

Implementation of a Learning Design Run-Time Environment for the .LRN Learning Management System

Jose Pablo Escobedo del Cid, Luis de la Fuente Valentín, Sergio Gutiérrez,

Abelardo Pardo, Carlos Delgado Kloos

Telematics Engineering Department
Carlos III University of Madrid
Av. Universidad, 30 E-28911 Leganés,
Spain

<http://gradient.it.uc3m.es>

Abstract: The IMS Learning Design specification aims at capturing the complete learning flow of courses, without being restricted to a particular pedagogical model. Such flow description for a course, called a Unit of Learning, must be able to be reproduced in different systems using a so called run-time environment. In the last few years there has been several tools implementing such functionality, but most of them are standalone and do not benefit from the integration with a Learning Management System. This paper describes GRAIL, an IMS Learning Design run-time environment supporting all three levels of the specification and fully integrated with the .LRN Learning Management System. The paper describes its functionality, explains how each level is implemented, shows the differences with other CopperCore based environments, and analyzes its integration with the rest of .LRN services and other specifications such as QTI and SCORM.

Keywords: Learning Design, services integration, .LRN, run-time environment

1 Introduction

The interest on capturing the ideal conditions for an effective learning in a formal paradigm capable to be reproduced in different instances led to the appearance of the so called "educational modelling languages". The objective was to describe learning as a process, that is, to include the different organizational aspects, resources, persons involved, etc.

The Learning Design Working Group within the IMS Global Consortium (henceforth simply IMS), started to work on a previously proposed language called EML which evolved to become the IMS Learning Design specification version 1.0 approved in February of 2003 (henceforth IMS-LD). As argued in Koper and Tattersall (2005) IMS-LD is based in the so called "script" metaphor where the learning process description is done in terms of the elements present in a theatrical play, a film, or a game. The components of a script capturing a learning process are: metadata, roles, acts, environments, role parts, sequence of activities and conditions. The description of a process in terms of these elements is generically known as a "unit of learning" or UoL.

But as with other specifications prior to this one, applications and learning management systems (LMS) need to make use of it. In the case of Learning Design, this usage quickly shaped into two related but clearly different aspects: how teaching staff would produce descriptions of their learning processes in IMS-LD (authoring problem), and how current LMS would support such descriptions (support problem).

The main challenge in the authoring problem is how to offer an intuitive interface of a non-trivial specification for any member of a teaching staff, a learning designer, without compromising its expressive power. The script metaphor is powerful yet it contains a rich set of mechanisms to capture as many different learning processes as possible. The solution points in the direction of powerful editing tools that support this creation process and help capture its structure with a simplified interface.

The support problem, on the other hand, assumes that UoLs have been properly produced by the learning designers and they should be deployed in an LMS and "executed" an arbitrary number of times with potentially different users. This paper is entirely devoted to this second problem. It presents the implementation of a run-time environment (RTE) capable of interpreting a previously produced UoL using the IMS-LD specification within the .LRN learning management platform.

As it is the case with all the specifications, they include barely no guidance as to how to implement an RTE capable of understanding the UoL and replicate the process captured by an author. This document presents the design decisions and structure of GRAIL (Gradient-lab RTE for Adaptive IMS-LDs in .LRN). GRAIL supports all three levels (Level A, B and C) of the specification. Aside from describing how the basic infrastructure is deployed to show students the material, control navigation, activity termination, etc. additional aspects required a more in depth analysis to provide an effective and intuitive solution. UoL/role management and service integration are aspects that required careful study in order to provide an interface suitable to handle an arbitrary number of instances of a large number of UoLs and make full usage of the services already available in .LRN.

Currently, GRAIL is being deployed and has been used for several test cases in higher educational institutions and is gradually being deployed in regular learning processes.

The rest of the document is organized as follows. Section 2 includes a brief recount of current initiatives in the area of IMS-LD RTEs. A description of the structure and capabilities present in .LRN is included in Section 3. Section 4 describes the architecture of the RTE as well as how its main features have been implemented. Section 5 covers how to integrate different services already present in .LRN with GRAIL. The specification offers guidance for only a subset of them, and the more generic issue of how any service is integrated is covered. The paper concludes with Section 6 drawing some conclusions and pointing future work.

2 Related work

Since the publication of version 1.0 of the IMS-LD specification, several tools appeared both in the context of UoL production and execution. For the sake of brevity, It follows a summary only of the most relevant ones.

