLOs may have limited access to javascript variables that may be used by the html page to pass parameters from the portal into the applet. Such parameter parsing can be done either in the `param` arguments of the applet. Note that SCO requires that the corresponding applet code, e.g. a java .jar file, is indicated as a resource in the `imsmanifest` file. This resource – the applet code itself - shall be found in the "lib" directory of the SCO package.
- In case the applet is a java applet, parameters may also be passed into the applet by means of the LiveConnect technology that allows javascript code to invoke java classes. Note that in the latter case the java method to be called **must** be a member of applet object to be rendered, due to a restriction in the internet explorer. For details on LiveConnect, see [11].
- Even though access to the applet itself is only granted by the portal if the user has been authenticated successfully, applet authors might want to implement a second-level access test at server side. This second level security test prevents users from simply downloading the HTML code rendering the content, and using this HTML code to gain access to the laboratory bypassing the access procedure of the portal. To enable the server-side security test, the LiLa portal provides a set of Java script variables including the user name and the URL of the booking system server; the contents of these variables can be forwarded to the applet using either the LiveConnect or any other suitable parameter passing mechanism. Note that it is up to the author of the applet code and the SCO to implement such mechanisms, and the LiLa portal interfaces only provide the framework to enable such tests, though does not provide recommendations for their optimal deployment.

- In case the content is rendered in a 3D virtual world using the Wonderland technology, the content shall contain a link to a jnlp (java webstart) download found on a server hosted by the institution running the laboratory. For more information on java webstart, see [12]. To ease the portal design, a web-page containing a link to a webstart application shall be named `webstart.html` and shall be indicated under this name in the `imsmanifest` file.
- For passive media such as text, PDF, audio or video streams, the web page shall contain a link to this media. Depending on the size and complexity of the media, media may be either an integral part of the SCO, or may be located on a server of a participating institution. In the former case, the media file(s) shall be placed into the "lib" directory of the SCO and shall be indicated as such in the `imsmanifest` file. Html files containing only media and no applet shall be called `media.html`, and shall be indicated as a resource of this name in the `imsmanifest`.

4.3 *Including metadata in LLOs*

Metadata describing LLOs follow the WP4 metadata set (including the data describing the Metadata describing LLOs follow the WP4 metadata set, including the data describing the necessity of having to book the experiment, and the URL of the booking server to contact. The metadata is to embed into the imsmanifest.xml file of the .zip package, specifically into the SCO section in the `<resource>` property of the XML file in a `<metadata>` tag. Note that such an extension creates only a conforming, but not a strictly conforming SCORM package.

If applicable, metadata elements of the SCORM standard, namely LOM (IEEE LTSC Learning Object Meta-data standard), having a meaning similar to those defined by WP4 may be added if those elements are optional, and shall be added if such metadata elements are defined to be mandatory by the SCORM standard, but are otherwise ignored by the portal. For details on SCORM metadata, see [13]. LLO metadata has to be represented in XML (i.e. using the DC/XML encoding) and have to be placed in the <`metadata`> tag of the imsmanfest, but outside of the `<lom:lom>` metadata.

4.4 *LLO Authentication and authorization*

The LiLa Portal is able to check the schedule and user permissions for experiments that are contained in a LLO as far as the LLO is launched in the LiLa Portal itself. However, as it has been mentioned in section 3.1.2, downloading LLOs to a LMS would bypass the mechanism used to check the availability of a reservation for an experiment by a user. This section discusses the authentication and authorization mechanisms used in LiLa and possible solutions to overcome this problem.

The LiLa Portal uses Shibboleth for authentication and authorization, which for the purpose of the following description means that the portal will need to have a Shibboleth Service Provider installed, which will be in charge of redirecting the requests for AA to a Shibboleth Identity Provider, that can be located elsewhere.

One possible solution to "export" the LiLa authorization and scheduling infrastructure of experiments to other systems is to both a) integrate scheduling information and Shibboleth, and b) include some way of checking AA and scheduling information within the LLOs.

