Journal of Web Engineering, Vol. 11, No.1 (2012) 023-050
© Rinton Press

DOMAIN SPECIFIC LANGUAGE FOR THE GENERATION OF LEARNING MANAGEMENT SYSTEMS MODULES

CARLOS ENRIQUE MONTENEGRO-MARÍN

Distrital University, Bogota
cemontenegrom@udistrital.edu.co

JUAN MANUEL CUEVA-LOVELLE OSCAR SANJUÁN-MARTÍNEZ VICENTE GARCÍA-DÍAZ

University of Oviedo. Oviedo-Spain.
cueva@uniovi.es *osanjuan@uniovi.es* *garciavicente@uniovi.es*

Received January 20, 2011
Revised November 23, 2011

Nowadays there are many research projects conducted in the areas of Learning Management Systems (LMS) and Model-Driven Engineering (MDE). These research projects have shown that there are LMS platforms with different architectures and inoperative to each other. The most significant contribution of MDE has been the creation of a common meta-metamodel. This meta-metamodel allows transformations between different models. This research work presents a LMS metamodel. The metamodel created is based on the study of five LMS platforms. The LMS metamodel is a global model that makes a bridge for the transformation of modules between the model and different LMS platforms, and it also presents the development of a Domain Specific Language (DSL) tool to validate the metamodel, the transformation process of the model with our DSL Tool to modules deployed in Moodle, Claroline and Atutor, and finally testing and validation of creating modules with LMS platforms VS creating modules with our DSL Tool.

Key words: Platform independent model, Model transformation, Model-Driven Architecture (MDA), Metamodel, Domain Specific Language (DSL).
Communicated by: B. White & M. Norrie

1 Introduction

At present there is an increasing interest in creating knowledge representation models in software engineering [1]. This has led to research works like, for instance, Model-Driven Architecture (MDA) [2], and the creation of modeling languages such as the Unified Modeling Language (UML) [3], The UML metamodel is viewed as defining the language for creating models, and the MOF as defining the language for creating metamodels. MOF allows serializing metamodels in XML Metadata Interchange (XMI) [4, 5]. There are other ways to generate XMI, e.g. generating metamodels with Ecore [6]. This paper presents a metamodel and Domain Specific Language (DSL) tool for Learning Management Systems (LMS), to create course modules on some LMS, independently of their platform. This is a view point to solution problem of interoperability between LMS platforms.

To create a LMS metamodel, the first step was to select and graphically represent 5 LMS platforms. The LMS platforms selected were: DotLRN, SAKAI, ATutor, Claroline and Moodle, because they have a good degree of acceptance and a good evaluation in most communities and working groups devoted to topics related to course content management and strategies in learning [7, 8]. The graphic representation was made with CMapTools [9] because it offers ease for collaborative Web work, has a variety of formats for exporting the map and has a large number of projects that have worked with it, such as [10-14]. For the knowledge acquisition a technique called "knowledge elicitation" was used [15], this technique uses natural language in the acquisition of requirements. The report of this research work is in [16]. The final result is a list of the similar modules between the LMS platforms and this is explained in Section 2. In Section 3, we review modules architecture in LMSs (Moodle, Claroline and ATutor). The next step was generating a LMS Metamodel based on previous work; this is shown in Section 4. The next step was constructing a DSL tool based on the modules of communications of the LMS Metamodel, this is shown in Section 5. In order to validate and test the Metamodel and the DSL tool, the transformation from a model in the DSL tool to modules in Moodle, Claroline and ATutor is necessary. This is explained in the Sections 6. Section 7 evaluates and validates the approach. Section 8 presents related works. And Section 9 presents conclusions and future research works.

2 Selecting tools and methodology for construction the LMS metamodel.

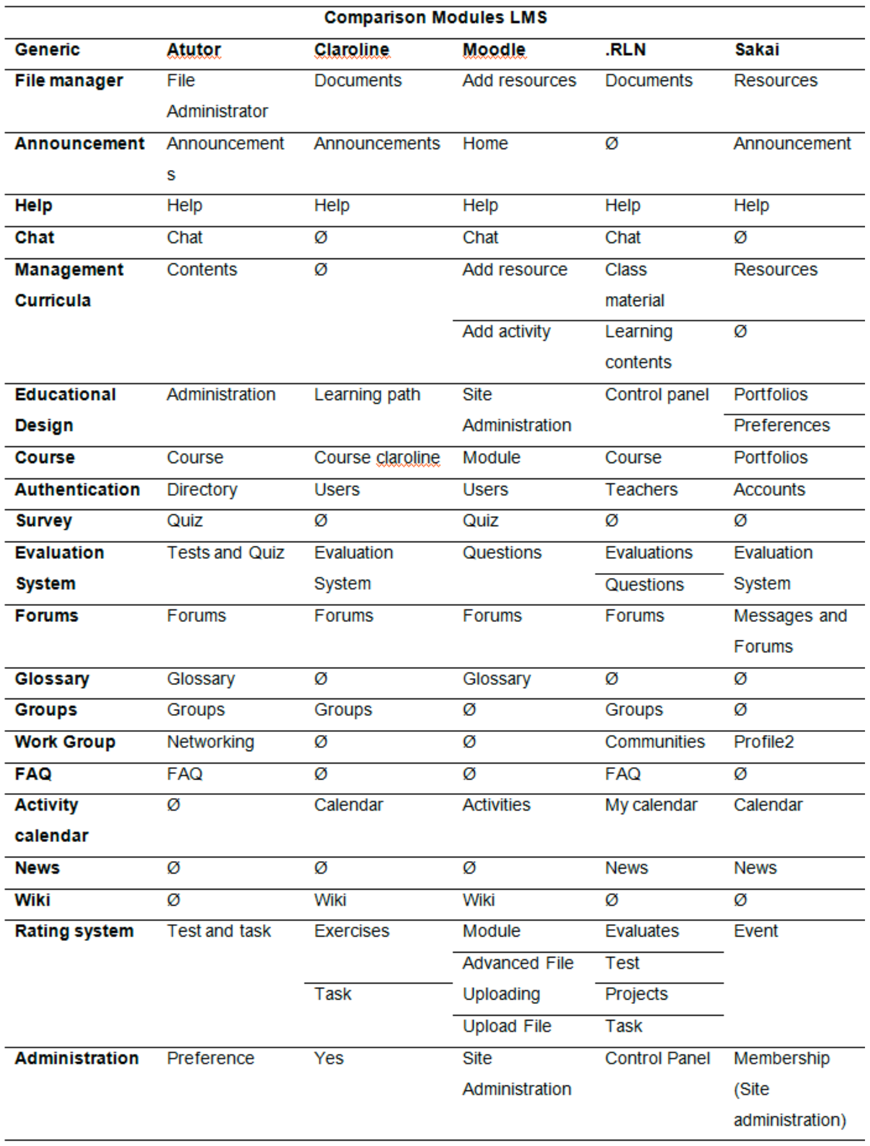
2.1 Selecting LMS Platforms

At present there are many Learning Management Systems (LMS), these LMS usually take some form of standardization in their construction. We will conduct a detailed study to analyze from the standpoint of computing, the content of some LMS to define a comparison table between them, the main criterion for selecting the platforms is the adoption of GPL licensing and open source, this condition with the aim of facilitating the analysis of its behavior and its modules. For the selection of LMS, we chose Moodle, Sakai, and DotLRN because works such as [17] show that Moodle is the most used open source LMS in Spanish Universities with over 45%. Furthermore, Sakai has 5%, and DotLRN has 4%. Claroline and ATutor were selected because they are also used worldwide. For instance, according to [18] each of these platforms are used by 3% of the Italian Universities.

2.2 Using of the previous platforms for the build metamodel

According to [19], a product family provides a context in which many problems common to family members can be solved collectively, in this sense, we separate the product family in three parts 1) common part, 2) specific part for the particular client needs and 3) repetitive schematic part, this last part will be refactored to be create with models [20]. To capture LMS characteristics we used Feature Models, that provides a formal way to capture commonalities and variabilities in features identified during domain analysis [21].

For the capture of commonalities characteristics, with the platforms selection of the previous section, we built a knowledge map each platform and adjusting the proposal of [22], that used a ontology as based for feature modeling, for a knowledge map. With this modeling, we did a comparison procedure between the modules of the LMSs, and finally we generated a comparison table of modules, which shows a first approximation of modules compatible between LMSs and a generic name for them to define the LMS metamodel [16]. The final result is shown in Table 1.



Comparison Modules LMS					
Generic	**Atutor**	**Claroline**	**Moodle**	**.RLN**	**Sakai**
File manager	File Administrator	Documents	Add resources	Documents	Resources
Announcement	Announcements	Announcements	Home	Ø	Announcement
Help	Help	Help	Help	Help	Help
Chat	Chat	Ø	Chat	Chat	Ø
Management Curricula	Contents	Ø	Add resource	Class material	Resources
			Add activity	Learning contents	Ø
Educational Design	Administration	Learning path	Site Administration	Control panel	Portfolios
					Preferences
Course	Course	Course claroline	Module	Course	Portfolios
Authentication	Directory	Users	Users	Teachers	Accounts
Survey	Quiz	Ø	Quiz	Ø	Ø
Evaluation System	Tests and Quiz	Evaluation System	Questions	Evaluations	Evaluation System
				Questions	
Forums	Forums	Forums	Forums	Forums	Messages and Forums
Glossary	Glossary	Ø	Glossary	Ø	Ø
Groups	Groups	Groups	Ø	Groups	Ø
Work Group	Networking	Ø	Ø	Communities	Profile2
FAQ	FAQ	Ø	Ø	FAQ	Ø
Activity calendar	Ø	Calendar	Activities	My calendar	Calendar
News	Ø	Ø	Ø	News	News
Wiki	Ø	Wiki	Wiki	Ø	Ø
Rating system	Test and task	Exercises	Module	Evaluates	Event
			Advanced File Uploading	Test	
		Task		Projects	
			Upload File	Task	
Administration	Preference	Yes	Site Administration	Control Panel	Membership (Site administration)

Table 1 Comparison between modules LMS.

2.2 The LMS platforms described by the metamodel

The metamodel does describe all LMS platforms, but describes the schematic repetitive parts of the five LMS and possibly other more. Namely, the refactored parts, for create models that serve specify for parts of the system with major abstraction level, because all is common but variable the whole family of LMS platforms, or at least the platforms studied. Other common parts as security, were not modeled because do not belong strictly to the domain of the LMS and therefore are not described in the metamodel.

Now it is necessary to explain how a module architecture in the LMS platforms is. This is necessary for learning to generate modules in specific platforms and is explained in the next section.