CopperCore [1] is the reference implementation of an IMS Learning Design engine and provides also a player based on such engine. The objective was to provide a source of background information for implementing other LD compliant players rather than for adoption. In practice it has been seen as a reference player and a test bed for running designs and validating other LD tools such as editors, authoring systems, etc.

The CopperCore goals included testing if the API design allowed a thin client to be built on top of the engine at a low cost; and giving an indication of expected IMS-LD players functionally (i.e. if it is not supported by CopperCore, there is no need to support it). Performance and scalability are not looked for by CopperCore, although they are expected to be one of the goals of production-level IMS-LD players. Additionally, every player looks for additional functionality that might be relevant for learning designs even if they are not part of the reference implementation: integration with other IMS specifications, an intuitive method for assigning users to roles, importing/exporting UoL functionality, more appealing user interfaces (in CopperCore, administrative tasks like user assignation to roles is performed with a command-line utility called CLICC), connection to external services (e.g. student and portfolio management), etc.

SLeD (Service-based Learning Design system) is a CopperCore based player that separates the player's functionality from the underlying engine, as depicted in Weller, Little, McAndrew and Woods (2006). It focuses on the use of web services: communication between the engine and the player is performed using web services, additional end user tools (e.g. conference systems) were built oriented to web services. SLeD presents more functionality than the basic CopperCore engine. For instance, the CopperCore engine is only able to indicate that a learning design needs a forum, while SLeD shows the forum, running it through the conferencing service.

Another CopperCore based player is the one included in the Reload project [2]. The project provides a family of tools related to IMS specifications, from the widely used IMS-CP packager to an IMS-LD editor. Its implementation is based on the Java and JBoss platforms. Its functionality include: import/removal of Learning Designs into the CopperCore engine with a simple graphical interface, automatically reading a learning design and populating the engine with a default run and active user for every role found within the manifest (custom users can be added too), etc.

3 The .LRN Platform

.LRN [3] is an enterprise-class open source platform for supporting e-learning and digital communities. The tool was originally developed at the Massachusetts Institute of Technology as a virtual learning environment, and it then evolved into a comprehensive platform including not only e-learning support but also generic web resources.

The platform is based in the Open Architecture Community System (OACS) [4**Error! Reference source not found.**], a toolkit for building scalable, community-oriented web applications. The toolkit structure is highly modular and .LRN is a set of modules that provide the additional features to deploy an e-learning environment.

OACS (and therefore .LRN) is tightly integrated with a relational database, both PostgreSQL and Oracle are currently supported. The web server that handles requests at the basic level is AOLServer [5**Error! Reference source not found.**], the America Online's open source web server. One of its main features is the integration in its core of a multi-threaded TCL interpreter which provides an effective solution for industrial strength type of services such as those present in large higher educational institutions.

As in most open source projects, there is a community around .LRN/OACS involving nearly 11,000 registered users. The community portal is itself based on this platform and coordinates the interaction between developers, users, technical personnel employed by higher education institutions and anybody interested on exchanging ideas, solutions and information about the tool.

The following institutions are a reduced sample of the type of environments in which this platform is currently being used (see [3**Error! Reference source not found.**] for a more detailed list including case studies):

- **MIT Sloan School of Management .LRN** hosts over 11,000 users and around 3,000 concurrent sessions. Estimates shown that .LRN was deployed and maintained with roughly 25% of the cost compared to solutions based on commercial software.
- **Harvard Univ. Executive Education Project.** An e-learning platform was required to provide the best combination of flexibility, enterprise-class foundations, strong user base, and cost-effectiveness. After deployment, the project director acknowledged that .LRN provided "a huge head start toward what we wanted, plus support for things we didn't originally anticipate".
- **Vienna Univ. of Economics and Business Admin.** It is one of the largest .LRN instances. It serves around 20,000 users, contains 26,000 learning resources, and averages 600 concurrent connections.
- **Universidad de Valencia.** This university required a platform to support its conventional teaching for 40,000 users. After surveying platforms such as Moodle, ATutor, WebCT and ILIAS, their choice was .LRN due to its combination of scalability and extensibility.

Several features make .LRN an effective and powerful e-learning platform. Its modular structure allows for very fast customization and prototyping of new applications. The user space is organized through a customizable set of portlets, each of them offering access to one of the various services

available. The underlying OACS toolkit provides an ever increasing set of web functionality most of them suitable to be adopted by the e-learning platform.

