Herramienta de modelado de dominio específico (DSM) para la creación de módulos en sistemas de gestión del aprendizaje (LMS)*

Modeling tool specific domain (DSM) modules for building on learning management system (LMS)

Artículo recibido: marzo de 2011
Artículo aprobado: julio de 2011

Carlos Enrique Montenegro Marín[**]
Paulo Alonso Gaona García[***]
Juan Manuel Cueva Lovelle[****]
Oscar Sajuan Martínez[*****]

Abstract

The general objective is to make domain-specific modeling for the construction of modules of learning management systems (LMS) platform independent. For this, the start point is a metamodel for the contruction a domain specific language (DSL), that with model-driven engineering (MDE) and applying the appropriate transformations are achieved from a independent model platform, deploy this model with LMS modules in moodle.

Key words

Model-Driven Architecture (MDA), metamodel, ecore meta-metamodel, Eclipse Modeling Framework (EMF), Meta Object Facility (MOF), Graphical Modeling Framework (GMF), moodle.

* Este artículo hace parte de los resultados que el grupo de investigación OOTLab de la Universidad de Oviedo (España) ha desarrollado en el área de Ingeniería Dirigida por Modelos.

** Candidato a doctor en ingeniería de la Universidad de Oviedo, España. Profesor de la Universidad Distrital Francisco José de Caldas, Bogotá, D. C. Correo electrónico: cemontenegrom@udistrital.edu.co

*** Candidato a doctor en ingeniería de la Universidad Universidad Pontificia de Salamanca, España. Profesor de la Universidad Distrital Francisco José de Caldas. Correo electrónico: pagaonag@udistrital.edu.co

**** Director del Departamento de Informática de la Universidad de Oviedo, España. Correo electrónico: cueva@uniovi.es

***** Profesor del Departamento de Informática de la Universidad de Oviedo, España. Correo electrónico: osanjuan@uniovi.es

Introducción

MDE (Model-Driven Engineering) o ingeniería dirigida por modelos es una propuesta que plantea el uso de modelos como eje fundamental en todo el ciclo de vida de un proyecto de *software*. Una de las aproximaciones de este planteamiento es MDA (Model-Driven Architecture), una de las propuestas de la OMG (Object Manage Group), la cual establece una serie de tecnologías para la construcción de *software*, bajo el esquema de MDE. La idea fundamental de MDA se basa en la transformación de modelos, a través de conversiones y partiendo de lo general (requerimientos) a lo particular (despliegue de la solución). Bajo este panorama, en este artículo se mostrará un metamodelo bajo *Ecore*; después, la construcción de una herramienta DSL (Domain Specific Languaje) gráfica, basada en ese metamodelo y con apoyo en Eclipse Modeling Framework (EMF), Meta Object Facility (MOF) y Graphical Modeling Framework (GMF). Finalmente, se mostrará como transformar un modelo construido con el DSL a código desplegable, en este caso, para la plataforma *moodle*, mediante *MOFScript*. Al terminar el artículo se realizará una discusión de las pruebas de la herramienta.

Ingeniería Dirigida por Modelos (MDE)

La ingeniería dirigida por modelos surge como la respuesta de la ingeniería de *software* a la industrialización del desarrollo del *software*, MDA (Model-Driven Architecture) es la propuesta de la OMG (Object Management Group), que centra sus esfuerzos en reconocer que la interoperabilidad es fundamental y también que el desarrollo de modelos permite el perfeccionamiento de otros, que luego, al unirse, proveerían la solución a todo un sistema e independizarían el desarrollo de las tecnologías empleadas (Mellor, S., Scott, K., Uhl, A. & Weise, D., 2004).

Se ha visto como el nivel de abstracción de los diferentes lenguajes de programación se ha incrementado durante décadas, un hecho que revela su evolución; así, por ejemplo, antes se hablaba de lenguajes en binario, luego del ensamblador, mas adelante de los lenguajes procedimentales, posteriormente de la orientación a objetos (esta ultima, apoyada en diversos artefactos como UML) y, ahora, hablamos de construcción de esos modelos como UML, a esto se le ha llamado MDA. Y, aunque el nivel de abstracción aumente, éste se acerca más al dominio específico sobre el cual se trabajará. De esta forma, cualquier persona podría llegar a realizar un modelo, empleando los conocimientos de su contexto, y, por ende, la complejidad se trasladaría a ver cómo ese modelo se convierte en una solución desplegable y funcional sobre cualquier tecnología específica.

Una buena interpretación de lo que es un modelo, un metamodelo y un meta-metamodelo se encuentra en los estudios de García Díaz y Cueva Lovello (2010), allí, un metamodelo son aquellas herramientas que permiten la creación de un modelo, es decir, la descripción de uno o varios elementos del dominio o mundo real; finalmente, el meta-metamodelo describe a esos metamodelos planteados, generando un grado de abstracción supremamente alto, en el cual coinciden todos los modelos (Figura 1).

