LPCEL Editor: A visual authoring approach for learning design

Jorge Torres\textsuperscript{a}, Jesús Reséndiz\textsuperscript{a}, Juan Manuel Dodero\textsuperscript{b}, Ignacio Aedo\textsuperscript{c} \textsuperscript{*}

\textsuperscript{a}Tecnológico de Monterrey, Epigmenio González 5000 Fracc. San Pablo, Querétaro 76130, México
\textsuperscript{b}Universidad de Cádiz, C/ Chile # 1,11002, Cádiz, Spain
\textsuperscript{c}Universidad Carlos III de Madrid, Avda. de la Universidad # 36, 28911, Madrid, Spain

Abstract

The IMS-LD is a widely used Educational Modeling Language and several compliant tools have been created in order to facilitate the authoring process, but they have a restricted level of expressiveness. To overcome this issue, the LPCEL Editor is presented in order to facilitate the authoring of complex learning scenarios by means of a visual tool that applies some concepts of the Model-Driven Development. The editor provides a broad level of expressiveness and allows the use of educational patterns and web services and transformations of the visual scenario into a document compliant with a target EML (IMS-LD, LAMS LD, etc.).

© 2011 Published by Elsevier Ltd. Selection and/or peer-review under responsibility of Multidisciplinary symposium on the design and evaluation of digital content for education.

Keywords: Learning Design; LPCEL; Visual authoring; Learning patterns; Model-Driven Development

1. Introduction

Learning Design is a set of actions concerned with the creation of activities, services and resources that must deliver a learning experience. Educational Modeling Languages (EMLs) provide a framework to design learning scenarios in a formal manner; they include the tools needed to represent all the activities, resources, services, etc. that must be included in the scenarios (Torres, Juarez, Dodero & Aedo, 2009a).

The capability of handling complex learning processes is desired for any EML (Torres, Juarez, Dodero & Aedo, 2009b). A Complex Learning Process (CLP) (Dodero, Torres, Aedo & Diaz, 2005) is the result of the dynamic integration of teaching methods and resources inside a learning process. The smallest unit of work for a CLP is known as a Unit of Learning (UoL) (Koper, 2001), which are events that the students need to complete in order to achieve specific learning objectives.

The IMS-LD (Koper & Tattersall, 2005) includes a language to describe learning experiences as UoLs and provides a framework for the representation of learning processes. The IMS-LD is restricted by its level of expressiveness (Torres et al., 2009a) and so are the authoring tools compliant with it.

To overcome this issue Torres, Dodero, Aedo & Diaz (2006) present the Learning Process Composition and Execution Language (LPCEL) which provides a framework to design courses using CLPs. The LPCEL is used to specify complex and dynamic structures; also, it can describe the execution of a learning process as well as the resources needed to complete it. However, there was not an implementation for the LPCEL and no greater expressiveness could be achieved in Learning Design.

The rest of this paper is structured as follows. Section 2 presents the issues related with authoring tools and the reasons to work in the matter. Section 3 describes the existing solutions and tools as a visual approach for authoring. Section 4 describes our visual approach to design a course using a web editor. Section 5 presents the evaluation of the tool. Section 6 presents some conclusions and future work.

2. Issues and motivation

In order to create or edit a course created for a particular EML, the instructor must have full knowledge of all of the low-level details (the XML tags and structure) of that EML. The task is even more complicated when it comes to make a good use of the different and advanced functionality of the chosen specification (Martinez, Moreno-Ger, Sierra-
To overcome some of these issues, visual editors have been created. The visual editors use graphical abstractions of the low-level and high-level details of the target EML in which the learning scenarios are described. The IMS-LD (Koper & Tattersall, 2005) provides a language to create a set of elements, such as learning activities, resources and roles that represent several components of a learning process. The IMS-LD level A describes each one of these elements in standard XML format, which represent the static part of the course and is considered as the low-level design of the course. The Level B of IMS-LD adds some level of dynamism, i.e. the learning flow can change according to certain conditions.

