Model-Driven Learning Design

Article *in* Journal of Research and Practice in Information Technology · August 2012

4 authors:

Juan Manuel Dodero
Universidad de Cádiz

Iván Ruiz-Rube
Universidad de Cádiz

Manuel Palomo-Duarte
Universidad de Cádiz

Jordi Cabot
Catalan Institution for Research and Advanc…

Model-Driven Learning Design

Juan Manuel Dodero, Iván Ruiz-Rube and Manuel Palomo-Duarte

Escuela Superior de Ingeniería, Universidad de Cádiz
C/ Chile 1, 11002 Cádiz, Spain
Email: juanma.dodero@uca.es, ivan.ruiz@uca.es, manuel.palomo@uca.es

Jordi Cabot

INRIA – École des Mines de Nantes
Rue Alfred Kastler, B.P. 20722 – F-44307 Nantes Cedex 2, France
Email: jordi.cabot@inria.fr

Learning Design is a framework of elements that are used for the formal specification of learning courses. Learning Design languages have been defined to facilitate the editing of online courses, usually including a number of technical formalisms that have proved to be scarcely comprehensible to non-technical staff. In this work we describe a model-driven development approach based on a Learning Design domain-specific language that makes it possible to author and generate learning design courses and to export them into languages and formats of common learning environments and course players.

Keywords: *Learning design, model-driven development, domain-specific languages*

ACM Classifications: *D.2.13 [Reusable Software]: Domain engineering, D.3.2 [Language Classifications]: Design languages, K.3.1 [Computer Uses in Education]: Computer-assisted instruction*

1. Introduction

Learning Design (LD) is an educational discipline that aims at converting a set of instructional or didactic principles into a learning environment (Smith and Ragan, 2005). From a technology-based viewpoint, the LD discipline aims at providing a framework of elements used for the formal specification of courses prepared to be played by a computer (Olivier and Tattersall, 2005). Such learning courses are normally hosted and run on software platforms known as Learning Management Systems (LMSs) or Virtual Learning Environments (VLEs), which often use a specific format to package and exchange the courses they store and manage. Because of this, Educational Modeling Languages (EMLs) and Learning Design Languages (LDLs) have been produced to express the software artefacts that result from the LD process.

Developing a generic LD language that makes it possible to author and deploy LD courses for the existing diversity of VLE platforms can be a challenge. The IMS LD specification (IMS, 2003) is a widely recognized LD language that has succeeded in doing so. Instructional designers and technology-enhanced learning researchers, however, have found it difficult to edit IMS LD courses (Burgos and Griffiths, 2005; Griffiths and Blat, 2005).

The authors of an LD may have a certain degree of contact with the conceptual structure of the LDL, depending on the authoring application and its usability. It seems reasonable to explore the

Manuscript received: 1 January 2012
Communicating Editor: José-Luis Sierra-Rodríguez

way in which learning designers engage with an LDL through an authoring application. However, the usability of the interfaces of the application (and hence users' effectiveness with them) does not reveal much about the complexity of understanding the EML. Derntl *et al* (2010) propose to tackle the complexity issue by implementing improved authoring metaphors in the tools rather than by providing improved versions of the LDL. Nevertheless, shifting complexity from the LDL side to the developer side would significantly increase the effort required to improve the usability of the tools.

This work deals with a model-driven approach to reduce the complexity of editing LDs (France and Rumpe, 2007). Model-Driven Development (MDD) and Model-Driven Engineering (MDE) provide a set of principles and techniques to formalize and represent specific knowledge about a particular interest. MDD is intended to build software artefacts more flexibly and quickly, based on the development and transformation of models. The main idea is to produce executable code through successive transformations of models provided at a higher level of abstraction. The expected benefit of MDD is a reduction in the complexity of LD development and, more importantly, in terms of reusability of the learning model of a course for different platforms.

The objective of the paper is to define a Domain-Specific Language (DSL) that allows an LD to be edited without being committed to the technical formalisms of a given LDL. The domain of such a DSL is the technology-enhanced LD discipline, which aims at formalizing how a set of people in specific groups and roles are engaged in the structure of learning activities, resources and services that describe a learning experience (Olivier and Tattersall, 2005). LD shares many features with the business process management domain (Karampiperis and Sampson, 2007), such as workflows, activities, and roles. In the field of learning experiences, however, *assessments* are especially distinctive (Strijbos, 2011). Assessments are inevitable when designing the activities, resources, and roles involved in a learning experience. Assessments have also been the basis of proposed extensions to IMS LD (Sitthisak and Gilbert, 2010). In this vein, assessments and assessment-based adaptations of the learning flow are the foundation on which our DSL is based.

2. Review of Related Works

LD languages define a model of elements (namely learning resources, activities, flow connectors, and user roles, among others) that can be combined to design the structure of a course. LD languages are thought to deliver a runnable version of the designed learning experiences to a course player engine (Martens and Vogten, 2005) that can run them. Such course players can also be embedded in specific learning platforms or VLEs (Escobedo *et al*, 2007). On the other hand, there are relevant examples of VLE-specific playable courses, such as LAMS (Dalziel, 2003), which do not use a standard LD language for the specification of courses. Another example of a widespread VLE that does not implement any LDL is *Moodle*, which uses a non-standard package format instead to store and backup courses (Dougiamas and Taylor, 2003). LD languages and VLE-specific course formats are very diverse. The aim of having a generic LDL can hinder the deployment of a course, especially if it has to be rendered for a specific VLE platform. In the following, major LDL approaches and their issues are studied, beginning with IMS LD as the main source of research in the LD domain.

2.1 IMS Learning Design

The IMS LD specification (IMS, 2003; Olivier and Tattersall, 2005) is a formal framework of components described in a technical specification consisting of three levels. The information

model of level A describes elements required to express a static LD, that is, a predefined structure of activities and roles in the course. Level B elements make it possible to describe the dynamic adaptations that can occur in the learning flow according to events and conditions that might happen during execution of the course (Koper and Burgos, 2004; Burgos *et al*, 2006). Finally, level C defines the kinds of notifications that can be issued to enable the running of course activities. All these components are described in a manifest file and packaged along with content resources in a Unit of Learning (UoL) for each learning course. The core concept around which IMS LD level A components are set is the activity, which can be of two different types, namely learning activities and support activities, depending on their relationship with the learning objectives of the course. Activities can be aggregated as activity-structures, which can be one of two kinds: sequence or selection. Someone playing a role (e.g. a learner or a member of staff) runs the activity. Activities and roles are associated by means of role-parts, which are instantiations of links between a role and an activity running within the method of the LD. The method is structured using the metaphor of a theatrical play, which includes plays and acts. Other IMS LD level A components are learning objects and services, which can be directly or indirectly – that is, through an environment – linked to an activity. The dynamic part of IMS LD is provided by level B elements, which are used to adapt the execution flow of the learning process (Koper and Burgos, 2004). Any event that might influence the learning flow is modeled by a number of properties that govern structured conditions to allow branching from one activity to another. An LD may include adaptations of the learning flow of activities through the use of properties that represent values on the branching decisions to be made, conditions to express combinations of conditional clauses on which the learning flow can branch, and showing or hiding elements to show or hide activities on each branching flow.

