

Integrating generic services in GRAIL, the Learning Design Run-Time Environment in .LRN

Luis de la Fuente Valentín, Abelardo Pardo, Carlos Delgado Kloos

Department of Telematic Engineering

Carlos III University of Madrid, Spain

lfuente@it.uc3m.es, abel@it.uc3m.es, cdk@it.uc3m.es

The recent trend to include a wider variety of technologically supported learning services in a learning experience has led to an increase in the complexity of both the production and deployment phases. Learning Design has been conceived as a formalism to capture the widest variety of pedagogical strategies. Lately, there have been several tools that provided support for both editing and interpreting a so called *Unit of Learning* (UoL). This paper focuses on the interpretation of a previously created Unit of Learning using Learning Design. The interpretation is provided completely embedded in .LRN an open-source, industry-class Learning Management System.

Keywords: Learning Design, web services, run-time environments

Introduction

When a learning experience is truly enhanced with the use of technology, the number of possible scenarios expands significantly. Today, the process of designing learning experiences may reach levels of complexity that were not possible a few years ago. This change is in part motivated by the availability of an enormous amount of electronic resources generically known as learning objects. From the technological point of view, there has been significant effort devoted to provide the right framework to promote the sharing of these learning objects. Numerous formalisms and some standards have been created to store metadata together with the resource. But as pointed out in Wiley 2007, this “engineer invasion” led to a “technical standards soup” that far from solving the problem it favored the appearance of informal frameworks that achieve the same effect. Authors are simply exchanging files, images, resources without the burden of using the proposed formalisms.

From the point of view of the information management, e-learning tools have clearly achieved maturity. But new more difficult obstacles are laying ahead. Once a learning designer has access to all the required resources, the problem is no longer how to manage them, but how to script its use by all the actors involved in a learning experience. From an initial version of a learning environment in which resources were simply made available to students, experiences now are moving toward crafted combination of multiple resources in which both students and teaching staff need to have a carefully designed strategy to achieve a truly effective learning experience.

This complexity has already been considered in some special learning scenarios such as collaborative learning. In a collaborative learning scenario users interact with peers through a set of activities to obtain a set of common objectives. Aside from the specific topic, one of the objectives of the learning process is to trigger productive argumentation among students (Jermann & Dillengourg 2003). The concept used to describe the details of the interaction between students and teaching staff is a script (see Hernández 2007 and Weinberger 2005).

With the advent of the new technological developments, and leaving aside the problem of achieving a truly positive contribution to a learning experience, the difficulties to coordinate the usage of resources such as discussion forums, wikis, blogs, remote simulators, virtual chat rooms, etc. have increased dramatically. Although a learning experience could be designed that simply deploys these resources whenever needed, more powerful and rich scenarios are possible if the interaction of all the actors in a learning experience with these new resources can be described in detail and used to adapt the experience itself.

The IMS Global Learning Consortium published in February of 2003 version 1.0 of the Learning Design specification (IMS Global Learning Consortium 2003). The proposed formalism tries to tackle precisely the problem of defining the interaction of multiple learning resources (in its widest interpretation) with

the actors participating in the experience. The formalism follows the analogy of a theatrical play. The play (learning experience) is divided into a sequence of acts in which several actors divided into so called *roles* perform different tasks in different scenarios. A scenario is a set of resources available to the actors participating in an act, and a given resource is allowed to have a different appearance depending on the actor using it.

The specification is divided into three levels of functionality. Level A includes the possibility of defining a set of activities in a static structure decided at design time. Level B provides a significant increase in flexibility because it allows the definition and manipulation of properties and conditions. With these two mechanisms, scenarios, environments, activities and even document visualization can be adapted based on observations captured in properties in previous activities. This enhancement allows experiences with a truly dynamic structure. Level C provides a marginal increase in functionality defining a notification mechanism by which actors can be advised of changes and events within the experience. The main goal of this specification is to offer an educational modeling language capable of capturing the widest possible range of pedagogical strategies used in learning experiences.