Visual editors for Learning Design have a set of tools and iconographic representations of each one of the elements for a specific EML (Paquette et al., 2006). Most of visual editors seek to create and represent the UoLs based on a certain EML. These approaches provide the visual models needed to represent the primitives of the chosen specification (activities, roles, learning flows, etc.) and once the visual representation is done, it is translated into the EML-specific XML files. Most of the current visual authoring tools are IMS-LD compliant; this implies that they are also restricted by the level of expressiveness provided by the IMS-LD. To overcome this problem, the Learning Process Composition and Execution Language (LPCEL) is proposed by Dodero et al. (2005) and Torres et al. (2006).

The main objective of the LPCEL is to represent Complex Learning Processes (CLPs) with a broader expressiveness, that is, to support advanced execution patterns in order to provide more functionality during the execution of a learning scenario (Torres et al., 2009a). Despite the expressiveness of LPCEL in comparison with IMS-LD, there was not an implementation that fully describes the LPCEL specification. The LPCEL has been chosen as the target EML because it has a broad support for complex learning processes (CLPs) and would be feasible to translate the same course design into other specifications.

There have been several approaches to create learning patterns for the IMS-LD, particularly for collaborative learning (Asensio et al., 2004). The objective of these patterns is to provide the means to reuse the solutions found for certain problems during the design of a course. Although the collaborative learning is widely used, there are several other learning techniques that are worth exploring, even more, if those patterns can be created and then reused in other learning methods.

The objective of this paper is to present a high-level of specific purpose authoring tool for the LPCEL specification, which can facilitate the authoring process of a course by means of a visual abstraction of elements as well as the use of patterns, web services and EML-specific translations.

3. Related Work

Instructional designers and teachers are required to know the low-level details (the XML tags and structure) of the Educational Modeling Language (EML) they are using to describe the design of learning scenarios. However, is it recommended for them to use a tool that facilitates the authoring process of a course, i.e. use graphical abstractions for the low-level primitives of a given EML (Martinez et al., 2008).

Here we present some visual approaches that simplify the authoring process of a course for the IMS-LD. The ReCourse editor (Griffiths et al., 2009) has the capability of displaying the flow of activities for a given UoL, but it has not a wide support for the Level B. The CompendiumLD (Brasher et al., 2008) is a tool resembling a mind-map where the user is able to create sequences of tasks, but it doesn't have a mechanism to translate the design to a standard EML. The MOT+ editor (Paquette et al., 2005) can create UoLs through its own graphical language, but the UoLs are only part of the Level A. The ASK LDT (Sampson, Karanpireris & Zervas, 2006) has a graphical representation for the primitives of the Level A and the Level B is managed through the properties, but the tool appears to be designed for IMS-LD experts. The UML4LD (Laforcade, 2007) is used to reverse-engineer an IMS-LD compliant XML file, however, one of its limitations is that it doesn't allow the editing or creation of learning scenarios. The LAMS tool (Dalziel, 2003) presents a set of options to design, manage and deploy learning scenarios; it uses its own EML named LAMS LD and its own graphical representation, but its level of expressiveness is narrower than the one provided by the LPCEL. The FLEXO editor (Dodero & Torres, 2010) provides a visual and text editor to manage the properties of the IMS-LD primitives in order to make a good use of the Level B of the specification, but is limited by the level of expressiveness of the IMS-LD. The COLLAGE tool (Hernandez et al., 2006) provides a set of patterns for collaborative learning environments, but there is no possibility for creating or modifying patterns.

In brief, most of the presented visual tools are IMS-LD compliant. There are some interesting approaches, such as LAMS and FLEXO, which can be used to easily design learning scenarios using their own EML (e.g. LAMS LD) with the possibility for such design to become IMS-LD compliant. However, they all are limited by the IMS-LD level of
expressiveness. Although the same can be said about the COLLAGE tool, the most interesting part of it is that it gives the user the possibility to design a course using patterns for collaborative learning scenarios.

