Transformational techniques for model-driven authoring of learning designs

Juan Manuel Dodero[1] **Colin Tattersall**[2] **Daniel Burgos**[2] **Rob Koper**[2]

[1] *Computer Science Department, Universidad Carlos III de Madrid, Madrid, Spain*

[2] *Educational Technology Expertise Centre (OTEC), Open University of the Netherlands, Heerlen, The Netherlands*

Abstract

Diverse authoring approaches and tools have been designed to assist the creation of units of learning compliant to current learning technology specifications. Although visual and pattern-based editors of Learning Designs (LD) can help to abstract the learning designer from the details of the specifications, they are still far from a high-level, integrated authoring environment. This paper analyzes the major approaches used to transform an abstract LD into a concrete unit of learning (UoL), according to three desired features: the use of patterns and other design techniques to abstract the specific representational details; the difference between the abstract source LD model and the concrete target UoL model; and the possibility of combining multiple models into a single environment. A classification is proposed for the LD techniques commonly found in the analyzed approaches, in order to underline its abstraction from the details of the underlying specifications. We have integrated such LD techniques in a unified Model-Driven Learning Design (MDLD) meta-modelling environment, which has been used to generate UoLs from a number of meta-models. The model-driven development process was studied on the creation of a IMS LD UoL for the Learning Networks' knowledge base.

Keywords: Model-driven development, learning design patterns, IMS Learning Design, unit of learning

1. Introduction

The increasing adoption of the IMS Learning Design (LD) specification has provided a common ground for the digital representation of learning designs. Learning design deals with exploiting prescriptions from instructional design theory plus examples of best practices and patterns of experience, which are then applied to develop concrete Units of Learning (UoL) [1]. Teachers can avail themselves of LD authoring tools to perform such a complex task, thus creating UoLs that adapt to specific instructional needs.

A recent report on this topic from the UNFOLD project shows that most tools are still too close to the technical formalisms and underlying specifications of the digital *artefacts* they assist to create [2]. For instance, although level B properties and conditions are powerful primitives to write adaptive UoLs [3], adding them may be actually burdensome, since they are too low-level and have to be expressed in XML. The significant shortage of high-level authoring tools prevents teachers from keeping their distance from the specification, so they have to learn and use technical concepts that are far from their pedagogical backgrounds and contexts [4] This becomes especially difficult if teachers have to learn more than one specification, as it happens in netUniversitè [5]; or if they have to create learning content, metadata, and learning designs separately and afterwards integrate them manually, as required by RELOAD [6].

Most computer-assisted authoring environments differentiate between the language used to interact with them (i.e. the *source* language), and the language of the artefact that is eventually generated (i.e. the *target* language). The source LD language can be graphical, as in MOT+LD, or textual, like in the IMS LD XML binding. Graphic tools rely on user-defined visual representations, from which XML descriptions of an IMS LD can be easily generated. However, shapes and drawing elements usually keep too close to the core IMS LD model, as it happens with ASK-LDT [7].

Some graphic tools usually describe a number of models that put together make up the overall source LD. When such generic and shared models are used as learning design elements, the resultant LD can be mapped to a single UoL that is compliant with the IMS LD specification. This approach is taken by MOT+LD [8] to develop models of knowledge, competences, pedagogical structure, materials and delivery, which are eventually transposed to concrete IMS LD UoLs. However, the occurrence of multiple models is not restricted to the LD source, but it also can occur in the target UoL. For instance, an IMS LD UoL can be readily merged with resources described with XHTML [3], QTI [9] and other specifications. In particular, integrating IMS LD and QTI requires clear-cut manipulations of manifest files to bind certain QTI elements (e.g. *score* and *feedback*) with the proper IMS LD Level B properties.

Some works such as the CPM meta-model [10] define a UML profile plugged into a commercial meta-CASE environment to provide abstract models, which constitute the basis of a model-based approach to engineer learning designs. Although not strictly comparable to the pedagogically neutral IMS LD, CPM provides a problem-based pedagogical modelling approach that requires knowledge of UML and a vendor tool, which can be still a barrier for teachers and specialists in pedagogy.