In the last five years since the first version of the specification has been published, several implementations of tools supporting the description and use of Learning Design have appeared. The main challenges faced by these tools is the significant semantic gap between the Learning Design formalism, and a intuitive design procedure for a Unit of Learning (UoL). The process of creating a UoL using this formalism is supported by several editors. The main reference is the Reload Learning Design Editor (Reload 2008). Although Reload supports all three levels of the specification, its adoption threshold is high due to the required familiarity of the user with the details contained in the Learning Design specification. As for the run-time engines, one of the most relevant is the CopperCore Engine (see CopperCore 2008), the first engine to support all three functional levels in the description. This tool is not a learning management system (LMS), but an engine suitable to be used by a generic LMS for interpreting and deploying units of learning complying with the Learning Design specification. The RELOAD project itself offers a run-time environment that uses the CopperCore engine to interpret units of learning described with Learning Design.

In this paper the use of GRAIL, a Learning Design run-time environment completely embedded in the .LRN Learning Management System is described. GRAIL is part of the .LRN Learning Management System and offers the possibility of uploading and executing units of learning described in Learning Design and to take full advantage of the functionality already provided by the LMS.



Figure 1.: Example of a user home page in .LRN

The .LRN Learning Management System

.LRN (.LRN 2008) is an enterprise-class open source platform for supporting e-learning and virtual communities. The tool was originally developed at the Massachusetts Institute of Technology as a virtual learning environment. The tool is based in the OpenACS Toolkit (OpenACS 2008) which provides an extensive set of functions and applications for generic virtual web communities. .LRN extends OpenACS by including functionality specifically conceived for e-learning experiences.

The early versions of LMSs emphasized only the aspect of managing a large set of digital resources as well as the typical administrative tasks associated with an educational institution. While such functionality is perfectly covered by current tools, very few were conceived from the concept of virtual community as .LRN has. A community is a set of users that share a virtual space and resources. In .LRN these resources include file storage, discussion forums, blogs, wikis, calendar, schedule, assessments, surveys, etc. Communities may have a hierarchical structure allowing sub-communities of users to belong to a given community thus sharing the higher level resources and services but at the same time having separated spaces for each sub-group. A user in .LRN may belong to multiple communities and is offered a highly customizable personalized home page with public file storage.

.LRN is being used in several higher level educational institutions with a large number of students such as Vienna University of Economics and Business Administration, Universidad de Valencia (Spain), Harvard University Executive Education Project (USA), etc. (see .LRN 2008 for a detailed list). Figure 1 shows an example of the information in the user home page.

The screenshot shows the user home page for the course 'Arquitectura de ordenadores'. The page is divided into several sections:

- Class Info:** Contains links for 'Sobre la asignatura', 'Profesores', 'Calendario detallado de sesiones', 'Material de prácticas', 'Foro de la asignatura', 'Sala de chat', and 'Canal de RSS'. It also includes social media icons for Google, netvibes, and Pageflakes.
- Staff List:** Lists the professors: Pablo Basanta Val, Raquel M. Crespo García, Carlos Delgado Kloos, Iria M. Estévez Ayres, and Abelardo Pardo.
- Schedule:** Shows a calendar view with dates [1 | 7 | 14 | 21 | 30 | 60] and a 'day' button.
- Anuncios:** Contains two announcements. The first is about technical sessions for the Linux group. The second is a weekly quiz titled 'El acertijo de la semana' with a question about train colors.
- Frequently Asked Questions (FAQs):** Lists common questions under the heading 'TTAO: Preguntas frecuentes', such as '¿Se pueden aprobar las dos partes, teoría y problemas, por separado?' and '¿Existen otras versiones de los apuntes de la asignatura?'.