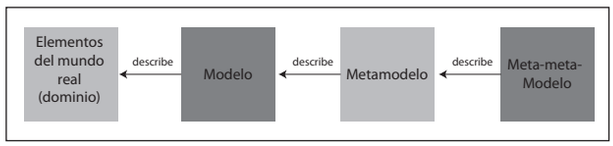

Figura 1. Modelo, metamodelo y meta-metamodelo

Según García y Cueva (2010) hay cuatro espacios de modelamiento: 1) los niveles base *M0*, que son los elementos del mundo real o módulos de *moodle*, para nuestro caso, 2) los niveles *M1*, que son los programas informáticos o la plataforma *moodle*, entre otros, 3) los niveles *M2,* constituidos por la especificación de UML, ODM, Java, C#, XML, u otras, y que –para nuestro caso– será el metamodelo a construir, y 4) los niveles *M3*, que son los de mayor abstracción. Básicamente, hay dos meta-metamodelos (M3) planteados: por un lado está MOF y por el otro EBNF. La idea de generar estos niveles de abstracción tan altos, es proveer un mecanismo común que permita, a través de la transformación de un modelo a otro, la interoperabili-

dad de los sistemas. Así, una definición explícita de lo que es MDA, la encontramos a continuación:

> The Model Driven Architecture (MDA) is a framework for software development defined by the Object Management Group (OMG). Key to MDA is the importance of models in the software development process. Within MDA the software development process is driven by the activity of modeling your software system. (Kleppe, A., Warmer, J. & Bast, W., 2003)

MDA se centra en la generación de modelos, por ello su ciclo de vida se enfoca hacia la creación de éstos, y, por lo tanto, no modifica para nada el ciclo tradicional de vida de desarrollo de *software*, tan solo cambia la generación de artefactos, por ejemplo: el paso de un PIM (Modelo Independiente de la Plataforma) a PSM (Modelo Específico de la Plataforma) y luego a código se realiza de manera automática y recibe el nombre de transformaciones a partir de los modelos generados; de esta forma, si hay un cambio en el sistema, se realiza en el modelo más global, y para desplegar la solución se vuelve a repetir el proceso de trasformaciones entre modelos. En la figura 2 se muestra el ciclo de vida del desarrollo de *software* con MDA.

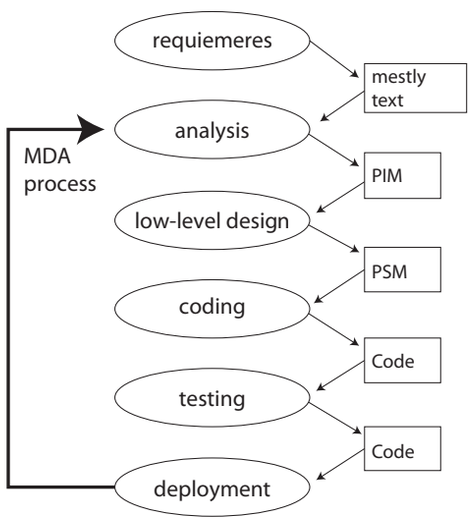

Figura 2. Ciclo de vida del desarrollo de software con MDA

Fuente: Kleppe, A., et al. (2003).

El artefacto que reúne los requerimientos del sistema se llama CIM (Modelo Independiente de la Computación); además, el resultado de modelar este sistema es un PIM que se hace a través de un DSL construido previamente. Este DSL genera, mediante un proceso de transformación, un PSM que con otra transformación se convierte en código desplegable o modelo específico de implementación (ISM) (ver figura 3). En este trabajo se expondrá la creación de ese DSL basado en un metamodelo para la creación de módulos de comunicación para un LMS, su posterior modelamiento con la herramienta DSL creada, luego la transformación a código compatible con la plataforma *moodle*, para finalmente hacer el despliegue y pruebas de los módulos creados.

Ingeniería Dirigida por Modelos (MDE) con Eclipse

Meta-metamodelo

El meta-metamodelo que se empleara será *Ecore*, éste se encuentra en el paquete org.eclipse.emf.ecore y es la especificación más alta que existe en la pirámide de los modelos (M3), sobre ella se construirá el metamodelo del proyecto. La especificación de *Ecore* se puede consultar en Others (2006).

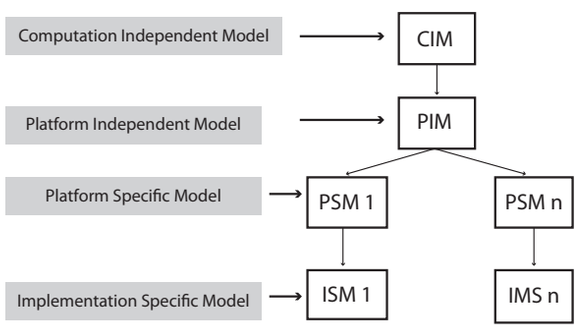

Figura 3. Modelos generados en MDA

Metamodelo

Para la utilización de *Ecore* en Eclipse es necesario tener instalado el *plugin* de EMF (Eclipse Modeling Framework), éste provee básicamente dos herramientas para construir un modelo basado en *Ecore*: el *Ecore Model*, que es un editor manual que funciona en un estilo de árbol de navegación para la creación del modelo basado en *Ecore* y el

Ecore Diagram, un editor gráfico similar a las herramientas gráficas para la creación de diagramas de clases UML. Cualquiera de las dos formas que se utilice para crear el diagrama basado en *Ecore*, genera un fichero XMI (Group, 2007) (XML Metadata Interchange), que es una especificación para el intercambio de diagramas, en nuestro caso, utilizaremos *Ecore Model.*

Para la construcción del metamodelo, en este trabajo se ha tomado únicamente lo concerniente a los módulos de comunicación que tiene un LMS, para ello hemos tomado sólo la parte que nos interesa del metamodelo LMS (planteado en Montenegro, C., Cueva, J. & Sanjuán, O., 2011), éste está construido sobre *Ecore* y se puede ver en la figura 4. El metamodelo, en esencia, posee seis módulos llamados *EClass* en *Ecore*, que son: *Forum*, *Chat*, *Wiki*, *Annoncement*, *News* y *Note*; además, todos están relacionados con la *EClass Communications*, pues el módulo *Communications* puede tener cero o muchos módulos de los anteriores. Igualmente, esta *EClass Communications* hereda de la *EClass Tools*, incluso todas las *EClass* están contenidas en la *EClass LMSModel*, esta relación es obligatoria en todo metamodelo pues es quien representará el contenedor de *EClasses*, y allí será en donde despleguemos nuestros módulos a modelar, o, mejor dicho, donde estarán contenidos. Así mismo, es necesario aclara que la *EClass LMSModel* solo almacenará cero o una *EClass Communications*, esto tiene sentido pues no puede haber más de un modulo *Communications* en un LMS, ya que éste, a su vez, almacena las herramientas de *Communications* (tal como *Forum*, *Chat*, *Wiki*, *Annoncement*, *News* y *Note*). Por último, cada *EClass* tiene sus propios atributos que la compone (Figura 4).