3 Modules Architecture in Learning Management Systems

3.1 LMS's study platforms

In this research work, three LMSs (Moodle Claroline and ATutor) were selected how sample of the population, because this represented more of 50% of the initial population sampling. The Moodle platform was selected because of [17] show that for the year 2009 this was used in more than 45% of Spanish Universities, and Claroline and ATutor were selected because these platforms use the same technology architecture like Moodle, for example, they use a Apache Web server, a MySQL data base and PHP programming language.

For the purpose of this work it is necessary to study the structure and creation of modules in the platforms before mentioned. In first instance the Moodle's module architecture is below:

3.1.1 Moodle architecture

The Moodle version used was 1.9 [23]. A good guide to learn how to create modules in Moodle is its Web page [24], other works that were used for the same purpose are [25] and [26]. The Moodle's modules are stored in a folder named Moodle/mod/module_created each one in a directory, the files and folders general structure is presented below:

- mod_form.php: a form to set up or update an instance of this module.
- lib.php: any/all functions defined by the module should be in here. If the module name is called widget, then the required functions include:
    - widget_install() - will be called during the installation of the module.
    - widget_add_instance() - code to add a new instance of widget.
    - widget_update_instance() - code to update an existing instance.
    - widget_delete_instance() - code to delete an instance.
    - widget_user_outline() - given an instance, return a summary of a user's contribution.
    - widget_user_complete() - given an instance, print details of a user's contribution.
    - widget_get_view_actions() / widget_get_post_actions() - Used by the participation report (course/report/participation/index.php) to classify actions in the logs table.

3.1.2 Claroline architecture

The Caroline's version was 1.8. For the development of modules in Claroline was used the presented in its Web page [27]. The Claroline's modules are stored in a folder named

Claroline/Claroline/module_created, each one in a directory, the files and folders general structure are presented below:

- entry.php: the entry point of the module. In other words is the first execution file.
- manifest.xml: describe the module.
- lib/:directory to store libraries to used.
- lib.php: functions used in the module.

3.1.3 ATutor architecture

In the case of ATutor the version was 2.0.2. For the development of modules in ATutor the one presented in its Web page[28, 29] was used. The ATutor's modules are stored in a folder named ATutor/mods/module_created, each one in a directory, the files and folders general structure are presented below:

- module.php: is the main module file which gets included whenever a page is loaded. It is required.
- module.xml: is used only for identifying the module for distribution and is only used when viewing a module's details. It is required.
- module_install.php is used when installing the module. It is required.
- module_backup.php is used when backing-up and restoring course content. It is optional.
- module_delete.php is used when deleting course specific content. It is optional.
- module_cron.php is used to run module related commands at specified intervals. It is optional.
- module.sql is used to add tables and/or modify data in the ATutor database. Also used to insert language into the language_text table. It is optional.
- lib/:directory to store libraries to used.
- dsllib.inc.php: functions used in the module.

The next step was to generate an LMS metamodel, this is the abstract syntax and this idea is explained in the next section.

4 LMS Metamodel

Before explaining the LMS metamodel, it is necessary to mention the works that have contributed to this proposal, like: [30] in which the goal is to provide a generalized architectural framework enabling an integrated specification of platform architectures, this contribution presents an original approach to the problem of integrating LMS platforms of different architectures. In particular it proposes an integration strategy of LMSs into a generalized model of an LMS with an example of this integration strategy at two sets of functionalities (access rights and learning object management). They modeled a Platform Independent Model (PIM) of two LMS systems (Moodle and OLAT) and showed how to map them to a generic General PIM. The framework still needs to be enriched by more examined LMS systems. [31] proposed a framework for e-learning platforms that allows us to separate the system specification into different complementary perspectives. Moreover, this framework isolates the final development process (technology) from the system modeling by using the MDA concepts provided to capture the system functionality and organize the specification of complex e-learning systems. On the other hand [32] present a framework for developing all kinds of applications based on the opinion of renowned experts in Model-Driven Engineering (MDE) and on the lessons learned from other authors.

The metamodel created is based on the study of LMS platforms [16]. This research work shows modules compatible between LMS platforms and a generic name for them, see Table 1.

In the process of generating the metamodel, the knowledge must be transformed to meet the needs of the model, thus the metamodel is not an exact representation of model with the CmapTool. For the mapping between the model with CmapTool and the metamodel, we used a proposal of [33], where they postulate a relationship between the elements in an ontology or model of domain (model with CmapTool in our case) and the corresponding elements in the metamodel. This mapping is a manual process.

For the construction of metamodel and DSL, we use Eclipse Modeling Components, this includes Eclipse Modeling Framework (EMF) [6], Graphical Modeling Framework (GMF) [34] and Editors for metamodels, among others. The LMS metamodel only represents some parts of the LMSs.

As the metamodel LMS has a correspondence with XMI, below is a simple example. The LMS can have zero or more Users, while the User can have a rol (assignetRol), e-mail (eMail), password and user name (userName). Fig. 1 shows the representation of User EClass in a graphical view and Ecore Explorer. Fig. 2 shows the correspondence between graphical view and XMI. The Eclipse plugin provides all these modes of visualization.

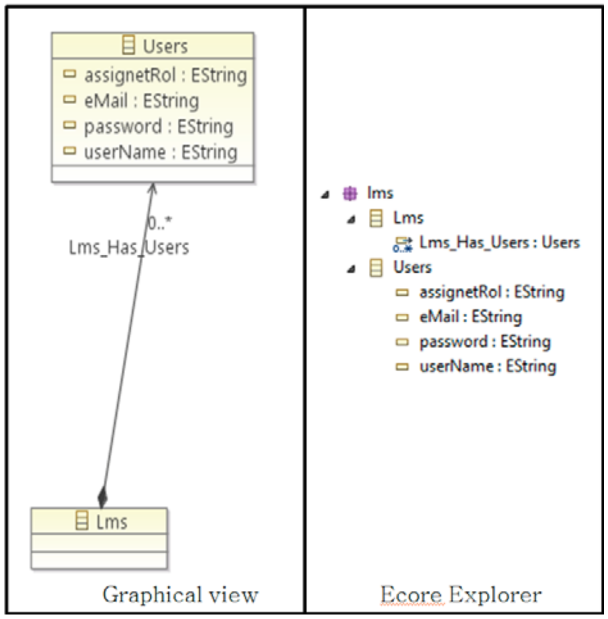
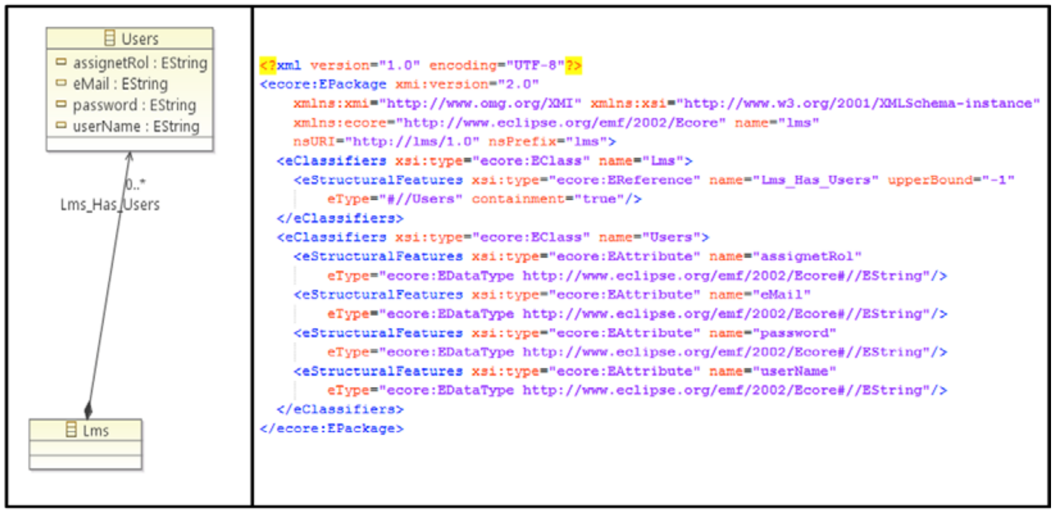


Figure 1 Representation of User EClass in a Graphical view and Ecore Explorer.



```
<?xml version="1.0" encoding="UTF-8"?>
<ecore:EPackage xmi:version="2.0"
    xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore" name="lms"
    nsURI="http://lms/1.0" nsPrefix="lms">
  <eClassifiers xsi:type="ecore:EClass" name="Lms">
    <eStructuralFeatures xsi:type="ecore:EReference" name="Lms_Has_Users" upperBound="-1"
        eType="#//Users" containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Users">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="assignetRol"
        eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="eMail"
        eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="password"
        eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="userName"
        eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  </eClassifiers>
</ecore:EPackage>
```

Figure 2 Correspondence between graphical view and XMI.

The result of conversion between the model with CmapTool and the metamodel is shown in the next three figures (Fig. 3, Fig. 4 and Fig. 5). The LMS has tools for: Course, Communications and Administration. The three elements have in common the attribute name (nameCourse). For this reason there is a Tools EClass and three EClass which inherit Course, Communications and Administration from it.

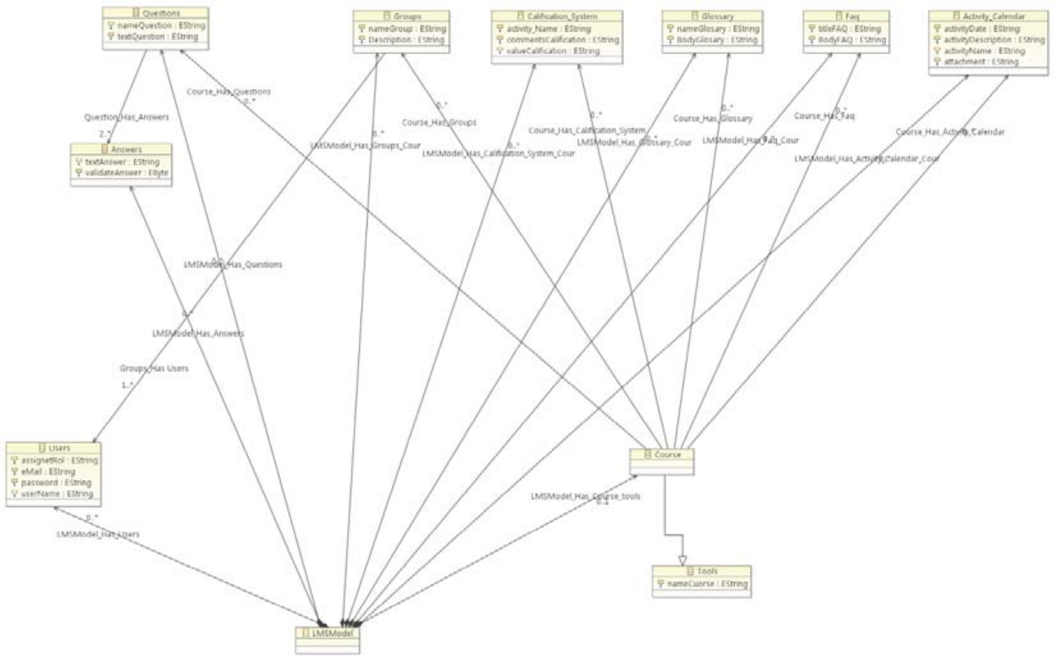


Figure 3 EClasses contained in Course EClass.