The IMS LD specification (IMS, 2003; Olivier and Tattersall, 2005) is a formal framework of components described in a technical specification consisting of three levels. The information model of level A describes elements required to express a static LD, that is, a predefined structure of activities and roles in the course. Level B elements make it possible to describe the dynamic adaptations that can occur in the learning flow according to events and conditions that might happen during execution of the course (Koper and Burgos, 2004; Burgos *et al*, 2006). Finally, level C defines the kinds of notifications that can be issued to enable the running of course activities. All these components are described in a manifest file and packaged along with content resources in a *Unit of Learning* (UoL) for each learning course. The core concept around which IMS LD level A components are set is the *activity*, which can be of two different types, namely *learning activities* and *support activities*, depending on their relationship with the learning objectives of the course. Activities can be aggregated as *activity-structures*, which can be one of two kinds: *sequence* or *selection*. Someone playing a role (e.g. a *learner* or a member of *staff*) runs the activity. Activities and roles are associated by means of *role-parts*, which are instantiations of links between a role and an activity running within the *method* of the LD. The method is structured using the metaphor of a theatrical play, which includes *plays* and *acts*. Other IMS LD level A components are *learning objects* and *services*, which can be directly or indirectly (that is, through an *environment*) linked to an activity. The dynamic part of IMS LD is provided by level B elements, which are used to adapt the execution flow of the learning process (Koper and Burgos, 2004). An LD may include adaptations of the learning flow of activities through the use of *properties* that represent values on the branching decisions to be made, conditions to express combinations of conditional clauses on which the learning flow can branch, and *show* or *hide* elements to select the visible activities of each branching flow.

Most significant editing approaches prevent users from authoring some IMS LD elements. For instance, environments, role-parts, properties, and conditions have been found to present significant challenges to teachers' understanding (Derntl *et al*, 2010). Consequently instructional designers ought to learn and use technical concepts that are far from their pedagogical background (Griffiths and Blat, 2005). There are also findings on the low scalability of IMS LD course artefacts (Gutiérrez-Santos *et al*, 2008), because of the excessive number of properties and conditions required to express branching learning flows. Authoring approaches that aim at facilitating the editing of IMS LD UoLs also focus on these issues (Martínez-Ortiz *et al*, 2009).

2.2 Exchange LD Languages

There is a conceptual distinction between *exchange* LDLs and *authoring* LDLs (Martínez-Ortiz *et al*, 2008). On the one hand, the features of an exchange LDL are intentionally defined to run the LD course on a player, but they are not of special interest to the learning designer. IMS LD is the major example of an exchange LDL that can deliver ready-to-run UoLs. It is difficult for non-technical people to understand and use the IMS LD level B, however (Griffiths and Blat, 2005). On the other hand, authoring LDLs are closer to the learning designer's common needs and practices, but are not readily playable. Some instances of authoring LDLs are E2ML (Botturi, 2006) and CompendiumLD (Conole and Weller, 2008). Most authoring LDLs define visual languages for both instructional designers and educational software developers (Caeiro-Rodríguez *et al*, 2010). In this section, we mainly focus on LDL issues as a means of delivering ready-to-run LD courses which can guide learners and instructors while running the LD on a player. The source LD abstractions required to edit such courses are sometimes excessively dependent on the underlying formalisms of the platform on which the course must be run. A lot of approaches use the IMS LD model as both the source and the target abstractions. Others restrict the source model to a subset of the IMS LD specification that does not present considerable difficulties for the engineering (i.e. source-to-target) or re-engineering (i.e. target-to-source) of LD courses based on model mappings (Martínez-Ortiz *et al*, 2009).

A deeper analysis of existing approaches to LDLs can be found in Dodero *et al* (2007), Caeiro-Rodríguez *et al* (2010), and Dodero *et al* (2010). In the following, this analysis is revisited and updated from the perspective of exchange LDL issues.

The *e-LD* approach (Martínez-Ortiz *et al*, 2007, 2009) provides a visual workflow-like source model that must be transformed to the IMS LD course. The visual source language supports the level B conditions by adding flowchart-like branching representations, which are more attainable for the pedagogical designer.

CompendiumLD (Conole and Weller, 2008) is a visual tool used to describe LDs that are delivered as a set of HTML pages in a way that is easily interpretable by both instructor and students. It is useful to sketch a sequence of learning activities, but it does not deliver any executable artefact that complies with any runnable LD specification. Eventually, the instructional designers should use a different tool to deploy and run the course as an IMS LD UoL.

Template-based approaches have also been used to generate the basic structure of IMS LD-based courses (Boticario and Santos, 2008; Hernández-Leo *et al*, 2006). The template abstractions are parametrized simplifications of the design of some course activities, and have to be instantiated to deliver the final UoL. For example, the *Collage* tool (Hernández-Leo *et al*,

2006) provides a visual abstraction based on design patterns to edit an LD. These approaches are usually restricted to a predefined set of patterns that come with the tool.

Authors can also use rich sequencing graphs (Gutiérrez-Santos *et al*, 2008) to define learning flow strategies, which can then be converted into runnable UoLs. This approach can solve the limitations of the IMS LD level B specification in defining a learning flow. Nevertheless, sequencing graphs are not still a friendly language for instructional designers and pedagogues.

UML4LD (Laforcade *et al*, 2008) introduces meta-modeling and model-driven principles as part of the LD authoring. A UML profile is provided in a meta-CASE tool to reversely engineer a UML diagram from an IMS LD level A-compliant UoL. Understanding the delivered models, however, requires certain technical knowledge of UML, which is a barrier for instructional designers. More recent works (Laforcade, 2010) use domain-specific modeling tools to build visual instructional design languages, but the approach still requires software engineering skills.

The main drawback of existing LDLs has to do with using the same LD language as a means of both authoring and exchange. All these solutions share the idea that the authored LD model is to be run on an IMS LD-specific player, which is not necessarily the case. Besides, if the potential of a LDL is reduced to the IMS LD metamodel, the resulting LDL can be rather limited, because VLE-specific features cannot be envisaged. That is the rationale of the model-driven approach that is described next.