The fact that OACS is a community-oriented toolkit has influenced and shaped .LRN into what it could be called a "communication oriented LMS". Most of the current LMS, focused at the beginning of their existence on providing content management for teaching staff and learners. .LRN, on the other hand, was conceived as a platform to facilitate communication among all the different actors in a learning experience.

Just as an example, ever since the first release, each .LRN user has a web folder shown in the login page to include both private as well as publicly accessible files. Each community of users has also its own area to exchange documents. Also derived from the community-oriented nature of the tool, there is a powerful user management model with rich functionality to handle groups of users and permissions.

Another differentiating feature of .LRN is its comprehensive notification mechanism. The underlying data model is object oriented, and each object may have a notification attached to it. Each time an object is modified, the notification information is processed and the proper email messages are sent. The user may choose to receive a notification whenever the most relevant objects in the platform are changed. This is especially effective for forum messages, shared files in a community, appointments, etc. The platform also allows each user to choose among instant, hourly or daily batched notification, providing a very effective mechanism to interact with the rest of the users.

But aside from these features, .LRN offers support for the most common specifications in e-learning. The SCORM player allows for the upload of course material stored using this format. If the administrator uploads a zip file with a SCORM package, the system installs its content in the common file storage area of a class or a community. A portlet in the student area shows the links to enter a special screen to visualize its content as well as its index. This SCORM support offers the teaching staff to see the percentage of course material covered by each student.

Tests and quizzes are also fully supported in .LRN through the IMS Question and Test Interoperability format [6]. Exam questions may be uploaded in this format as well as managed through the editing capabilities of the platform. Exams present inside a SCORM package are handled seamlessly by the tool by invoking the rendering engine and showing the content to the student. Teaching staff may manipulate both exam content, results and statistics within the platform.

A large number of educational institutions have SCORM support as one of their requirements. Full scale experiences, from content production to course management are being performed at several higher education institutions that together with the user community contribute to constantly increase its robustness as well as its functionality.

The latest official release of .LRN, fully supports the IMS-LD specification. The provided run-time environment, called GRAIL, supports all three levels (Level A, B and C). There are several pilot experiences under way that are showing how the language offers a simple vehicle to take full advantage of the functionality offered by the platform.

4 Run Time Environment Design

GRAIL (Gradient-lab RTE for Adaptive IMS-LD in .LRN) is the IMS-LD run-time environment implemented in .LRN. It has been conceived to be used within the context of a .LRN community, a set of users sharing resources such as documents, forums, calendar, schedule, etc. A regular course is simply an instance of one of these communities. As it is usually the case, and IMS-LD is no exception, specifications do not include details on how the RTE must be implemented. This usually translates into a wide set of decisions that need to be taken by the design team. They relate to important aspects of the usability and effectiveness of the run-time environment and therefore need to be carefully considered. It follows a brief description of the main aspects taken into account when designing GRAIL. This description is given separately for each one of the three levels of the specification.

Level A: The core engine

Level A is the core of the specification. By supporting this level, a mechanism to statically sequence content is provided. Even though the capability of dynamically (that is, while the UoL is being used) deciding which content to show is include in Level B, there are some other important issues that need to be properly managed at this level such as: roles, environments, plays, acts, role parts and services.

The starting point for the RTE is the import of a previously packaged UoL. The users simply uploads the file containing the unit and the platform immediately parses its content, divides it into functional fragments and stores its content in the relational database. The implementation philosophy behind .LRN/OACS includes a tight integration with the relational database. GRAIL was designed following also this philosophy, so all operations are performed from/to such database. This organization has the advantage of facilitating a simple integration with typical administrative functionality already present in the platform. As an example, simple operations over a UoL -such as renaming it, modifying its description, etc. - are implemented based on the generic functionality to edit elements in a database table. Other non-trivial operations such as monitoring the usage of the UoL become very easy to implement once the information is stored in the relational database.

Once the community (or course) administrator has uploaded the file containing the UoL, an administrative page is shown (see Figure 1) allowing the creation of new instances or "runs".

