Citación: Montenegro, C. E., Cueva, J.M., Sanjuán O., y Gaona P.A. (2010). Desarrollo de un lenguaje de dominio específico para sistemas de gestión de aprendizaje y su herramienta de implementación "KiwiDSM" mediante ingeniería dirigida por modelos. En: *Ingeniería, Vol. 15, No. 2.* 67-81

Report

Desarrollo de un lenguaje de dominio específico para sistemas de gestión de aprendizaje y su herramienta de implementación "KiwiDSM" mediante ingeniería dirigida por modelos

Developing a domain-specific language for learning management systems, and a corresponding implementation tool ("KiwiDSM") through model-driven engineering

Carlos Enrique Montenegro Marín
Universidad Distrital
Francisco José de Caldas
Facultad de Ingeniería
cemontenegrom@usitrital.edu.co

Juan Manuel Cueva Lovelle
Universidad de Oviedo
Departamento de Informática
cueva@uniovi.es

Óscar Sanjuán Martinez
Universidad de Oviedo
Departamento de Informática
osanjuan@uniovi.es

Paulo Alonso Gaona García
Universidad Distrital
Francisco José de Caldas
Facultad de Ingeniería
pagaonag@udistrital.edu.co

Resumen

El artículo presenta la creación de KiwiDSM: herramienta de lenguaje de dominio específico (DSL), que apoyada en ingeniería dirigida por modelos (MDE), permite modelar módulos que conforman un sistema de gestión del aprendizaje (LMS) en el área de comunicaciones; esta herramienta es independiente de la plataforma. La validación de la propuesta se realizó desplegando un modelo hecho con KiwiDSM sobre una plataforma LMS particular llamada *Atutor*. Las pruebas realizadas evidenciaron que al trabajar con MDE se reduce el tiempo y esfuerzo en la creación y despliegue de los módulos modelados sobre *Atutor* y que el metamodelo planteado es compatible con los requerimientos de dicho LMS.

Palabras clave: Lenguaje de Dominio Especifico (DSL), Ingeniería Dirigida por Modelos (MDE), Arquitectura Dirigida por Modelos (MDA), Transformación de Modelos, Sistema de gestión del Aprendizaje (LMS), XML Metadata Interchange (XMI), Modelo y Metamodelo.

Abstract

This article presents the creation of a domain-specific-language (DSL) tool called WikiDSM, which is supported by model-driven engineering (MDE) and allows modeling the different modules that conform learning management systems (LMS), particularly in the field of communications. Such a tool is platform-independent. The validation of the proposal was performed by deploying a model built with WikiDSM on Atutor platform. The tests showed that using MDE reduces the time and effort when creating and deploying the Atutor-modeled modules. Moreover, it is shown that the meta-model proposed is compatible with Atutor requirements.

Key words: Domain Specific Language (DSL), Model Driven Engineering (MDE), Model Driven Architecture (MDA), Transformations of Models, Learning Management System (LMS), XML Metadata Interchange (XMI), Model, Metamodel.

1. Introducción

Dentro de la gran gama de dificultades que existen en el área de los sistemas de gestión del aprendizaje (LMS), destaca el problemas de la interoperabilidad entre ellos como lo aseguran Grob y Moreno [1, 2], éste articulo presenta una solución al problema desde el enfoque de la ingeniería dirigida por modelos (MDE) y también demostrará mediante un caso práctico la hipótesis que al trabajar con MDA (Model Driven Architecture) el tiempo y esfuerzo en la generación de soluciones se reducen.

El enfoque a utilizar parte de desde la perspectiva de uso de materiales u objetos de aprendizaje para migración de contenidos bajo especificaciones y/o estándares y los problemas que aparecen de forma repetitiva durante el desarrollo de software; además en muchas ocasiones estos problemas pertenecen a un dominio concreto, para dar solución a estos problemas se puede utilizar un GPL (General Purpose Language) como es Java y C# o se puede recurrir a la utilización de un DSL (Domain Specific Language), que no es más que un lenguaje definido específicamente para resolver los problemas de un dominio concreto.

A partir del contexto de DSL se puede hablar de DSM (Modelado de Dominio Especifico) que tiene su origen en la existencia de muchos desarrollos de software similares para un mismo dominio, una parte común y una parte variable. La parte común podría desarrollarse utilizando las técnicas de desarrollo tradicionales y la parte variable podría desarrollarse utilizando un DSL y es bajo este contexto que se pretende dar solución al problema plateado.

El artículo se organiza de la siguiente forma: primero una pequeña descripción de la metodología utilizada y del problema, luego un estado del arte acerca de ingeniería dirigidas por modelos y lenguajes de dominio especifico para ubicar al lector en el contexto del trabajo, después se explicará mediante un caso práctico y enfocado al problema anteriormente descrito, como aplicar la ingeniería dirigida por modelos al contexto de los LMS, en éste sentido se utilizará metodológicamente para su desarrollo las etapas que define el proceso de despliegue de una solución bajo MDE (ver Figura 1), por ello se hablará acerca de la construcción de la herramienta KiwiDSM, la siguiente sección tratará la temática de el metamodelado con KiwiDSM, posteriormente se explicará el proceso de generación de código, después se hablará de las pruebas de despliegue realizadas, por último las conclusiones y referencias del trabajo.

2. Metodología

Metodológicamente hablando se han contemplado las siguientes etapas:

1. Recopilación de información: Generar una base documental que soporte la elaboración de la investigación en las áreas de: estándares de representación de contenidos para plataformas virtuales de aprendizaje (LMS), estructura de plataformas LMS, modelamiento de plataformas LMS, ontología y construcción de un metamodelo LMS, construcción de DSLs basados en metamodelos.
2. Desarrollo y elaboración de la propuesta: Una vez realizada toda la base documental se procederá al desarrollo y elaboración de la propuesta en el siguiente orden: modelado y construcción de una ontología para LMS, generación de un metamodelo y un

DSL, asistido por ontologías, aplicación de MDA y transformaciones de modelos.