3. Model-driven Learning Design

The principles of MDD/MDE are applied in this work by the provision of a DSL and a tool that facilitates the transformation between an authoring LD model (that is known by domain experts) and the exchange LDL model that eventually has to be delivered. According to MDD terminology, the exchange LDL is the Platform-Specific Model (PSM), while the authoring LD provides the Platform-Independent Model (PIM). In the LD domain, common exchange LDLs (such as IMS LD or Moodle packages) can be considered as platform-specific languages for the implementation of learning environments. The result of the MDE process has to be the delivery of executable PSM-compliant course artefacts, which can be accomplished in two ways. The first way is to edit models with domain-specific software engineering modeling tools (Laforcade, 2010). The second is to define our own Learning Design Domain-Specific Language (LDDSL) in which LD-specific PIM models can be written. We have opted for this second option, which does not require authors to learn software engineering tools before providing them with an acceptable level of abstraction, flexibility, and control to edit an LD. By using simple annotations, our framework automatically generates PSM-compliant course artefacts from PIM models, allowing non-expert users to specify and generate new learning environments.

3.1 Learning Design DSL

The original requirements of the project included the development of an editing tool with a visual interface and a text-based interface, since the former might pose accessibility issues. In the following, the LDDSL is defined in terms of its abstract syntax or metamodel and its concrete syntax. For the concrete notation, we have defined a text-based and a visual LD syntax; however, here we will focus on the text-based notation.

3.1.1 Abstract syntax

The metamodel of LDDSL is depicted in Figure 1. We restrict our analysis to the core elements, which are used to describe learning activities, learning flows, and learning assessments, as summarized in Table 1.

The core concept is the *Activity* class. Activities are associated to a set of *Resources*, which represent both learning resources and services that are required to deploy them in a Web environment. Both simple and composite activities can be defined. The *Flow* class represents conditional expressions used to alter the learning flow, which can be branched to other activities, depending on the result of a conditional *Expression*. Since assessments are the usual way to adapt the learning flow, the

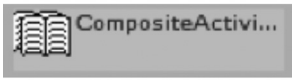
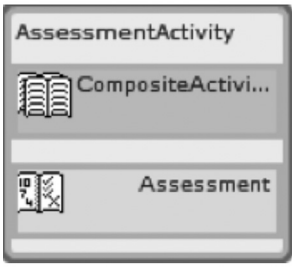
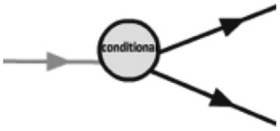
Name	Description	Icon
SimpleActivity	A simple activity associated to a resource and an assessment, to be included within an assessment activity or composite activity	
CompositeActivity	An activity consisting of other activities, to be included within an assessment activity or another composite activity	
AssessmentActivity	An entity consisting of an activity plus an assessment	
Assessment	An entity composed of a set of assessment parameters, which will be the output from an assessment activity or the input to a conditional flow	
SimpleFlow	A directed connection between two activity entities; the target entity will be run after the source entity	
BranchingFlow	A directed connection between two or more activities, with one source activity and multiple target activities; it may contain a conditional expression that determines which target activity will be run after the source activity and when this will occur	

Table 1: Summary of LDDSL elements that represent learning activities, learning flows, and learning assessments.

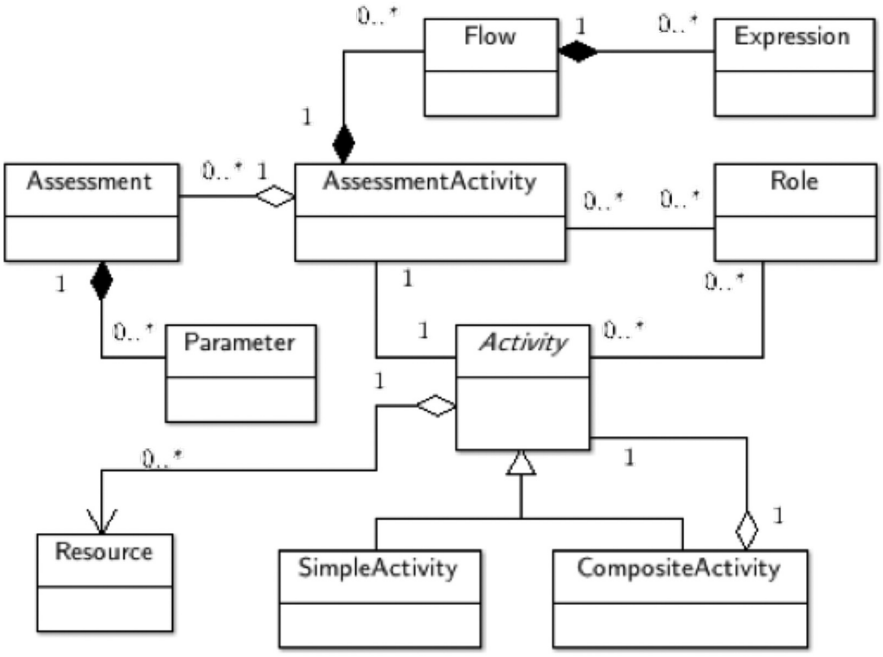


Figure 1: LDDSL abstract syntax to describe learning activities, learning flows, and assessments.

AssessmentActivity links an activity with an *Assessment* required for conditional adaptations of the flow. The *Assessment* class holds the *Parameters* required for the assessment. Finally, a set of *Roles* can be defined and mapped to activities and assessment activities, in order to provide advanced models of assessment (e.g. self-assessment and peer assessment).

3.1.2 Concrete syntax

The concrete syntax of the text-based version of LDDSL is defined using ANTLR (Parr and Quong, 1995). Listing 1 depicts a condensed version of the LDDSL grammar that covers assessments, activities, flows, and roles. For brevity, basic data types, expressions, and custom tokens (e.g. ID, STR, NATURAL) are not described here.

```
course : courseTag 'course' ID '{' roles courseBody '}' ;
courseBody : assessmentActivity * ;

// Assessment activities
assessmentActivity :
    ( activityTags ) ('begin') 'assessment_activity' ID
    '{' ( compositeActivity assessment | simpleActivity assessment | ) learningFlow '}' ;
compositeActivity :
    ( activityTags ) 'composite_activity' ID
    '{' activityRole 'title' STR activity * assessmentActivity * '}' ;
simpleActivity :
    ( activityTags ) 'simple_activity' ID
```

```
    '{' activityRole 'title' STR activityResource '}' ;
activity :
    simpleActivity | compositeActivity ;
activityResource :
    'resource' STR ;
assessment :
    assessmentProcType '{' assessmentRoles 'title' STR assessmentParameters '}' ;
assessmentProcType :
    'assessment' | 'self_assessment' | 'peer_assessment' ;
assessmentParameters :
    assessmentParameter * ;
assessmentParameter :
        'collection' ID collection
    |   'natural' ID rank
    |   'boolean' ID ( 'true ' | 'false' ) ;
    |   'string' ID (STR) ;

// Learning flow
learningFlow :
        branchingFlowTag 'flow' ':' condition
    |   branchingFlowTag 'flow' ':' ID | ;
branchingFlowTag :
    (   layoutTag | flowTag )+ | ;

// Roles
roles :
    |   'roles' '{' ( role )* '}' ;
assessmentRoles :
    assessorRole ( activityRole ) | ;
assessorRole :
    'evaluator_role' ID (',' ID)* ;
activityRole :
    'teacher_role' ID (',' ID)* | 'learner_role' ID (',' ID)* | ;
```

Listing 1: ANTLR grammar of the text-based LDDSL.