Unit of Learning Name ▲	Creation Date ◆		
Geo-Quiz-3	07/24/2006 16:40	create new run	
Quo Builder	07/24/2006 16:42	create new run	
What is Greatness?	07/24/2006 16:44	<i>Deleted</i> (make it live)	

Figure 1: Administrative portlet for UoL management

The platform needs to support for several instances or "runs" of the same UoL simultaneously running in a course. This functionality translates into a database structure in which common information about the UoL is stored in a set of global tables, whereas a second set of tables are replicated with the information specific for each run. Instantiating a run means creating a new set of tables with the proper initial values. Also, with each run appears the notion of its status. A run may be in three possible states: role-management required, active or finished. After a run has been instantiated, it requires for its roles to be defined. This step is described in detail later in this section. When users are actually visualizing the UoL, its state is active. Once all users finished the unit, its state is finished.

Figure 2 shows the portlet with the status information for all the runs of a given community or course.

Run (IMS-LD) Name ◆	Status ◆	Creation Date ▲		
Candidas. The great unknown (II)	✓	03/06/2007 13:04	View members Monitor	
Geo-Quiz-3	▶	03/06/2007 13:05	View members Monitor	
Quo Builder	🕒	03/06/2007 13:05	Manage Members	
Candidas. The great unknown (II)	▶	03/06/2007 13:06	View members Monitor	

Figure 2: Administration page with the list of runs in a community

A significant part of the IMS-LD specification is based in the use of roles, therefore, role management is one of the most relevant issues to be addressed when implementing a run-time environment. The .LRN platform contains a very powerful user/permission model which lent itself nicely to the implementation of role management. A user in IMS-LD is directly mapped to a user in .LRN. Therefore, all users within a community (with their permissions) are automatically considered as potential users of a UoL. The player creates a .LRN user group containing all users in the community when the UoL is instantiated. The two reasons for creating this new group are to guarantee that no modifications are performed during the use of the UoL (as stated in the specification) and to simplify permission management within the run.

Once this user group has been created, its users need to be associated with one or more role instances derived from the roles included in the UoL. This is a mandatory step before the UoL can be used by regular users. A role may have several role instances derived from it. These instances are mapped to the concept of sub-groups within .LRN. Group/sub-group support within .LRN also considers nesting, thus offering the perfect match to implement role nesting within the users of a UoL. More precisely, for each newly created run, the administrator assigns users to the needed role instances as well as nested instances. Once the process has finished, the UoL is suitable to be started moving into the "active" state.

Depending on the role structure and the number of students in a course, the task of creating role instances and assigning users to roles can be confusing. Providing an intuitive interface for this task was identified as an important requirement for GRAIL. Figure 3 shows a screen capture of the user interface for this task. The interface shows restrictions for role creation such as maximum or minimum number of students per instance. This information is directly obtained from the UoL description.

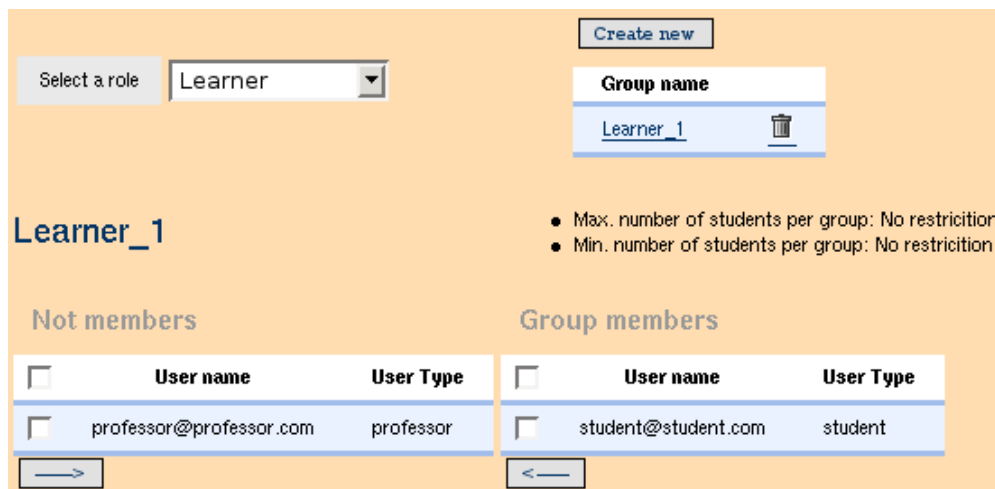


Figure 3: Roles administration page

According to the specification, in Level A, users only have access to a subset of the activities and services depending on the completed activities at a certain point in time for each role instance. When the manifest of the UoL was parsed, all activities and resources, aside from the relational database, were also stored in the document repository of the community or course. Thus, access to these resources needs to be consistent with the described policy in the UoL. Users may access to a subset of them depending on their location in the course activity tree. This is where the .LRN permission scheme becomes useful. Since the structure of the relational database in .LRN is object oriented, each stored item has a unique object identifier, and the permission system granularity allows for the definition of different permissions for each object. These objects are created when the UoL is initially parsed and the initial permissions allow no operation by regular users. When a user interacting with an active UoL reaches certain activity, the permissions of its corresponding object (documents, resources, services)