Because of the limitations of the previously mentioned approaches, a tool that can describe a more expressive EML is needed, so the user can be able to represent CLPs and be also able to create and make use of different learning patterns. The tool must be based on an expressive language, such as the LPCEL.

4. The LPCEL Web Editor for learning design

The LPCEL Editor is composed by a set of elements that can facilitate the authoring process of learning scenarios. The architecture of the LPCEL Web Editor is composed by three main components as depicted in Fig. 1: (1) The Visual Editor, (2) The Intermediate Representation, (3) The EML-Specific Converters, (4) The Learning Patterns and (5) The Use of web services.

![Fig. 1. LPCEL Editor Architecture](image)

4.1. Visual Editor

Model Driven Development (MDD) is a software engineering approach consisting of techniques to produce models rather than computer programs (Atkinson & Kuhne, 2003). The MDD uses models that are less bound to a target implementation and are more closer to the problem being solved, enabling the user to focus on solving problems rather than the implementation issues.

Some of the characteristics presented by MDD are very suitable for the learning design domain. The Model-Driven Learning Design (Dodero, Tattersall, Burgos & Koper, 2006) is an interesting approach because it proposes a set of high-level authoring tools (visual abstractions) and how they can be used to produce XML files compliant with a specific EML. Tools using this approach must allow the users to design a course without needing them to know specific low-level XML details about a particular EML.

The LPCEL Editor uses a subset of the Business Process Model and Notation (BPMN) 2.0 (OMG, 2010) as the visual layer to create an abstraction for the LPCEL primitives. The BPMN is a graph-oriented language where the flow of the nodes can be specified in a simple fashion. The subset used by the LPCEL Editor can be divided into three groups: Basic Elements, Simple elements and Artifacts. Using these elements, the user can be more focused on the pedagogical intention and the learning objectives of a given learning course, without being aware of the XML implementation details for the same scenario.

4.2. Intermediate Representation
The Intermediate representation language for the LPCEL visual abstraction is created in plain text using JSON. The JavaScript Object Notation (JSON) is a lightweight, text-based, human-readable data interchange format. The JSON structure is created to correctly represent the selected state of the visual abstraction created in the LPCEL Editor. One of the main reasons to use JSON as the intermediate language is because it is considered as best suited for JavaScript applications (Nurseitov, Paulson, Reynolds & Izurieta, 2009). The LPCEL Editor uses JavaScript to handle some elements of the visual layer and the same language is used to create and process the intermediate representation.

4.3. EML-Specific Converters

The LPCEL Editor uses the visual component as the main authoring tool. Using this method, as recommended by the MDD approach, the users are able to focus only in the educative intentions of the activities and not in the low-level details. However, the visual abstraction only makes sense from a pedagogical point of view but it cannot be executed by a specific engine; the visual representation must be translated into a particular EML. The LPCEL Editor supports the use of plugins in order to create such translations. The EML-specific plugins must be created by the user; this requires programming skills and in-depth understanding of the target EML, but that discussion is out the scope of this paper.

The translation of the intermediate representation is feasible because of the broad level of expressiveness provided by the LPCEL. It also means that, although translations are possible, the user must acknowledge that some level of expressiveness could get lost in the process. For example, if a course is saved as an XML file compliant with the LPCEL specification, it is guaranteed that it will preserve a broad level of expressiveness. However, if the same scenario is transformed into a file compliant with IMS-LD, some expressiveness will get lost due to the maximum level possible within that specification.

4.4. Learning Patterns

The importance of identifying reusable collaborative learning patterns is highlighted by Hernández, Pérez & Dimitriadis (2004). However, the same can be truth for any learning pattern, because they can facilitate the design of learning scenarios. Patterns must be abstract enough to avoid implementation details and must provide sufficient information about their potential uses.

