Nuevos aliados en el diseño de asignaturas: UML y MDA para profesores

Jesús Martínez Cruz

Departamento de Lenguajes y Ciencias de la Computación

Universidad de Málaga

ETSI Telecomunicación, Campus Teatinos

29071 Málaga

jmcruz@lcc.uma.es

Resumen

La construcción de Software de Comunicaciones y servicios para Redes ha evolucionado enormemente en los últimos años: existen nuevas tecnologías de red, estándares, protocolos, *middleware*, lenguajes y librerías de soporte que permiten diversos enfoques para conseguir los objetivos generales de las asignaturas universitarias orientadas al diseño e implementación de aplicaciones distribuidas. En la mayoría de los casos, los contenidos y actividades planteadas estarán condicionados por la selección concreta de un lenguaje de programación y de un sistema operativo de ejecución. Esta dependencia se hace aún más fuerte en las asignaturas enfocadas a aspectos de bajo nivel, como la construcción de aplicaciones con *Sockets*. En este artículo se propone una metodología basada en el lenguaje unificado de modelado (UML) y la arquitectura dirigida por modelos (MDA) para la confección de contenidos genéricos (independientes del lenguaje o plataforma) y su posterior refinamiento en función de tales selecciones, restricciones o preferencias del profesor (o estudiantes). El objetivo de esta metodología es contar con un soporte semiautomático que ayude al profesor en la planificación dinámica de distintos itinerarios formativos a medida. El marco de prueba para su evaluación lo constituyen las asignaturas de Software de Comunicaciones impartidas en la Universidad de Málaga.

1. Introducción

Al igual que las redes de ordenadores y sus protocolos han evolucionado a lo largo de los últimos treinta años, también lo han hecho los servicios distribuidos. Gran parte del éxito en la aparición de nuevas aplicaciones se debe al uso de estándares y librerías de alto nivel (*middleware*) que aíslan al programador de los protocolos y características del entorno de red sobre los que se ejecutan. Sin embargo, la disponibilidad de estos recursos juega a veces un papel negativo en la implementación de software de comunicaciones eficiente: abuso de protocolos de aplicación y Servicios Web, deficientes implementaciones con *Sockets*, estrategias inadecuadas en el diseño de servidores... En muchos casos, da la impresión de que las aplicaciones resultantes son eficaces sólo por incorporar una mayor funcionalidad a un bajo coste de desarrollo, dejando a un lado otros detalles importantes para que el producto sea robusto y escalable. No cabe duda que nuestros ingenieros deben tener presente que un buen software de comunicaciones exige del conocimiento extenso en campos ya maduros como la programación concurrente, la ingeniería de protocolos y, por supuesto, la ingeniería del software. Y he aquí donde los docentes en Telemática con responsabilidad en la impartición de esta materia se enfrentan al dilema de qué impartir y cómo hacerlo.

Tradicionalmente, las asignaturas relacionadas con el Software de Comunicaciones se plantean objetivos diferentes de enseñanza en función de la ti-

tulación, curso, optatividad/troncalidad y número de créditos asignados. Además de estos factores, existen otros que hacen que la asignatura se planifique de diferentes formas en determinados años. Por ejemplo, los lenguajes de programación y sistemas operativos que conoce cada promoción son un factor clave en la preparación de la asignatura. Si el objetivo es utilizar la interfaz de *Sockets* en C para UNIX, pero una promoción no conoce bien el sistema operativo, hay que planificar bastantes tareas de programación de sistemas, seguramente a costa de no poder profundizar luego en aspectos de diseño avanzado de servidores. Si, por el contrario, los alumnos han cursado ya programación orientada a objetos y han practicado concurrencia en sistemas operativos, la asignatura puede centrarse más en patrones de diseño para comunicaciones donde la orientación será la de conseguir aplicaciones robustas y escalables. Estas situaciones corresponden, por ejemplo, a los cursos de segundo ciclo en Ingeniería de Telecomunicación e Ingeniería Informática de la Universidad de Málaga, respectivamente. Mucho más complejas son las situaciones en las que la asignatura es cursada por estudiantes de distintas promociones y/o titulaciones (libre configuración). En estos casos, una planificación rígida suele afectar significativamente al rendimiento de algún grupo de estudiantes.

Partiendo de la base de que diseñar una asignatura con todos estos condicionantes supone un esfuerzo ímprobo para cualquier docente, sería más que interesante poder manejar objetivos, planificaciones o actividades en diversos niveles de concreción. De esta forma, cada nivel supondría un grado más de refinamiento en el diseño, fruto de la aplicación de alguno de los factores reseñados anteriormente. No cabe duda de que, para obtener un modelo para asignaturas acorde a esta visión, se debería contar con una metodología rigurosa que diera soporte (automático o semi-automático) a la descripción, manipulación y generación de dichos niveles, hasta la obtención del diseño final.