To implement exchange LDL-specific extensions to LDDSL, attribute-oriented programming techniques have been applied (Pawlak, 2005). Such techniques use tagged annotations (usually having an @ prefix) that can be included in the source model before certain source elements. For instance, an *@imsld* tag before an activity description can be used to describe the IMS LD-specific type of activity to be generated. An excerpt of the ANTLR grammar rules that define extension tags is shown in Listing 2.

```
courseTag :
        '@moodle ' | '@imsld ' | '@lams ' | '@lpcel ' | ;
activityTags :
        (   layoutTag | activityTag )+ | ;
activityTag :
        '@imsld ' ':' imsldActivityAttr
    |   '@moodle' ':' moodleActivityAttr
```

```
   |   '@lpcel' ':' lpcelActivityAttr ;
flowTag :
        '@imsld ' ':' imsldFlowAttr | '@lpcel' COLON lpcelFlowAttr ;
imsldActivityAttr :
        'learning_activity' | 'support_activity'
   |    imsldActivityStructure ;
imsldActivityStructureType :
        'activity_structure' imsldActivityStructureType ;
imsldActivityStructureType :
        'sequence' | 'selection' ;
imsldFlowAttr :
        'conditional ' condition | 'optional' ;
moodleActivityAttr :
        'lesson' | 'chat' | 'forum' | 'wiki' | 'survey' | 'choice' | 'resource' ;
lpcelActivityAttr :
        'while' booleanExpr | 'transactional' | 'join' | 'num_join' NATURAL ;
lpcelFlowAttr :
        'split' | 'split_min' NATURAL | 'split_max' NATURAL ;
```

Listing 2: Attribute-oriented tagged extensions to describe IMS LD, Moodle, and LPCEL platform-specific models.

Attribute-oriented programming techniques reduce the redundancy between the abstract and concrete syntax of the DSL (Krahn *et al*, 2007). Tagged annotations can be detached from the source specification, which can then be reused for designs having another LDL or VLE as the PSM. In the visual environment, such attribute-oriented extensions are deployed as forms that make it possible to give values to specific properties.

3.1.3 Transformations

From an LDDSL source specification, specific code of an exchange LDL can be generated. For that aim, first we defined a number of transformations between the LDDSL source model and the IMS LD model to enable the LD course to be run on regular IMS LD engines. Second, another set of transformations were defined to consider specific VLEs as the PSM. In this case, the target of such transformations can be Moodle or other VLEs that include their own specific types of activities, role mappings, and learning flows, among other details. PIM-to-PSM transformations are driven by the tagged attributes that are specific to the target LDL or VLE. The details of IMS LD UoLs and Moodle exchange packages are defined by the permitted attribute values of *@imsld* and *@moodle* tags, respectively. The LDL PSM of the editing tool can be extended to include other exchange LDLs. For instance, we have easily extended the concrete notation (i.e. the last rules of Listing 2) in order to configure a tool variant prepared for the LPCEL language (Torres *et al*, 2006). LPCEL is an LDL that uses improved synchronization mechanisms for learning processes (e.g. concurrent split-join branching flow) not considered in other LDLs.

3.2 LDDSL Editing Example

The LDDSL editing environment is available through a stand-alone tool implemented using ActionScript. It can be used to design and export playable LD courses. The editing interface provides common LDDSL abstractions (such as activities, roles, learning flows, and assessments)

to edit a source LD specification. For that aim, it provides a text-based and a visual interface. The LDDSL source of a course can be visually edited in the tool or directly edited in a text view. The current version of the editing environment can generate IMS LD UoLs, LPCEL learning processes, and Moodle packages. For the tool serialization notation we took advantage of the text-based concrete syntax, and extended the language with additional *@layout* tags that hold layout information (e.g. the position of forms and connectors in the canvas). That provides a rudimentary form of reverse engineering from LDDSL specifications, which can be modified with external text editors and imported again in the tool.

Listing 3 depicts the LDDSL representation of a course, designed to cover the main elements of a regular LD situation. The first sentence is the specification of roles, which defines a new *peer* role that is defined as an extension of learner and evaluator predefined roles and is intended for the final assessment activity. After the definition of roles, four *assessment_activities* are defined. They are surrogates of simple activities (*ae1*, *ae2* and *ae4*) and one composite activity (*ae3*). Activity *ae1* is marked with the *begin* flag and is connected by a conditional flow with *ae2* and *ae3*. Assessment activities are wrappers of regular *activities* that encompass all assessment-related information. This is a common design technique for extending a class interface without changing it. In this way, each activity is provided with an assessment specification which describes the type of assessment (i.e. self-assessment, peer assessment, or regular assessment), the assessment parameters, and the roles that must develop it. The *ae1* activity renders as its output a natural parameter that can be read later to branch the learning flow. For example, after *ae1*, the flow will branch to *ae2* activity, looking for some learning support if the *grade* value is below 5. Assessments may include a *resource* that defines the application or service to be used for assessment. In *ae2* and *ae4*, two assessment instruments hosted in the http://evalcomix.uca.es/ service are used for this purpose.

```
course sample_course {
   roles { role peer extends learner, evaluator }
   begin assessment_activity ae1 {
      simple_activity as11{
         learner_role learner
         teacher_role teacher
         title "Beginning"
      }
      assessment {
         evaluator_role evaluator
         teacher_role teacher
         natural grade [0,10]
      }
      flow: if: grade < 5
         then: ae2
         else: flow: ae3
      }
   assessment_activity ae2 {
      @imsld:support_activity
      simple_activity as21 {
         learner_role learner
         teacher_role teacher
```

```
            title "Support"
        }
        assessment {
            evaluator_role evaluator
            resource "http://evalcomix.uca.es/instrument/view?id=85cf15"
            natural grade [0,10]
            boolean finished
            collection scale ["good", "regular", "poor"]
        }
        flow: ae3
    }
    assessment_activity ae3 {
        @imsld:learning_activity
        @imsld:activity_structure sequence
        composite_activity ac1 {
            learner_role learner
            teacher_role teacher
            title "Content"
            simple_activity as31 {
                title "Activity 1"
                resource "http://docs.google.com/document/d/1CA7e47dSi"
            }
            simple_activity as32 {
                title "Activity 2"
                resource "http://docs.google.com/document/d/jD6kd273DS"
            }
        }
        assessment {
            evaluator_role evaluator
            teacher_role teacher
        }
        flow: ae4
    }
    assessment_activity ae4 {
        simple_activity as43 {
            learner_role learner
            teacher_role teacher
            title "Final Activity"
        }
        assessment {
            evaluator_role peer
            teacher_role teacher
            resource "http://evalcomix.uca.es/instrument/view?id=74fd04"
            collection final_grade ["fine","medium","poor"]
        }
    }
}
```

Listing 3: Text-based representation of a learning course using LDDSL.