3. Despliegue de la propuesta: Construir varios módulos LMS empleando el DSL creado y aplicar transformaciones de modelos hasta obtener su despliegue en una plataforma LMS.
4. Pruebas del contenido: Probar el funcionamiento del contenido desplegado en la plataforma LMS previamente seleccionada.

Se aclara que los resultados a esbozar en el presente artículo son los relacionados con el área de ingeniería dirigida por modelos en el contexto de los LMS.

3. Interoperabilidad entre plataformas LMS

Autores como Grob y Moreno [1, 2] han identificado un problema de interoperabilidad entre plataformas LMS, si este problema no existiera se podría migrar un curso que está desplegado sobre la plataforma LMS-A a la plataforma LMS-B manteniendo todos sus contenidos y funcionalidades, pero esto no es posible ya que cada LMS está construido bajo un estándar diferente o si están bajo el mismo estándar cada una hace su propia interpretación. En este artículo se plantea la solución a este problema desde la perspectiva de los modelos, para ello se aumentará el nivel de abstracción para obtener un *framework* independiente de la plataforma, con el cual se pueda modelar un LMS para su posterior despliegue en una u otra plataforma especifica.

Existen propuestas orientadas a trabajar éste problema con MDA como las de Grob, Moreno y Bizoòová [1-5] pero que no incluyen los módulos que componen a un LMS ni la utilización de un DSL, también existen otros planteamientos de solución como el de Muñoz [6] que se orientan a la utilización de servicios pero agrega el problema de la mantenibilidad de mas plataformas; en el presente estudio se propone una herramienta de lenguaje de dominio especifico, que empleando MDA desplegará módulos en una plataforma LMS especifica como es Atutor [24].

4. Ingeniería Dirigida Por Modelos (MDE)

La ingeniería dirigida por modelos surge como la respuesta de la ingeniería de software a la industrialización del desarrollo de software, MDA es la propuesta de la OMG (Object Management Group), que centra sus esfuerzos en reconocer, que la interoperabilidad es fundamental y el desarrollo de modelos permite el desarrollo de otros modelos que luego al ser juntados proveerían la solución a todo un sistema e independiza el desarrollo de las tecnologías empleadas [7]. MDA plantea que el artefacto que reúne los requisitos del sistema se llama el CIM (Modelo Independiente de la Computación), el resultado de modelar éste sistema es un PIM (Modelo Independiente de la Plataforma) que se hace a través de de un DSL construido previamente. Éste DSL genera a través de un proceso de transformación un PSM (Modelo Especifico de la Plataforma), que por ultimo a través de otra transformación se convierte en código desplegable ó ISM (Modelo Específico de Implementación), éste proceso se visualiza en la Figura 1.

4.1. Metamodelos

Los metamodelos según Atkinson [8] son uno de los pilares de la ingeniería dirigida por modelos. Sin embargo, cuando se trabaja con modelos se suelen encontrar dificultades en la definición de los metamodelos por la gran variedad de representaciones posibles para los mismos y la ausencia de manuales que guíen su definición.

4.1.1. Metamodelos y la Arquitectura de Metadatos

Un modelo es una abstracción de la realidad, que la presenta de una manera simplificada. Un metamodelo es un modelo que describe un Lenguaje de Modelado (LM), con el que se describen otros modelos.

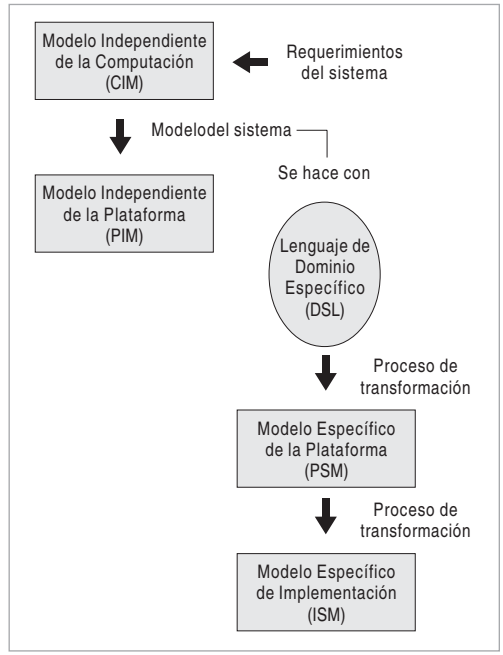


Figura 1. Proceso de despliegue de una solución completa con MDE

La noción de metamodelo se basa en la arquitectura de metadatos que se muestra en la Figura 2, adoptada por el consorcio OMG en la especificación del Meta-Object Facility (MOF) [9]. Ésta figura muestra la diferencia entre información, modelos, metamodelos y meta-metamodelos. MOF nombra cada una de estas capas o niveles con los nombres M0, M1, M2 y M3 respectivamente. Es importante fijarse en que estos conceptos están en capas diferentes y poseen significados diferentes aunque usen la misma notación.

El nivel más alto es el nivel de los meta-metamodelos, M3. Algunos ejemplos de lenguajes en éste nivel son MOF [9] usado por OMG, ECore [10] usado por EMF (Eclipse Modeling Framework), y GOPRR (Grafo, Objeto, Propiedad, Relación, y Rol) [11]. Las instancias de los lenguajes de M3 son los metamodelos, que definen LMs y corresponden al nivel M2. Un ejemplo bien conocido de metamodelos es la definición de UML (Unified Modeling Language) [12]. Luego están