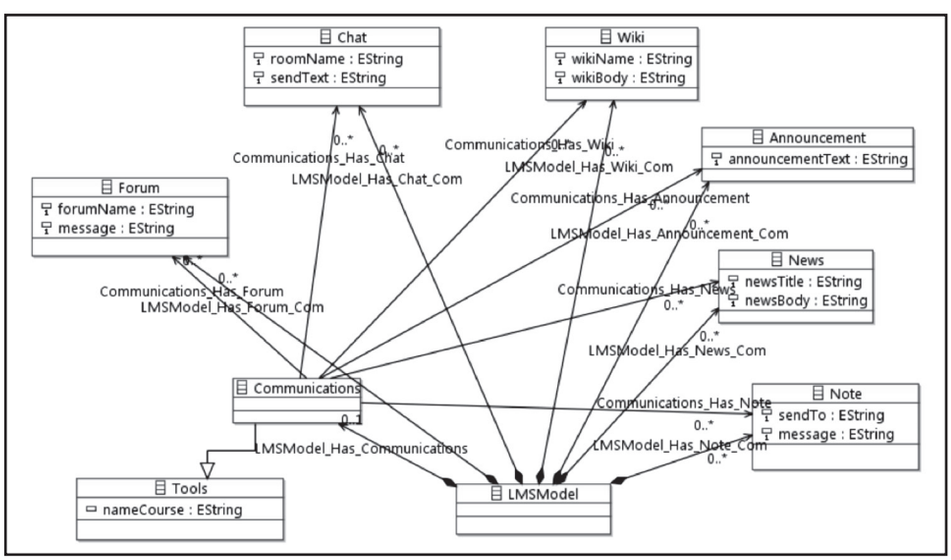

Figura 4. Metamodelo LMS de módulos de comulación

Construcción del editor para el modelo o DSL

Como se está trabajando bajo Eclipse, para esta etapa se empleó *EMF Tooling* (Graphical Modeling Framework Tooling) que hace parte del proyecto GMP (Graphical Modeling Project) (Foundation, 2010). El proceso para la construcción del DSL Gráfico se visualiza en la figura 5, además, una excelente guía para el desarrollo de herramientas de este tipo la encontramos en el estudio de Budinsky, F., Steinberg, J., Merks, E., Ellersick, R. & Gorse, T. (2009) y en los tutoriales que ofrece su web oficial (Foundation, 2010).

Según el *dashboard* de la figura 5, lo primero que se debe crear es el *Domain Model*, este corresponde al LMS metamodelo, descrito en la sección anterior. El siguiente paso es crear el *Domain Gen Model*, que es un modelo que permite transformar automáticamente el modelo *Ecore* a código fuente. El código se genera aplicando patrones de transformación. El resultado es un conjunto de clases *java*, que serán utilizadas más adelante en la herramienta DSL. Es muy importante en las propiedades del *Domain Gen Model*, llenar el campo *Base Package* con el nombre del paquete en minúscula, como lo muestra la figura 6.

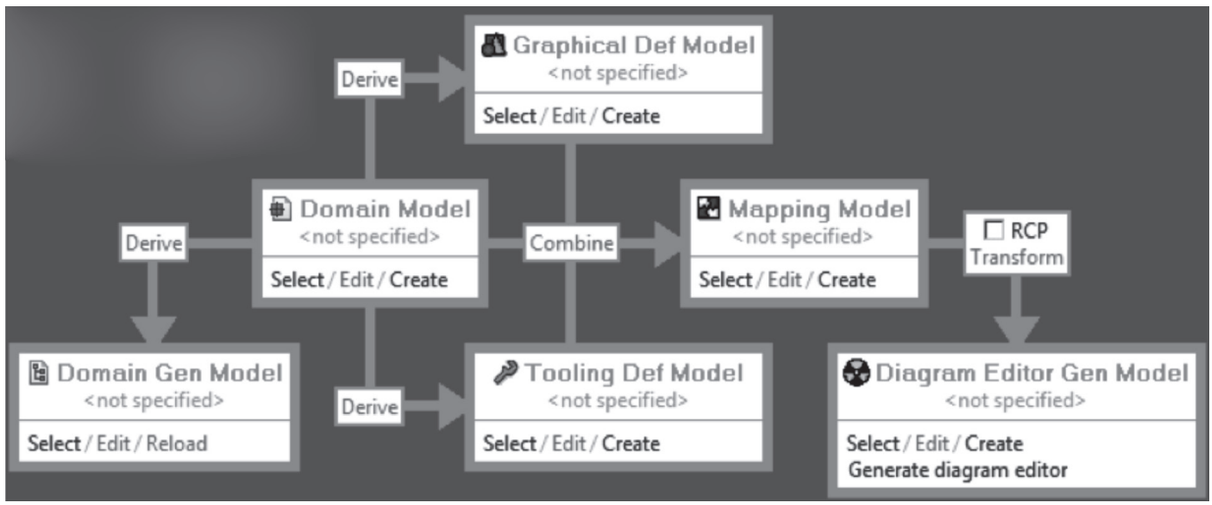

Figura 5. GMF Overview o Dashboard

Fuente: Foundation (2010)