To overcome the former issues, learning design patterns have been proposed as a means to incorporate template-based pedagogy in a half-cooked learning design [11]. LD pattern-based tools as Collage [12] are a powerful way to alleviate teachers and designers' inconveniences caused by the undesirable proximity to the IMS LD specification. The seamless integration of patterns into unified LD methods and tools is still required, so they become the foundation of model-based design approaches, which are the focus of this work.

The rest of this paper is structured as follows: first, a characterization of major LD authoring environments is presented, and some challenges observed in the analyzed tools are detailed, specially focused on the use of design abstractions as patterns. Then a classification of relevant pattern and idiom-based LD techniques is provided. An explanation of how such techniques can be integrated in a model-driven development environment is followed, for which a meta-modelling tool and a case study have been developed and tested. Finally, we provide some conclusions and future work.

2. Challenges of LD authoring

The background challenge found in many authoring approaches is how to raise the level of abstraction at which teachers operate when they create a learning design. Instead of dealing directly with low-level descriptions of LD, software engineering design techniques are commonly used to shortcut the way to the final UoL. This section discusses some issues and provides a characterization of the most widespread authoring tools. The feature selection criterium for the characterization has been the provision of abstraction facilities to isolate users from the specification of the final UoL and resources. Such features are structured in the following categories:

- *Design abstractions*: This category encompasses whether and which learning design abstractions (e.g. patterns) are used to raise the abstraction level, and if different design patterns can be combined and applied to the same UoL. This challenge can be difficult or even meaningless, and it is comparable to the combination of different source models, usually undertaken by defining meta-models and model transformations. Examples of this category are MOT+, CPM and Collage.

- *Source vs. target model*: There is a difference between the source model (i.e. the one to be known by the user to compose an LD) and the target model used by the resultant UoL. For instance, some approaches (e.g. MOD+LD, CPM) are founded on source pedagogical models, which are eventually transposed into the target IMS LD model for the generated UoL, whilst others (e.g. RELOAD, ASK-LDT, Collage) consider IMS LD as both the source and target model.

- *Multiple models*: Some development tools allow the combination of elements from different models or specifications within the same target UoL, while others are model-specific. For instance, the netUniversitè environment [5] allows including IMS LD and QTI elements together in the same UoL through a single authoring environment. Other tools as RELOAD strictly adhere to a number of learning technology specifications, but do not allow combining them in the same UoL. The combination of models can also be carried out on multiple source models, as in MOT+.

The model-based engineering of learning situations [13] uses the CPM meta-model to produce dedicated models that fit a concrete learning situation. The dedicated model is afterwards merged with an abstract model describing part of the context. Therefore, the objective of keeping learning designers distant from the technological details of the UoL or learning product can be achieved by integrating higher-level abstractions into computer-assisted authoring environments. Such abstractions can be manipulated and transformed to produce the eventual UoL, as described below.

3. Transformational techniques for learning design

In software engineering, several design techniques have proven to be helpful in automating software manipulations [14]. These are classified according to the cross-linguality and significance of the transformations required to map, merge and extend software models. Although they are applied on regular software artifacts, they can also be used to map an abstract LD into concrete UoLs. When it comes to LD, such techniques are summarized in the following categories:

- *Shortcutting*: generation of LD instances from parameterized templates that require less coding. Design patterns, idioms and the like are included in this category [15].
- *Mapping*: generation of an LD instance from mixing more than one partly-completed design.
- *Refactoring*: to perform modifications on an LD instance to become more efficient or reusable, without changing its original purpose [16].
- *Extending*: adding a number of LD elements which cannot be represented with a given LD language or it becomes so costly that it is better to use elements from a different language.

3.1 Shortcutting patterns and idioms

A design pattern names, abstracts, and identifies the key aspects of a common design structure that make it useful for creating a reusable design [17]. Pattern-based solutions are applied to LD with different levels of abstraction. On one hand, pattern languages define high-level collections of patterns and the rules to combine them, so raising the abstraction level of course design [18]. On a lower abstraction level, when a learning design pattern is specific to an LD language, it becomes an *idiom* [15]. An idiom is a pattern that describes how to implement particular aspects of an LD using the features of a given LD language (notably IMS LD). Although IMS LD can be used to represent higher-level LD patterns, when it comes to generating concrete UoLs, the required transformations have to be aware of IMS LD details such as level B properties and conditions. Idioms provide an abstract expression of such details in IMS LD. Next, a purposeful catalogue of IMS LD idioms is proposed to support the development approach described thereafter.