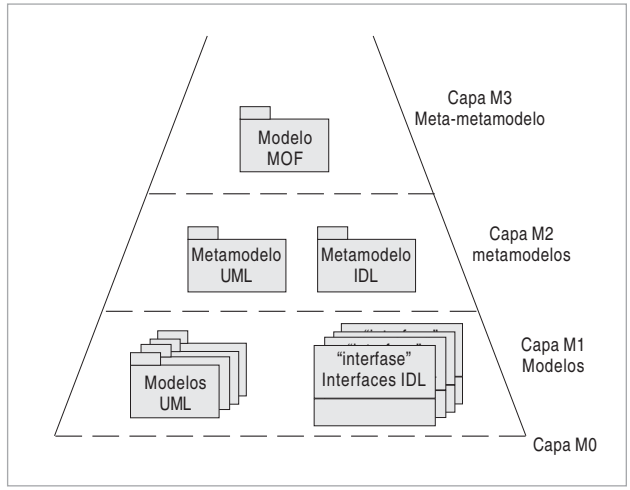


Figura 2. Arquitectura de Metadatos en MOF

los modelos que representan, por ejemplo, diseños de aplicaciones, estos son instancias de los metamodelos y constituyen el nivel M1, cualquier diseño concreto de una aplicación en UML podría servir como ejemplo. Por último se encuentra el nivel M0, que sería la implementación de los diseños, por decirlo en términos de programación orientada a objetos, en el nivel M0 se considerarían los objetos de las clases en ejecución y en el nivel M1 estarían las definiciones de las clases de objetos. Para éste trabajo todo lo que haga referencia a modelos se serializará en ficheros XMI (XML Metadata Interchange).

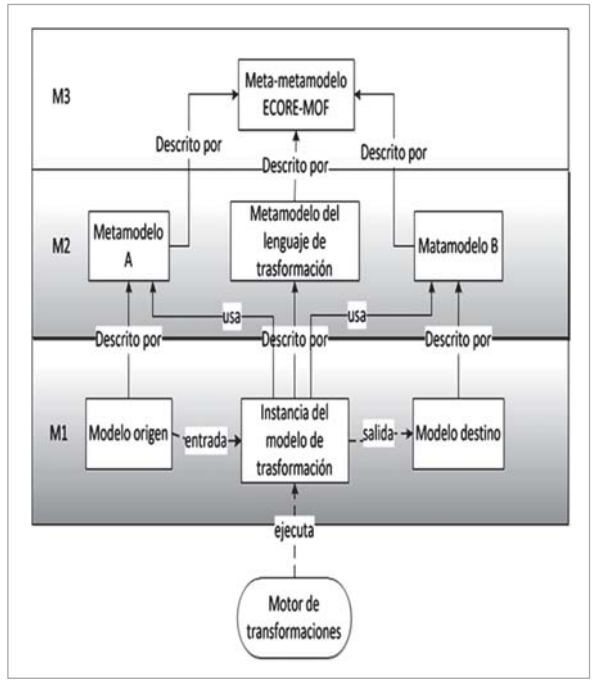


Figura 3. Transformaciones entre modelos

4.1.2. XML Metadata Interchange (XMI)

El estándar de la OMG XMI [9] es una especificación basada en XML para compartir metadatos MDA. Actualmente XMI es la base para conseguir interoperabilidad entre herramientas de apoyo a MDA, una de las grandes ventajas de XMI es que sobre él se pude almacenar todos los modelos basados en MOF utilizando un mismo esquema. Sin embargo, tiene inconvenientes, como su complejo formato para ser comprendido por humanos o que muchas herramientas que trabajan con XMI no se ajustan a las especificaciones de la recomendación.

4.2. Transformaciones Entre Modelos (M2M)

Una transformación de modelos significa convertir un modelo de entrada (o un conjunto de modelos) descritos por un determinado metamodelo, en otro modelo descrito por un segundo metamodelo, como se muestra en la Figura 3.

5. Lenguajes De Dominio Específico (DSL)

Los DSL intercambian generalidad por expresividad en un dominio limitado. Ofrecen notaciones y construcciones orientadas a un dominio particular, presentando ganancias sustanciales en términos de expresividad y facilidad de uso en comparación con los GPL para el dominio en cuestión. Estas ganancias se corresponden en mejoras de la productividad y reducción de los costes de mantenimiento.

5.1. Tipos de DSL

Los DSLs se pueden clasificar desde un punto de vista de la construcción del lenguaje en:

- Internos: Utilizan un determinado lenguaje anfitrión para darle la apariencia de otro lenguaje concreto. Un ejemplo claro son lo que actualmente se conoce como Fluent Interfaces [13].
- Externos: Tiene su propia sintaxis y es necesario un parser para poder procesarlos. Un ejemplo claro de DSL externo es SQL (Structured Query Language).

Desde el punto de vista del formato del lenguaje:

- Textuales: La mayoría de los lenguajes informáticos son textuales y están formados por un conjunto ordenado de sentencias. Un ejemplo muy conocido de DSL textual es SQL utilizado para realizar consultas a una base de datos. Una forma de crear DSLs textuales es mediante la creación de una determinada gramática (por ejemplo utilizando EBNF - Extended Backus-Naur Form) y posteriormente crear o utilizar un parser para dicha gramática, que en etapas posteriores puede interpretar el DSL o generar código.
- Gráficos: En los últimos años están ganando gran aceptación los lenguajes gráficos, podrían citarse como ejemplo UML. La creación de un lenguaje grafico es similar a la de un lenguaje textual, la única diferencia es que en lugar de usar texto para representar los conceptos, se utilizan conectores y figuras simples.

Desde el punto de vista del dominio del problema:

- Horizontales: Los DSL horizontales son aquellos en los que el cliente que utilizará el lenguaje no pertenece a ningún dominio específico. Un ejemplo son los editores visuales de entornos de desarrollo que permiten generar interfaces de usuario automáticamente (por ejemplo Windows Forms de visual Studio).
- Vertical: A diferencia de los DSL horizontales, el cliente que utilizará el lenguaje pertenece al mismo dominio que el lenguaje en sí. Por ejemplo un lenguaje de definición de encuestas, los usuarios finales serían los expertos en estadística encargados de definir dichas encuestas.

5.2. Partes de un DSL

Los lenguajes de dominio específico son el elemento principal de cualquier solución de dominio específico. El lenguaje se define como un metamodelo, una notación específica y generalmente una herramienta informática que lo soporta para facilitar la usabilidad. La idea básica es no utilizar conceptos de lenguajes de programación de propósito general. En su lugar, se utilizan conceptos y reglas de dominio.