Figure 2.: Example of the home page of a course in .LRN

As it can be seen, the main page is divided into small portions called *portlets*. Each of them shows a different aspect of the communities the user belongs to (forums, file-storage, schedule, etc). This philosophy has been shown to adapt itself nicely to the complex user communities that frequently arise within an educational institution that go beyond pure teaching duties. A class is conceived as a special instance of a virtual community that is usually attached to a semester, a department and/or a school, its

file storage space is initially divided into special sections (handouts, submissions, lecture notes, etc) and contains special user profiles (professor, teaching assistant, student, etc).


The virtual community hosting a course has a similar look and feel as shown in Figure 2. The course page is also divided into *portlets* each of them showing information about different services. The figure shows the upper right portlet with some popular links to the course description, detailed schedule, lab material, discussion forum (hosted by the same platform), a chat room (externally hosted in a different platform), etc. The page contains additional portlets with announcements, information about the teaching staff and course member list (students and teaching staff), a frequently asked question list, and a Wiki. The community used for the example has more than 100 members and follows a blended learning approach. Conventional face-to-face sessions are complemented with additional resources (forum, wiki, faq, etc) which are offered with the help of technology.

This example tries to illustrate the increase in complexity that appears when deploying a course using such a variety of services and resources. For example, a simple discussion forum with a set of users may be hosted in multiple generic web applications free of charge. But embedding these resources in a community solely devoted to a course has obviously multiple advantages. Only members of the course are allowed to read/write posts in this forum, teaching staff may be given special rights to administer the forum, a file complementing the information discussed in a post may be stored in the area specially dedicated to the community, the faq will contain only questions relevant to the course, etc. The more services are included in a virtual community, the more complicated is to coordinate their functionality within the community.

But specifying, deploying and coordinating these services requires a formal approach, a formalism to capture this structure and be able to replicate it as many times as needed. This is the role of specifications such as Learning Design. The analogy used to illustrate the potential power of this paradigm is the theatrical play (IMS Learning Design Information Model). The structure, organization, activity sequencing and all other factors of a course could be captured in the same way than a theatrical play captures how a piece needs to be performed. Such piece may then be performed multiple times by simply interpreting the script. Pushing this paradigm a bit further, there would be a set of truly *masterpieces* which could be deployed and enjoyed in multiple locations without the effort of creating them.

GRAIL: Learning Design as an additional portlet

GRAIL is the run-time environment capable of interpreting a UoL described in Learning Design within any virtual community or course in .LRN and deploy the required resources. This functionality assumes that the UoL has been previously designed using an external editor. It offer then almost no editing capability. Whenever a GRAIL instance is created in a community, there are two main functionalities that are deployed. An extra portlet is deployed both at the community and home user page showing the UoLs in which the user is participating, their role in such unit, its status (running or terminated) and the date in which such UoL was created. Figure 3 shows an instance of a GRAIL portlet for a user that is participating in three UoLs with the profesor role and one with the student role.



Unit of Learning Name	Role(s) in Run	Status	Creation Date
PHP/Drupal	Profesor	▶	11/22/2007 20:36
PHP/Drupal	Profesor	▶	11/22/2007 20:41
PHP/Drupal	Profesor	▶	11/22/2007 20:44
PHP/Drupal	Alumno	▶	11/26/2007 13:03

Figure 3.: GRAIL portlet with the information about four UoLs where the user is participating

This portlet shows information only about those UoLs that have been uploaded in the platform and instantiated. The instantiation of a UoL is a required step before it can be fully deployed in a community. The main task to perform in this stage is the assignment of users to the different roles considered. Without this information the platform is unable to show the appropriate resources to each user. Once the instantiation process has finished, the platform does not allow this user distribution to be modified. This restriction might seem too restrictive at a first glance, but it needs to be enforced because all the

privileges to access the different resources are derived from the role in which a user is included. Changes during the execution of the UoL in this membership would require re-evaluation of all the permissions which is costly procedure that would need to be executed every time a new event is detected in the unit.