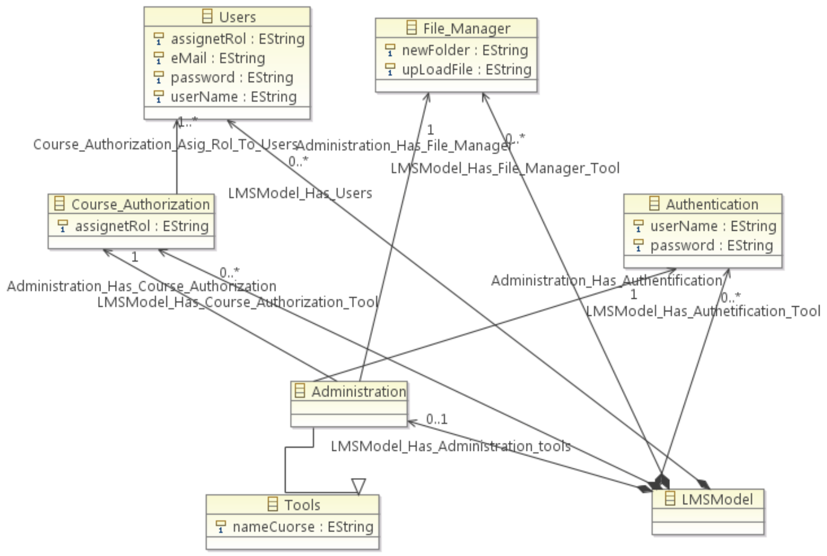


Figure 4 EClasses contained in Administration EClass.

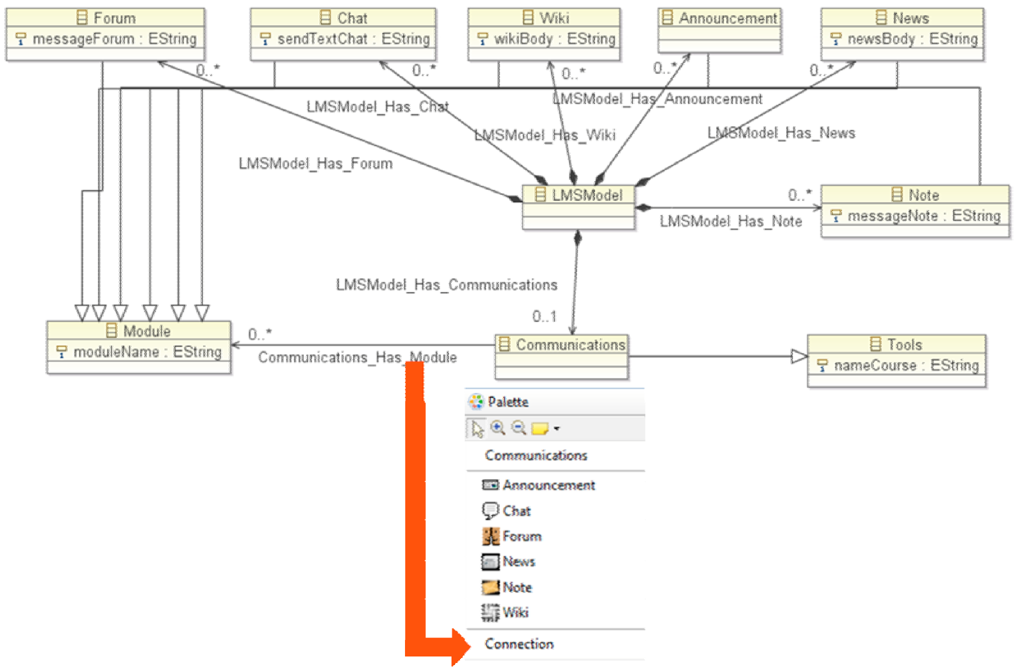


Figure 5 EClasses contained in Communications EClass and Single connection between Communication EClass and Module EClass.

The LMSModel EClass is a container, which stores all the elements of the abstract syntax. One EClass that stores the elements of the model is necessary for the definition of all metamodels. This is similar to a canvas in programming. The LMSModel EClass contains: Users, Authentication, Course_Authorization, File_Manager, Announcement, Chat, Forum, News, Note, Wiki, Activity_Calendar, Calification_System, Faq, Glossary, Groups, Questions, Answers, Course, Communications and Administration.

An aim of the research work is modeling modules of courses for LMSs, and it was felt that the basic modules for a LMS, are those contained in the Communication EClass (Fig 5). In this metamodel the Module EClass is a generalization of Forum, Chat, Wiki, Announcement, News and Note modules. This generalization is necessary to take a single connection with the Communication EClass, the relation between Communication EClass and Module Eclass is named Communications_Has_Module, and this relationship represents a unique connection in the DSL tool. This is shown in the Fig. 5.

The next section describes the construction of the visual tool called KiwiDSM for the Communication modules with the LMS metamodel, this metamodel is shown in Fig. 5 and corresponds to abstract syntax, the next step is create a concrete syntax and this idea is explained in the next section.

5 Construction of the DSL tool

A domain-specific language is a tailor-made language that does only provide abstractions suitable for one particular problem domain, in this case the communications modules of LMSs. For the construction of the DSL tool there are basically two technologies used, EMF and GMF [34]. These technologies were used in the metamodel definition. The GMF solution improves EMF/GEF (Eclipse Graphical Editing Framework) with several metamodels that help to define concrete and abstract syntaxes, their mappings and a simple user-interface customization.

The construction from DSL tool was conducted under the Eclipse platform and this should have EMF, GEF and GMF plugins installed. Fig. 6 shows the steps for the creation of a DSL tool.

According to Fig. 6 the first step is to create the Domain Model, which corresponds to LMS metamodel of Fig. 5. The next step is to create the Graphical Definition Model, which is used to define the nodes, links, etc. that will be displayed on your diagram. The result is a file with the following structure; a Canvas at the root with shapes gallery containing basic Rectangle, Label, and Polyline Connection elements. These are used by corresponding Node, Diagram Label and Connection elements to represent the topics from the domain model. The next step is a Tooling Definition Model, which is used to specify the palette, creation tools, actions, etc. for the graphical elements. The result is a file with the following structure; a top-level “Tool Registry” element, under which we find a Palette. The palette contains a Tool Group with Creation Tool elements for topics nodes and links for subtopic elements.

The Mapping Model is the next step, here the three models are merged: the Domain Model, the Graphical Definition Model, and the Tooling Definition Model. The final step is creating a Diagram Editor Generation Model. The generator model sets the properties for code generation. The final result is shown in Fig. 7. This is the view of KiwiDSM.

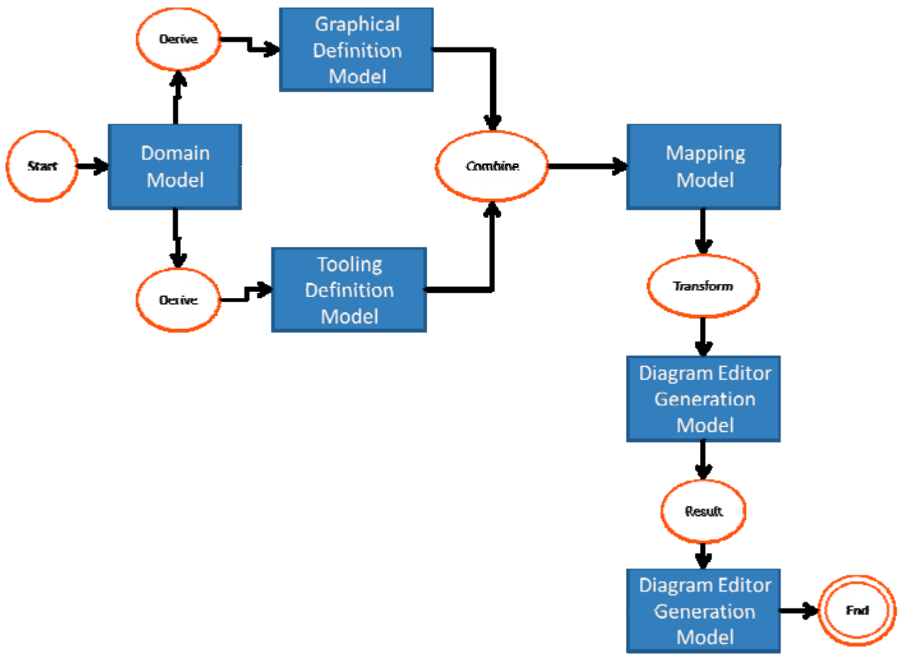


Figure 6 Steps to create DSL tool.

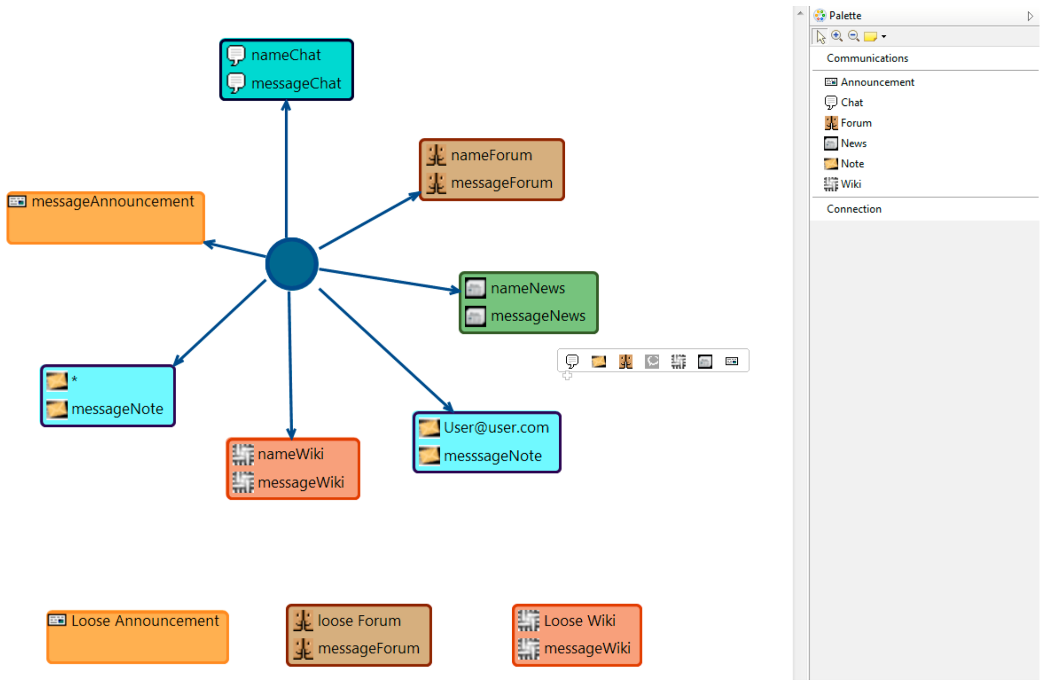


Figure 7 KiwiDSM: DSL tool for modeling LMS modules in some LMS platforms.