La sintaxis abstracta de un lenguaje especifica su estructura, es decir, las construcciones, propiedades y conectores que pueda tener dicho lenguaje. La sintaxis concreta es necesaria para especificar la notación específica con la que los usuarios del lenguaje podrán utilizarlos. Idealmente cada concepto del dominio y del lenguaje se mapearán a una representación en la notación especifica. Es importante reseñar que una misma sintaxis

abstracta podría tener diferentes sintaxis concretas.

Como ya se ha puesto en contexto lo que es la ingeniería dirigida por modelos y los lenguajes de dominio especifico, en éste trabajo se construirá un DSL grafico. En el siguiente apartado se explicará el desarrollo de la propuesta.

6. Construcción de KiwiDSM

6.1. Meta-metamodelo

El meta-metamodelo que se empleará 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 [14].

6.2. Metamodelo

Para la utilización de Ecore en Eclipse es necesario tener instalado el plugin de EMF, éste plugin provee básicamente dos herramientas para construir un modelo basado en Ecore, una el Ecore Model que es un editor manual y funciona en un estilo de árbol de navegación para la creación del modelo basado en Ecore, la otra es el Ecore Diagram siendo éste un editor gráfico similar a las herramientas gráficas para la creación de diagramas de clases UML. Cualquiera de las dos forma que se utilice para crear el diagrama basado en Ecore, genera un fichero XMI[9].

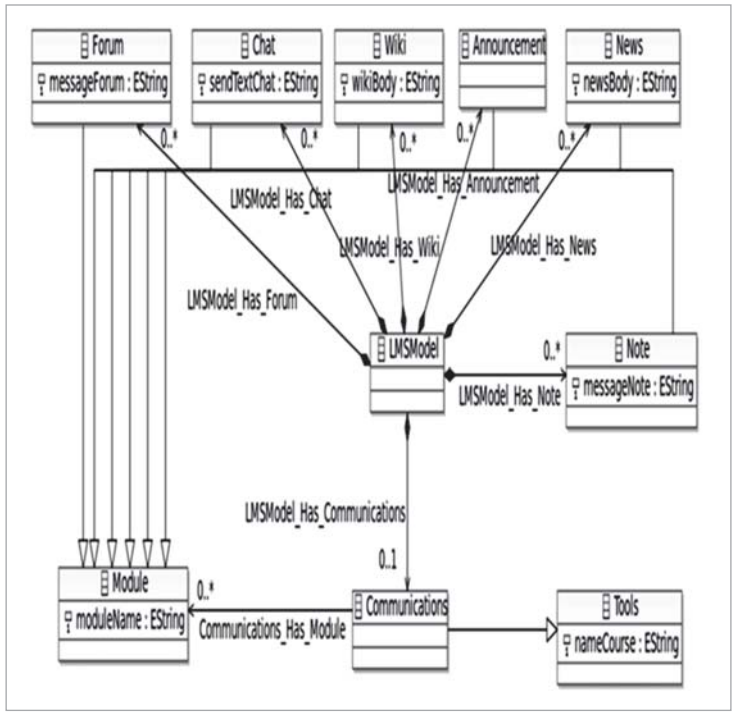


Figura 4. Metamodelo para módulos de comunicaciones de un LMS.

Para la construcción del metamodelo, se ha acotado el dominio únicamente a lo concerniente a los módulos de comunicaciones que tiene un LMS, por ello, ésta propuesta se basa en el metamodelo LMS planteado en el trabajo de Montenegro [15], éste metamodelo 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, todos están relacionados con la EClass Module mediante herencia, ya que todos los módulos poseen el atributo moduleName. La razón para definir esta herencia fue para eliminar la gran variedad de conexiones que habían en la herramientas DSL por una sola que está representada por la relación entre las EClass Communications y Module, de ésta forma una Communications puede tener cero o muchos Module; ésta EClass Communications hereda de la EClass Tools y todas las EClass que se visualizarán en KiwiDSM están contenidas en la EClass LMSModel, ésta relación es obligatoria en todo metamodelo pues es quién representará el contenedor de EClasses, y allí será en donde se desplieguen los módulos a modelar o mejor dicho donde estarán contenidos. Es necesario aclarar que la EClass LMSModel solo almacenará cero o una EClass Communications, esto tiene sentido pues no puede haber más de un módulo Communications en un LMS, ya que éste a su vez almacena las herramientas de Communications tales como Forum, Chat, Wiki, Annoncement, News y Note. Por último cada EClass tiene sus propios atributos que la compone.

6.3. Construcción del editor para el modelo o DSL

Como se está trabajando bajo eclipse, para ésta etapa se empleó EMF Tooling (Graphical Modeling Framework Tooling) que hace parte del proyecto GMP (Graphical Modeling Project)[16]. El proceso para la construcción del DSL gráfico se visualiza en la Figura 5; para mayor información para este tipo de herramientas el lector puede referirse a Budinsky [10] ó los tutoriales que ofrece su Web oficial [17].

Según el *dashboard* de la Figura 5, lo primero que se debe crear es el Domain Model, éste corresponde al metamodelo LMS 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.

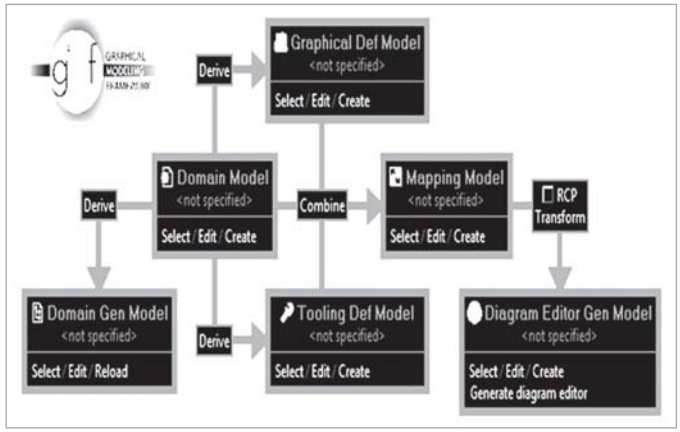


Figura 5. GMF Overview o dashboard. Tomado de T. E. Foundation [17]

Seguidamente, se crea el Graphical Def Model, éste es usado para definir las figuras, nodos, conexiones, etc. 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.