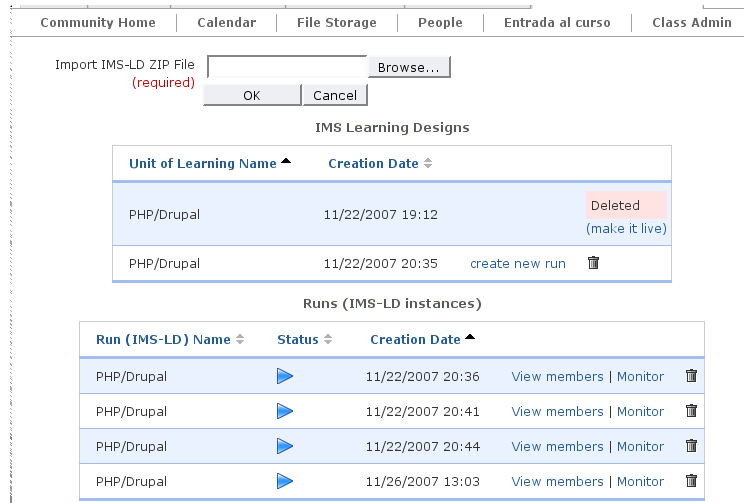


Figure 4.: Interface to upload and instantiate UoLs

Figure 4 shows the administrative interface offered only to community or course administrators to upload (or import) a new UoL, instantiate a new “run” from a previously uploaded UoL, or manage currently instantiated runs.

Once a UoL has been uploaded and instantiated, it is made available to users through the GRAIL portlet. When selected, a separated screen is opened with the information distributed as shown in the example of Figure 5.

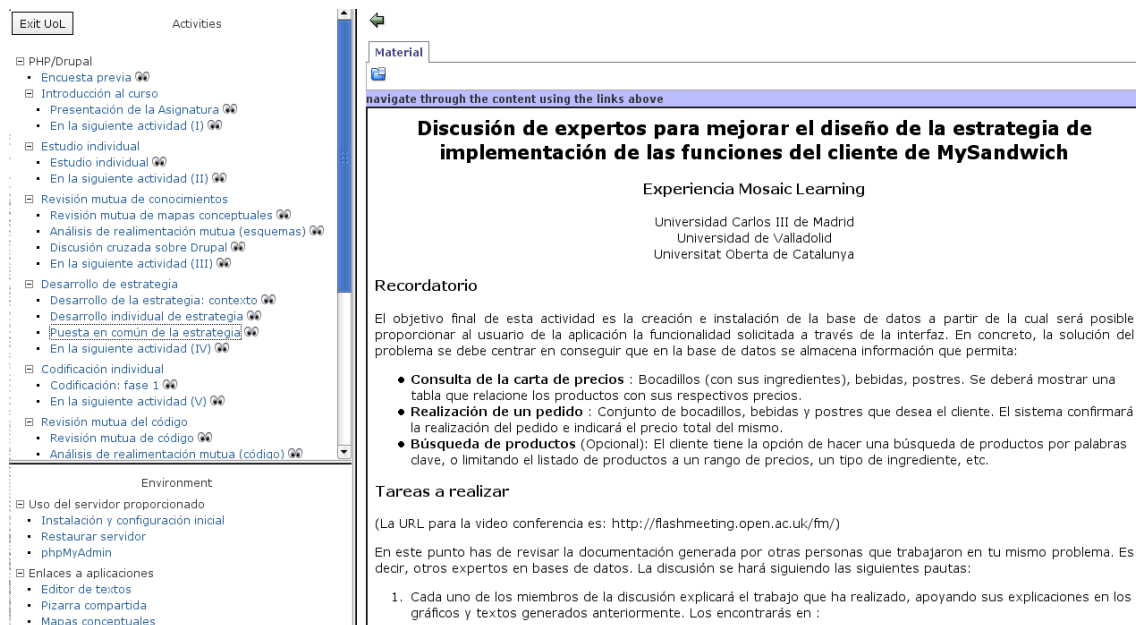


Figure 5.: Example of three areas UoL shown to the user

The upper left area contains the hierarchically organized set of activities already visited by the user. The visibility of these activities is controlled by the run-time environment based on the part of the UoL being enacted and the user role. The lower left area contains the *Environment* which shows additional resources required in the activity. In this example, the environment contains links to additional documents, but also to additional tools such as virtual boards, conceptual maps and shared text editor. These applications were remote services specially allocated for the UoL.