Luego de que el *Domain Gen Model* esté creado, se crea el *Graphical Def Model*, este es usado para definir las figuras, nodos, conexiones, etc., que serán mostradas en nuestro diagrama. Para la creación del *Graphical Def Model*, la hoja de ayuda lanzará un asistente para la creación del *Simple Graphical Definition Model*. Seleccione la carpeta que contiene el modelo en el proyecto, el nombre del fichero será "modellms.gmfgraph", luego elija el modelo *ecore* o el metamodelo LMS "modellms.ecore", y, en la siguiente página del asistente, escoja la *EClass* a mapear, esta es la clase que contiene todos los elementos para nuestro caso "LMSModel".

En la última página del asistente, seleccione la mínima cantidad de opciones elementos (*element*), conexiones (*link*) y etiquetas (*label*) para la herramienta DSL. Por el momento, solo estamos interesados en obtener un conjunto mínimo de elementos para validar el modelo, en este caso las herramientas de *Communications* (Figura 4), la siguiente figura muestra los elementos, conexiones y etiquetas seleccionadas.

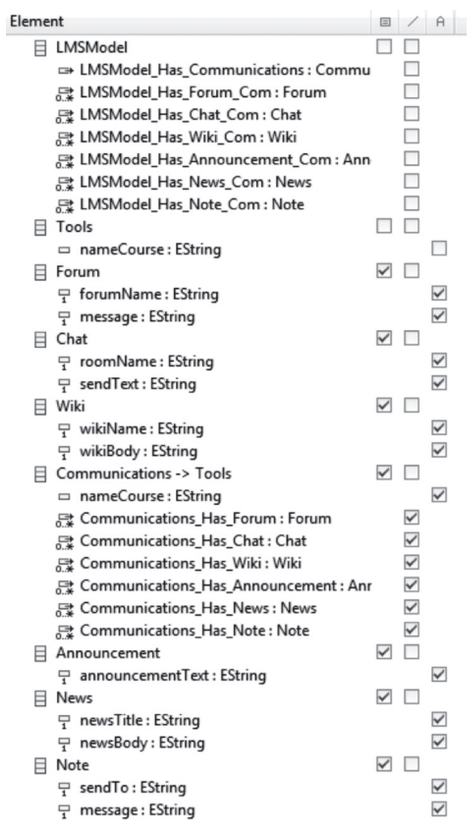

Figura 7. Graphical Definition Model: elementos, conexiones y etiquetas seleccionadas

Fuente: elaboración propia

El resultado es un fichero con la siguiente estructura: un Canvas (lienzo) en la raíz con una galería de figuras base que contiene elementos de Rectángulos, Etiquetas y Conexiones de Polilineas. Estas son usadas por el correspondiente elemento Nodo, Etiqueta del diagrama y Conexión para representar los temas del *domain model*. Estos elementos pueden ser configurados. La siguiente figura muestra algunos elementos de "modellms.gmfgraph".

Figura 8. Algunos elementos de Graphical Definition Model

Fuente: elaboración propia

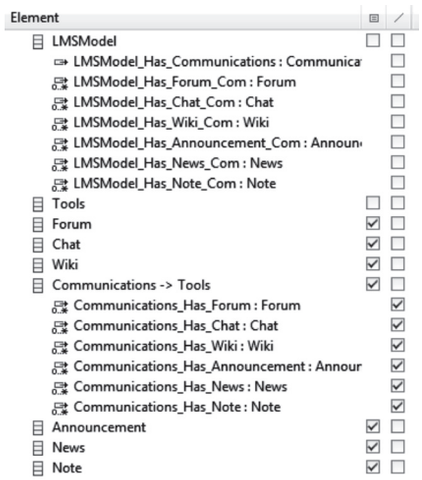

Figura 9. Elementos de Tooling Definition Model: elementos y conexiones seleccionadas

Fuente: elaboración propia

El siguiente paso en el *dashboard* es el *Tooling Def Model*, este es usado para especificar la paleta (*Pallete*) de herramientas de creación, acciones, etc., para los elementos gráficos. La hoja de ayuda nos guiará en un proceso muy similar al anterior, para comenzar con la creación de un *Simple Tooling Definition Model*. De hecho, los dos pasos son muy similares, ya que el dominio del modelo "modellms.ecore" será cargado y se escogerán las posibles herramientas que se necesitan, esto se muestra en la figura 9.

Si observamos el modelo provisto por nosotros, vemos que hay un elemento en el nivel superior (*Tool Registry*), en el que encontramos una paleta (*Palette*). La *Palette* contiene un *Tools Group* con elementos de tipo *Creation Tool* para los nodos tema y conexiones para elementos de subtemas que fueron identificados por el asistente. Más adelante, reorganizaremos y modificaremos estos elementos, por el momento, lo dejaremos por defecto y realizaremos la definición del *mapping*. El resultado es un fichero con el modelo proporcionado por nosotros, cuyos elementos pueden ser reorganizados. La figura 10, muestra el resultado.