Listing 3 also shows an example of PSM annotations including tagged values to represent two IMS LD activities: one is an IMS LD *support-activity* configured with a teacher-based assessment service, and the other is a *sequence*-type *activity-structure* for an IMS LD *learning-activity* that provides access to a pair of Google Docs resources.

4. Evaluation

For the evaluation of our approach we will follow first an analytical method and then a case study evaluation, as proposed by Hevner *et al* (2006). The former will evaluate the quality of LDDSL as a design artefact. It serves to examine the structure of an LDDSL specification for static qualities such as computational complexity. We assume that IMS LD is expressive enough to fulfil many LD situations (van Es and Koper, 2006) and it can be used to represent most learning scenarios. Since LD languages are Turing-complete, it is pointless to compare them only from the viewpoint of expressiveness (van der Aalst and van Hee, 2002). Kiepuszewski *et al* (2003) evaluate workflow languages based on suitability, which responds to whether the language provides direct support for patterns which appear frequently in domain-specific designs. With that aim, a case study evaluation has also been designed, consisting of a set of common LD situations. The aim is to find out the main source of complexity in the final courses that are delivered in a target language. We have selected the IMS LD specification as the target language in order to test if IMS LD is limited in expressing varied learning flows (Gutiérrez-Santos *et al*, 2008).

4.1 Analysis of Complexity

The complexity of an LD specification can be expressed as a function of the evaluable attributes in the graph of the learning activities. Graph nodes represent activities; graph edges represent the flow between activities. The complexity of a course can be described as an expression of the following variables:

a_1 = Number of source activities, that is, activities in which a flow starts
a_2 = Number of sink activities, that is, activities in which a flow ends
a_3 = Number of clauses in conditional branching flows

The type and number of IMS LD elements required to represent each element of LDDSL is computed as follows. Table 2 shows the types of IMS LD elements required to implement each adaptation feature of the conceptual model, the number of elements, and the formulas used to compute the overall number of elements. Simple, composite, and assessment activities require a finite number of IMS LD level A activity elements. Conditional branching flows require a number of IMS LD level B conditional elements that is a function of

$$\sum_{i=1}^{3}(c_i a_i)$$

which has O(n) complexity, with $n = max(a_i)$ and c_i constants. If conditional flows are chained, the complexity increases by a multiplicative factor of n due to the additional nested branching that is required. So the complexity of IMS LD specifications is O(n^k), with k being the maximum nesting level of conditional flows. In contrast, an equivalent LDDSL specification will only require a constant number of additional elements for each conditional branch, so the complexity remains linear.

LDDSL element	IMS LD element	Count	Total
SimpleActivity	learning-activity	1	3
	learning-activity-ref	2	
CompositeActivity	activity-structure	1	2
	activity-structure-ref	1	
AssessmentActivity	support-activity	1	7
	support-activity-ref	2	
	locpers-property	2	
	datatype	2	
BranchingFlow	activity-structure	2	$9a_1 + 4a_2 + a_3 + 5$
	activity-structure-ref	2	
	if	a_2	
	then	a_2	
	else	a_2	
	and, or	a_3+1	
	greater-than, less-than	a_3	
	is, is-not	a_3	
	show, hide	a_2	
	property-ref	$2(a_1 + a_2)$	
	property-value	$2(a_1 + a_2)$	
	complete	a_2	

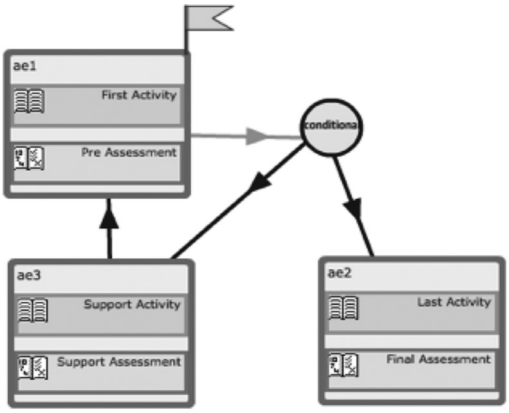
Table 2: Type and number of IMS LD elements required to represent the elements of the LDDSL metamodel.

This evaluation could be complemented with an analysis of the performance of the UoL in an IMS LD player. Nevertheless, the purpose of our research is not to compare the performance of different LD specifications, but to evidence the lesser complexity of editing LD adaptations with LDDSL.

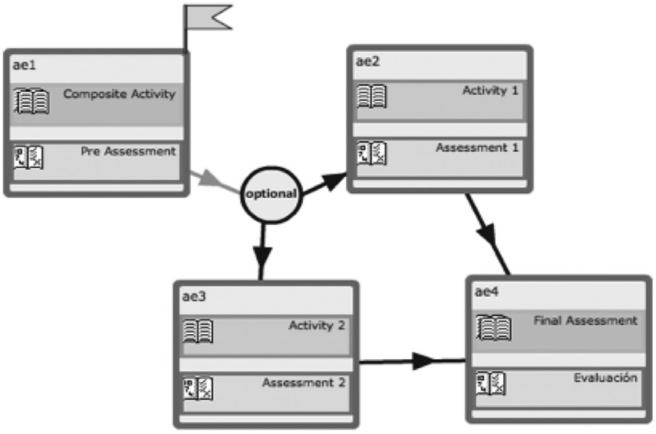
4.2 Learning Design Cases

For the second evaluation, three common LD cases have been designed. These situations are characterized by the kinds of activity elements included (simple and composite), the types of assessments, and the kinds of flow primitives included. The flow of activities can be sequential or conditionally branched. The learning flow can go forward or backward, so that loops can be defined for remediation activities. Either the teacher or the learner role can carry out assessments. The cases have been designed to cover all the learning flow possibilities of the LDL.