Although many kinds of IMS LD idioms can be identified, depending on what elements of the IMS LD specification have to do with the idiom, we only selected those that provide a higher-level abstraction for level B elements, like properties and conditions, which are particularly difficult to abstract in authoring tools. Each idiom is described in Table 1 by a UML activity diagram and an associated script containing the binding to the IMS LD information model, which drives the transformation used to generate the UoL. This way, the designer only selects the desired activity idiom from the catalog, and the appropriate transformation is then applied. For brevity we summarize only a handful of IMS LD idioms used to describe common structures of learning activities:

- *Alternative activities* (vid. Table 1(a)): Only one of a number of activities is executed depending on a previous condition

- *Forked activities* (vid. Table 1 (b)): The learning flow is split among a number of concurrent activities, which since then will run in parallel.

- *Rendezvous* (vid. Table 1(c)): Two or more concurrent learning activity flows meet on a synchronization point (i.e. the *rendezvous*), and then continue together on a single flow. If there are flows that arrive later, they have to wait for the rest.

- *Guard-synchronized activities* (vid. Table 1(d)): The rendezvous synchronization is augmented by the satisfaction of a boolean condition. When it is false, all learning flows must wait; when it becomes true, the behavior is like the rendezvous idiom.

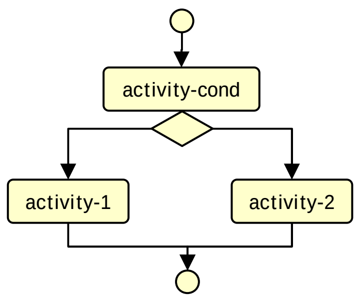
For clarity, the activity idioms described above include only two activities, but the cardinality can be extended by the iterative application of a two-activity idiom, without losing generality. The binding scripts avoid the verbosity of the IMS LD XML binding. From the scripts of Table 1, a simple substitution of parameters —marked with $— is applied to generate the XML-based UoL chunks. These are not still fully-fledged UoLs, since they miss the role and method role-part definition. Nevertheless, they are closer to the sought-after runnable condition of the UoL [19].

3.2 Mapping

The generation of UoL chunks from the idiom scripts is a one-to-one mapping from the model used to express the idiom (i.e. from the idiom script language) to the model in which the UoL chunk is based (i.e. the IMS LD XML binding). Nevertheless, other kinds of transformations may require conformance to more than one model, in which case more complex mappings are required. On the one hand, one-to-many model mappings are useful to keep the user away from knowing more than one model. For instance, the teacher can create an LD containing both activities and assessments without needing to know about IMS LD or QTI and, more importantly, without having to know the way of connecting elements from both specifications —based upon binding level B properties and QTI item variables. On the other hand, many-to-one model mappings (sometimes called model merging) allow the definition of separate aspects on the development of a learning artefact, as well as the division of responsibilities in a group of designers.

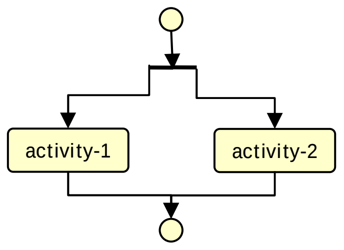
One-to-one mappings are used to transform between single meta-models. These mappings can be intra-language or inter-language, depending on the source and target meta-models. In intra-language mappings, a simple sequence of modelling actions describes the transformation from a model to another complying with the same meta-model (for instance, a transformation to enlarge the number of activities of an IMS LD). In inter-language mappings, the source and target meta-models are different (for instance, a transformation from UML to IMS LD). The mapping effort depends on the semantic distance between the concepts of both models (e.g. UML 2.0 activities, lanes, and object nodes versus IMS LD activities, roles, and environments). In such an example, activity diagram lanes are mapped to IMS LD roles, as usually done in the IMS LD specification.

In a sense, one-to-many model mappings can be considered as one-to-one mappings between the source model and an instance of the non-explicit meta-model formed by the combination of the target models. For this reason, we do not focus on the mapping cardinality, but on defining appropriate meta-models instead.