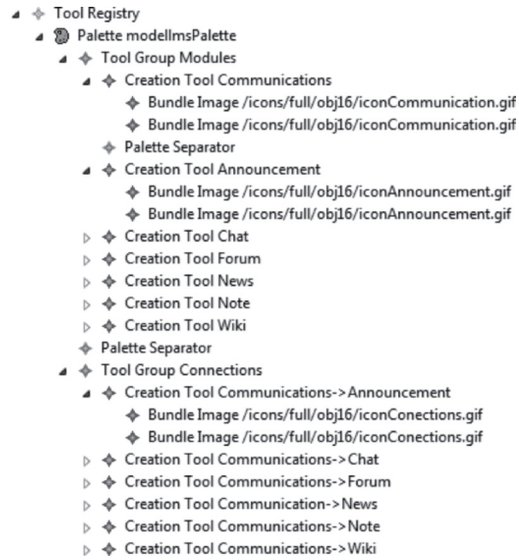

Figura 10. Elementos de Tooling Definition Model

Fuente: elaboración propia

El *mapping model* es el siguiente paso en el *dashboard* y combina los tres modelos: el *Domain Model*, el *Graphical Def Model* y el *Tooling Def Model*. Continuado con nuestra hoja de ayuda se siguen las instrucciones para poner en marcha el asisten-

te de *Guide Mapping Model Creation*. Seleccione la carpeta que contiene al modelo, el nombre del archivo para nosotros será *modellms.gmfmap* y elija los modelos *modellms.ecore*, *modellms.gmfgraph* y *modellms.gmftool*. En la siguiente página seleccione *LMSModel* como el elemento raíz, luego en aceptar el *modellms Palette* del tooling model seleccionado, después aceptar el *modellms diagram canvas*, seleccionándolo, y hacer clic en finalizar. El resultado se muestra en la figura 11.

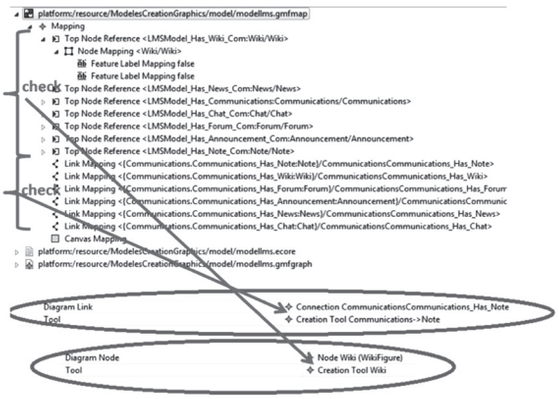

Figura 11. Mapping del modelo y propiedades

Fuente: elaboración propia

Nota: Los elementos del *mapping* tienen problemas (Foundation, 2007). Por ejemplo, uno de los pasos que tiene que hacerse de forma manual es el *Labels Mapping* de la *graphical definition*. El resultado debe ser similar al de la figura anterior. Mientras que en la vista de propiedades de cada elemento se debe verificar el correcto *mapping* del *Diagram Node* con *Tool* y *Diagram Link* con *Tool* (ver Figura 11)El último paso es la creación del *Diagram Editor Gen Model*. Allí, el modelo generador *modellms.gmfgen* establece las propiedades para la generación de código, similar al *EMF genmodel*. Para conseguir esto, haga clic derecho en el fichero del *mapping* "*modellms.gmfmap*" y seleccione *Create generator model*. Cuando se le solicite, mantenga el nombre predeterminado *modellms.gmfgen*, haga clic derecho en el fichero *modellms.gmfgen* y seleccione *Generate diagram code* para continuar. Si todo va bien, usted verá un cuadro de diálogo con el mensaje "*Code generation completed successfully*". Cierre el cuadro de dialogo, y observe el nuevo plugin *ModelesCreationGraphics.diagram* en su *workspace*.

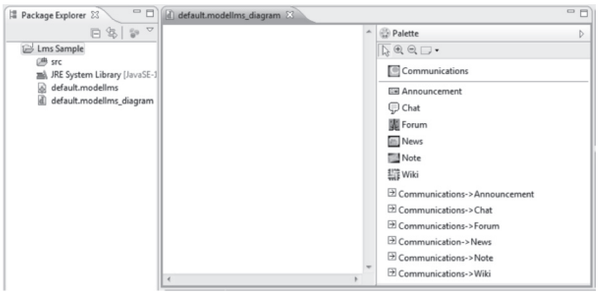

Figura 12. Herramienta DSL para modelar módulos en plataformas LMS

Fuente: elaboración propia

Ahora que hemos generado el *plugin* necesario para nuestra herramienta DSL, que además está en ejecución en un nuevo *workspace*, se la puede probar utilizándola. La configuración por defecto es de una nueva *Eclipse Application* en tiempo de ejecución y debería funcionar bien, mientras que existe una opción de ejecutar una configuración mínima que incluye solo los *plugins* generados en la ejecución y sus dependencias de una configuración de org.eclipse.platform.ide.

Por último, hablaremos de la creación de un proyecto vacio (*File->New->Java Project*). En la sección *Examples* aparecerá una nueva opción llamada *Modellms Diagram*, que creará su nuevo diagrama. Esta es la herramienta DSL para modelar los módulos de una plataforma LMS, y se muestra en la figura 12.

Modelo

En la sección anterior se describió cómo crear la herramienta DSL para modelar módulos de un LMS, ahora vamos a ver cómo utilizar dicha herramienta para poder obtener un modelo de ella. El diagrama que se obtenga como resultado de emplear el editor de DSL, tendrá asociado un fichero XMI(Group, 2007) que basará su sintaxis en el metamodelo creado.

El funcionamiento de la herramienta es muy sencillo. Para crear los módulos basta con arrastrar los nodos de la *Pallete* al área de trabajo, rellenar los campos y conectarlos respetando las siguientes reglas:

- Un cursos sólo puede tener un modulo de *Communications*.
- El modulo de *Communications* tiene cero o muchos: *Announcements*, *Chat*, *Forum*, *News*, *Note* y *Wiki*.

