Mashups in Learning Design: pushing the flexibility envelope

Luis de la Fuente Valentín* Derick Leony Abelardo Pardo Carlos Delgado Kloos

Department of Telematics Engineering, Carlos III University of Madrid, Spain
http://gradient.it.uc3m.es

Abstract. Internet is evolving toward highly personalized user spaces. Mashups, web applications that combine data from different sources into a single tool, are providing users with higher levels of flexibility to surf the net. These new environments can be used to offer students an attractive learning environment. At a first glance, this flexibility might be contradictory with the tendency toward capturing the interaction and structure of a learning experience in languages such as IMS Learning Design. However, this paper shows an example in which these contexts can be combined to exploit both advantages. The mash-up language LISL is used to describe a flexible learning environment in the context of a Unit of Learning enacted in a learning management system.

Keywords: Design, flexibility, mash-up, service integration

1 Introduction

Learning management systems (LMS) have been evolving steadily over the last years. In the early stages, technology allowed to access most of the resources required in a learning experience such as documents, exams, quizzes, simulators, etc. The technological support in this initial stage could be defined as *information-based*. But the evolution of communication technology in general has had its effect also in the learning realm. The new scenarios are not relying only on access to information but on the interaction between all the stakeholders. Learning technology is then clearly shifting from an information-based to a *communication-based* paradigm. The tendency in the last years has shown how technology enhanced learning has been steadily adopting several interaction patterns that emerge in Internet at large.

The unfolding landscape is that of a community of users with a highly personalized environment prepared to interact with a large number of available services. The success of personalization platforms such as iGoogle, Netvibes, Pageflakes (just to mention a few) together with their potential to increase the effectiveness of a learning experiences sustains this claim. Mash-ups appear in this context and are web applications that combine data from different sources into a single tool. These sources are generically known as services.

But with the possibility of combining a set of services comes the challenge of

* Corresponding author: lfuente@it.uc3m.es

inserting such applications in current specifications used to formally capture the interaction and resources needed in a learning experience. The IMS Learning Design specification [1] (henceforth IMS LD of simply LD), provides a language capable of defining the interaction that takes place in a learning environment. While the framework claims to be generic, it is oriented toward capturing interaction at the level of the different activities within a learning experience. Furthermore, although the specification includes a property-based adaptation paradigm, these properties need to be defined at design time and therefore typically refer to aspects that are statically included in the unit of learning.

The emerging scenarios provided by the use of mash-ups needs to be combined with the description at a higher level of a Unit of Learning. The solution explored in this paper is to combine both paradigms. On one hand, the Learner Interaction Scripting Language [2] (henceforth LISL) is a language defined to create a learning environment [1] as a web-application mash-up: an aggregation of user interfaces from different tools that are combined to achieve learning outcomes. These mash-up based environments are then placed in Unit of Learning IMS LD environments, providing tools to perform course activities. The paper describes how such integration has been done in Grail [3], the Learning Design run-time environment included in the Learning Management System .LRN [4]. As a result, a Unit of Learning is used to capture the higher level organization and resource of a learning experience, but at the level of an activity, students are offered a highly flexible learning environment described in LISL.

The rest of the paper is organized as follows. Section two defines the concept of flexibility when it is applied to the enactment phase of the course life-cycle. Next, section three discuss how this type of flexibility can be offered by the joint use of IMS Learning Design and the LISL language. In section four, a flexible Unit of Learning is exposed as practical example. The article concludes with a summary of main ideas depicted in the text.

2 Flexible learning scripts

The term “script”, when applied in a pedagogical context, refers to the method to structure a learning process [5]. Course participants are guided through the flow of activities previously defined by a set of authors in a script. The level of coercion in this script will influence on the course success. Over-scripting, or a too detailed set of steps that must be followed, may reduce course effectiveness, while a too flexible scheme may not produce the expected interactions [6]. The trade-off then is to create a set of instructions detailed enough to guarantee a successful learning experience, yet leaving room for certain flexibility.