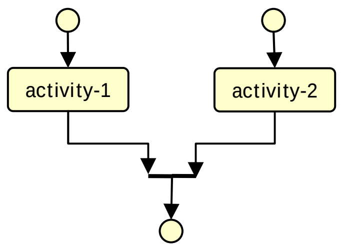
```
locpers-property $LP-COND { datatype: boolean; initial-value: false; }
learning-activity $A1, $A2;
activity-structure alternatives {
    structure-type: selection; number-to-select: 1;
    activities: $A1, $A2; }
activity-structure alt-activ-idiom {
    structure-type: sequence;
    activities: $A-COND, alternatives; }
condition {
    if completion($A-COND)
        if ($LP-COND) then { show($A1); hide($A2) }
            else { show($A2); hide($A1) }; }
```

(a) Alternative activities idiom



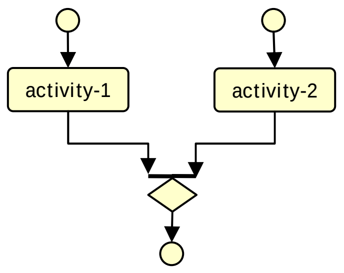
```
learning-activity $A1, $A2;
activity-structure fork-activ-idiom {
    structure-type: selection; number-to-select: 2;
    activities: $A1, $A2; }
```

(b) Forking activities idiom



```
locpers-property comp_a1, comp_a2, comp_rv {
    datatype: boolean; initial-value: false; }
learning-activity $A1 { on-completion: comp_a1=true; }
learning-activity $A2 { on-completion: comp_a2=true; }
learning-activity rendezvous {
    isvisible: false; complete-activity: when comp_rv==true; }
activity-structure forked {
    structure-type: selection;  number-to-select: 2;
    activities: $A1, $A2; }
activity-structure rendezvous-idiom {
    structure-type: sequence; activities: forked, rendezvous; }
condition {
    if (comp_a1 and comp_a2) then comp_rv=true; }
```

(c) Activity rendezvous idiom



```
locpers-property comp_a1 { datatype: boolean; initial-value: false; }
locpers-property comp_a2 { datatype: boolean; initial-value: false; }
locpers-property comp_rv { datatype: boolean; initial-value: false; }
locpers-property $LP-COND { datatype: boolean; initial-value: false; }
learning-activity $A1 { on-completion: comp_a1=true; }
learning-activity $A2 { on-completion: comp_a2=true; }
learning-activity rendezvous {
    isvisible: false; complete-activity: when comp_rv==true; }
activity-structure forked {
    structure-type: selection;  number-to-select: 2;
    activities: $A1, $A2; }
activity-structure guard-sync-idiom {
    structure-type: sequence; activities: forked, rendezvous; }
condition {
    if (comp_a1 and comp_a2 and $LP-COND)
        then comp_rv=true; }
```

(d) Guard-synchronized activities idiom

Table 1. IMS LD activity structure idioms. Each idiom is described by a UML activity diagram (left column) and a binding script (right column) from which the IMS LD XML elements are generated

3.3 Refactoring

Refactorings are intra-language mappings usually applied to improve the efficiency or reusability of a model. Although refactoring transformations do not really encompass different abstraction levels for the source and target model, they can be easily automated so that users only have to select the appropriate refactoring method to be applied. For instance, the application of the IMS LD activity rendezvous idiom (see Table 1(c)) results in adding local personal properties to hold the completion of each activity. On another hand, if each activity contained a QTI item to mark their finalization, the IMS LD-to-QTI mapping would result in the definition of additional properties for the same purpose. Automated refactoring inspections can help to abbreviate the eventual UoL and merge such duplicated properties. This does not add a new functionality, but improves the efficiency of the target UoL.

3.4 Extending

In inter-language mappings, either the source or the target meta-model can define concepts that are not present in the other. For instance, UML 2.0 does not have a form to model roles, like IMS LD does. Such cases are usually dealt by extending the source meta-model. For example, CPM uses the UML extension mechanism based on profiles to augment the set of available modelling elements. On another hand, if the target meta-model has no means to represent certain elements of the source model, then the target model can lose information. For instance, with ASK-LDT the user can define and configure a library of activity types, which are eventually transformed to IMS LD activities only. But these can be only of two kinds, namely learning or support activities. In these cases, extensibility could be both an advantage and a risk, since it would offer a chance to enhance the expressive power, but it may compromise reusability and interoperability.