El siguiente paso en el *dashboard* es el Tooling Def Model, éste es usado para especificar la paleta (Pallete) de herramientas de creación, acciones, etc, para los elementos gráficos. Allí existe un elemento en el nivel superior "Tool Registry", en el que se encuentra la paleta. La paleta contiene un "Tools Group" con elementos de tipo "Creation Tool" para los nodos tema y conexiones para elementos de subtemas.

El Mapping Model es el siguiente paso en el *dashboard*, el Mapping Model combina los tres modelos: el "Domain Model", el "Graphical Def Model", y el "Tooling Def Model".

El último paso es la creación del Diagram Editor Gen Model. El modelo generador establece las propiedades para la generación de código, similar al EMF Gen Model. A partir de éste modelo se obtiene un *plugin* para eclipse que contiene la herramienta DSM construida. La apariencia de la herramienta se muestra en la Figura 6.

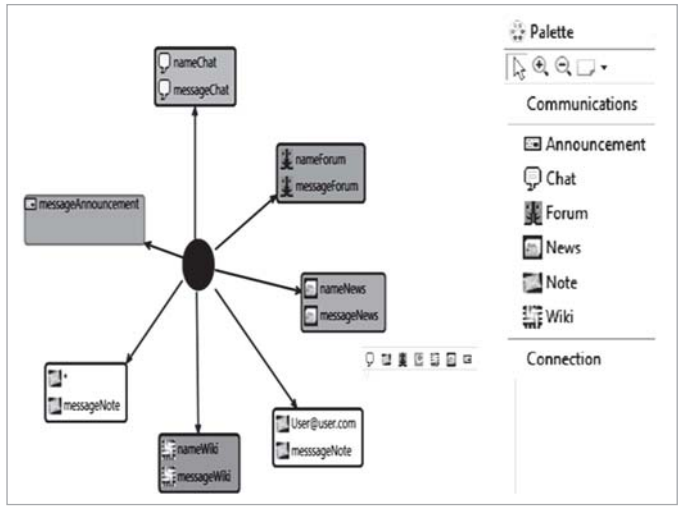


Figura 6. Apariencia de KiwiDSM

7. Modelado utilizando KiwiDSM

En la sección anterior se describió como se creó la herramienta KiwiDSM para modelar módulos de un LMS, ahora se mostrará cómo utilizar dicha herramienta para poder obtener un modelo de ella. El diagrama que se obtenga como resultado de emplear el editor de KiwiDSM, tendrá asociado un fichero XMI [9] que basará su sintaxis en el metamodelo creado.

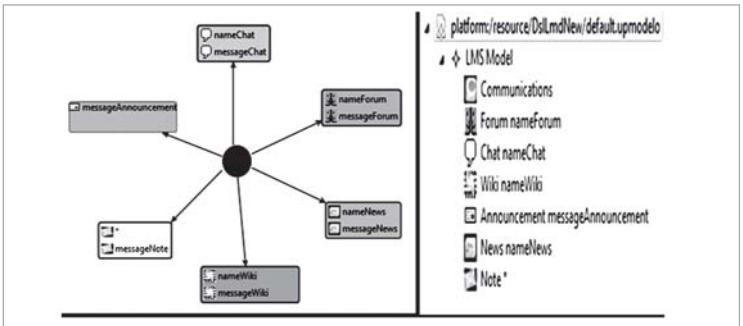
El funcionamiento de la herramienta es muy sencillo, para crear los módulos basta con arrastrar los nodos de la "Pallete" al área de trabajo, rellenar los campos y conectarlos respetando las siguientes reglas:

- Un curso solo puede tener un módulo de "Communications", esto es validado por la herramienta.
- El módulo de "Communications" tiene cero o muchos: "Announcements", "Chat", "Forum", "News", "Note", y "Wiki".
- 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 ésta herramienta se desplegarán en un solo tema del curso, con lo cual es necesario considerar los nombres de los campos en cada elemento (Nodo o Módulo) como genéricos, para no tener que cambiar el modelo cada vez que se despliegue sobre el curso. En la Figura 7 se puede visualizar un ejemplo, esta figura también muestra la relación de cada nodo y con el código XMI que genera la herramienta.

Algunas observaciones a tener en cuenta en el modelado son:

- Para enviar una "Note" a todos los inscritos en un curso, el primer campo debe estar marcado con asterisco ("*").
- Para enviar una "Note" a un solo inscrito en un curso, en el primer campo debe ir el correo completo con el que la persona se inscribió en la plataforma virtual. Si éste correo no corresponde con el de la persona el mensaje no será enviado.
- Todos los campos en los nodos deben estar diligenciados, pues aquellos campos en los cuales el usuario no ponga información serán tomados en blanco y la herramienta no procesará esos nodos.
- Todos los campos aparecen con información por defecto pero ésta información solo es de carácter orientativo, no implica información en los nodos.




```
<?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>
```

Figura 7. Modelo generado con KiwiDSM y su correspondencia con XMI.

En resumen, hasta el momento se ha hablado de cómo se creó KiwiDSM (M1), basada en un metamodelo generado (M2) y que a su vez se estructura sobre el meta-meta modelo Ecore (M3), el siguiente paso es convertir éste nuevo modelo que se obtuvo con KiwiDSM y hacerle una transformación a código (M0), ésta idea se explicará en la siguiente sección.

8. Proceso de generación de código a partir del modelo hecho con KiwiDSM

En éste último paso, existen varias tecnologías que se integran a Eclipse, como son Acceleo [18], Jet [19], Xpand [20] y MOFScript [21], 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, éste tipo de tecnologías reciben el nombre M2T o Model to Text. Para éste caso se empleará MOFScript [21].

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 su uso no es complicado de entender. El lector puede referirse al manual para el uso de MOFScript aportado por Oldevik [22].