The attitude of students cannot be predicted during the authoring phase, but can be monitored while the course is being enacted specially if deployed in a LMS. Ideally, the run-time environment within the LMS should be able to react to special situations and/or minor changes in the script. This capability of changing the course behavior while is being enacted is known as *run-time flexibility* and is analyzed in detail in [7]

Typical changes in a script have a wide range of variation: structural changes are

1 Here and hence, the term environment does not mean IMS LD environment, unless it is specifically stated by the proper use of the acronym

needed if an activity is highly dependent on previous results but they are not available at design time; content might need to be changed when participants detect anomalies in any resource; changes in group members might be needed if students drop-out. Based on these different scenarios, the flexibility required in a learning environment can be classified in different ways.

Depending on to the scope of the changes, flexibility is required at either the micro or macro script. The macro-script level refers to the high level learning flow: the activities that are present in a course, how artifacts are produced and consumed by these activities, how and when participants can be monitored and evaluated, time scheduling, or any other factor that affects the course as a whole. On the other hand, flexibility at micro-script level is related to activity-centered modifications. For example, how a single activity can be properly performed, including the tools to be used, the available content and expected interactions among peers and resources, etc.. Thus, flexibility at the micro-script level allows to re-define how an activity is performed. Typical learning scripts combine structures at both levels.

But a second type of flexibility appears depending if it is applied to teaching staff or students. Teaching staff (not necessarily authors) are in charge of supervising that the activities are taking place as expected in the script. They might need to perform changes to guarantee that the objectives can be reached. One simple example is an unexpected homework deadline extension. Students usually are not allowed to change the learning flow, but their interaction with their peers or resources can vary significantly, and therefore, the learning environment should also offer certain degree of flexibility at least to customize to certain extent the learning environment.

3 Integration of Learning Design and LISL

In computer supported learning scenarios, scripts can be formalized during the authoring phase of a course using a modelling language. All the interactions of a learning flow are then captured in a single file, which can be deployed in a compliant platform, where the enactment takes place. This section analyzes how runtime flexibility can be provided combining two different scripts: IMS Learning Design and LISL.

Macro-scripts

The IMS Learning Design specification [1] provides a framework where a wide range of pedagogies can be expressed. It is divided into different levels of complexity, each of them complements the previous one with more functionality. Thus, level A introduces the vocabulary and sets the basis of sequencing capabilities. Level B adds properties and conditions to the course definition, and allow to define several sequencing rules based on the state of the course. IMS LD is therefore a specification that matches perfectly with the concept of macro-scripts. Course structural changes during runtime can be managed by a proper use of LD properties, but they must be anticipated during the design phase. Taking advantage of this potential use of properties, LD offers an significant level of runtime flexibility [8].

From the teacher point of view, content can be made visible depending on the value of a given property, that can be modified by tutors. It is also possible to capture certain events to take place at a given time. By capturing those times in a property, modifications are easily allowed. Group management can also be captured in part by properties, allowing to dissolve and regroup students if required.

Typically, the described learning flow cannot be modified by students, but they can take decisions that affect their learning path. This is the case of profile-based statements: if students can modify their own profile, they will be able to select the kind of activity they prefer, without breaking the course constraints.

Although certain degree of flexibility can be achieved during run-time, it is at the cost of capturing these changes with numerous properties and conditions. The main consequence is that all possible changes need to be anticipated during the authoring phase. This aspect hinders significantly the possibility of performing fine-grained modifications.

Micro-scripts

The LD specification is clearly oriented to macro-script creation and therefore is not appropriate to scaffold the interaction process that occurs within an activity. A new approach is required that allows participants to select their preferred environment for the required activity. The LISL language can be used to create micro-scripts that create, manage and maintain fully-customized learning environments, and can be used to offer flexibility to students while enacting an activity [2].

The LISL language allows to define **actions** to perform in a given activity, **objects** that are produced, modified or consumed by actions, and **tool** to perform requested task. Objects can be defined by a URL, and tools must be related to a URL. LISL statements are close to natural language, so that they can be read as follows: "perform action A on object B using tool C". When the script is played, all required tools are opened in the working environment with mash-up visualization techniques.

Simplicity of LISL script creation and modification was one of the design premises of the language. As a result, the personalization of working environments becomes a really agile task that can be performed by course authors and modified by students during runtime.

Implementing the Mash-up Integration on Learning Design

In order to integrate a LISL-based mash-up with a LD player, the used mash-up templates need to be simplified to achieve a plain look. This interface simplification and the use of technologies such as AJAX helped to perform the integration very intuitively and in an unobtrusive way.