The right portion of the screen is where the resources directly associated with the activity are shown.

One of the main tasks of the run-time environment is to monitor the activity of the users in the UoL and interpret these events as specified. At the simplest level, Level A, of the specification, this translates into deciding the subset of the activity hierarchy that is visible for each user at a given time. When supporting Level B of the specification, this task is significantly more complex. Whenever an event is produced, the run-time environment needs to evaluate the set of properties within the UoL and propagate the derived changes, if any, to all resources.

Monitoring the enactment of a Unit of Learning

Several learning experiences were deployed using GRAIL, and as a consequence, numerous empirical observations were derived. Perhaps the most important is the excessive rigidity derived from a production cycle in which the editing and enactment phases are loosely coupled. More precisely, performing changes in the UoL while it is being enacted is a complex task. In our experience, small changes in a running UoL are more common than desired, and therefore, a run-time environment which offers no possibility of changing any aspect of a UoL is too restrictive.

The proposed solution to alleviate this problem is to offer a *monitoring function*. Administrative staff are given access to the list of properties in a running instance and the possibility to change any of their values at any time during the enactment. From the point of view of the implementation, a change in the value of any variable in the monitor is perceived as a regular event that prompts the re-evaluation of all the properties and the changes reflected in the UoL resources.

Although far from providing full flexibility with respect to changes, this solution has been shown to be a reasonable compromise between the amount of changes allowed and their impact in run-time engine. Figure 6 shows an example of the information provided by the monitoring functionality.

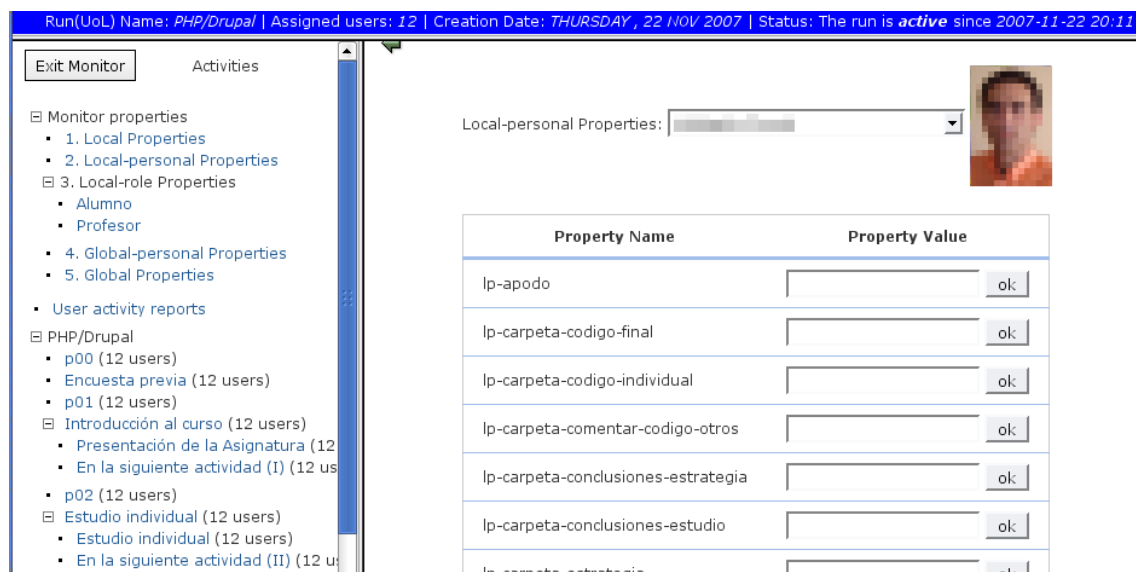


Figure 6.: Example of property modification in the monitoring menu

The left part of the screen provides access to all the properties defined in the UoL as well as access information for each activity. The right side of the screen shows the information selected. In this example, the screen shows the set of local-personal properties for a previously selected user. The effect of any change in these properties is immediately reflected in the UoL being enacted.