Para poder comprender el contenido del fichero MOFScript es necesario entender cómo se desarrolla un módulo para la plataformas LMS Atutor, cuya guía se encuentra en su web oficial [23, 24]. Los detalles de este tema están por fuera del alcance de este artículo.

Para transformar el modelo que se obtiene con KiwiDSM es necesario considerar que este modelo estará basado en el metamodelo LMS construido anteriormente. 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 éste caso *upmodelo*, también se debe dar un nombre al modelo enviado, para éste caso se llamo *mlms*. Lo anterior se puede visualizar el siguiente fragmento de código Figura 8.

```
texttransformation modellmsTransformationAtutor (in
mlms:"upmodelo") {//Cuerpo de la transformación}
```

Figura 8. Declaración de una transformación en MOFScript.

La funciona principal o punto de partida para las transformaciones en MOFScript están contendidas en la función "main()"

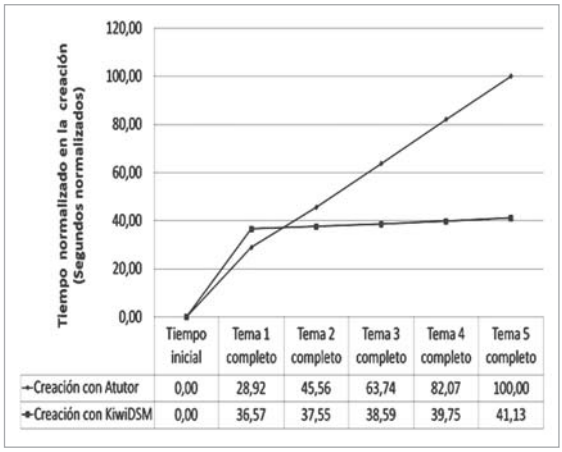
Finalmente aplicando las tecnologías de transformación sobre el modelo creado se obtiene el código en el formato definido para su despliegue sobre la plataforma seleccionada Atutor. El proyecto completo está disponible para su descarga en *http://www.vicegd.com/KiwiDSMCode.rar*.

9. Pruebas de despliegue

El objetivo de esta sección es validar el metamodelo planteado así como comprobar la validez de la siguiente hipótesis "al trabajar con MDA (Model Driven Architecture) el tiempo y esfuerzo en la generación de soluciones se reducen". Las pruebas medirán el tiempo y esfuerzo (usabilidad) de los usuarios para crear exactamente los mismos módulos en Atutor y KiwiDSM; el esfuerzo será medido, de acuerdo a la definición de Yamada [25], como la "cantidad de ocasiones en las que usuario selecciona o ingresa algún tipo de información al sistema". Cada uno de los usuario creará 5 temas y cada uno de los temas tendrá en éste orden: 1 Chat, 1 Forum, 1 Wiki, 1 Announcement, 1 News y 1 Note a todas las personas inscritas en ese curso, la información de cada módulo será la misma en todos los casos y los valores son normalizados con respecto al mayor, las pruebas se aplicaron a un total de 16 personas con conocimientos básicos en el uso de Atutor.

Las mediciones se realizaron tanto a la plataforma Atutor como a KiwiDSM y abarcaron: 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, tiempo y esfuerzo total en la creación de los cinco temas acompañados de sus herramientas. En las pruebas se ha considerado que el envío de las notas (mensajes) a los inscritos en el curso son una herramienta del tema. Para poder comparar las mediciones se hizo necesario que todos los módulos fueran creados en el mismo orden para los dos sistemas.

La Figura 9 muestra los tiempos promedio normalizados al mayor (el mayor tiempo para éste caso es el tiempo total de crear todos los temas con sus módulos en Atutor) por tema, allí se ve que la diferencia al final de crear el primer tema, es tan solo de 7.65% más tiempo, en crear los módulos en KiwiDSM que en Atutor, pero ésta diferencia cambia a partir del segundo tema en adelante, logrando reducir hasta en un 58.87% los tiempos; también se observa que los tiempos para la creación de módulos en Atutor crecen más rápido que con la herramienta KiwiDSM.



	Tiempo inicial	Tema 1 completo	Tema 2 completo	Tema 3 completo	Tema 4 completo	Tema 5 completo
Creación con Atutor	0,00	28,92	45,56	63,74	82,07	100,00
Creación con KiwiDSM	0,00	36,57	37,55	38,59	39,75	41,13

Figura 9. Comparación de tiempos entre la creación de módulos usando la plataforma Atutor Vs la creación de módulos con la herramienta KiwiDSM.

La Figura 10 muestra un comportamiento muy similar pero midiendo el esfuerzo promedio normalizado al mayor (el mayor esfuerzo para éste caso se obtiene en la creación de todos los módulos para el primer tema con KiwiDSM y despliegue sobre Atutor), allí la diferencia al final de crear el primer tema, es de 17.98% menos esfuerzo con Atutor que con la herramienta KiwiDSM, pero desde el segundo tema en adelante, el esfuerzo en la creación de módulos con la herramienta KiwiDSM, es por lo menos 71.16% menor que al hacerlo en Atutor.

Las Figuras 11 y 12 muestran el tiempo promedio total y esfuerzo total normalizados al mayor valor (que para éste caso se obtienen al crear todos los temas con sus módulos en Atutor), en la creación de 5 temas cada uno con sus respectivos módulos, y se concluye que la creación de módulos con la herramienta KiwiDSM, es por lo menos 58.87% más eficiente que al hacerlo en Atutor.

Los resultados de estas pruebas evidencian que al trabajar con modelos se disminuye el tiempo y el esfuerzo del usuario durante la creación de módulos para una plataforma LMS, caso particular, Atutor. Igualmente de esta manera se verifica el funcionamiento del metamodelo, ya que no se genero ningún fallo en la creación de los módulos con la herramienta DSL, que tiene como base fundamental el metamodelo LMS planteado.