On the side of the LD player, the mash-up is referenced as a regular learning object within the environment of a given activity. Thus, the LD player will provide an independent highly customizable learning environment while remaining unaware of the level of freedom provided by the mash-up.

4 An Example

The proposed scheme to combine the two paradigms in this paper can be easily demonstrated with a simple UoL containing the previously discussed features. This section shows the deployment of a UoL as a proof of concept.

The deployment environment is based on the use of .LRN [4], a LMS built over the functionality provided by the OpenACS web toolkit. .LRN includes GRAIL as the LD run-time environment and was further modified to include Mupple, a LISL interpreter [2] and a visualization screen.

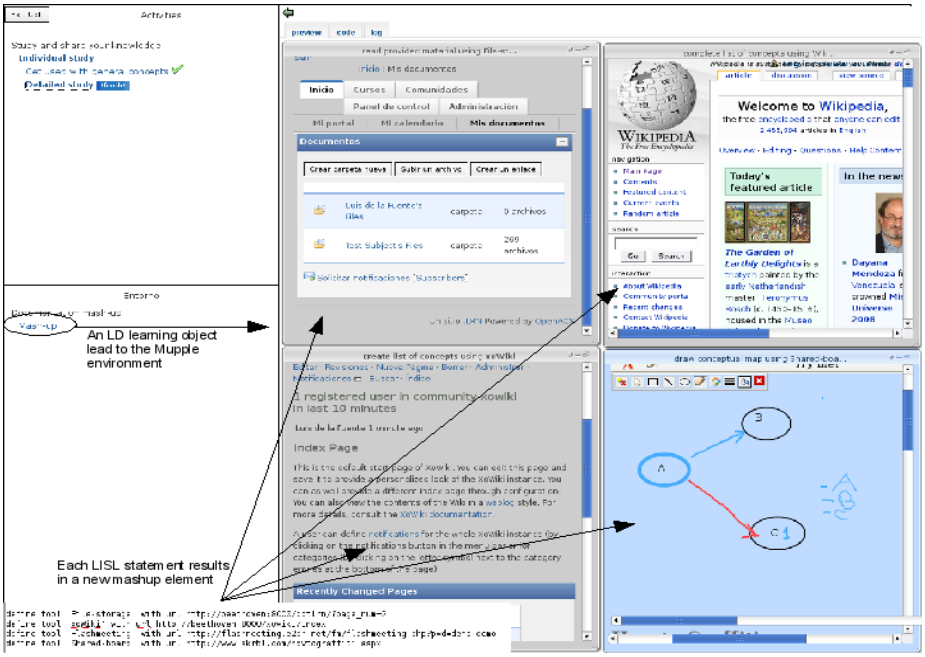

Fig. 1. Resulting working environment when combining LD and LISL.

In the UoL used in the example, the flow consists of two sessions, modeled as **acts** in LD. In the first session, learners have to study a set of initial documents and write a list of the main concepts. Then, they have to explore more deeply these concepts and draw a conceptual map linking all of them graphically. The activity starts with the reading of the documents available in the course repository. Then, students must use Wikipedia to clarify those concepts that not properly understood. Finally, a wiki tool allows to write the list of main concepts and a mind-mapper tool allows to create conceptual maps. The resulting structure is depicted in Figure **1**.

Students access the activity environment and are able to modify it either modifying the source code, or using the graphical interface that allows to open, close, move and iconify each box. With this structure, an additional step (i.e. search a concept in Google), or the modification of one of the tools used by existing steps (i.e. one student prefers cmaptools for conceptual maps) can be easily achieved.

Monitor facilities are provided to allow the teacher to see what is happening in the course. This monitoring functionality is offered at the level of the LD run-time environment and it is intended to monitor at the level of macro-scripts. As it can be seen Figure **2**, the platform allows the course administrator to see how many students have accessed a given activity. The same application allows also to change the termination condition of any activity or act, and to change the collection of learning objects present in an environment.