In the software engineering field, patterns have been divided into different categories according to the need they are intended to solve. However, and for educational purposes, we pose the need to have only two levels: High-level and Low-level detail. The high-level patterns provide the overall set of activities that need to be fully described by the user, while the low-level patterns are those detailed set of activities. For example, a task in a high-level pattern would only specify that the learner must review some topic, while the low-level detail would be the detailed activities, e.g. read chapter one, make essay, make test.

There is also the need for a formal specification to store educational patterns. Based on the classification proposed by Gamma, Helm, Johnson & Vlissides (1993) for the software engineering patterns, we suggest a set of fields (structure) that can describe learning patterns for the LPCEL Editor. For this structure proposal, we only take into consideration the fields that have some educational relevance. Using the pattern structure and classification depicted in Table 1, users can successfully make use of the patterns that are included in the editor and they are also able to manage the patterns they create.

Table 1. Learning Pattern Structure

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>The name for the pattern</td>
</tr>
<tr>
<td>Intent</td>
<td>What does the pattern do? What is the intent? What particular design problem does it address?</td>
</tr>
<tr>
<td>Motivation</td>
<td>Scenarios in which the pattern is applicable</td>
</tr>
<tr>
<td>Applicability</td>
<td>The situations in which the pattern can be applied</td>
</tr>
<tr>
<td>Participants</td>
<td>Describe the actors involved in the execution of the pattern</td>
</tr>
<tr>
<td>Collaborations</td>
<td>Describe how the participants collaborate to carry out the activities specified in the pattern</td>
</tr>
<tr>
<td>Diagram</td>
<td>The graphical representation of the pattern (preview of the structure previously saved)</td>
</tr>
<tr>
<td>See also</td>
<td>List of other patterns that are closely related</td>
</tr>
</tbody>
</table>

4.5. Use of web services
A web service is a software system designed to support interoperable machine-to-machine interaction over a network and are normally described using the WSDL standard (W3C, 2004). The LPCEL Editor supports the integration and testing of web services, which is possible due to its level of expressiveness.

The LPCEL Editor provides the tools needed to include and test educational web services for learning scenarios. The objective of this functionality is to facilitate the inclusion of educational web services by means of a simple wizard instead of asking for XML tags and attributes. Basically, the user only makes use of a wizard where some of the key components of the web services are requested (the URL, service, operation, etc.) while the remaining information is gathered automatically. Before the process is completed, the editor provides the mechanism to test the web services so the user can be sure that the selected service is still useful. The objective of testing a web service during the design stage is to verify if a particular web service or operation is still available or functional, this way the user can check if the service is still operational in order to avoid runtime errors in the execution stage of the learning scenario.

5. Evaluation

The LPCEL Editor has been subject to a case study evaluation as recommended by Hevner, March, Park & Ram (2004) and it is done using an actual course design. In the Monterrey Institute of Technology, Mexico, inside the B.S. Computer Systems Engineering, there is a modality in which some courses are gathered together so the students can be able to deliver a complex project. The objective is to simulate real life scenarios in which they will make decisions in order to deliver what the clients demand.

During the evaluation periods, the students must deliver an incremental version of the final system; such a milestone consists of the following tasks: The leader manages the requirements and assigns them to the other teammates; the programmer implements the requirements as planned; the tester verifies the work done by the programmer; the documenter writes the software documentation. Once these stages are done, it is first evaluated by the client and later for each one of the teachers who evaluate the students according to the specific criteria of their course. Finally the teachers decide the final grade for each one of the students in their respective course (in this example are Course A, B and C).

If the teacher designing this course needed to use a specific EML to describe such process, it would be complicated to use the low-level XML details of that EML. Fig. 2 presents only a small part of the XML tags of the LPCEL representing case study; it focuses on the set of roles and the first two activities for the students (manage and implement requirements) as the definition of the role that will perform each task.

```xml
<LPCEL xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="lpcel.xsd"> ...
</Roles>
</Complex-Learning-Process>
</Component-CLP>
</Complex-Learning-Process>
</Component-CLP> ...
</LPCEL>
```

Fig. 2. LPCEL XML representation of the course design

From the example above, it is evident that even designing a simple scenario using XML low-level details can be very difficult for the users (e.g. teachers) who are not used to work with specific XML tags and structures. The task
gets more complicated when the same learning scenario needs to be designed for different EMLs, in such cases, the user must manually write the same scenario for the different EMLs.