The operation of DSL tool is very simple. To create modules, select a module of the palette, drag and drop in the model diagram workspace, fill in the fields and connect nodes with the connection of the pallet, following the next guidelines:

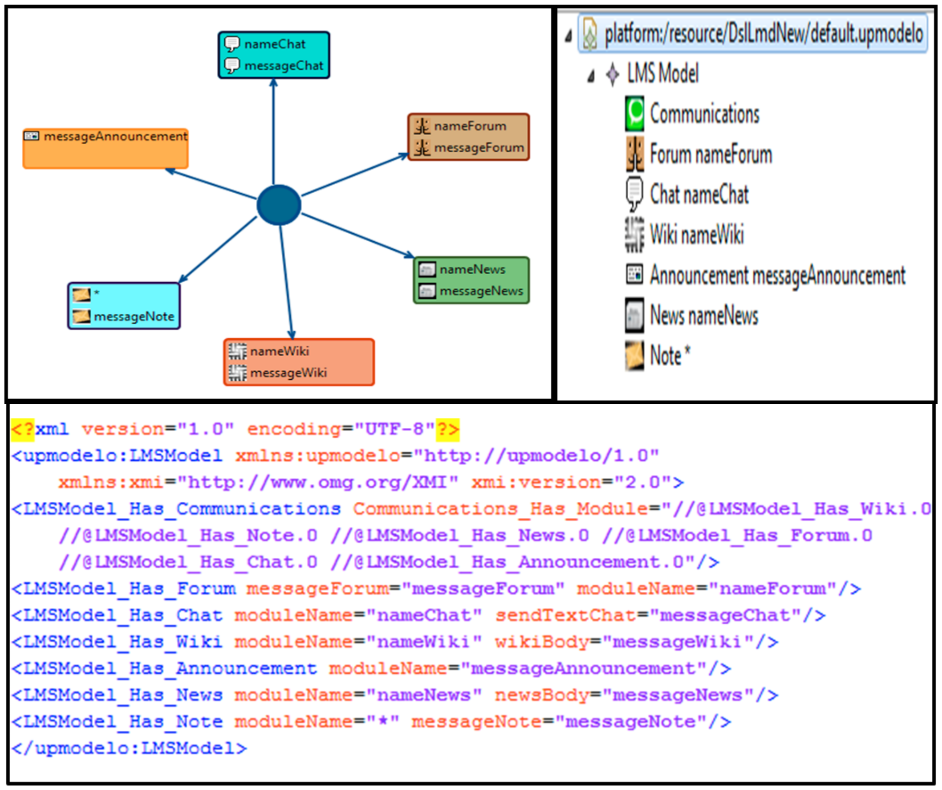
- A course can have a single module "Communications".
- The module "Communications" has zero or many: "Announcements", "Chat", "Forum", "News ", "Note" and "Wiki".

The tool validates the following cases:

- It only allows a single module "Communications".
- When you generate the code to be deployed, those nodes that are loose (no connection) will not be considered for the creation of the course.

It is very important to consider that the model with this tool is deployed on a single topic of the course; therefore it is necessary to consider the names of the fields in each element (node or module) as generics, to avoid change the model in every deployment on the course.

The Fig. 8 shows an example of a model and the relationship of each node and connection with the XMI file that generates DSL tool.



```
<?xml version="1.0" encoding="UTF-8"?>
<upmodelo:LMSModel xmlns:upmodelo="http://upmodelo/1.0"
    xmlns:xmi="http://www.omg.org/XMI" xmi:version="2.0">
<LMSModel_Has_Communications Communications_Has_Module="//@LMSModel_Has_Wiki.0
    //@LMSModel_Has_Note.0 //@LMSModel_Has_News.0 //@LMSModel_Has_Forum.0
    //@LMSModel_Has_Chat.0 //@LMSModel_Has_Announcement.0"/>
<LMSModel_Has_Forum messageForum="messageForum" moduleName="nameForum"/>
<LMSModel_Has_Chat moduleName="nameChat" sendTextChat="messageChat"/>
<LMSModel_Has_Wiki moduleName="nameWiki" wikiBody="messageWiki"/>
<LMSModel_Has_Announcement moduleName="messageAnnouncement"/>
<LMSModel_Has_News moduleName="nameNews" newsBody="messageNews"/>
<LMSModel_Has_Note moduleName="*" messageNote="messageNote"/>
</upmodelo:LMSModel>
```

Figure 8 Sample of a model and the relationship of each node and connection with the XMI file.

Observations to be considered in the modeling with this DSL tool:

- If you want to send a "Note" to all those enrolled in a course, the field "sendTo" should be filled with a “*”.
- If you want to send a "Note" to a single enrolled in a course, the field "Send To" should be filled with the email address, with which the person is enrolled in the virtual platform. If this email address does not match that of the person, the message will not be sent.
- It is necessary to fill in all fields in the nodes. For those fields in which the user does not the fill, information will be taken as null and it will not be processed.

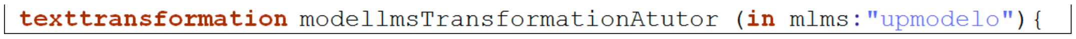
In summary, this section talked about how to create a DSL visual tool, based on a metamodel, which in turn is based on the Ecore meta-metamodel, the next step is to convert this new model, that was obtained with the DSL tool and make a transformation to code, and this idea is explained in the next section.

6 Transformation and deployment of model with KiwiDSM in LMS Platforms

In this section the problem of transforming a model to code for deployment on the specific platform is presented, this type of transformation is called “Model to Text” or “M2T”. MOFScript is the technology used to make this parser [35]. It is an Eclipse’s plug-in that works with special language rules. The next step is the transformation to code from the model created with the KiwiDSM tool.

For this we have three files with the templates required for each LMS, these files are located in the Eclipse project where the DSL tool was developed. These are: “modellmsTransformationMoodle.m2t” for Moodle, “modellmsTransformationClaroline.m2t” for Claroline and “modellmsTransformationATutor.m2t” for ATutor. The model generated was applied to the M2T transformation file and like a result we have the files required to install a module with all the model requirements specified by the user in the DSL tool, just we need to compress, store or install the folder in the LMS's specified route in section 3.

MOFScript is a language based on rules, introduced by the OMG, for model transformations to text (M2T). It can be installed as an Eclipse plugin. A good user guide to start development in MOFScript is [36]. In MOFScript the first step is to define the start model to the MOFScript’s template, to do this it is necessary to declare the transformation with “texttransformation”. You need to add a parameter with the name of the metamodel created (modellms) with the DSL tool with the extension “.ecore”, as it can be seen in the follow code line (see Fig. 9).



```
texttransformation modellmsTransformationAtutor (in mlms:"upmodelo"){
```

Figure 9 Declaration the transformation in MOFScript.

If you need to use variables, the next step is to add these just after the above line, in this case we add some variables to control creation of files and modules, like you can see in the follow code lines (see Fig. 10).

```
var numForumY:integer = 0;
var numForumN:integer = 0;
var numForumSN:integer = 0;
var numAnnouncementY:integer = 0;
var numAnnouncementN:integer = 0;
var numAnnouncementSN:integer = 0;
var numChatY:integer = 0;
var numChatN:integer = 0;
var numChatSN:integer = 0;
var numWikiY:integer = 0;
var numWikiN:integer = 0;
var numWikiSN:integer = 0;
var numNewsY:integer = 0;
var numNewsSN:integer = 0;
var numNewsN:integer = 0;
var numNoteY:integer = 0;
var numNoteN:integer = 0;
var numNoteSN:integer = 0;
var numChatmsn:integer = 0;
var numWikitext:integer = 0;
```

Figure 10 Variable declaration in MOFScript..

The principal task or starting point for transformations in MOFScript are contending in the "main ()" which is declared as follows (see Fig. 11).

```
mlms.LMSModel::main () {
```

Figure 11 Main function “main” in MOFScript.

To create a new file you use the "file" as shown below (see Fig. 12 or 13).

```
mlms.LMSModel::main () {
```

Figure 12 Creating a file in MOFScript.

```
('mod_form.php');
```

Figure 13 Creating a file in a folder with MOFScript.

Finally, to write the strings in that file should go between the character ('). For the a function declaration, you must first putting the word "module" followed by the operator “: :” and then the code as follow (see Fig. 14).

```
module::header()
{'<?php
define('+quote+'AT_INCLUDE_PATH'+quote+','+quote+'../../include/'+quote+');
require  (AT_INCLUDE_PATH.'+quote+'vitals.inc.php'+quote+');
require(AT_INCLUDE_PATH.'+quote+'lib/mysql_connect.inc.php'+quote+');

authenticate(AT_PRIV_PLANTILLADSL);

$_pages['+quote+'mods/plantilladsl/index.php'+quote+']['+quote+'title'+quot
e+']  = '+quote+'plantilla_dsl'+quote+';

require(AT_INCLUDE_PATH.'+quote+'../mods/plantilladsl/lib/dsllib.inc.php'+q
uote+');
'
}
```

Figure 14 MOFScript header function for ATutor.

To invoke functions, it is called in where needed, as in the metamodel are contained EClass all in one EClass called LMSModel, this relationship is represented by a connection. Then validate the model if there is a model (Eclass) Communications relates to LMSModel through LMSModel_Has_Communications, this is show below, see Fig. 15.

```
header();
if (self.LMSModel_Has_Communications != null){
```

Figure 15 Call to a function and provide validation if there is any model EClass Communications.

Below, it is shown how to count how many modules a model has, regardless if they are connected or not (see Fig. 16).

```
numForumN = self.LMSModel_Has_Forumm.size();
numAnnouncementN = self.LMSModel_Has_Announcement.size();
numChatN = self.LMSModel_Has_Chat.size();
numWikiN = self.LMSModel_Has_Wiki.size();
numNewsN = self.LMSModel_Has_News.size();
numNoteN = self.LMSModel_Has_Note.size();
```

Figure 16 Counter modules within a model.