10. Conclusiones

Se ha descrito una herramienta DSL que se basa en un metamodelo LMS; la herramienta funciona para la creación de módulos independientes de la plataforma y solo cubre algunos módulos de comunicaciones. Las pruebas a la herramienta han demostrado que los tiempos y esfuerzos necesarios en la creación de diversos módulos en una plataforma LMS se minimizan en más del 50% y que este rendimiento mejora a medida que se utiliza más la plataforma. El siguiente paso es probar con mas módulos y distintas plataformas, y comprobar si éste comportamiento se mantiene.

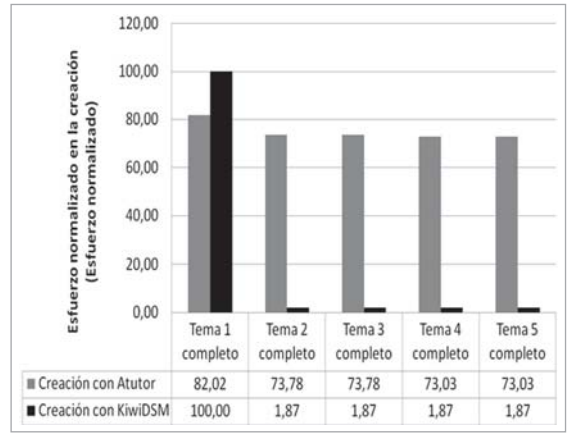


Figura 10. Comparación de usabilidad (esfuerzo) entre la creación de módulos usando la plataforma Atutor y la creación de módulos usando la herramienta KiwiDSM.

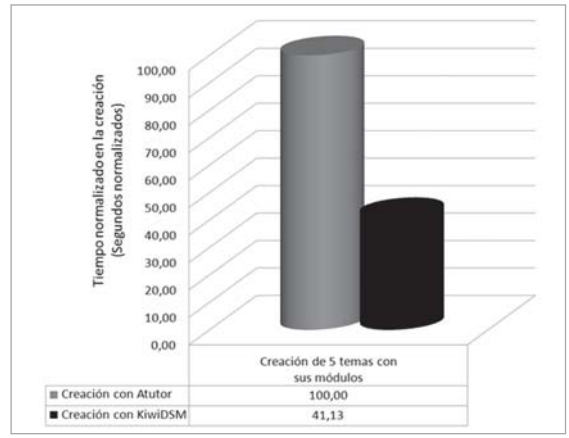


Figura 11. Comparación de tiempo total en la creación de módulos usando la plataforma Atutor Vs la herramienta KiwiDSM.

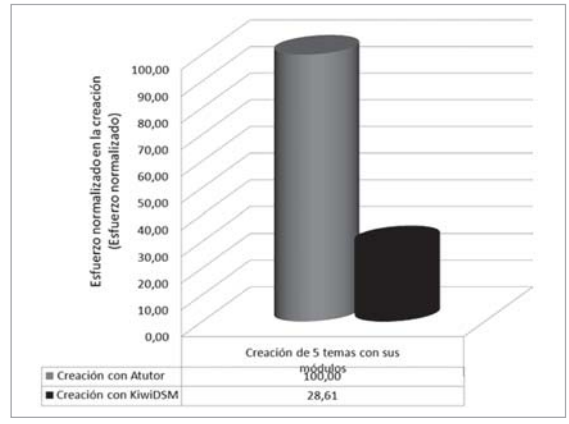


Figura 12. Comparación de usabilidad (esfuerzo) total en la creación de módulos usando la plataforma Atutor Vs la herramienta KiwiDSM.

El metamodelo planteado es acorde al LMS trabajado y su validez se soporta en el correcto despliegue del modelo creado con KiwiDSM. Dicho metamodelo podría extenderse para integrar más módulos y plataformas LMS. De igual forma es posible que, a partir de la definición de este metamodelo, se plantee un proceso inverso en el cual los módulos que ya estén creados en un LMS específico se conviertan a instancias del metamodelo y de esta forma establecer un mecanismo de transformación a cualquier otra plataforma LMS.

En esta propuesta se han explorado los niveles M0, M1, M2 y M3 planeados en la ingeniería dirigida por modelos (MDE). El metamodelo LMS construido representa el nivel M2, el meta-metamodelo Ecore es el nivel M3, el resultado de modelar los módulos con KiwiDSM representa al nivel M1 y finalmente, las conversiones y despliegue de esos módulos construidos en Atutor representan el nivel M0. El alcance de esta propuesta se limitó a la utilización de MDE para crear un modelo independiente de la plataforma y lograr un despliegue real de este sobre una plataforma LMS.

Finalmente, respecto al diseño de la herramienta DSL, mediante la aplicación del patrón Fabrica Abstracta se solucionó el inconveniente de mantener muchos conectores para un mismo módulo. Se deja abierta la posibilidad de plantear otros mecanismos que se apoyen en los patrones de diseño orientados por objetos, aplicables a este tipo de situaciones en ingeniería de metamodelos.

11. Referencias bibliográficas

[1] Grob, H. L. et al. (2010). eleed - Model Driven Architecture (MDA): Integration and Model Reuse for Open Source eLearning Platforms, *FernUniversität Hagen, CampusSource Muenster, Germany*.

[2] Moreno, N. y Romero, J. R. (2005). Recent Research Developments in Learning Technologies (2005), en: *The A MDA-based framework for building interoperable e-learning platforms*, Badajoz, Spain.

[3] Bizoòová, *Z.* et al. (2007). Model Driven E-Learning Platform Integration, en: *2nd European Conference on Technology Enhanced Learning EC-TEL PROLEARN 2007 Doctoral Consortium*, *Crete, Greece*. 8-14.

[4] Bizonova, Z. y Ranc, D. (2007). Model Driven LMS Platform Integration, en: *Telecommunications, 2007. AICT 2007. The Third Advanced International Conference*. 25-25.

[5] Bizonova, Z. y Ranc, D (2008). Interoperability and Reuse Between Systems in eLearning, en: *World Conference on Educational Multimedia, Hypermedia and Telecommunications 2008*, *Vienna, Austria*. 1700-1705.

[6] Muñoz-Merino, P. J. *et al.* (2009). Enabling interoperability for LMS educational services, en: *Computer Standards & Interfaces, Vol. 31*. 484-498.