Fig. 6 depicts the scenario in which Shibboleth is used to perform User Authentication and Authorization, while the authorization related to a LLO is carried out by a specific "Token based LLO Authorization Server" (TLAS).

A LLO includes some information or security token, the LLO AA Token, for example in the SCO Manifest *adlcp:datafromlms* data element. LiLa could update *adlcp:datafromlms* "on the fly", if needed, when the LLO is downloaded to be included in an external LMS. A LLO is designed to get this information before launching the Remote Laboratory Applet, by retrieving the value *cmi.launch_data*.

Once the LLO AA Token is retrieved, the LLO will contact the TLAS using the token and in return, the TLAS will provide some piece of information which is needed to start the virtual experiment / remote laboratory applet. This piece of information will be passed as a parameter to the applet when launched. In the case of a remote laboratory applet, the parameter could be the url needed to contact the remote laboratory server, which might include authorization parameters depending on the level of integration of the remote laboratory with LiLa authorization and scheduling mechanisms. In the case of a virtual experiment, in which the applet is does not need access to a server, the parameter could be some access code needed to unlock the virtual experiment. The communication between the LLO and the TLAS will be based on AJAX, using the XMLHttpRequest object.

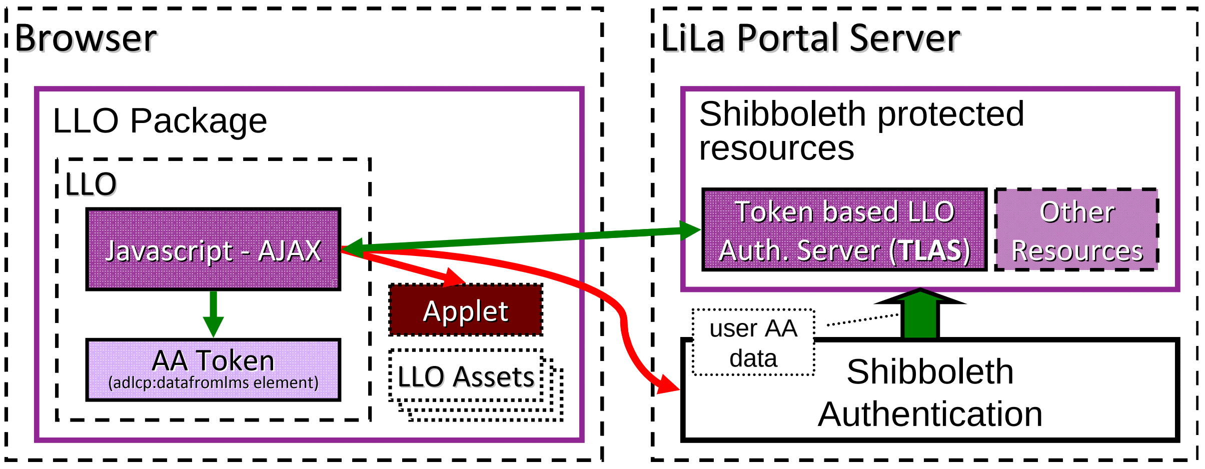

Browser
LLO Package
LLO
Javascript - AJAX
AA Token
(adlcp:datafromlms element)
Applet
LLO Assets
LiLa Portal Server
Shibboleth protected resources
Token based LLO Auth. Server (TLAS)
Other Resources
user AA data
Shibboleth Authentication

Fig. 6: Authentication and Authorization for LiLa Learning Objects

The task of the Token based LLO Authorization Server (TLAS) will be aided by Shibboleth. This server will be accessed through an Apache httpd server and protected by a Shibboleth Service Provider. This means that the TLAS will be able to perform the LLO authorization task using the user AA data provided by Shibboleth. This will make it possible to have an authorization scheme where a LLO is only authorized if the user belongs to a certain organization that has obtained the access rights to the LLO. In this schema, the AA token includes information that links the LLO and the organization.

In the case of a RR-LLO (see section 3.1.2), the TLAS could also be linked to the LiLa scheduling infrastructure, in order to only give access to the applet in case there is a valid reservation for the user. This would benefit remote laboratories that could use the LiLa reservation system even when the LLOs are used in external LMSs.