After all modules are checked, you need to get into the module Forum which are within the model and connected to the Communications module, it is validated if the fields are filled, if not filled a counter incrementing by 1 the variable Forum. It is determining that the modules are connected but have no information. If they are filled, a new line is printed in the file along with two tab spaces. Invoking the corresponding function in Moodle, Claroline or ATutor with input parameters names entered from the model created and finally increments a counter of the modules created correctly (see Fig. 17).

```
self.LMSModel_Has_Communications.Communications_Has_Module-
>forEach(modul:mlms.Module)
{
 if (modul.oclGetType()='Forum'){
 var forum:mlms.Forum;
 forum = modul;
 if (forum.moduleName = null or forum.messageForum = null)
 {
  numForumSN = numForumSN+1;
 }
 else
 {
  newline(2);
  tab(3);
  '$forum_name= '+quote+forum.moduleName+quote+';
  $forum_desc= '+quote+forum.messageForum+quote+';
  add_forum_dsl($forum_name,$forum_desc);'
  numForumY = numForumY+1;
 }
}
```

Figure 17 Reviewing the Forum requested in MOFcript modules for ATutor.

The Table 2 shows the MOFScript instructions and the generated files, is shown the MOFScript instructions for each LMS used in the present work and the files that were generated.

LMS	MOFScript instructions.	Files Generated
Moodle	• file ('mod_form.php'); • encabezado(); • final(); • createlibphp(); • file ('lib.php');	• mod_form.php • lib.php
Claroline	• file ('entry.php'); • encabezado(); • final(); • createlibphp(); • file ('plantilladsl.lib.php');	• entry.php • plantilladsl.lib.php
ATutor	• file f1('plantilladsl/index.php'); • encabezado(); • file fchat('dslchat/'+numChatmsn+'.message'); • file fwiki('dslwiki/'+'FrontPage.'+numWikitext); • final(); • createlibphp(); • file ('/lib/dsllib.inc.php'); • createsql(); • file ('module.sql'); • modulephp(); • file ('module.php'); • moduleuninstall(); • file ('module_uninstall.php'); • moduleinstall(); • file ('module_install.php'); • modulexml(); • file ('module.xml');	• plantilladsl/ • ../index.php • ../dlschat/2.message • ../dslwiki/FrontPage.2 • ../lib/dsllib.inc.php • ../module.sql • ../module.php • ../module_uninstall.php • ../module_install.php • ../module.xml

Table 2 MOFScript instructions and files generated.

In a development environment with the KiwiDSM tool, the teacher creates the specific model. When finish the design, the teacher needs to give to an Administrator, the generated file from the tool to make the transformation and deploy in the selected LMS platform.

Once the Administrator receives the files sent by the teacher, for this sample scenario "default.modellms" and "default.modellms_diagram".

1. Change the file extension "default.modellms" to "default.ecore".
2. Open Eclipse and apply the changes to the MOFScript file “modellmsTransformationMoodle.m2t” or “modellmsTransformationClaroline.m2t" or "modellmsTransformationATutor.m2t", it depends on the LMS selected by the teacher.

This transformation will generate the files mentioned above; when the process is finished, it is necessary compress the folder with the files. The installation depends the LMS's administrator, for sample in ATutor is necessary log in as Administrator, go to the Modules's option and Install Modules, and select he file zip that was created in the transformation and click on the install selection. When the installation is finished in the case of ATutor, the module needs to be "enabled". If you want to uninstall click on the button "uninstall".

With these steps the teacher will see the module with all the specified in the model developed with the KiwiDSM tool and can be activated in the course. Like we comment the transformation and deploy depends on the model specific created with KiwiDSM and the LMS platform selected. The source code of KiwiDSM, the MOFScript templates and the LMS platforms modules, it is available in http://www.vicegd.com/KiwiDSMCode.rar.

7 Tests, validation and discussions about creating modules with LMS platforms VS creating modules with our DSL Tool

7.1 Research experiment

The tool KiwiDSM was tested in the three LMS platforms. The objective is to test the performance using the tool that performs a specific modeling domain for the construction of modules LMS independent of the platform, against the creation of these modules manually. In this case like we commented we used Moodle, Claroline and ATutor to measure parameters of usability (effort) [37] and time (in seconds) in each of the mentioned cases. This test was presented by sixteen people with basic knowledge of the LMS platforms, each one with the instructor role created five topics in his course, using the DSL tool and the platform LMS with the ordinary way.

The measurement of time was made for both environments and we took the next:

- The time for addition of information for each tool.
- The time for addition of information on all tools that makes a topic.
- The total time in adding information on the tools of the five topics.

The measurement of effort getting the amount of times the user had to select or enter any data into the system:

- Effort in adding information for each tool.
- Effort in adding information on all tools for a topic.
- Total effort in adding information on the tools of the five topics.

Chat	Name	Chat
	Text	Welcome to Chat
Forum	Name	Forum
	Text	Welcome to Forum
Wiki	Name	Wki
	Text	Welcome to Wiki
Announcement	Text	New topic
News	Name	New topic
	Text	New topic available
Notes student1@uniovi.es	Send to	student1@uniovi.es
	Text	New topic available
Notes All Users	Text	New topic available

Table 3 Data to create specific model in KiwiDSM and LMS platforms.

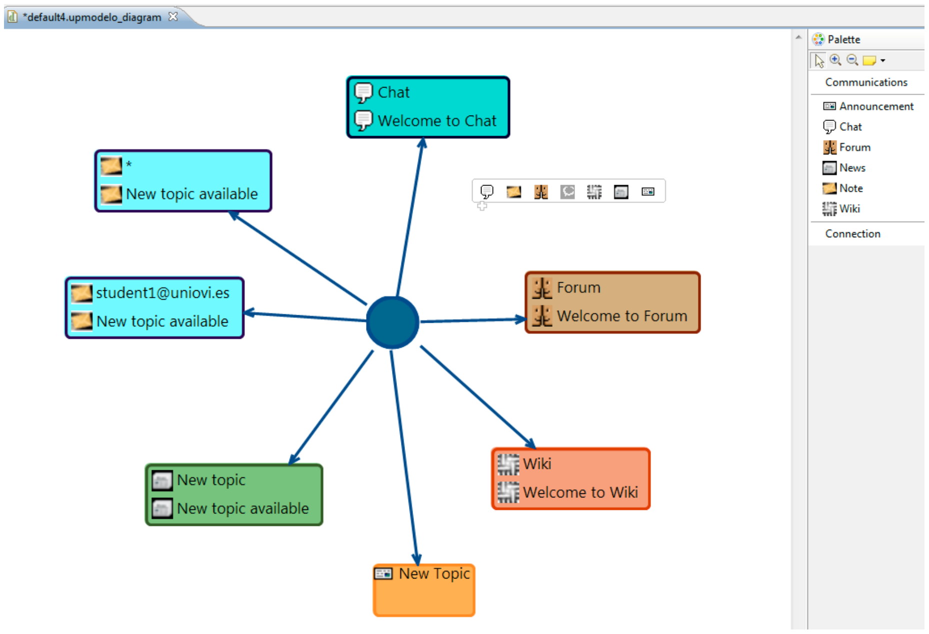


Figure 18 Specific model for the test.

For the test it is necessary to compare the time of model construction in the DSL tool with each LMS platform. For this in KiwiDSM, first the instructor needs to create the module and then the link, and not create all the nodes and later all the links. As the test seeks to measure the time and level of effort in adding information to each module, for this is necessary to give the data (see Table 3), the final specific model you can see the final specific model in Fig. 18.

7.2 Results

Fig.19 shows the average time of creating five modules per topic in Moodle. The difference to create the first topic, is a 1.75% more time to create the modules in Moodle than with the DSL tool, but this difference grows from the second topic onwards, until it has a difference of 66.65% faster build the modules with the DSL tool than with Moodle. It is observed that the time to create modules in Moodle grows faster than with our DSL tool. Fig. 20 shows the average effort of creating five modules per topic, and presents a similar behavior. The difference after creating the first topic is a 12.68% less effort to create the modules in Moodle than with the DSL tool, but from the second topic onwards, the effort in the creation of modules with the DSL tools is at least 63% less than creating them in Moodle.

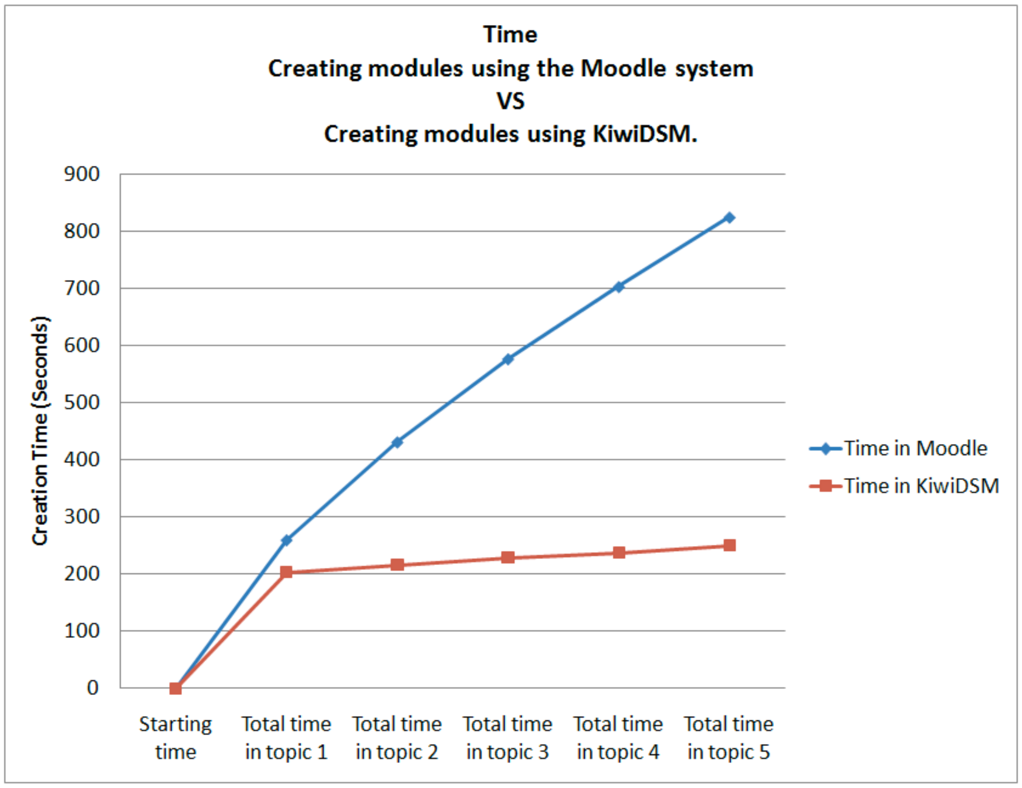


Figure 19 Comparison of average times between creating modules using the Moodle system VS creating modules using KiwiDSM.