are modified, and access is granted. The group/sub-group structure in .LRN allows this same technique to be used when a set of resources needs to be made available to a role or sub-role.

Level B: Managing properties

Level B extends the specification to include properties and conditions that are dynamically updated and evaluated with values obtained during the execution of the UoL. By using properties, a learning design author may capture the state of the UoL at a certain point. By using conditions, it is possible make the UoL behave differently, for example showing different activities, when properties change their value. This level also defines the behaviour of the *imsldcontent*. An *imsldcontent* resource is a XHTML document containing special elements to visualize and obtain a property or a group of property values. These elements are included in the document through the use of a different namespace.

Other IMS-LD run-time environments such as CopperCore [1] use a finite state machine (FSM) model to evaluate properties and conditionally sequence content. For each imported UoL, a FSM is created and an activity tree is attached to each state and shown to the user when the state is reached as shown in Vogten, Tattersall, Koper, Van Rosmalen, Sloep and Martens (2004).

As described in the specification, condition evaluation must lead to a newly calculated activity tree containing only those resources visible for a user and/or a role. Properties and conditions may be related with arbitrarily complex dependencies. When a property value is changed, all conditions referring to it need to be evaluated. But such evaluation may prompt new value changes in more properties. As a consequence, the run-time environment applies these steps iteratively until no property value changes, thus reaching the new state. This process has the risk of entering an infinite loop (property changes do not reach a stationary value, but they oscillate) which would denote an incorrect UoL, but the run-time environment needs to provide some mechanism to avoid it.

The approach taken in GRAIL is slightly different from a state machine. No predefined states are defined when loading the UoL, instead they are calculated on demand when the UoL is executed. The run-time environment stores the dependency relationship between properties and conditions. The initial value of all conditions is obtained when a new run of the UoL is instantiated. From that point on, when a property changes its value, only the related conditions are re-evaluated. To avoid infinite loops, the system stops evaluating conditions if a property changes its value more times than a given threshold that must be chosen high enough.

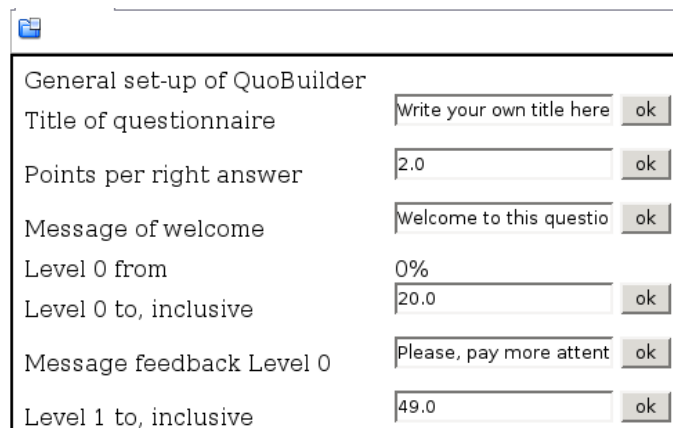
At parse time, conditions are stored in the database as defined in the UoL description, with their XML code. Condition evaluation amounts to parsing this XML replacing properties by their corresponding values. This scheme offered a simple implementation at a negligible computing cost and facilitates a hypothetical edition of conditions when an error is detected.

Another key issue to be considered at this level is the management of *imsldcontent* resources. The run-time environment needs to detect the presence of such elements and perform two types of operations. If the element is of the view-property type, its content must be replaced by the current value of the specified property. If, on the other hand, the element is of type set-property, its content needs to be replaced by a form in which the user may introduce a value which will later be assigned to the given property. Also, these documents may contain elements with condition-controlled visibility classes. Although these classes are not part of the IMS-LD specification, a mechanism needs to be implemented to deal with them.