[7] Stephen, J. M. et al. (2004). MDA Distilled: Principles of Model-Driven Architecture, *Addison Wesley*.

[8] Atkinson, C. y Kuhne, T. (2003). Model-driven development: a metamodeling foundation, en: *Software, IEEE, Vol. 20*. 36-41.

[9] OMG. (2007). MOF 2.0/XMI Mapping, Version 2.1.1, ed: Object Management Group. 120.

[10] Budinsky, F. et al. (2009). EMF: Eclipse Modeling Framework: *Addison-Wesley*.

[11] Tolvanen, J. P. et al. (1993). An integrated model for information systems modeling, en: *System Sciences, 1993, Proceeding of the Twenty-Sixth Hawaii International Conference. Vol.3*. 470-479

[12] OMG. (1999). OMG Unified Modeling Language Specification, Version 1.3, ed: Object Management Group. 808.

[13] Freeman, S. y Pryce, N. (2006). Evolving an embedded domain-specific language in Java, en: *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*, *Portland, Oregon, USA*.

[14] I. C. a. others. (2010). *Package org.eclipse.emf.ecore*. Consultado: http://download.eclipse.org/modeling/emf/emf/javadoc/2.6.0/org/eclipse/emf/ecore/package-summary.html (5 de diciembre, 2010).

[15] Montenegro, C. et al. (2011). Generation of metamodel in ecore with start point in an ontology for learning management systems (lms), *Journal of Web Engineering, (en prensa)*.

[16] T. E. Foundation. (2010). Graphical Modeling Project (GMP). Consultado: http://www.eclipse.org/modeling/gmp/ (2 de diciembre, 2010).

[17] T. E. Foundation. (2011). GMF Tutorial. Consultado: http://wiki.eclipse.org/GMF_Tutorial. (6 de agosto, 2011)

[18] Obeo. (2010). Acceleo. Consultado: http://www.eclipse.org/acceleo/. (1 de diciembre, 2010).

[19] T. E. Foundation. (2010). M2T-JET. Consultado: http://wiki.eclipse.org/M2T-JET. (5 de diciembre, 2010)

[20] T. E. Foundation. (2010). XPand. Consultado: http://wiki.eclipse.org/Xpand. (5 de diciembre, 2010).

[21] T. E. Foundation. (2010). MOFScript Consultado: http://www.eclipse.org/gmt/mofscript/. (2 de diciembre, 2010)

[22] Oldevik, J.. (2010). MOFScript User Guide Version 0.8 (MOFScript v 1.3.6), 2009. Consultado: http://www.eclipse.org/gmt/mofscript/doc/. (5 de diciembre, 2010)

[23] ATutor. (2010). Module Development Documentation. Consultado: http://atutor.ca/development/documentation/modules.html#structure. (5 de diciembre, 2010)

[24] ATutor. (2010). ATutor Learning Managment Tools. Consultado: http://atutor.ca/ (5 de diciembre, 2010)

[25] Yamada, S. et al. (1993). Software-reliability growth with a Weibull test-effort: a model and application, *Reliability, IEEE Transactions on,* Vol. 42.100-106.

Carlos Enrique Montenegro Marín

Ingeniero de Sistemas de la Universidad Distrital "Francisco José de Caldas", de Bogotá, Colombia. Obtuvo su título de Maestría en Ciencias de la información y las comunicaciones en la Universidad Distrital "Francisco José de Caldas de Bogotá, Colombia. Es estudiante del Doctorado en Sistemas y servicios informáticos para internet en la Universidad de Oviedo, de Oviedo, España. Actualmente se desempeña como profesor de la Facultad de Ingeniería en la Universidad Distrital "Francisco José de Caldas", de Bogotá, Colombia, y pertenece como investigador al grupo GIIRA donde realiza estudios sobre Ingeniería Dirigida por Modelos MDE.

Juan Manuel Cueva Lovelle

Ingeniero Superior E.T.S.I. de Minas de Oviedo de, Oviedo, España. Obtuvo su PhD en ingeniería en la Universidad Politécnica de Madrid, España. Catedrático de Escuela Universitaria de Lenguajes y Sistemas Informáticos de la Universidad de Oviedo (España). Director de la Escuela Universitaria de Ingeniería Técnica en Informática de Oviedo (Universidad de Oviedo) desde Julio-1996 a Julio-2004. Socio de ATI y miembro con voto de ACM e IEEE. Sus áreas de investigación son Tecnologías Orientadas a Objetos, Procesadores de Lenguaje, Interacción Persona-Ordenador e Ingeniería Web. Ha dirigido varios proyectos de Investigación y tesis doctorales en Ingeniería Informática. Es autor de libros, artículos y comunicaciones a congresos. Coordina el laboratorio de investigación de Tecnologías Orientadas a Objetos del departamento de Informática de la Universidad de Oviedo (www.ootlab.uniovi.es).

Óscar Sanjuán Martínez

Ingeniero Informático de la Universidad Pontificia de Salamanca, España. Obtuvo su título de PhD en Ingeniería Informática en la Universidad la Universidad Pontificia de Salamanca, España. Se ha desempeñó en el sector empresarial de la informática a nivel mundial, ha sido catedrático en el área de informática en la Universidad la Universidad Pontificia de Salamanca y la Universidad de Oviedo, España, también ha sido profesor invitado en varias universidad a nivel mundial. Actualmente se desempeña como profesor e investigador en el departamento de informática de la universidad de Oviedo, España.

Paulo Alonso Gaona García

Ingeniero de sistemas de la Universidad Distrital "Francisco José de Caldas", de Bogotá, Colombia. Obtuvo su título de Maestría en Ciencias de la información y las comunicaciones en la Universidad Distrital "Francisco José de Caldas de Bogotá, Colombia. Es estudiante del Doctorado en Informática de la Universidad la Universidad Pontificia de Salamanca, España. Actualmente se desempeña como profesor de la facultad de Ingeniería en la Universidad Distrital "Francisco José de Caldas", de Bogotá, Colombia, y es director del grupo de investigación GIIRA.