However, this task can be facilitated by using the visual elements provided by the LPCEL Editor. The same course design and sequence of activities mentioned in the case study can be represented using the LPCEL Editor visual approach as depicted in Fig. 3. One of the main properties of the visual representation is how easy it presents the pedagogical intention of the sequence of tasks, and how intuitive and understandable it can be for the teacher. It is important to notice how the learning scenario structure (sequence of activities, the roles and which tasks every role performs) can be understood just by analyzing the visual representation. Using this visual abstraction the user can focus on the learning objectives and how the complete learning scenario must be executed rather than the low-level implementation details.

![Visual representation of the course design](image-url)
On the other hand, there is also the intermediate representation which is based on JSON. The file can serve several purposes, e.g. as an exchange file (other than XML) and it can also be read and understood by the user. Because of the human-readable property of the intermediate representation, the user can understand the complete learning scenario without uploading it first into the tool. This property becomes important when the user only need to make minor adjustments to the intermediate representation.

Using this intermediate representation and because of the LPCEL level of expressiveness, it is possible to create XML files compliant with a specific EML. The plugins are yet to be implemented, but given the broad level of detail stored in the intermediate representation, as well as the level of detail the user can manipulate using the visual tool, the EML-specific extensions are feasible. The user must also take into consideration the fact that a 1 to 1 mapping between two different EMLs are not possible. Because of the different levels of expressiveness found for each EML, it is not possible to translate from the intermediate representation to a specific EML without losing some expressiveness in the process (except for LCPEL).

The use of patterns for the LPCEL editor can be achieved by using the tools it provides to store, group and reuse the patterns preloaded within the tool, but also for those created by the user. To illustrate the different levels of detail for the pattern classification presented before, we make us of the same case study.

The modality presented in the case study by the Monterrey Institute of Technology, Mexico, is based on a Project Oriented Learning (POL) method. According to Martin (2005), the POL method consists of a set of high-level sequence of activities and roles. This high-level detail of activities can be seen in Fig. 4, and it presents a set of complex activities such as the parallel, sequence, and sub-process. Notice that this is only a part of the complete POL process, but is a good example of the nature of the high-level classification, because it only presents some high-level activities but not the specific tasks.

![Fig. 4. Example of POL High-level detail pattern](image)

For example, if this pattern is applied to the case study for the individual task of the students, then there is a need to change the activity for a complex sequence. Making these changes using plain XML would be a complicated task as can be seen in Fig. 5. In this example the task Manage Requirements has been divided into Get all requirements and Write requirements. It is important to notice that, again, the user must have in-depth knowledge of the target EML in which the course is being designed because some advanced tasks might not be so obvious to perform by non-expert users.

![Fig. 5. LPCEL XML complex sequence](image)
The advantages of using visual elements to represent complex learning scenarios is highlighted in the last example, mainly because creating a sequence of activities needs only one visual element while doing the same task using the XML low-level details needs not only several lines of code but also an in-depth understanding of the target EML.

If the user needed a more advanced low-level arrangement of tasks for the sequence activity, then using the XML tags and structure would be another complicated task. However, it can be facilitated using the LPCEL Editor. For instance, if the individual activity for the learner is a very complex set of resources and activities, then it is easier to use the editor rather than write the XML tags and structure, i.e. if it was difficult to divide a basic activity into a complex sequence then it would be more complicated to create a complex structure like the one depicted in Fig. 6, where an external resource is specified for the activity A.

![Fig. 6. Complex sequence of activities](image)

The user is also able to store this kind of complex structures as patterns if it is a good solution for a recurrent problem, then the user can set the complete structure as a low-level pattern. It must be saved as a low-level detail pattern because it contains specific task and resources.