The approach followed in .LRN consists on modifying the stored resource and creating a temporary file which is then delivered to the user browser. This processing is done entirely by the server. GRAIL manages this task by using the tDOM library, which offers a very efficient DOM implementation [7]. This tool provides an efficient environment to transform *imsldcontent* elements into its correct XHTML to be delivered to the users.

An alternative for this server based approach using tDOM would be to delegate this task into the browser and use the Extensible Stylesheet Language Transformation (XSLT) [8]. However this

solution would have browser compatibility problems difficult to solve and would increase the user requirements, so the decision was taken to perform this process entirely at the server side. Figure 4 shows an example of a resource of type `imsldcontent` where properties are both visualized and entered by the user by means of forms.



The screenshot shows a web form titled "General set-up of QuoBuilder". It contains several input fields, each followed by an "ok" button. The fields and their values are: "Title of questionnaire" with the placeholder text "Write your own title here"; "Points per right answer" with the value "2.0"; "Message of welcome" with the placeholder text "Welcome to this questio"; "Level 0 from" with the value "0%"; "Level 0 to, inclusive" with the value "20.0"; "Message feedback Level 0" with the placeholder text "Please, pay more attent"; and "Level 1 to, inclusive" with the value "49.0".

Figure 4: Example of an `imsldcontent` resource type displayed by the RTE

Activity attributes such as visibility or starting time must be stored separately for every run, role and user. Visibility, like properties, can be modified in level B by using conditions. Thus, for each run, the visibility attributes are instantiated for every user. Properties are also instantiated for every role, user or run, depending on the property type.

Level C: Notifications Support

Level C extends the specification by introducing notifications, which are messages that are sent when some event occurs. These notifications may optionally change the visibility attribute of activities of a given role.

GRAIL implements this functionality by leveraging in the notification system already present in .LRN/OACS. Despite its similarity in the name, this system was conceived to notify users through email when an object in the system changed. Its main usage is to subscribe to changes in forums, documents, events, etc. This functionality is modified by the run-time environment to send the email when the event specified in the UoL is produced.

5 Integration Issues

One of the key features of the IMS Learning Design specification is the capability of managing applications based not only resources, but also services. The advantage of using services is to have content consisting of run-time interactive activities. Koper and Tattersall (2005) explains that the difference between learning objects and services is that the location of a service is created when a UoL is instantiated. These services are usually based on server applications which are able to enhance the learning process. Although the specification does not determine the applications to be used, the requirements they must offer are clearly defined.

Since services are an important part of the specification, the run-time environment must be able to perform a transparent interaction with external services. Ideally, the results derived from this interaction must be seamlessly embedded in the current environment as actual content without any disruption of the learning process. The underlying OpenACS toolkit in which GRAIL is based has a highly modular architecture facilitating the interaction between different tools, thus facilitating the integration of services in the run-time environment.

5.1 Service Integration

The specification defines the set of services which must be supported by a compliant run-time environment. Depending on the type of information the service manages, they are divided into two groups: communication and information oriented.

- **Communication oriented services.** Allow users to communicate with each other, making possible the use of collaborative learning designs. The considered services are conference and sendmail.
- **Information oriented services.** Allow users to search and manage information about the UoL. The index/search service allows users to find information inside the course, while the monitor service shows information related to its state.

As discussed in Section 3, .LRN is a communication oriented LMS and therefore the asynchronous *conference* service is directly mapped to its forum service. The functionality to create several forums within a community offers the perfect underlying support to implement the *conference* service.

When created in a .LRN, forums are only visible to the users of the community in which they are instantiated. Since GRAIL is used in the context of a community (or a course), only people enrolled in the course are able to join the discussions. These forums may be used with different management policies. Each role must then be able to join the forum with a different set of permissions. The user types included in the specification are: observer, participant, conference-manager and administrator. All of them are easily mapped into .LRN groups as in the case of roles.

Whenever a run of a UoL containing a forum is created, a new forum instance is also created. At this point, no user has access to the service, therefore it is invisible. When users reach an activity where the forum must be used (that is, the environment contains the asynchronous *conference* service) their role membership is checked and the right permissions are consequently assigned to the forum becoming visible in the user space (as any other forum in .LRN) even once the course has finished. Although not part of the specification, forums in .LRN support some extra functionality, for example, email notification for new posts, moderation, etc.