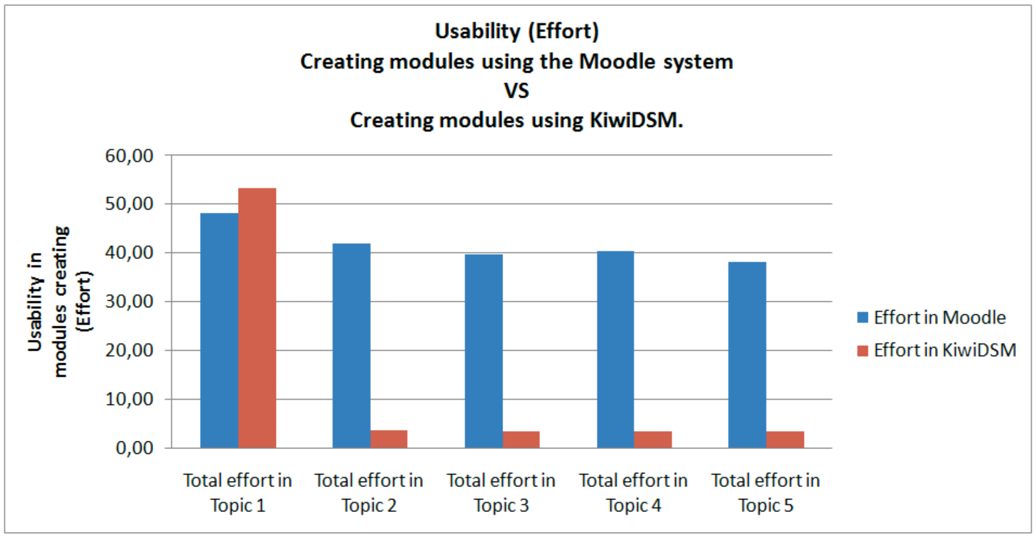


Figure 20 Comparison of average effort between creating modules using the Moodle system VS creating modules using KiwiDSM.

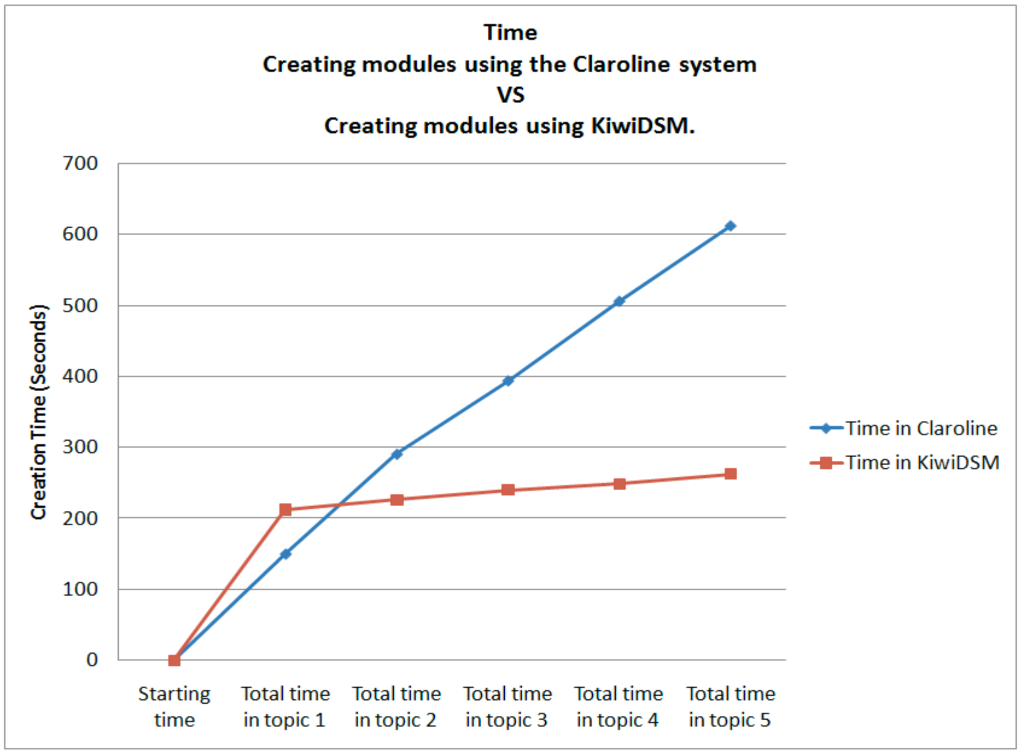


Figure 21 Comparison of average times between creating modules using the Claroline system VS creating modules using KiwiDSM.

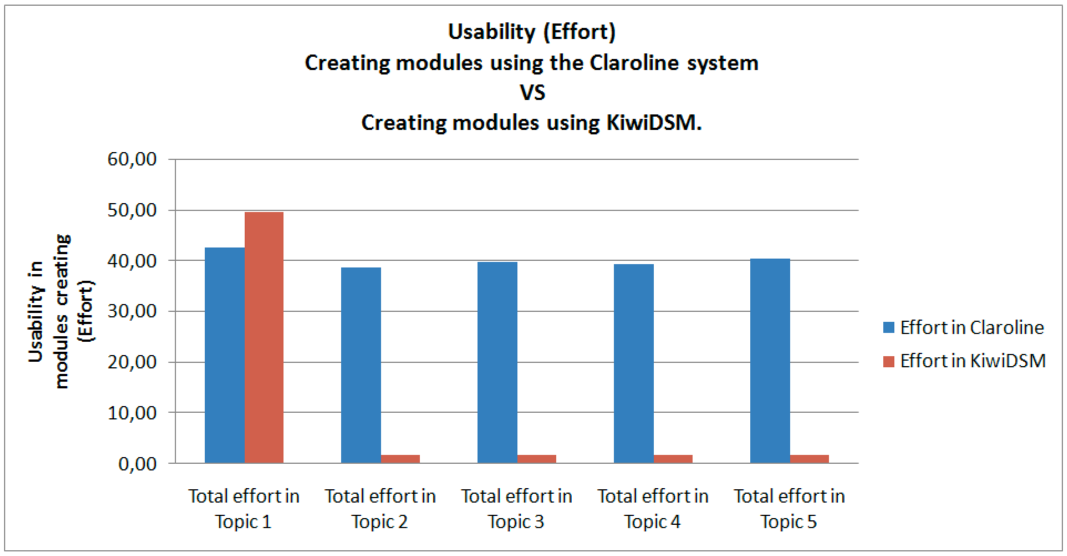


Figure 22 Comparison of average effort between creating modules using the Claroline system VS creating modules using KiwiDSM.

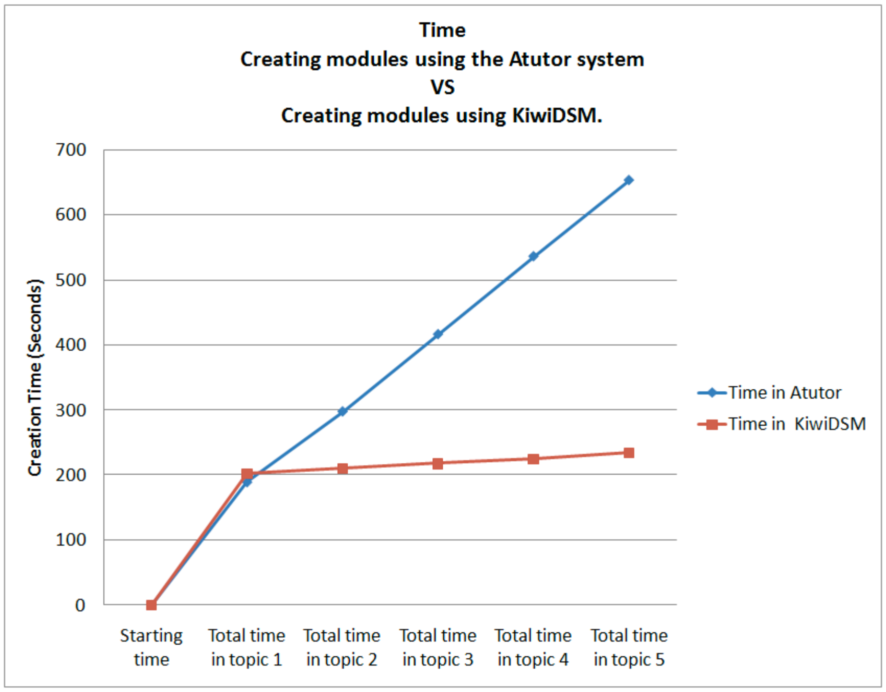


Figure 23 Comparison of average times between creating modules using the Atutor system VS creating modules using KiwiDSM.

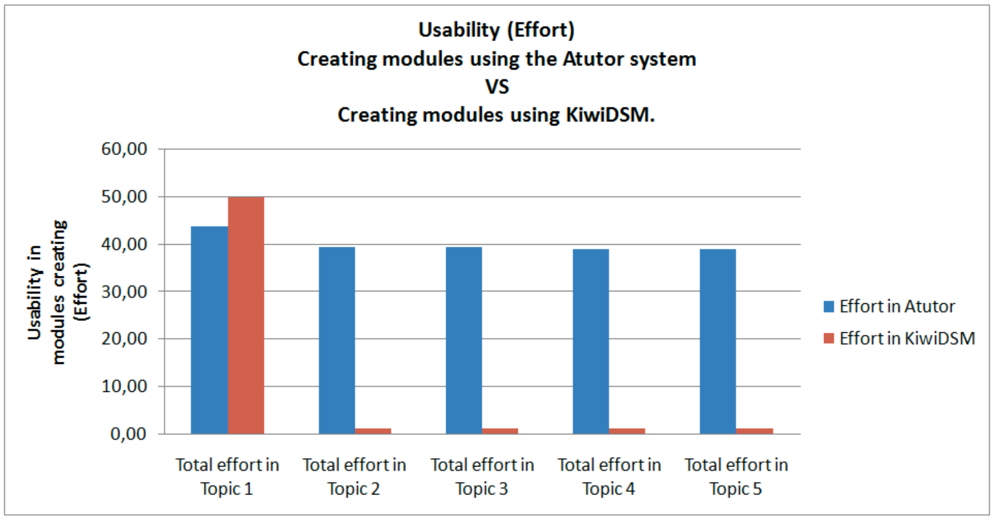


Figure 24 Comparison of average effort between creating modules using the Atutor system VS creating modules using KiwiDSM.