4. Model-driven learning design

Non-representational LD techniques such as the described above enable the *Model-Driven Learning Design* (MDLD). The intention of MDLD is to develop specific learning technology-supported software artefacts such as learning designs. It enables tools to be provided for specifying a system independently of the platform that supports it, as well as for transforming the system abstract specification into the more concrete specification for a particular platform. Therefore, the objective of MDLD is to raise the abstraction level at which learning technology systems are designed. As a consequence, the MDLD integrates diverse non-representational abstractions to specify an LD.

The purpose of the MDLD is to integrate non-representational techniques in a single environment that relieves the learning designer from lower-level LD authoring tasks. In order to test it in practice, we have built a meta-modelling tool based on Eclipse and the FXL plug-in [20], which provides two complementary views of model-based development:

- The *activity modeler* view (vid. Figure 1) uses the Business Process Execution Language (BPEL) plug-in to model activities at the PIM level.
- The *transformation modeler* view (vid. Figure 2) provides a graphic user interface to edit transformation pipelines from the PIM to the PSM level.

The model-driven transformation process is defined as a pipeline, which receives a number of source models on the input, and generates other models on the output. This process is accomplished in a number of steps defined through the transformation modeler. On the other hand, graphical editing of the models is carried out with the activity modeler. In its current prototype, the activity modeler allows editing learning activity structures, which are enough to define the IMS LD activity idioms.

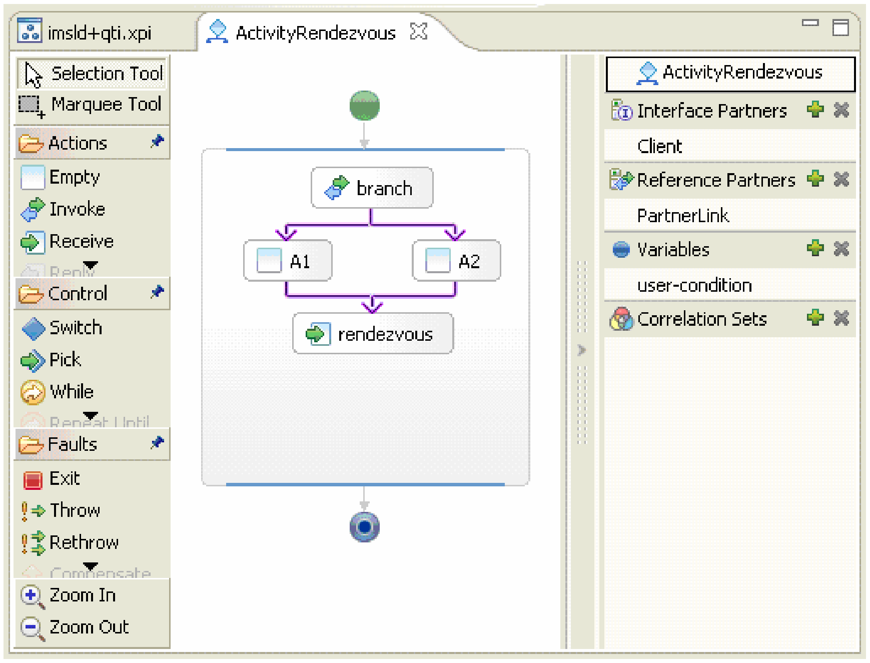


Figure 1. Activity modelling view of the MDLD environment. The screen represents the BPEL modelling of an activity rendezvous pattern, from which the corresponding IMS LD idiom is later generated.