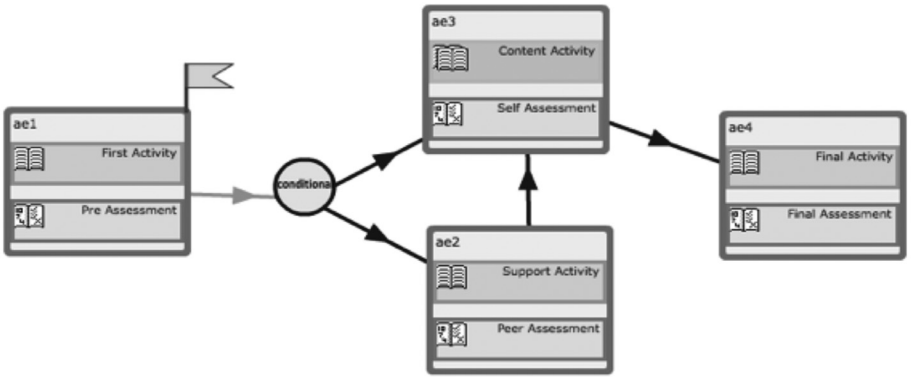
- **Case 1** (see Figure 2a and Listing 4) describes a set of activities and a conditional branching that can flow forward and backward until the *Last* activity is reached. If remediation is needed, the flow is directed to the *Support* activity, which has a backward connection to the *First* activity. Assessments that determine the need for remediation are carried out by the teacher role.
- **Case 2** (see Figure 2b and Listing 5) describes a set of activities and a non-conditional branching flow so that the learner can select between two ways of action (i.e. *Activity 1* or *Activity 2*). Assessments are carried out by the learner roles on a peer-assessment basis.



(a) Simple activities connected by a backwards conditional flow.



(b) Composite activities connected by an optional branching flow.



(c) Simple and composite activities connected by a conditional branching flow.

Figure 2: View of three LDDSL courses that consist of assessment activities and branching flows.

- **Case 3** (see Figure 2c and Listing 6) describes a set of activities and a conditional branching that flows forward. After the beginning activity, the flow can branch forward to the *Content* activity or to the *Support* activity. Assessments that determine the need for remediation are carried out by each learner role as self-assessment.

```
course case_1 {
    @layout:(50,63)
    begin assessment_activity ae1 {
        simple_activity as11 {
        learner_role learner
        teacher_role teacher
        title "First Activity"
    } assessment {
        evaluator_role evaluator
        teacher_role teacher
        boolean ok false
    }
    @layout:(273,104)
    flow: if: ok = true
        then: ae2
        else: flow: ae3
    }
    @layout:(285,229)
    assessment_activity ae2 {
    simple_activity as21 {
        learner_role learner
        teacher_role teacher
        title "Last Activity"
    }   assessment {
       evaluator_role evaluator
       teacher_role teacher
    }
  }
  @layout:(53,231)
  assessment_activity ae3 {
    simple_activity as31 {
       learner_role learner
       teacher_role teacher
       title "Support Activity"
    } assessment {
       evaluator_role evaluator
       teacher_role teacher
    } flow: ae1
 }
}
```

Listing 4: Text LDDSL specification of the course of Figure 2a.

```
course case_2 {
    begin assessment_activity ae1 {
        composite_activity ac11 {
            title "Composite Activity"
            simple_activity as111 {
                title "Simple Activity"
            }
        } assessment { ... }
        @imsld: optional
        flow: ae2, ae3
    }
    assessment_activity ae2 {
        simple_activity as21 {
            title "Activity 2"
        } assessment { ... }
        flow: ae4
    }
    assessment_activity ae3 {
        simple_activity as31 {
            title "Activity 3"
        } assessment { ... }
        flow: ae4
    }
    assessment_activity ae4 {
        composite_activity ac41 {
        title "Final Assessment"
            simple_activity as411 {
                title "Simple Activity"
            }
        } assessment { ... }
    }
}
```

Listing 5: Text LDDSL specification of the course of Figure 2b.

```
course case_3 {
    roles {
        role peer extends learner, evaluator
    }
    begin assessment_activity ae1 {
        simple_activity as11 {
            title "First Activity"
        } assessment {
            evaluator_role teacher
            boolean ok false
        } flow: if: ok = false
            then: ae2
            else: flow: ae3
    }
    assessment_activity ae2 {
```

```
        simple_activity as21 {
            title "Support Activity"
        } peer_assessment {
            evaluator_role peer
        } flow: ae3
    }
    assessment_activity ae3 {
        composite_activity ac31 {
            title "Content Activity"
            simple_activity as311 { ... }
            simple_activity as312 { ... }
        } self_assessment {
            evaluator_role learner
        } flow: ae4
    }
    assessment_activity ae4 {
        simple_activity as41 {
            title "Final Activity"
        } assessment {
            evaluator_role teacher
        }
    }
}
```

Listing 6: Text LDDSL specification of the course of Figure 2c.

The measurements of complexity of the IMS LD-based UoL generated in each evaluation case study are shown in Table 3. Some conclusions can be drawn from these results:

- Level A structural elements introduced by simple and composite activities are not the source of much complexity, as observed in cases 1 and 3. An exception is the need to include as many references to learning activities as the number of simple activities defined within each composite activity. The latter is observed in case 2, which has two composite activities including other simple activities.
- The main source of complexity for assessment activities is the need to include many activity references and properties, as obliged by the level B primitives that are generated.
- Non-conditional branching flows do not add much complexity, as shown in case 2. This is because they can be represented with simple references and attributes belonging to level A.
- Conditional branching flows are the major source of complexity, as shown in cases 1 and 3. The required number of level B elements is greater than in any other situation.

The main source of complexity added to the delivered IMS LD UoL artefact is due to the adaptation details required by level B of the IMS LD specification. Such details are automatically generated from an LDDSL specification. For instance, a single entity of LDDSL, such as a conditional flow, requires a number of elements that is O(1), versus the O(n) required if the level B properties and conditions of IMS LD are edited directly.

LDDSL element	IMS LD element	Case 1	Case 2	Case 3
SimpleActivity	learning-activity	3	10	9
	learning-activity-ref	9	14	14
		12	***24***	***23***
CompositeActivity	activity-structure	12	4	3
	activity-structure-ref	2	4	4
		4	***8***	***7***
AssessmentActivity	support-activity	3	4	4
	support-activity-ref	6	6	7
	locpers-property	4	4	7
	datatype	4	4	7
		17	***18***	***25***
BranchingFlow	activity-structure	2	2	2
	activity-structure-ref	2	2	2
	if	3	2	3
	then	3	2	3
	else	1	0	0
	and, or	2	0	0
	greater-than, less-than	2	0	0
	is, is-not	1	0	0
	show, hide	3	2	3
	property-ref	8	8	8
	property-value	8	8	8
	complete	3	2	3
		38	***28***	***32***

Table 3: Complexity of the IMS LD elements that represent the evaluation cases.