La herramienta valida los siguientes casos:

- Solo permite un modulo de *Communications*.
- Los enlaces sólo pueden corresponder entre el nodo *Communications* y su respectiva herramienta, por ejemplo, *Communications -> Chat*, sólo sirve para conectar el nodo *Communications* con el nodo *Chat*. La herramienta no permite utilizarlo en otro caso.
- Cuando se genere el código a desplegar, aquellos nodos que estén sueltos (sin conexión) no serán tenidos en cuenta para la creación del curso.

Es muy importante tener en cuenta que los elementos modelados con esta herramienta se desplegaran en un solo tema del curso, con lo cual es necesario considerar los nombre de los campos en cada elemento (Nodo o Modulo) como genéricos, para no tener que cambiar el modelo cada vez que se despliegue sobre el curso. En la figura 13 se puede visualizar un ejemplo, esta figura también muestra la relación de cada nodo y conexión con el XMI que genera la herramienta.

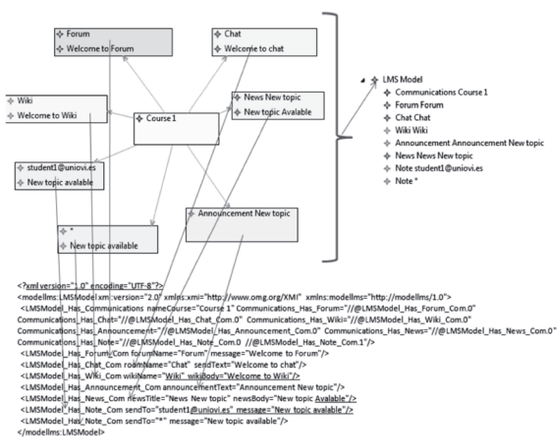

Figura 13. Ejemplo de un modelo en nuestra herramienta DSL y su correspondencia con XMI

Fuente: elaboración propia

Observaciones a tener en cuenta en el modelado:

- Si se desea enviar una *Note* a todos los inscritos en un curso, en el campo *Send To* debe ir un asterisco (*).
- Si se desea enviar una *Note* a un solo inscrito en un curso, en el campo *Send To* debe ir el correo completo con el que la persona se inscribió en la plataforma virtual. Si este correo no corresponde con el de la persona el mensaje no será enviado.
- Recuerde diligenciar todos los campos en los nodos, pues, aquellos campos en los cuales el usuario no digite información, serán tomados en blanco y la herramienta no procesará esos nodos.
- Todos los campos aparecen con información por defecto, pero esta información sólo es de carácter orientativo, es decir, no implica información en los nodos.

En resumen, en esta sección se habló acerca de cómo crear un herramienta DSL gráfica (M1), basada en un metamodelo generando (M2) y que, a su vez, se basa en el meta-meta modelo *Ecore* (M3). El siguiente paso es convertir este nuevo modelo que se obtuvo con la herramienta DSL y hacerle una transformación a código (M0), esta idea se explicará en la siguiente sección.

Proceso de generación de código

En este último paso, existen varias tecnologías que se integran a Eclipse, como son *Acceleo* (Obeo, 2010), *Jet* (Foundation, 2010), *Xpand* (Foundation, 2010) y *MOFScript* (Foundation, 2010), todas emplean el mismo principio, la creación de reglas de transformación basándose en un metamodelo. Estas reglas, serán aplicadas al modelo para generar código en el lenguaje deseado. Además, este tipo de tecnologías reciben el nombre M2T o *Model to Text*. La transformación de un modelo a otro se llama M2M o *Model to Model*; algunas herramientas como ATL (Foundation, 2010) u *OperationalQVT*, realizan esta tarea. Para nuestro caso se empleará *MOFScript* (Foundation, 2010).

MOFScript es un lenguaje basado en reglas, presentado por la OMG para realizar transformaciones de modelo a texto (M2T). Se puede instalar como un *plugin* de eclipse y tanto su sintaxis como su uso resultan bastante sencillos. Un excelente manual para el uso de *MOFScript* es Oldevik (2009).

A continuación mostraremos los pasos a seguir para generar código a partir del modelo LMS, creado con nuestra herramienta DSL. Para poder comprender el contenido del fichero *MOFScript* es necesario entender cómo se desarrolla un módulo para *moodle* y para ello las mejores guía las encontramos en su página web (Moodle, 2011), vínculo *Develoment*, o en los trabajos de Ivorra (2009) y González (2009). Nosotros no ahondaremos en este tema pues no es el objetivo del artículo.

Para transformar el modelo que se obtiene con la herramienta DSL creada, hay que tener en cuenta que este modelo estará basado en el metamodelo LMS, construido anteriormente. *MOFScript* tiene su propia sintaxis que pude ser consultada en Oldevik (2009).