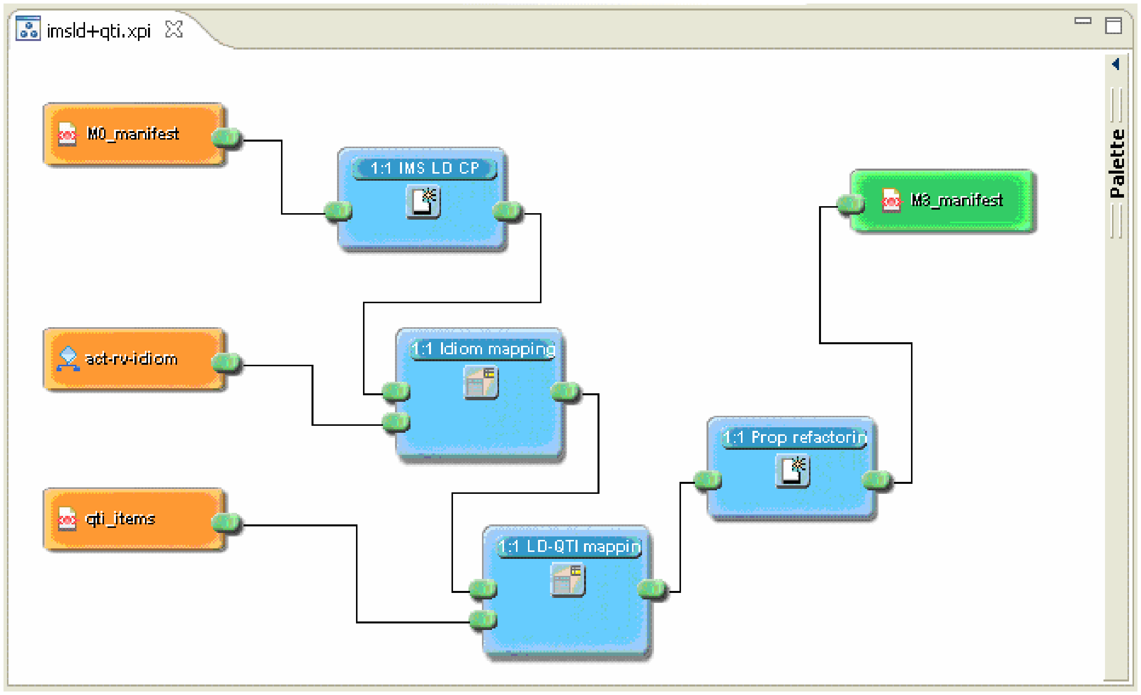


Figure 2. Transformation modelling view of the MDLD environment. The screen represents the transformation pipeline used to generate the IMS LD manifest file of a UoL through a number of transformational steps.

The transformation pipeline has to be defined only once —i.e. the teacher is not required to use the transformation modeler. The teacher only has to choose the LD idiom and select the QTI items from the library, and then apply the transformation. On the other hand, the teacher is not forced to define a concrete IMS LD activity structure, but she can simply select it from the library of IMS LD idioms. Nonetheless, the activity modeler provides the learning designer with a higher-level way of dealing with activities.

Activity abstractions are represented with BPEL. The expressive limitations of BPEL are clear (e.g. concerning user roles), since it is not designed for instructional design. To overcome this lack of expressiveness, an extension was provided by adding scripts to each IMS LD idiom. Other alternatives to specify activity idioms' details, such as using UML profiles and tagged activity diagrams [10], were also explored. Nevertheless, BPEL was used because of its simplicity.

In order to show the possibilities of the MDLD, the meta-modelling tools described above have been used to change the traditional way of creating an IMS LD-based UoL into a series of transformation commands. The target UoL of the case study is based on an adaptive IMS LD taken from the Learning Networks' knowledge base [3], but elements from other specifications have been also incorporated —a set of QTI items grouped in IMS LD activities, which are synchronized by means of level B properties and conditions. Adding such items to a UoL may be a hindrance when the learning designer is not acquainted with the IMS LD and QTI specifications. Therefore, the objective of the case study is to show how non-representational LD techniques included in the MDLD development environment facilitate the creation of an LD exemplar consisting of different models. In particular, the learning design technique used to abstractly represent the activity structure has been the *activity rendezvous* IMS LD idiom.