5. Conclusion and Future Work

This work describes a model-driven approach to edit LDs and presents LDDSL, an LD-specific language that relieves the instructional designer of needing to know software development techniques, course exchange formats, and technical details of LD specifications. With the model-driven approach, the complexity of using an LD language is reduced so that the reusability of the learning models for different learning platforms can be improved. Both standard and specific LD models, such as IMS LD and Moodle, are generated from an LDDSL model simply by annotating it with LDL-specific tags that complement it. A different annotation on the same model would be enough to generate the same learning environment on another platform.

The LDDSL provides learning designers with a set of editing elements that can be more suitable than exchange LDL elements. LDDSL is not more expressive than IMS LD, but it contains a set of frequently recurring structures (such as assessments and learning flows) that, for a learning

designer, can become more convenient than IMS LD level B elements. Although such structures can be directly expressed with IMS LD technical details, the LDDSL tool implementation generates an equivalent IMS LD specification representing such patterns, which can be more easily understood by the learning designer. The elements of LDDSL have been analysed and compared with the corresponding ones of IMS LD, with the result that: (1) IMS LD is expressive enough to represent common LD situations described in LDDSL; and (2) LDDSL has less complexity for expressing learning flows and learning assessments. An authoring tool is provided with a visual and a text-based editing interface to design an LDDSL specification from which an exchange LDL-based course can be generated. An attribute-oriented tagging approach has been followed to extend LDDSL courses with details of a specific exchange LDL or learning platform.

The LDDSL approach has been previously analysed with respect to the generation of IMS LD-compliant units of learning and Moodle 1.x courses (Dodero *et al*, 2010). Tagging has also been used to implement a rudimentary form of reverse engineering from an LDDSL text-based specification. Nevertheless, reverse engineering from fully fledged IMS LD UoLs or Moodle packages is still an issue that needs to be explored. In this vein, the Moodle 2.x platform is expected to provide a more widespread user environment test bed. Particularly interesting is the improved model of conditional activities and service integration of the latest Moodle versions, which were not available in former versions of the platform. In addition, the way in which users can engage with the LDDSL language through the authoring tool must be further investigated. Users' effectiveness with the tool interface may confuse the complexity of understanding the DSL model with the usability of the tool, so a careful co-design experience must be carried out.

Acknowledgements

This work has been sponsored by a grant from the ASCETA project (ref. P09-TIC-5230) of the Andalusian Government Research and Development Programme and a research stay grant (ref. EST2011-075) from the University of Cádiz.

References

BOTICARIO, J.G. and SANTOS, O.C. (2008): A standards-based modelling approach for dynamic generation of adaptive learning scenarios. *Journal of Universal Computer Science* 14(17): 2859–2876.

BOTTURI, L. (2006): E2ML: A visual language for the design of instruction. *Educational Technology Research and Development* 54(3): 265–293.

BURGOS, D. and GRIFFITHS, D. (2005): The UNFOLD project. Understanding and using learning design. Open University of the Netherlands.

BURGOS, D., TATTERSALL, C. and KOPER, R. (2006): Representing adaptive eLearning strategies in IMS learning design. In *Proc. International Workshop on Learning Networks for Lifelong Competence Development*, KOPER, R. and STEFANOV, K. (eds), Sofia, Bulgaria, 54–60.

CAEIRO RODRÍGUEZ, M., DERNTL, M. and BOTTURI, L. (2010): Visual instructional design languages. *Journal of Visual Languages & Computing* 21(6): 311–312.

CONOLE, G. and WELLER, M. (2008): Using learning design as a framework for Supporting the Design and Reuse of OER. *Journal of Interactive Media in Education*, 2008(may), http://www-jime.open.ac.uk/jime/article/view/2008-5. Accessed 19-Apr-2012.

DALZIEL, J. (2003): Implementing learning design: The learning activity management system (LAMS). In *Interact, Integrate, Impact: Proc. 20th Annual Conference of ASCILITE*, CRISP, G., THIELE, D., SCHOLTEN, I., BARKER, S. and BARON, J. (eds), Adelaide, Australia, 593–596.

DERNTL, M., NEUMANN, S., GRIFFITHS, D. and OBERHUEMER, P. (2010): Investigating teachers' understanding of IMS learning design: Yes they can! In *Sustaining TEL: From Innovation to Learning and Practice*, WOLPERS, M., KIRSCHNER, P., SCHEFFEL, M., LINDSTAEDT, S. and DIMITROVA, V. (eds), LNCS 6383: 62–77, Springer.

DODERO, J.M., MARTÍNEZ DEL VAL, Á. and TORRES, J. (2010): An extensible approach to visually editing adaptive learning activities and designs based on services. *Journal of Visual Languages & Computing* 21(6): 332–346.

DODERO, J.M., TATTERSALL, C., BURGOS, D. and KOPER, R. (2008): Transformational techniques for model-driven authoring of learning designs. *Proc. 6th International Conference on Web-based Learning*, Edinburgh, UK, LNCS 4823: 230–241, Springer.

DOUGIAMAS, M. and TAYLOR, P.C. (2003): Moodle: Using learning communities to create an open source course management system. *Proc. World Conference on Educational Multimedia, Hypermedia and Telecommunications*, Cheasepeake, VA, 171–178, AACE.

ESCOBEDO DEL CID, 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* 2007(sep). http://www-jime.open.ac.uk/jime/article/view/2007-7. Accessed 19-Apr-2012.

FRANCE, R. and RUMPE, B. (2007): Does model driven engineering tame complexity? *Software & Systems Modeling* 6(1): 1–2.

GRIFFITHS, D. and BLAT, J. (2005): The role of teachers in editing and authoring units of learning using IMS learning design. *Advanced Learning Technology* 2(4): 243–251.

GUTIÉRREZ-SANTOS, S., PARDO, A. and DELGADO-KLOOS, C. (2008): Authoring courses with rich adaptive sequencing for IMS learning design. *Journal of Universal Computer Science* 14(17): 2819–2839.

HERNÁNDEZ-LEO, D., VILLASCLARAS-FERNÁNDEZ, E.D., JORRÍN-ABELLÁN, I.M., ASENSIO-PÉREZ, J.I., RUIZ-REQUIES, I. and RUBIA-AVI, B. (2006): COLLAGE, a collaborative learning design editor based on patterns. *Educational Technology & Society* 9(1): 58–71.