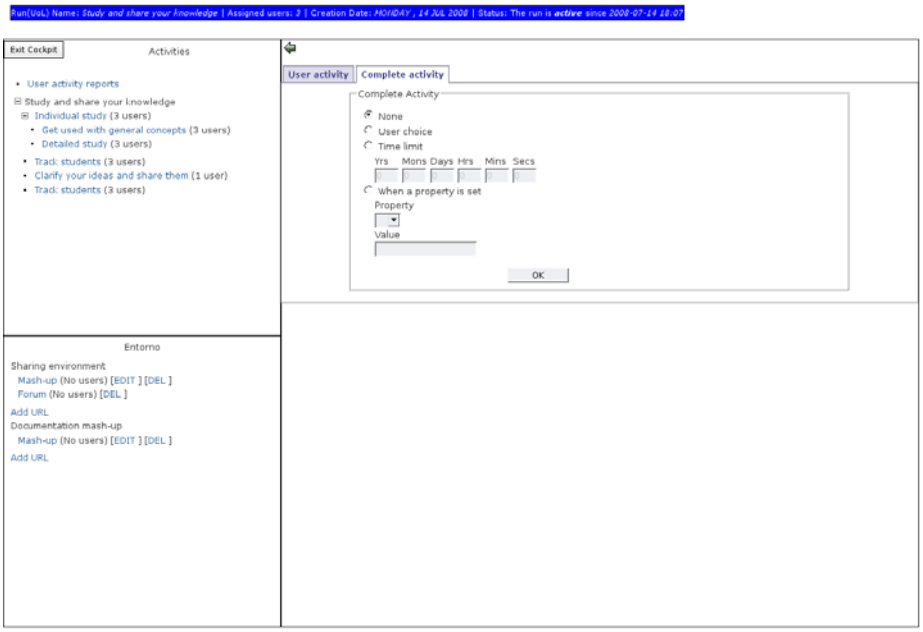

Fig. 2. Access to course details and tracking facilities.

5 Conclusions

This paper presented how flexibility on learning courses can be achieved with the joint use of LD to define the learning flow (macro-script) and LISL to detail scaffolding details on activities (micro-scripts). Flexibility is required to appear in all the course life-cycle. The proposed working scheme is focused on the run-time phase.

In this stage of the course, teachers have to be able to modify content, re-configure services, change termination conditions on an activity or its visibility. Students should have the possibility of adjusting the learning environment to their personal needs: select the tools to use and change their own profile.

The Learning Design specification offers a reasonable degree of flexibility at run-time. Level B properties and conditions can be used to modify the course behavior even if it is already running. However, this changes must be explicitly stated during the authoring phase. Used to provide a framework where activities can be performed, the LISL language allow students to easily configure the environment and share preferences with peers.

An instance of the .LRN LMS including GRAIL, the IMS LD run-time environment has been enhanced with Mupple, a LISL interpreter. This platform has been used to test this course design paradigm. An example Unit of Learning has been created and deployed, showing the potential of the architecture.

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

References

1 "IMS Learning Design specification," http://www.imsglobal.org/learningdesign/, Feb. 2003, [On line].

2 F. Wild, F. Mödritscher, and S. E. Sigurdarson, "Designing for change: Mash-up personal learning environments," *eLearning Papers*, no. 9, July 2008.

3 L. de la Fuente Valentín, A. Pardo, C. Delgado Kloos "Experiences with grail: Learning design support in .lrn," in *TENCompetence Open workshop on current research in IMS Learning Design and lifelong competence development infrastructures*, June 2007.

4 "The .LRN platform," http://dotlrn.org, Oct. 2007, [On line].

5 D. Hernández Leo, D. Burgos, C. Tattersall, and R. Koper, "Representing computer-supported collaborative learning macro-scripts using ims learning design," in *EC-TEL (Posters)*, 2007.

6 P. Dillenbourg, "Over-scripting CSCL: The risks of blending collaborative learning with instructional design." in *Three worlds of CSCL. Can we support CSCL*. Heerlen: Open University Nederland, 2002, pp. 61–91.

7 P. Dillenbourg and P. Tchounikine, "Flexibility in macro-scripts for computer-supported collaborative learning," *Journal of Computer Assisted Learning*, vol. 23, no. 1, pp. 1–13, February 2007.

8 L. de la Fuente Valentín, A. Pardo, J. I. Asensio Pérez, Y. Dimitriadis, and C. Delgado Kloos, "Collaborative learning models on distance scenarios with learning design: a case study," in *ICALT '08: Proc. of the eighth IEEE International Conference on Advanced Learning Technologies*. Santander, Spain: IEEE Computer Society, 2008.