A second integration is that of the *sendmail* service, which allows a user to send an email to a set of users. The recipients of such message can be indicated in the UoL by setting the corresponding role-ref attribute. Level B allows these recipients to be defined as the content of a property which is read at run-time. In both cases, the interaction with a SMTP server is required. Role and property-based *sendmail*, are supported in GRAIL using the underlying functionality offered by .LRN which allows for a great level of simplification. From the point of view of users, emails are sent by simply filling the minimum fields in a form. The rest of data such as destination, interaction with the SMTP server, subject, etc. is directly obtained from the UoL and the underlying platform.

A chat tool embedded in the LMS would also be a desirable conference service. Latest releases of .LRN include some basic chat functionality, but it is now undergoing significant improvements. When fully deployed, GRAIL will be extended as to offer the possibility of instantiating it as a service in a UoL.

The support for the monitor service has a straightforward implementation derived from the Level B support. Since properties are handled by the run-time environment, the service consists of displaying the current values of the properties included in the service definition and therefore requires no additional tool. A monitor can be seen as a concrete case of the more generic *imsldcontent*, where a subset of properties can be easily visualized with the proper XHTML document.

5.2 Integration with additional tools

.LRN as a Learning Management System contains additional services that are not part of the IMS-LD specification but can be readily included in a UoL. In this section, a brief description of these services

as well as how they can be easily integrated in a UoL is described. Figure 5 summarizes this integration as well as the previously described services.

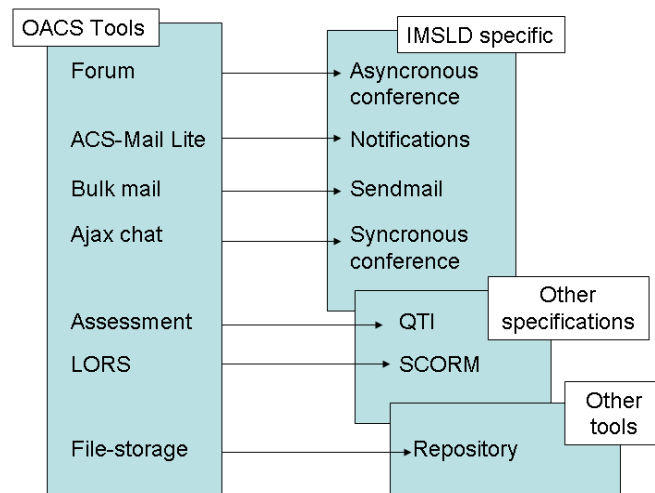


Figure 5: .LRN functionality and its potential mapping to LD services

In the current version of GRAIL, the following additional functionality can be within a given UoL.

File Storage

The possibility of sharing a common space for documents and folder is available in .LRN through its package called "file storage". Combined with the permission model, it allows users to share documents among different groups, communities, courses or even have their own private space. An immediate usage of such facility is to store all the resources present in a UoL. A more specialized usage would be to take this feature as part of an activity and make explicit usage of it. For example: write the main conclusions of the chat sessions held within the users in your role and publish it in the common area for the course. In the case of .LRN, since such area is always available, this type of activity may already be considered in any UoL.

QTI Tests

As described in [9], the QTI specification can be easily integrated with Content Packaging. This interaction is easily captured by GRAIL because .LRN offers support for QTI through the Assessment package. If an activity includes a set of questions in QTI, the rendering engine is invoked and its result is shown to the user. The platform then stores the received answers and returns the proper feedback to the user (as stated in the QTI specification).

SCORM

The integration of Learning Design and SCORM is still a pending issue. SCORM 2004 does not consider Learning Design as part of its format, but it is probable that future revisions would consider it to model the learning process as already pointed out in Tattersall, Burgos, Vogten, Martens and Koper (2006). Still, at a basic level, it is possible to take advantage of the combination of these two specifications. GRAIL supports the inclusion of a SCORM package as one of the resources of the UoL. Its treatment is similar to the one described for QTI Tests. Once it has been identified as a SCORM resource, the proper package within .LRN is invoked (more precisely, the LORS package) and it takes care of the visualization of its content packaging structure.