Lo primero que se debe hacer es definir el modelo de entrada a las plantillas *MOFScript*, para ello se debe declarar la transformación con *texttransformation*, darle un nombre a la trasformación y enviarle como parámetro de entrada el nombre del metamodelo con extensión *.ecore*, para nuestro caso, *modellms*. También se debe dar un nombre al modelo enviado, el nuestro se llamó *mlms*. Esto se puede visualizar en el siguiente fragmento de código (Figura 14).

```
texttransformation  modellmsTransformationMoodle (in mlms:»modellms») {
```

Figura 14. Declaración de una transformación en MOFScript

La funciona principal o punto de partida para las transformaciones en *MOFScript* están contendidas en la función "*main*" que se declara como se muestra a continuación:

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

Figura 15. Declaración de una transformación en MOFScript

Para crear un nuevo fichero se utiliza la instrucción *file* como se ve a continuación:

```
file (‹mod_form.php›);
```

Figura 16. Creación de fichero mod_form.php en MOFScript para moodle

```
        module::encabezado()

        {

‹<?php

require_once($CFG->dirroot.›+quote+›/course/moodleform_mod.php›

+quote+›);

require_once($CFG->dirroot.›+quote+›/course/lib.php›+quote+›);

require_once(‹+quote+›../config.php›+quote+›);

class mod_plantilladsl_mod_form extends

 moodleform_mod {

 function definition() {‹

        }
```

Figura 17. Función encabezado en MOFScript para moodle

Y para escribir en ese fichero las cadenas deben ir entre el carácter " ‹ ". Para la declaración de funciones, primero se debe anteponer la palabra *module*, seguida del operados "::" y luego el código correspondien-

te. A continuación se muestran la función encabezado para las plantillas de transformación de *moodle*.

Para invocar a las funciones sencillamente se las llama en donde sean necesarias. Al igual que en el metamodelo, todas las *EClass* están contenidas en una sola llamada *LMSModel*. Así mismo, esta relación se representa con una conexión, a continuación se valida si en el modelo existe un módulo (*EClass*) *Communications* que se relaciona con *LMSModel* a través de la relación *LMSModel_Has_Communications*, tal y como se muestra a continuación:

```
        encabezado();

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

Figura 18. Función encabezado en MOFScript para moodle

Para contar cuántos módulos (*EClass*) hay en los modelos se utiliza la función "*size*", provista por las conexiones; esto se debe hacer por cada nodo. A continuación se muestra un ejemplo para los módulos *Forum*.

```
self.LMSModel_Has_Forum_Com.size();
```

Figura 19. Contador de módulos Forum dentro de un modelo

Posteriormente se recorren todos los módulos que están dentro del modelo y conectados al modulo *Communications*, y se valida si en éstos los campos están rellenados, si no lo están, incrementa en 1 un contador de los módulos que están conectados pero no tienen información, si están rellenados se imprime una nueva línea en el archivo acompañado de dos espacios de tabulador, la invocación a la función correspondiente en *moodle* con parámetros de entrada, los nombres ingresados en el modelo y finalmente se incrementa un contador de los módulos creados correctamente. A continuación se muestra el código correspondiente para el módulo *Forum* en *moodle*; esto se debe hacer por cada módulo.

```
elf.LMSModel_Has_Communications.Communica-
tions_Has_Forum->forEach(forum:mlms.Forum)

{

        if (forum.forumName = null or forum.mes-
sage = null){

        numForumSN = numForumSN+1;

        }

        else{

        newline(1);

        tab(2);

        ‹def_forum_dsl (‹+quote+forum.
forumName+quote+›,›+quote+forum.
message+quote+›)›;

        numForumY = numForumY+1;