But the monitoring functionality by itself does not offer any UoL flexibility. More precisely, changes in the UoL are allowed as long as they are reflected in a property during the design phase. An example will clarify the expected compromise. A course is designed such that in a certain activity, access to a given application is provided to all the students. The URL locating such application is not known when the UoL is being designed. Any value inserted in an activity or environment will for sure need to be modified once the UoL is instantiated in a concrete environment.

The proposed solution is to capture the value of this location in a property. Any activity or environment resource referring to such application will in turn refer to the property. When the UoL is finally deployed in a concrete environment and instantiated, the monitoring functionality in GRAIL will allow the administrative staff to change the value of this property to its correct value at any point in time.

The main advantage of this approach is that it easily allows for a higher level of customization to be embedded in a UoL. A more generic UoL that later is customized to a concrete environment is more likely to be reused. But the approach has also disadvantages. The most significant one is the effect it has in the design phase. A learning designer needs to be aware of the possibility of properties being changed during the enactment and formulate as many aspects suitable to change in the UoL as properties. This technique could be taken to the extreme as to include every single title, sentence, name, parameter, etc. as a property. This situation would translate into an excessive number of properties increasing the complexity of the deployment phase.

The second disadvantage of this approach is the limited number of changes that can be supported. More sophisticated structural changes in a UoL such as replacing an entire act by another, cannot be easily reflected into property values, unless they are fully anticipated at design time.

Tight integration of generic services

A second limitation of the Learning Design specification in its current version is the lack of a generic description to tightly integrate generic services. The specification offers three categories for services: send-mail, conference and index-search. This division, aside from not being exhaustive, does not include low level details about how these services interact with the rest of resources. By “tight” integration, we refer to a service that can be instantiated from within a UoL, and the UoL may receive information about the events that happened internally. This level of integration is clearly not contemplated in the specification nor provided by any run-time environment.

An example of a service with tight integration could be a discussion forum for all the actors in a given act that receives several parameters from the run-time environment and returns to the environment additional values reflecting aspects such as the level of participation, number of posts per user, etc. With such integration, a UoL could be designed to adapt its content based on the values returned by the previous instance of the forum. Assuming a discussion forum that receives a parameter to specify a person to moderate and returns the number of posts done by each user, a UoL might instantiate one of such forums and select as moderator the person with the highest number of posts in a previously used forum.

In general, the assumption to achieve such integration is that a service needs to identify a set of input properties that are assigned upon its instantiation, and a set of properties that are returned as results during its operation. This paradigm is compatible with two requirements: it allows data to be exchanged between a run-time environment and a service in both directions, and it is compatible with the properties currently allowed in the Learning Design specification.

The full deployment of this paradigm imposes two requirements, one in the design phase and a second one in the run-time environment. During the design phase, a learning designer should have a catalog of possible services to choose from. Each of these services needs a description of both the properties required for its instantiation and those properties returned while is being used. With this information, the designer may include the service in the UoL and derive conditional behavior depending on the data obtained from the service. Most of the current Learning Design editors do not have support for browsing an extensible catalog of generic services. Some of them offer a restricted set of services previously selected.

In the run-time environment, the proposed integration can be implemented with an extra layer which performs the mediation with the generic services. This layer is similar to conventional plugins already present in other software tools. Figure 7 shows the structure to integrate the plugin inside a run-time environment. This layer is aware of the type of interface offered by one or many similar services and arbitrates the exchange of information between them and the run-time environment.