The transformation pipeline followed in the case study is depicted in Figure 3. From left to right, the input to the pipeline is the M_0 manifest file, which does not contain any IMS LD information at all. First, a content package (CP) transformation is executed to prepare it for holding an IMS Learning Design. The result is the M_1 manifest. Second, M_2 adds to M_1 an empty activity structure according to the activity rendezvous idiom (*act-rv-idiom*). Afterwards, all the activities are filled in with QTI items, which are bound to the UoL by means of two level B properties and one condition (notably to bind the *score* and *feedback* QTI elements), thus resulting in M_3. Since the LD-QTI mapping added some redundancy on properties when it is combined with the rendezvous idiom, a final refactoring stage is executed to overcome this issue.

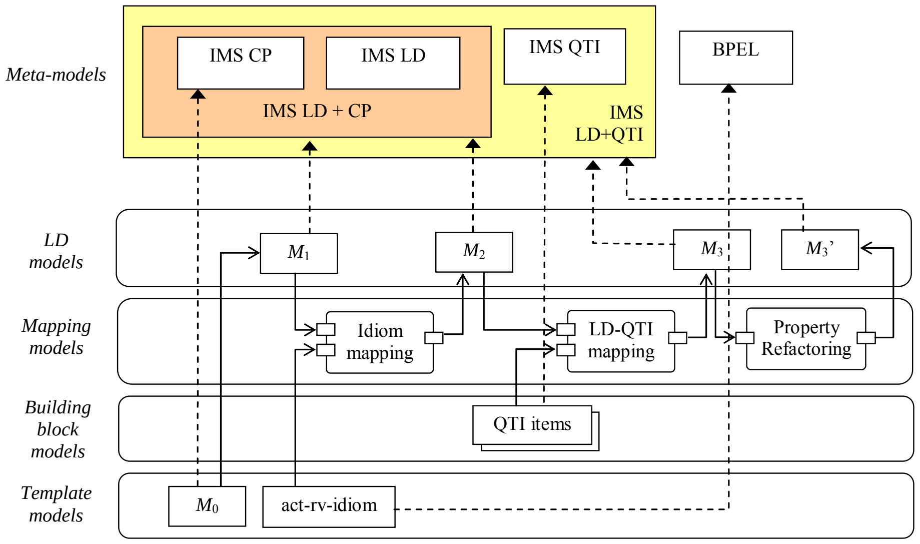


Figure 3. Scheme of the MDLD transformation pipeline of the case study. Models can represent templates, building blocks, mappings, or exemplars, depending on their granularity level and completion status [19]. Each artefact is an instance of a meta-model from the top of the figure, represented as dotted arrows.

The complete MDLD process delivers the M_3' UoL, which becomes functionally identical to M_3, but less verbose in using level B properties. Artefact M_3 contains elements from both IMS LD and QTI specifications. To generate this, it was only required to select the LD idiom and QTI items from respective libraries containing templates and building blocks. According to the terminology of Hernández-Leo *et al.* [19], M_3 and M_3' are exemplars (i.e. ready-to-run UoLs); M_1 and M_2 are LD templates (i.e. partly completed exemplars), M_0 is a non-LD template; QTI items are building blocks (i.e. partly completed UoL chunks).

5. Conclusion and Future Works

This paper presents a characterization of LD techniques aimed at providing high-level LD authoring environments. We have classified relevant design pattern-based approaches and enlarged them with others such as IMS LD idioms, mappings, extensions, and refactoring. All of them are combined in a model-driven learning design environment which enables the overall objective of distancing the author from the IMS LD specification. The experience of using the MDLD environment shows that the learning designer can generate an IMS LD UoL by combining several learning technology specifications, without requiring a detailed knowledge of them.

Although the list of idioms described in this paper is far from exhaustive —it only contained those which model IMS LD activity structures—, it was sufficient to test how they facilitate creating IMS LD level B UoLs. Nevertheless, if different LD roles were involved along with activities, the list of idioms should be extended to express the role-part collaborations with IMS LD. On the other hand, not all languages are equally suitable to express such idioms. Although BPEL resulted useful due to its simplicity, its expressive limitations (e.g. concerning the modelling of user roles) corresponds to the mapping case in which the source meta-model lacks concepts which the target model has. In the future, other transformational techniques based on extending the source modelling language are needed to overcome such limitations.

As a consequence, not all abstract modelling languages are equally suitable for LD authoring, and none of them is completely satisfactory to model complex aspects of the IMS LD specification. This motivated the need to define a Domain-Specific Language (DSL) for the educational domain, with which the MDLD environment can be extended.