        }

}
```

Figura 20. Recorrido para los módulos Forum en MOScript para moodle

Finalmente, aplicando las tecnologías de transformación sobre el modelo creado se obtiene el código en el formato definido para su despliegue sobre *moodle*.

Pruebas y validación de la herramienta DSL

El objetivo de esta sección es validar el metamodelo planteado y comprobar si es verdadera la hipótesis: "Las herramientas para diseño, basado en modelos, reducen el tiempo de desarrollo en los proyectos". Las pruebas medirán el tiempo y esfuerzo (usabilidad) de los usuarios para crear exactamente los mismos módulos en *moodle* y la herramienta DSL creada, además, cada uno de los usuario creará 5 temas con el siguiente orden establecido: 1 *Chat*, 1 *Forum*, 1 *Wiki*, 1 *Announcement*,

1 *News* y 1 *Note*. Para todas las personas inscritas en ese curso la información de cada modulo será la misma, en todos los casos, y los valores se han normalizado al mayor.

De acuerdo con Yamada, S., Hishitani, J. y Osaki, S., (1993) podemos definir el esfuerzo como la "cantidad de ocasiones en las que usuario selecciona o ingresa algún tipo de información al sistema". La medición de tiempos y esfuerzo se realizó para los dos sistemas (la herramienta DSL y la plataforma LMS moodle) y se midió: tiempo y esfuerzo en la creación de cada herramienta por tema; tiempo y esfuerzo en la creación de todas las herramientas que conforman un tema: finalmente, tiempo y esfuerzo total en la creación de los cinco temas acompañados de sus herramientas. En las pruebas hemos considerado que el envío de las notas (mensajes) a los inscritos en el curso son una herramienta del tema. Para poder comparar los tiempos, es necesario que todos los módulos sean creados en el mismo orden para los dos sistemas.

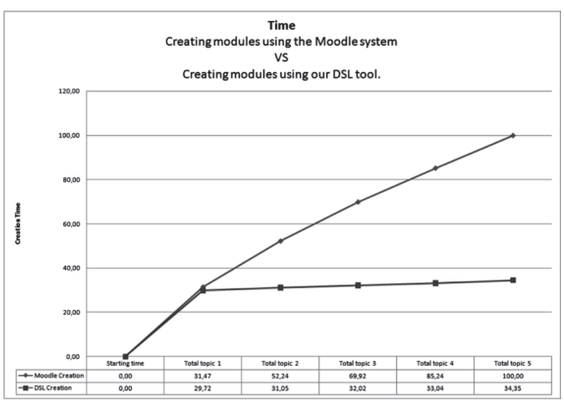

Figura 21. Comparación de tiempos en la creación de módulos en moodle vs. la herramienta DSL creada

Fuente: elaboración propia

La Figura 20 muestra los tiempos promedio normalizados por tema, allí se ve que la diferencia, al final de crear el primer tema, es tan solo 1.75% mayor, cuando se crean los módulos en *moodle* que con la herramienta DSL; sin embargo, esta diferencia empieza a crecer desde el segundo tema en adelante, hasta llegar a un 66.65% más rápido al crear los módulos con la herramienta DSL que con *moodle*. La Figura 21 muestra un comportamiento similar pero, midiendo el esfuerzo promedio normalizado, la diferencia al final de crear el primer tema es de 12.68% menos esfuerzo con *moodle* que con la herramienta DSL, no obstante, desde el segundo tema en adelante, el esfuerzo en la creación de módulos con DSL es, por lo menos, 63% menor que al hacerlo en *moodle*. El tiempo promedio total y esfuerzo total normalizados en la creación de 5 temas, cada uno con sus respectivos módulos, es por lo menos 65% más eficiente con la herramienta DSL que al hacerlo en *moodle*.

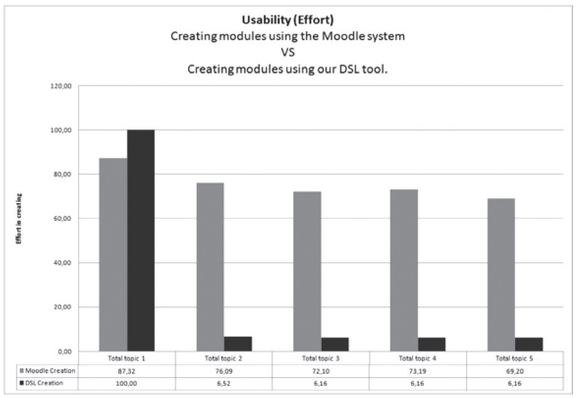

Figura 22. Comparación de esfuerzo en la creación de módulos en moodle vs. la herramienta DSL creada

Fuente: elaboración propia

Las anteriores pruebas, demuestran que al trabajar con modelos se disminuye el tiempo y esfuerzo del usuario en la creación de módulos para una plataforma LMS, en este caso *moodle*. Igualmente, el metamodelo funciona, ya que no se generó ningún fallo en la creación de los módulos con la herramienta DSL, que tiene como base fundamental el metamodelo LMS planteado.

Bibliografía

Budinsky, F., Steinberg, J., Merks, E., Ellersick, R. & Gorse, T. (2009). *EMF: Eclipse Modeling Framework*. USA: Addison-Wesley.

Foundation, T. E. (2007). "bug in map generation of the GMF tutorial". En *Retrieved* [en línea]. Disponible en: https://bugs.eclipse.org/bugs/show_bug.cgi?id=189410 [consultado en enero de 2011].

Foundation, T. E. (2010). "ATL". En *Retrieved*, [en línea]. Disponible en: http://www.eclipse.org/atl/ [consultado en diciembre de 2010].

Foundation, T. E. (2010). "GMF Tutorial". En *Retrieved*, [en línea]. Disponible en: from http://wiki.eclipse.org/GMF_Tutorial [consultado en diciembre de 2010].

Foundation, T. E. (2010). "Graphical Modeling Project (GMP)". En *Retrieved*, [en línea]. Disponible en: http://www.eclipse.org/modeling/gmp/ [consultado en diciembre de 2010].

Foundation, T. E. (2010). "M2T-JET". En *Retrieved*, [en línea]. Disponible en: http://wiki.eclipse.org/M2T-JET [consultado en diciembre de 2010].

Foundation, T. E. (2010). "MOFScript". En *Retrieved*, [en línea]. Disponible en: http://www.eclipse.org/gmt/mofscript/ [consultado en diciembre de 2010].

Foundation, T. E. (2010). "XPand". En *Retrieved*, [en línea]. Disponible en: http://wiki.eclipse.org/Xpand [consultado en diciembre de 2010].

García Díaz, V. & Cueva Lovelle, J. (2010). *Ingeniería Dirigida por Modelos*. Oviedo.

González, A. (2009). *Guía de apoyo para el uso moodle 1.9.4 Usuario Desarrollador. Informática.* Oviedo: Universidad de Oviedo.

Ivorra, R. (2009). *Tutorial: Creación de un módulo actividad. Moodle (1.9.3)*.

Kleppe, A. Warmer, J. & Bast, W. (2003). *MDA Explained: The Model Driven Architecture™: Practice and Promise*. Addison Wesley.

Mellor, S., Scott, K., Uhl, A. & Weise, D. (2004). *MDA Distilled: Principles of Model-Driven Architecture.* Addison Wesley.

Montenegro, C., Cueva, J. & San Juan, O. (2011). "Generation of metamodel in ecore with start point in an ontology for learning management systems (LMS)". En *Journal of Web Engineering*.

Moodle. (2011). "Moodle". En *Retrieved*, [en línea]. Disponible en: http://moodle.org/ [consultado en febrero de 2011].

Obeo. (2010). "Acceleo". En *Retrieved*, [en línea]. Disponible en: http://www.eclipse.org/acceleo/ [consultado en diciembre de 2010].

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

O. M. G. (2007). *MOF 2.0/XMI Mapping, Version 2.1.1*. Object Management Group**:** 120.

Others, I. C. a. (2006, 2006). "Package org.eclipse.emf.ecore". En. *Retrieved*, [en línea]. Disponible en: http://download.eclipse.org/modeling/emf/emf/javadoc/2.6.0/org/eclipse/emf/ecore/package-summary.html [consultado en diciembre de 2010]-

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