Fig. 21 shows the average time of creating five modules per topic in Claroline. In this case the difference to create the first topic, is a 15.69% more time to create the modules in the DSL tool than with Claroline, but this difference changes from the second topic onwards, until it has a difference of 51.62% faster build modules with the DSL tool than with Claroline. It is observed that the time to create modules in Claroline grows faster than with our DSL tool too. Fig. 22 shows the average effort of creating five modules per topic, and presents a similar behavior. The difference after creating the first topic is a 18.39% less effort to create the modules in Claroline than with the DSL tool, but from the second topic onwards, the effort in the creation of modules with the DSL tools is at least 69.35% less than creating them in Claroline.

Fig. 23 shows the average time of creating five modules per topic in Atutor. In this case the difference to create the first topic, is a 7.65% more time to create the modules in the DSL tool than with Atutor, but this difference changes from the second topic onwards, until it has a difference of 58.87% faster build modules with the DSL tool than with Atutor. It is observed that the time to create modules in Atutor grows faster than with our DSL tool too. Fig. 24 shows the average effort of creating five modules per topic, and presents a similar behavior. The difference after creating the first topic is a 17.98% less effort to create the modules in Atutor than with the DSL tool, but from the second topic onwards, the effort in the creation of modules with the DSL tools is at least 71.16% less than creating them in Atutor.

Finally, conclusion of this section is that modules creation with the DSL tool, reduced time significantly, and has performing better when created many modules. “When created many modules with DSL tool, the performance is better”.

7.3 Validation

The tests that were performed at the DSL tool model properties were based on the requirements defined in the metamodel, in order to validate the model built to meet the requirements of the metamodel. These tests were mainly created in the platform modules LMS: Moodle, Claroline and ATutor, which have the necessary transformations of the model to them and which were explained in previous sections.

We defined a set of tests that describe the activity that can be done with the DSL tool to be deployed in platforms LMS (Moodle, Claroline and ATutor), these tests correspond to different behaviors that can be given to fulfill a requirement. For each test, we made the following formalization:

Test = {<formula,result>} where,

formula Є to composition <input, output>

result = 1 (true), if the formula models a valid behavior and 0 (false), if the formula models an invalid behavior.

These tests included valid and invalid behaviors. If the test goes well in an invalid behavior indicated that the model is flawed. Each test is modeled with a Petri net [38], to observe their behavior and thus to find errors in the model.

A Petri Net is a quadruple R = {P, T, I, O} where:

P is a finite and nonempty set of nodes, given as conditions.

T is a nonempty finite set of transitions, given as events.

$P \cap T = 0$

I: $P \times T \rightarrow$ Input Function

O: $T \times P \rightarrow$ output function

Described below are tests defined for the creation of each module.

Wiki module creation:

Consider the following nodes and transitions:

P = {p1, p2, p3, p4, p5, pEnd}

T = {t1, t2, t3, t4, t5, t6}

p1 = Waiting elements in the Canvas.

p2 = Node Communications created in Canvas.

p3 = Node Wiki created in Canvas.

p4 = Data wikiName assigned to the Wiki.

p5 = Data wikiBody assigned to the Wiki.

pEnd = Module Wiki deployed on the platform.

t1 = Communications was selected node from the tool palette and placed on the Canvas.

t2 = Wiki node was selected in the tool palette and placed on the Canvas.

t3 = Connection was selected of the tool palette, and connected: the Communications node with Wiki node.

t4 = information was assigned to the field node wikiName Wiki.

t5 = information was assigned to the field node wikiBody Wiki.

t6 = proper changes are made and deployment on the LMS platform.

Fig. 25 shows the petri net for the creation of the Wiki module.

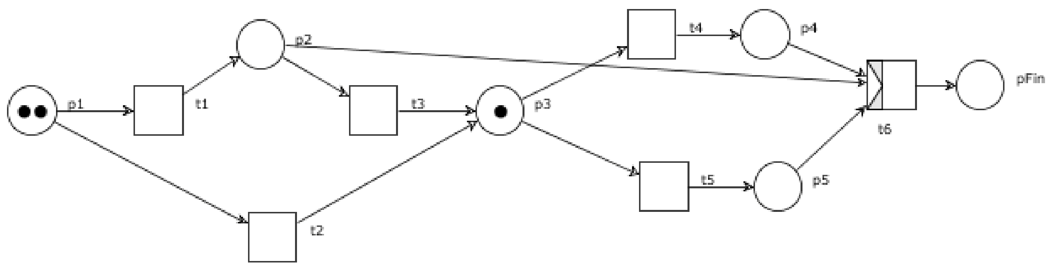


Figure 25 Petri rule for Wiki module.

Valid behavior

Test1 = {<p1,t1>, <t1,p2>, <p1,t2>, <t2,p3>, <p3,t4>, <p3,t5>, <t4,p4>, <t4,p5> , <t6,pEnd>, 1}

In this trial, it was obtained as a result 1, which indicates that this behavior occurs in the model and therefore meets the requirement. Below is graphically the end of the test in Petri net, simulating the behavior and function entry and exit in each transition (Fig. 26).

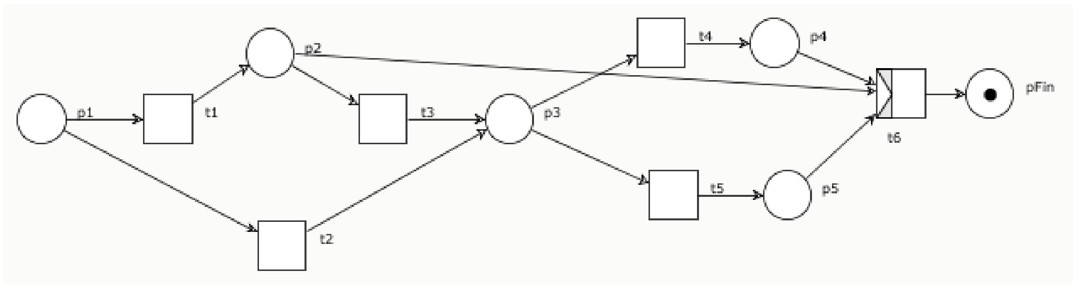


Figure 26 Valid behavior.

Invalid behavior

Test2 = {<p1,t2>, <t2,p3>, <p3,t4>, <t4,p4>, <p3,t5>, <t5,p5>, 0}

In this trial, it was obtained as a result 0, which indicates that this behavior does not occur in the model. Below is graphically the end of the test in Petri net, simulating the model, the behavior, function entry and exit in each transition (Fig. 27).

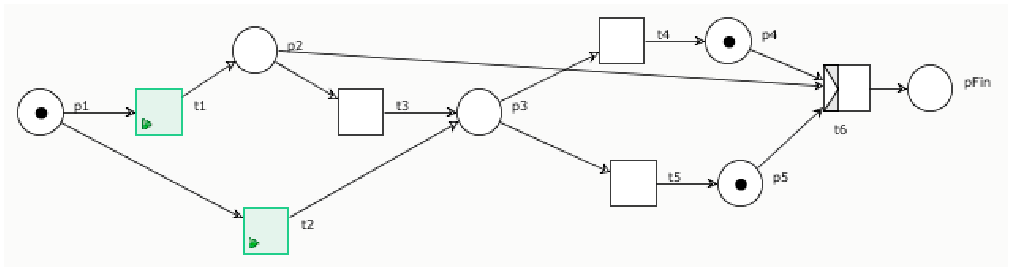


Figure 27 Invalid behavior.

For all the cases in the modules creation was defined a set of Petri nets and validated each one like the Forum module. With this validation we demonstrated that any module created with the KiwiDSM tool is correct.

8 Related works

Most of the related work has already been cited in the previous sections. Work like that of Bizoňová [30], The goal is to provide a generalized architectural framework enabling an integrated specification of platform architectures. This platform-independent framework can then be used to specify and classify existing or future Learning management systems and to simplify migration of data between different kinds of e-learning systems.

The relevance of the research works like [39, 40], but mainly [41] in the proposal of LMS ontologies or model of domain. Less directly related are the papers debating between ontology or model of domain and metamodel, as originally discussed in [33, 42-44], the major characteristic is to provide an equivalence between ontologies (model of domain) and metamodels.

Another interesting issue is MDE, research works like [6, 32, 45-47]. These works provide the knowledge base in: MDA, MOF, Ecore, metamodels, meta-metamodels, EMF and XMI.

About the framework used to define metamodels, it is appropriate to think about existing metamodels and their representations for Objects and connections. There are two main protagonists in this scene: OMG with MOF [4], and Eclipse Modeling Project with Ecore and EMF [48], which is the basis of this work.

The importance and roles of visual representations led to a new interdisciplinary research area about visual design for Technology Enhanced Learning (TEL). The main objective is to support and enhance the quality of the design of TEL systems and to facilitate idea sharing, collaboration, and reuse[49]. But visual representations can also be applied in other areas with DSL, such as [47, 50-54].

9 Conclusions and Future Research Work

This research work offers a LMS Metamodel that integrates several LMS platforms, allowing to create modules in a visual DSL tool: KiwiDSM and convert them automatically to three LMS platforms, an approach which can be seen in [55].

In this Article is presented a complete case of applied MDA in a real project. The Metamodeling techniques for domain analysis enable definition of domains as metamodels that serve both for capturing domain knowledge and for validating particular applications in a domain. And there are four levels M0, M1, M2 and M3 [56]. We have explored all these levels. The LMS Metamodel belongs to M2 level, the Level M3 is a meta-metamodel Ecore, and the next step is to create the M1 level. To create this level it is necessary to generate a Domain Specific Language (DSL) and for this, Graphical Modeling Framework (GMF) was used [57]. This DSL provides the user a tool for generating LMS modules for independent platforms. The level M0 is a model to text "M2T" transformation. In this case MOFScript [35] has been used. This is an area for new investigations with other technologies.

The EMF and Modeling Components of Eclipse project provide a way to make metamodels in Ecore and validate your correct structure. The point of view of [33], which identifies a “bridge” between the conceptualizations represented by metamodels and those by ontologies, particularly foundational and domain ontologies and identifying the difference between these latter two in the software engineering/modeling context is very important. This is an area for new investigations.

The above evidence demonstrates that by working with models, you reduce the time and effort to create modules for some LMS platform, in this case (Moodle, Claroline and ATutor) and further than the metamodel work, as it did not generated any flaw in the creation of visual tool modules in the KiwiDSM tool, which is based on the metamodel LMS created.

For future work, it is necessary to implement the LMS metamodel and the DSL tool in another EClass not only communications, to test and apply a new conceptual interaction between the metamodel, the DSL tool and the final user. Another is to create software to incorporate the generated files in the LMS platforms automatically when just the transformation is finished.