The LPCEL Editor provides the tools to manage the patterns by requiring the user to fill specific fields about the new patterns. An example of some sections of the web form can be depicted in Fig. 7 and is completely based on the pattern structure proposed in Table 1. There are two particular fields that are important to consider because of their relevance: The Diagram and See also fields.

Although all of the fields are necessary and serve specific purposes for the user and the editor, the Diagram and See also fields are more useful from a design point of view. The first field presents a preview of the diagram. This is particularly useful when the user needs to see if a selected pattern is, in fact, a good solution to a given problem. The last field represents the possibility for the user to navigate among similar pattern in order to find the one that resolves best a specific problem.

![Fig. 7. Web form to save solutions as patterns](image)

In Fig. 6 and external resource was specified for the activity A. Using the visual tools, this task was done using one simple icon (the resource element). However, the LPCEL specification requires some information about the web service that is being set for the activity. Fig. 8 presents only the critical information (URL, Service name, port, namespace etc.) that any user would need to set for a web service in the LPCEL. Also, it is important to mention that the user would need to create the web service client (web forms) in order to use the selected web service in the execution stage of the learning scenario (as specified by the LPCEL specification).

```xml
<LPCEL xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="lpcel.xsd">
...
```
Using the XML tags, the process of assigning web services to activities can be very difficult for two reasons. One is that there is some information that the user needs to get from the WSDL file describing the web service, and a non-expert user might not know where to get such information. Two, the user must place this information in the right structure and using the appropriate LPCEL XML tags. Of course, the same task is just as difficult when it comes to writing the same web service description for other EMLs (if the specification supports this characteristic).

It is important to highlight how easy it is for the user to attach an external web service to a task using the LPCEL Editor. In this case, such a process can be accomplished using the Web Services Wizard as presented in Fig. 9. The user only needs to select the web service and operation, configure the web service client (web forms) and finally test the web service (optional). So all of the work needed to complete the XML description of the web service can be summarized in three simple steps where the user only select the same basic information, but in an intuitive and easy-to-use manner. Once the main steps of the wizard are completed, all of the other important information about the web service is gathered automatically by the LPCEL editor; the web service client is also auto-generated.

Using the wizard allows the user to set all of the information about the web services that will be used by the activities in the execution stage. There is no need for the user to write XML tags to describe the service. Also, the user can use the wizard in order to verify the correct functionality of the web service or even if it is still functional; in this example it is just the sum of two numbers.

Fig. 8. LPCEL XML web service description

Fig. 9. Web Services Wizard
6. Conclusions

In this work we have developed a visual authoring tool compliant with the LPCEL specification. The objective of this tool is to simplify the authoring process of complex learning scenarios. Most of the current visual approaches are IMS-LD compliant; however, they are restricted by the level of expressiveness of that specification. The LPCEL was chosen as the target EML because of its broad level of expressiveness.

Using a visual approach for the authoring of a course allows the user to focus only on the high-level details of the course (such as pedagogical intention, teaching methods, etc.) without being aware of the XML low-level details of the target EML. The user only works with the visual tool to design a course and the translations can be achieved through the use of plugins.

The design of learning scenarios can also be facilitated through the use of learning patterns. Proving the necessary tools for the user to create, edit and make use of several patterns is a particular objective for this visual authoring tool, because it can enable the reutilization of known and good solutions to common problems found among several learning scenarios and by reusing proven solutions, the non-expert users can learn from expert instructional designers or highly skilled teachers.

The use of educational web services is another characteristic that complex learning scenarios must have. It is important for the user to have a more user-friendly mechanism to specify web services for the activities; a tool that helps to accomplish this goal is another improvement for the authoring process, mainly, because the user does not need to know the complete XML structure to represent them.

There are some characteristics that need to be improved, such as the EML-specific plugins which are yet to be implemented and because a 1 to 1 transformation is not possible, there are some issues that must be handled accordingly in such a process. The collaborative authoring of a course is also being considered, so the users can design a course as a team.

References


282