Other alternatives have been considered. For example, AA attributes and scheduling information could be provided exclusively through Shibboleth. However, this would mean that in order to take advantage of remote laboratory reservations, external LMS would need to include a Shibboleth Service Provider.

Another alternative is to include the functions to check AA and scheduling status in the LMSInitialize() function that will be provided by the LiLa API Adapter in the LLO wrapper portlet. In this case, a LLO does not need to include any specific javascript code to perform the authentication and authorization, but when it is used in external SCORM compliant LMSs systems, no AA status check would be made unless the LMS own LMSInitialize() function is also modified. This could be an interesting alternative, since some of the most popular LMS are open source and LiLa could integrate the LLO AA and scheduling mechanism in LMSs such as ILIAS or Moodle.

5 Links to other workpackages

There are a number of interactions with the rest of LiLa workpackages that have an effect on the work of the workpackages:

- The main concerned workpackage is WP4, since this document is describing how to interface with the different contents that have to be provided by WP4. So, the software architecture provided in this document requires that the virtual laboratories and remote experiments are provided as LiLa Learning Objects (LLO), which are an special SCORM encapsulation comprising:
    - The specific applet or web content that run the virtual laboratory or provide access to the remote experiment.
    - Some metadata that identify the experiment and that will be accessible by WP3 tools for searching and cataloguing the contents.
    - Access control functionality for interacting with the Virtual Portal authentication components in order to force that the experiments are only executed by the adequate authorized students.
- Work Package 3 is also affected with this specification, since most of the components and tools that should be designed in WP3 have to be executed and integrated in the Virtual Portal. So, it will be needed to design the tutoring system components as different portlets that will be integrated in the virtual portal, interfacing also the content metadata by means of the SCORM interface.
- Work Package 5 is affected for the technical evaluation, since the chosen approach in WP2 based on portlets technologies can have some effects on the quality of experience perceived by the students, in terms of performance, accessibility, etc. WP5 needs to address the evaluation of this quality of experience taking into account the technological decisions provided in this document.

References

[1] Alejandro Abdelnur, Stefan Hepper. JSR 168: Portlet Specification, Java Community Process. 2003. http://jcp.org/jsr/detail/168.jsp

[2] Hepper, Stefan. JSR 286: Portlet Specification 2.0. Java Community Process. 2008. http://jcp.org/jsr/detail/286.jsp

[3] Advanced Distributed Learning Initiative, Sharable Content Object Reference Model (SCORM) Version 1.2. The SCORM Content Aggregation Model. 2001. http://www.adlnet.org

[4] ILIAS (Integriertes Lern-,Informations- und Arbeitskooperations-System) Learning Management. http://www.ilias.de/docu/.

[5] Shibboleth, A Project of the Internet2 Middleware Initiative. http://shibboleth.internet2.edu/

[6] CAS - Central Authentication Service - JA-SIG Wiki. http://shibboleth.internet2.edu/

[7] Remote Authentication Dial In User Service (RADIUS). IETF RFC 2865. Management. http://tools.ietf.org/html/rfc2865

[8] MOODLE (Module Object-Oriented Dynamic Learning Environment) Learning Management. http://moodle.org

[9] GlassFish Web Space Server, Sun Microsystems. http://www.sun.com/software/products/webspace/index.xml

[10] GlassFish v2.1.1, GlassFish Community. https://glassfish.dev.java.net/

[11] JavaScript to Java Communication (Scripting). http://java.sun.com/j2se/1.5.0/docs/guide/plugin/developer_guide/js_java.html

[12] Java SE Desktop Technologies - Java Web Start Technology. http://java.sun.com/javase/technologies/desktop/javawebstart/index.jsp

[13] IMS Meta-data Best Practice Guide for IEEE 1484.12.1-2002 Standard for Learning Object Metadata. http://www.imsglobal.org/metadata/mdv1p3pd/imsmd_bestv1p3pd.html