References

1. Shaw, M., Software engineering education: a roadmap, in Proceedings of the Conference on The Future of Software Engineering. 2000, ACM: Limerick, Ireland. p. 371-380.
2. OMG, *MDA Guide Version 1.0.1*. 2003.
3. OMG, *OMG Unified Modeling Language Specification, Version 1.3*. 1999.
4. Group, O.M., *MOF 2.0/XMI Mapping, Version 2.1.1*. 2007, Object Management Group. p. 120.
5. Atkinson, C. and T. Kühne. *{The role of metamodeling in MDA}*. in *Proc. UML 2002 Workshop Software Model Eng*. 2002.
6. Budinsky, F., et al., *EMF: Eclipse Modeling Framework*. 2009: Addison-Wesley.
7. Márquez, V.J.M., *Estado del arte del eLearning. Ideas para la definición de una plataforma universal*, in *DEA, Departamento de Lenguajes y Sistemas Informáticos*. 2007, Universidad de Sevilla: Sevilla - Spain.
8. SIGOSSEE, O.S.f.E.i.E.G. *Evaluación de las plataformas LMS*. 2010 [cited 2010 October]; Available from: http://www.guidance-research.org/sigossee/join/sp/.
9. CmapTools, I. *IHMC CmapTools*. 2010 [cited 2010 October]; Available from: http://cmap.ihmc.us/.
10. Rodriguez, M.A., J.D. Barrios, and E.S. Schultz, *THE USE OF AN INNOVATION CLASSROOM A Perspective in the Introduction of ICT in Elementary Schools*. Csedu 2009:

Proceedings of the First International Conference on Computer Supported Education, Vol I, ed. J. Cordeiro, et al. 2009, Setubal: Insticc-Inst Syst Technologies Information Control & Communication. 173-180.

11. Roy, D. and Ieee, *Using Concept Maps for Information Conceptualization and Schematization in Technical Reading and Writing Courses: A Case Study for Computer Science Majors in Japan*. 2008 Ieee International Professional Communication Conference. 2008, New York: Ieee. 341-352.

12. Brine, J., I. Wilson, and D. Roy, *Using moodle and other software tools in EFL courses in a Japanese IT university*. 2007 Cit: 7th Ieee International Conference on Computer and Information Technology, Proceedings, ed. T. Miyazaki, I. Paik, and D. Wei. 2007, Los Alamitos: Ieee Computer Soc. 1059-1064.

13. Canas, A.J., et al., *Concept maps: Integrating knowledge and information visualization*, in *Knowledge and Information Visualization: Searching for Synergies*, S.O. Tergan and T. Keller, Editors. 2005, Springer-Verlag Berlin: Berlin. p. 205-219.

14. Canas, A.J., et al., *Using WordNet for word sense disambiguation to support concept map construction*, in *String Processing and Information Retrieval, Proceedings*, M.A. Nascimento, E.S. DeMoura, and A.L. Oliveira, Editors. 2003, Springer-Verlag Berlin: Berlin. p. 350-359.

15. R. Farenhorst, et al. *What's in Constructing a Domain Model for Sharing Architectural Knowledge?* in *Proceedings 18th International Conference on Software Engineering & Knowledge Engineering (SEKE 2006)*. 2006. San Francisco, CA, USA: July 5-7.

16. Montenegro, C., et al., *MODELING AND COMPARISON STUDY OF MODULES IN OPEN SOURCE LMS PLATFORMS WITH CMAPSTOOL*. International Journal of Interactive Multimedia and Artificial Intelligence newsletter, 2010.

17. Álvarez, V., *Voice Interative Classroom, a service-oriented software architecture to enable cross-platform multi-chanel access to Internet-based learning*, in *Computer science*. 2010, University of Oviedo: Oviedo.

18. Campanella, S., et al., *E-learning platforms in the Italian Universities: the technological solutions at the University of Bari*. WSEAS TRANSACTIONS on ADVANCES in ENGINEERING EDUCATION, 2008. **5**(1).

19. Parnas, D.L., *On the Design and Development of Program Families*. Software Engineering, IEEE Transactions on, 1976. **SE-2**(1): p. 1-9.

20. Stahl, T. and M. Voelter, *Model-Driven Software Development: Technology, Engineering, Management*. 2006: Wiley.

21. Lenz, G. and C. Wienands, *Practical Software Factories in .NET (Practical)*. 2006: Apress.

22. PENG, X., et al., *Ontology-based feature modeling and application-oriented tailoring*. Vol. 4039. 2006, Berlin, ALLEMAGNE: Springer. XIII-444 p.

23. moodle. *Development:NEWMODULE Documentation*. 2010 [cited 2011 Jan]; Available from: http://docs.moodle.org/en/Development:NEWMODULE_Documentation#Getting_started.

24. Moodle. *Moodle*. 2011 [cited 2011 Feb]; Available from: http://moodle.org/.

25. Ivorra, R., *Tutorial: Creaciòn de un mòdulo actividad. Moodle (1.9.3)*. 2009.

26. Gonzàlez, A., *Guìa de apoyo para el uso moodle 1.9.4 Usuario Desarrollador*, in *Informatica*. 2009, Universidad de Oviedo: Oviedo.

27. Consorcio Claroline. *Claroline*. [Internet] 2008 2008 [cited 2010 October]; Available from: http://www.claroline.net.

28. ATutor. *Module Development Documentation.* 2011 [cited 2011 03/07/2011]; Available from: http://atutor.ca/development/documentation/modules.html#structure.

29. ATutor. *ATutor Learning Managment Tools*. 2011 [cited 2011 03/07/2011]; Available from: http://atutor.ca/.

30. Bizoňová, Z., D. Ranc, and M. Drozdová. *Model Driven E-Learning Platform Integration*. in *2nd European Conference on Technology Enhanced Learning EC-TEL PROLEARN 2007 Doctoral Consortium*. 2007. Crete, Greece: CEUR-WS.org.

31. MORENO, N. and J.R. ROMERO, *A MDA-based framework for building interoperable e-learning platforms*, in *Recent Research Developments in Learning Technologies (2005)*, A. MÉNDEZ-VILAS, et al., Editors. 2005: Badajoz, Spain (2005).

32. García-Díaz, V., et al., *TALISMAN MDE Framework: An Architecture for Intelligent Model-Driven Engineering*, in *Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing, and Ambient Assisted Living*, S. Omatu, et al., Editors. 2009, Springer Berlin / Heidelberg. p. 299-306.

33. Henderson-Sellers, B., *Bridging metamodels and ontologies in software engineering.* Journal of Systems and Software, 2011. **84**(2): p. 301-313.

34. Foundation, T.E. *GMF Tutorial.* 2010 [cited 2010 Dec]; Available from: http://wiki.eclipse.org/GMF_Tutorial.

35. Foundation, T.E. *MOFScript* 2010 [cited 2010 Dec]; Available from: http://www.eclipse.org/gmt/mofscript/.

36. Oldevik, J., *MOFScript User Guide Version 0.8 (MOFScript v 1.3.6)*. 2009.

37. Yamada, S., J. Hishitani, and S. Osaki, *Software-reliability growth with a Weibull test-effort: a model and application.* Reliability, IEEE Transactions on, 1993. **42**(1): p. 100-106.

38. Peterson, J., *Petri Net Theory and the Modeling of Systems*. 1981: Prentice Hall PTR.

39. Díaz-Antón, G. and M.A. Pérez, *TOWARDS AN ONTOLOGY OF LMS A Conceptual Framework*, in *8th International Conference on Enterprise Information Systems*, Y. Manolopoulos and J. Filipe, Editors. 2006: Paphos - Cyprus.

40. Heiyanthuduwage, S.R. and D.D. Karunaratne, *A Learner Oriented Ontology of Metadata to Improve Effectiveness of Learning Management Systems.* International Journal of the Computer, the internet and management, 2006. **14**.

41. Srimathi, H., *Knowledge Representation of LMS using Ontology*. International Journal of Computer Applications, 2010. **Volume 6– No.3**.

42. Aßmann, U., S. Zschaler, and G. Wagner, *Ontologies, Meta-models, and the Model-Driven Paradigm*, in *Ontologies for Software Engineering and Software Technology*, C. Calero, F. Ruiz, and M. Piattini, Editors. 2006, Springer Berlin Heidelberg. p. 249-273.

43. Guizzardi, G., *Ontological foundations for structural conceptual models*. 2005: Enschede. p. 416.

44. Devedzi\, V. and \#263, *Understanding ontological engineering.* Commun. ACM, 2002. **45**(4): p. 136-144.

45. García-Díaz, V., et al., *TALISMAN MDE: Mixing MDE principles.* Journal of Systems and Software, 2010. **83**(7): p. 1179-1191.

46. García-Magariño, I., R. Fuentes-Fernández, and J.J. Gómez-Sanz, *Guideline for the definition of EMF metamodels using an Entity-Relationship approach.* Information and Software Technology, 2009. **51**(8): p. 1217-1230.

47. Pardillo, J. and C. Cachero, *Domain-specific language modelling with UML profiles by decoupling abstract and concrete syntaxes.* Journal of Systems and Software, 2010. **83**(12): p. 2591-2606.

48. Foundation, T.E. *Eclipse Modeling Framework Project (EMF).* 2010 [cited 2010 November]; Available from: http://www.eclipse.org/modeling/emf/.

49. Laforcade, P., *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, 2010. **21**(6): p. 347-358.

50. Zdun, U., *A DSL toolkit for deferring architectural decisions in DSL-based software design.* Inf. Softw. Technol., 2010. **52**(7): p. 733-748.

51. Alvares, M., *Diseño y Construcción de lenguajes específicos de dominio asistidos por ontologías*, in *Informatica*. 2010, Universidad de Oviedo: Oviedo. p. 90.

52. Garcia, A., *CADAM: Construcción de Aplicaciones Web, para el diseño de Asignaturas basada en Modelos*, in *Informática*. 2010, Universdiad de Oviedo: Oviedo. p. 250.

53. Wile, D., *Lessons learned from real DSL experiments.* Science of Computer Programming, 2004. **51**(3): p. 265-290.

54. Tratt, L., *Model transformations in MT.* Science of Computer Programming, 2007. **68**(3): p. 196-213.

55. Jouault, F., et al., *ATL: A model transformation tool.* Science of Computer Programming, 2008. **72**(1-2): p. 31-39.

56. Stephen, J.M., et al., *MDA Distilled: Principles of Model-Driven Architecture* 2004: Addison Wesley. 176.

57. Corporation, E. *Eclipse Modeling Project*. 2011; Available from: http://www.eclipse.org/modeling/.