Acknowledgement

This work is partly funded by the MODUWEB project (TIN2006-09768) from the Spanish Ministry of Science and Technology.

References

[1]. R. Koper, "An introduction to learning design". In *Learning Design: A Handbook on Modelling and Delivering Networked Education and Training* (Koper R. and Tattersall C., Eds.), pp 3-20, Springer, Berlin, 2005.

[2]. D. Burgos, and D. Griffiths, *The UNFOLD Project. Understanding and Using Learning Design*. Open University of The Netherlands, Heerlen, 2005.

[3]. R. Koper and D. Burgos, "Developing advanced units of learning using IMS Learning Design level B", *Advanced Technology for Learning* 2(4), 252-259, 2004.

[4]. D. Griffiths and J. Blat, "The role of teachers in editing and authoring units of learning using IMS Learning Design", *Advanced Technology for Learning* 2(4), 243-251, 2005.

[5]. E. Giacomini, P. Trigano and A. Sorin, "A QTI editor integrated into the netUniversitè web portal using IMS LD", *Journal of Interactive Media in Education*, 2005(9).

[6]. C.D. Milligan, P. Beauvoir and P. Sharples, “The Reload learning design tools”, *Journal of Interactive Media in Education*, 2005(7).

[7]. D.G. Sampson, P. Karampiperis and P. Zervas, “ASK-LDT: A Web-based learning scenarios authoring environment based on IMS Learning Design”, *Advanced Technology for Learning* 2(4), 2005.

[8]. I. De la Teja, K. Lundgren-Cayrol, and G. Paquette, “Transposing MISA learning scenarios into IMS units of learning”, *Journal of Interactive Media in Education*, 2005(13).

[9]. H. Vogten, H. Martens, R. Nadolski, C. Tattersall, P. Van Rosmalen, and R. Koper, “Integrating IMS Learning Design and IMS Question and Test Interoperability using CopperCore Service Integration”, *Proc. of the Learning Networks for Lifelong Competence Development Workshop*, p. 43, Sofia, Bulgaria, 2006.

[10]. T. Nodenot, P. Laforcade, C. Sallaberry and C. Marquesuzaa, “A UML profile incorporating separate viewpoints when modeling co-operative learning situations”, *Proc. of the First International Conference on Information Technology: Research and Education*, p. 605, IEEE Press, Newark, USA, 2003

[11]. P. McAndrew, P. Goodyear, and J. Dalziel, “Patterns, designs and activities: unifying descriptions of learning structures”, *International Journal of Learning Technology* 2(2), 216-242, 2006.

[12]. D. Hernández-Leo, E.D. Villasclaras-Fernández, I. M. Jorrín-Abellán, J.I. Asensio-Pérez, I. Ruiz-Requies, and B. Rubia-Avi, “COLLAGE: A collaborative learning design editor based on patterns”, *Educational Technology and Society* 9(1), 58-71, 2006.

[13]. T. Nodenot, C. Marquesuzaa, P. Laforcade, and C. Sallaberry, “Model-based engineering of learning situations for adaptive web based educational systems”, *Proc. of the Thirteenth International World Wide Web Conference*, p. 94, ACM Publications, New York, USA, 2004.

[14]. G. Caplat and J.L. Sourrouille, “Model mapping using formalism extensions”, *IEEE Software* 22(2), 44-51, 2005

[15]. F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture. Volume 1. A System of Patterns*, Wiley, 1996.

[16]. M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts, *Refactoring: Improving the Design of Existing Code*, Addison-Wesley, 1999.

[17]. E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1994.

[18]. J. Bergin, “A pattern language for initial course design”, *ACM SIGCSE Bulletin* 33(1), 282-286, 2001.

[19]. D. Hernández-Leo, A. Harrer, J. M. Dodero, J. I. Asensio-Pérez, and D. Burgos, “Creating by reusing learning design solutions”, *Proc. of the Eighth International Symposioum on Computers in Education*, León, Spain, 2006.

[20]. C. Reichel C, R. Oberhauser, “XML-based Programming Language Modeling: An Approach to Software Engineering”, *Proc. of SEA*, MIT Cambridge, MA, USA, November 2004.