HEVNER, A.R., MARCH, S.T., PARK, J. and RAM, S. (2004): Design science in information systems research. *MIS Quarterly* 28(1): 75–105.

IMS (2003): *IMS Learning Design*. http://www.imsglobal.org/learningdesign. Accessed 19-Apr-2012.

KARAMPIPERIS, P. and SAMPSON, D. (2007): Towards a common graphical language for learning flows: Transforming BPEL to IMS learning design level A representations. *Proc. 7th International Conference on Advanced Learning Technologies*, Niigata, Japan, 798-800, IEEE Computer Society Press.

KIEPUSZEWSKI, B., TER HOFSTEDE, A.H.M. and VAN DER AALST, W.M.P. (2003): Fundamentals of control flow in workflows. *Acta Informatica* 39(3): 143–209.

KOPER, R. and BURGOS, D. (2004): Developing advanced units of learning using IMS learning design level B. *Advanced Technology for Learning* 2(4): 252–259.

KRAHN, H., RUMPE, B. and VÖLKEL, S. (2007): Integrated definition of abstract and concrete syntax for textual languages. *Proc. 10th International Conference on Model Driven Engineering Languages and Systems*, Nashville, TN, 286–300, Springer.

LAFORCADE, P., ZENDAGUI, B. and BARRÉ, V. (2008): A domain-specific-modeling approach to support scenarios-based instructional design. *Proc. European Conference on Technology-Enhanced Learning*, Maastricht, The Netherlands, 185–196, Springer.

LAFORCADE, P. (2010): A domain-specific modeling approach for supporting the specification of visual instructional design languages and the building of dedicated editors. *Journal of Visual Languages & Computing* 21(6): 347–358.

MARTENS, H. and VOGTEN, H. (2005): A reference implementation of a learning design engine. In *Learning design. A Handbook on modelling and delivering networked education and training*. KOPER, R. and TATTERSALL, C. (eds), 91–108, Berlin, Springer.

MARTÍNEZ-ORTIZ, I., MORENO-GER, P., SIERRA-RODRÍGUEZ, J.L. and FERNÁNDEZ-MANJÓN, B. (2007): Supporting the authoring and operationalization of educational modelling languages. *Journal of Universal Computer Science* 13(7): 938–947.

MARTÍNEZ-ORTIZ, I., SIERRA, J.L. and FERNÁNDEZ-MANJÓN, B. (2008): A flow-oriented visual language for learning designs. *Proc. 7th International Conference on Web-based Learning*, Jinhua, China, 486–496, Springer.

MARTÍNEZ-ORTIZ, I., SIERRA, J.L. and FERNÁNDEZ-MANJÓN, B. (2009): Authoring and reengineering of IMS learning design units of learning. *IEEE Transactions on Learning Technologies* 2(3): 189–202.

OLIVIER, B. and TATTERSALL, C. (2005): The learning design specification. In *Learning design. A Handbook on modelling and delivering networked education and training*, KOPER, R. and TATTERSALL, C. (eds), 21–40, Berlin, Springer.

PARR, T.J. and QUONG, R.W. (1995): ANTLR: A predicated-LL(k) parser generator. *Software: Practice and Experience* 25(7): 789–810.

PAWLAK, R. (2005): Spoon: Annotation-driven program transformation – The AOP Case. *Proc. 1st International Middleware Workshop on Aspect-Oriented Middleware Development*, Grenoble, France, 1–6, ACM Press.

SITTHISAK, O. and GILBERT, L. (2010): Extension of IMS LD for improved pedagogical expressiveness in assessment. *Proc. International Computer Assisted Assessment Conference: Research into E-Assessment*, Southampton, UK.

SMITH, P. and RAGAN, T.J. (2005): *Instructional Design*, 3rd ed., John Wiley & Sons.

STRIJBOS, J-W. (2011): Assessment of (computer-supported) collaborative learning. *IEEE Transactions on Learning Technologies* 4(1): 59–73.

TORRES, J., DODERO, J.M., AEDO, I. and DÍAZ, P. (2006): Designing the execution of learning activities in complex learning processes using LPCEL. *Proc. 6th International Conference on Advanced Learning Technologies*, Kerkrade, The Netherlands, 415–419, IEEE Computer Society Press.

VAN ES, R. and KOPER, R. (2006): Testing the pedagogical expressiveness of IMS LD. *Journal of Educational Technology & Society* 9(1): 229–249.

VAN DER AALST, W.M.P. and VAN HEE, K.M. (2002): Workflow management: Models, methods, and systems. MIT Press.

Biographical Notes

Juan Manuel Dodero

Juan Manuel Dodero *is an associate professor of the University of Cádiz, Spain. He has a CS degree from the Polytechnic University of Madrid and a CS PhD from the Carlos III University of Madrid. Formerly he has been a lecturer of the Carlos III University of Madrid and has worked as a research and development engineer for Intelligent Software Components S.A. His main research interests are web science and engineering and technology-enhanced learning, fields in which he has authored numerous contributions for scientific journals and research conferences. He is an ACM member, founder and committee member of the SIGCSE Spanish Chapter, member of the software process improvement research group and director of the Libre Software and Open Knowledge Office at the University of Cádiz.*

Iván Ruiz-Rube

Iván Ruiz-Rube *is an assistant lecturer at the informatics department of the University of Cádiz. He has a CS degree and a MSc in software engineering and technology from the University of Seville. His fields of research are software process improvement and model-driven engineering and the semantic web, in which he has published several papers in international conferences and journals. Formerly, he has worked as a software engineer for Spanish consulting companies such as EVERIS and SADIEL.*

Manuel Palomo-Duarte

Manuel Palomo-Duarte *is a lecturer of the department of informatics of the University of Cádiz, in which he teaches courses on videogame development and operating systems using open source software. He obtained a CS degree from the University of Seville and a CS PhD from the University of Cádiz. He is currently coordinator of the CS degrees at the school of engineering, member of the software process improvement research group, and has been the former director of the Libre Software and Open Knowledge Office of the University of Cádiz.*

Jordi Cabot

Jordi Cabot *is currently leading the AtlanMod team, an INRIA research group at École des Mines de Nantes (France). Previously, he has been a post-doctoral fellow at the University of Toronto, a senior lecturer at the UOC (Open University of Catalonia) and a visiting scholar at the Politecnico di Milano. He received the BSc and PhD degrees in computer science from the Technical University of Catalonia. His research interests include conceptual modeling, model-driven and web engineering, formal verification and social aspects of software engineering. He has written more than 50 publications in international journals and conferences in the area. Apart from his scientific publications, he writes and blogs about all these topics in his Modeling Languages portal (http://modeling-languages.com). He is a member of the IEEE and the ACM.*