When the run-time environment instantiates a service, a set of parameters is obtained from the description and sent to the remote service. The plugin should also be aware of the issues regarding remote user login and/or registration. The result of a successful instantiation is a set of URLs to be assigned in the proper locations of the available resources. During the enactment of the UoL, some of the events that take place

in the remote service are notified to the environment by setting certain properties. The run-time environment, with the described functionality, evaluates again the entire environment and makes the appropriate adjustments.

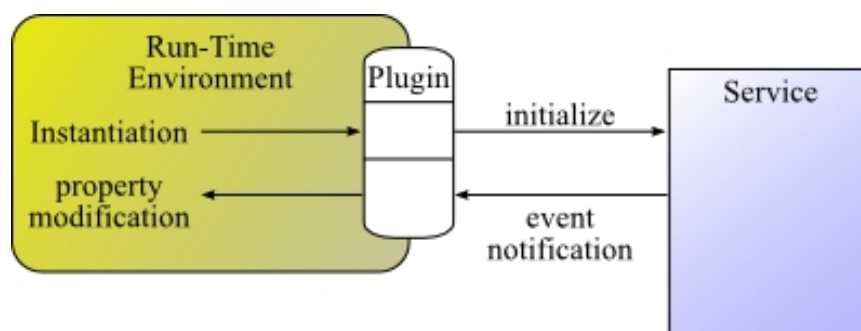


Figure 7.: Information exchange between environment and service

With this architecture, if a new service is available, a detailed description of its properties needs to be available to the learning designers and a new plugin needs to be developed and integrated in the run-time environment.

Conclusions

GRAIL, the run-time environment for Learning Design in .LRN has been described. The application is embedded within the LMS thus taking advantage of the already present virtual communities. Several pilot experiences have shown the rigidity of current design and deployment paradigms for Learning Design. A monitoring functionality has been added in order to provide some basic support for incorporating changes to the UoL while it is being enacted. The possibility to instantiate generic services has been described based on a plugin-based paradigm in which services can be considered by instructional designers and fully integrated within a UoL by exchanging information both at instantiation as well as enactment time.

Acknowledgments

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

References

- Wiley, D. (2007). *Openness, Localization, and the Future of Learning Objects*. Keynote Address, BCNET 2007 Conference, April 2007.
- Jermann, P. and Dillengourg P. (2003). *Elaborating new arguments through a CSCL scenario*. Arguing to Learn: Confronting Cognitions in Computer-Supported Collaborative Learning Environments, 2003, p. 205-226 Amsterdam: Kluwer, 2003.
- Hernández Leo, D and Burgos, D., and Tattersall, C and Koper, R,(2007). *Representing Computer-Supported Collaborative Learning macro-scripts using IMS Learning Design*. Second European Conference on Technology Enhanced Learning, CEUR Workshop Proceedings, EC-TEL'07, Crete, Greece, September 2007.
- Weinberger, A., Ertl, B., Fischer, F., & Mandl, H. (2005). *Epistemic and social scripts in computer-supported collaborative learning*. Instructional Science, 33(1), 1-30, Kluwer.
- IMS Global Learning Consortium (2003). IMS Learning Design Information Model. v1.0, Technical Specification. <http://www.imsglobal.org/learningdesign> [viewed 15 March 2008].
- CopperCore (2008). The CopperCore Project Home Page: <http://coppercore.org> [viewed 15 March 2008].
- RELOAD (2008). Reusable eLearning Object Authoring and Delivery. Project Home Page: <http://www.reload.ac.uk> [viewed 15 March 2008]
- Escobedo, J. P., De la Fuente Valentín, L, Gutiérrez, S, Pardo, A and Delgado Kloos, C (2007). *Implementation of a Learning Design Run-Time Environment for the .LRN Learning Management System*. Journal of Interactive Media in Education. September 2007
- The .LRN Platform. <http://www.dotlrn.org> [viewed 15 March 2008]
- The OpenACS Development Platform. <http://www.openacs.org> [viewed 15 March 2008]