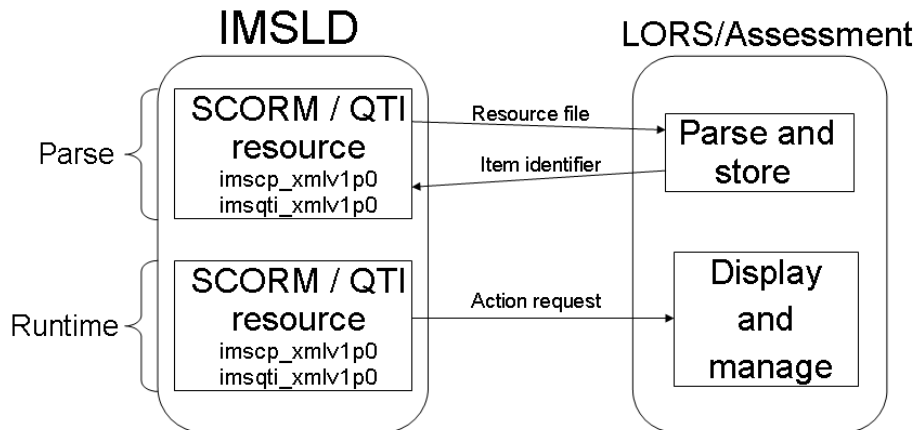


Figure 6: Usage of QTI and SCORM support within a UoL in .LRN

Figure 6 illustrates the management of QTI and SCORM resources within a UoL. At the parsing phase, the auxiliary tools are invoked to read the proper description files. Later, when the UoL is active, whenever an activity requiring any of these tools is invoked, control is transferred to them for rendering, and their result sent to the user browser. In this situation, GRAIL behaves as a simple intermediary.

From the interaction among different tools within the same LMS, a deeper integration problem arises. For example: consider a QTI test which is part of a UoL. Its management may be perfectly covered by the QTI engine, but a tight integration would allow the UoL to contain conditions and properties derived from the results obtained in such test. This, in the current Learning Design specification is not considered.

More generally stated, the issue of exchanging information between the UoL and the different services is yet to be solved. By providing a framework capable of capturing and describing such interaction, the UoLs that could be designed would take full advantage of functionality already present in most Learning Management Systems.

6 Conclusions and Future Work

The implementation of GRAIL, a Learning Design run-time environment within .LRN supporting all three levels of the specification has been presented. Its main difference with other run-time environments is the tight integration with a Learning Management System. Such integration has the advantage of leveraging in already present functionality such as user/permission management, group infrastructure, service implementation or integration with other services.

As for experimental results, the run-time environment is in its initial deploying phase in several higher educational with encouraging preliminary results. Instructions to use a fully operational trial version can be found in [10].

Future work is oriented toward exploring two avenues. On one hand, extend the current set of services available in .LRN with additions such as chats. On the other hand, provide a generic interface to facilitate the exchange of information among the different environments attached to services and the UoL.

Acknowledgements: Work partially funded by *Programa Nacional de Tecnologías de la Información y de las Comunicaciones*, project TSI2005-08225-C07-01/02

References

Koper, R., and Tattersall, C. (2005), *Learning Design. A handbook on Modelling and Delivering Networked Education and Training*. Springer Verlag.

Weller, M., Little, A., McAndrew, P., and Woods, W. (2006), *Learning Design, Generic Service Descriptions and Universal Acid*. IEEE Educational Technology and Society.

Vogten, H., Tattersall, C., Koper, R., Van Rosmalen, P., Sloep, P., and Martens, H. (2004), *Designing a Learning Design Engine as a collection of Finite State Machines*. International Journal on E-Learning.

Tattersall, C., Burgos, D., Vogten, H., Martens, H., and Koper, R. (2006), *How to use IMS Learning Design and SCORM 2004 together*. International Conference on SCORM 2006, Tamkang University, Taipei.

Footnotes

[1] CopperCore, The IMS Learning Design Engine: <http://www.coppercore.org>

[2] The RELOAD Project: <http://www.reload.ac.uk>

[3] The .LRN Platform: <http://www.dotlrn.org>

[4] The Open Architecture Community System: <http://openacs.org>

[5] The AOLServer web server: <http://www.aolserver.com>

[6] IMS Question and Test Interoperability: <http://www.imsglobal.org/question/>

[7] tDom manual: <http://www.tdom.org/doc-index>

[8] Extensible Stylesheet Language Transformations: <http://www.w3.org/TR/xslt>

[9] IMS Question and Test Interoperability Integration Guide *Revision*: 8 June 2006:
http://www.imsglobal.org/question/qtiv2p1pd2/imsqti_intgv2p1pd2

[10] The Gradient Lab, Department of Telematic Engineering, Carlos III University of Madrid:
<http://gradient.it.uc3m.es/>