Este artículo presenta un modelo para el diseño de contenidos y actividades de una asignatura mediante herramientas habituales de Ingeniería del Software. En primer lugar, se propone el uso del lenguaje unificado de modelado (UML) [8] para estructurar bloques temáticos, contenidos y sus relaciones de forma visual, así como para definir la planificación y actividades. Cabe citar que el uso de UML en contextos pedagógicos está poco explorado, aunque ya existen algunos trabajos relacionados que avalan esta propuesta [3, 2]. Una novedad aquí consiste en la posibilidad de que las asignaturas se diseñen de forma genérica, sin tener en cuenta factores como la titulación, curso o número de créditos. La aplicación de dichos factores se realizará en un paso posterior, obteniendo asignaturas concretas específicas para un determinado contexto educativo y plan de estudios. Para conseguir este objetivo se propone el uso de métodos provenientes de la Arquitectura Dirigida por Modelos (MDA) [4] que permitirían, además, la generación automática de recursos tales como la documentación del curso, o guiones de prácticas específicos para un lenguaje de programación, librería y sistema operativo. Los modelos de asignatura generados de esta forma se almacenan en formato estándar XML [11], y son susceptibles de manipulación automática o interactiva, utilizando herramientas CASE conocidas, como Eclipse [10].

La organización del artículo es la siguiente: en la sección 2 se introducen los conceptos de UML y MDA que forman la base del diseño de cursos; en la sección 3 se utilizan dichos conceptos como ejemplo para el diseño de una asignatura de Software de Comunicaciones, junto a las posibilidades y formas de implementarlo. Por último, se exponen las conclusiones del trabajo.

2. UML y MDA

UML es el lenguaje de modelado visual estandarizado por el Object Management Group (OMG) para el análisis y diseño de sistemas. Tradicionalmente utilizado en software, UML está bien soportado por herramientas gráficas tanto comerciales como de código abierto, que permiten la elaboración de distintos tipos de diagramas o vistas de un sistema, permitiendo describir tanto su estructura como el comportamiento de sus elementos. Uno de los diagramas más utilizados es el denominado Diagrama de Clases, en el que aparecen distintos clasificadores y sus relaciones. La fig. 1 muestra un ejemplo de elementos habituales en este tipo de diagramas. Los dos primeros clasificadores que aparecen (arri-

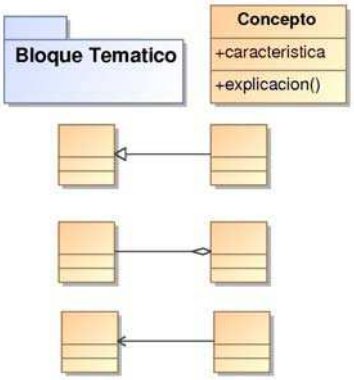

Figura 1: Algunos elementos de un diagrama de clases UML

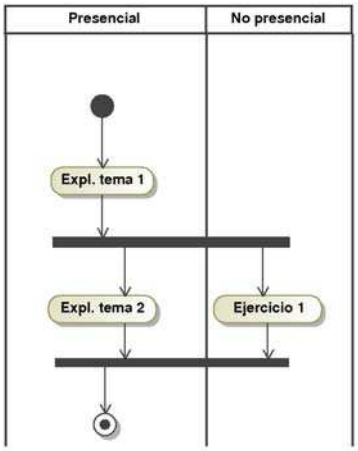

Figura 2: Ejemplo de elementos de un diagrama de actividades UML

ba) son un paquete y una clase, respectivamente. Los paquetes son contenedores de clase u otros paquetes. Las clases son contenedores de datos, junto a las operaciones que permiten su manipulación. En el ejemplo de la figura, el paquete *Bloque Temático* podría contener a la clase *Concepto*, que incorpora atributos (*característica* es un dato), y operaciones (como el método *explicación*). También aparecen en la figura tres tipos de relaciones habituales entre clases. De arriba abajo se describe una generalización (herencia de atributos y métodos), una agregación (una clase contiene a otra) y una asociación (una clase tiene acceso a las operaciones de otra).

Respecto al comportamiento, existen distintos diagramas ampliamente utilizados. Uno de los que más mejoras ha sufrido en la versión 2 de UML ha sido el denominado Diagrama de Actividades. En este diagrama se describe el flujo de ejecución de actividades (acciones) y resulta especialmente útil cuando se plantean transformaciones de UML a código final, tal y como se verá más adelante. La fig. 2 muestra un ejemplo de actividad con distintas acciones delimitadas por un inicio (círculo negro) y un final (círculo negro sobre blanco). La semántica del diagrama establece que el tránsito de una acción a otra se realiza de forma secuencial. En la figura, la primera acción que se acomete se denomina *Explicación tema 1*. También existe la posibilidad de realizar acciones en paralelo, indicándose con una barra horizontal *fork*, de la que parten otras dos acciones: *Explicación tema 2* y *Ejercicio 1*. Para sincronizar el punto en el que se ha de esperar a que terminen las actividades en paralelo, se utiliza de nuevo una barra horizontal *join*. En la fig. 2 también puede observarse que las acciones se encuadran dentro de columnas. Estos separadores, opcionales, se denominan *swimlanes* y permiten dotar de semántica adicional al diagrama, pudiendo especificarse que las acciones pertenezcan tanto a clasificadores UML existentes como a otros elementos. Por ejemplo, los *swimlanes* de la fig. 2 enmarcan acciones de tipo *Presencial* y *No presencial*. Hay que indicar que los Diagramas de Actividad carecen de semántica de tiempo, por lo que habitualmente se complementan con otro tipo de diagramas, denominados Diagramas de Interacción Temporal [8], para permitir la planificación temporal de actividades.

MDA es otra iniciativa del OMG que parte de la premisa de mantener separada la especificación de un sistema (modelo) de su implementación. Los modelos MDA se estructuran en capas de diferente nivel de abstracción. De esta forma, existen los modelos independientes y los dependientes de la plataforma. Los modelos independientes de la plataforma (Platform Independent Models o PIM) contienen aquellos elementos que no están sujetos a un

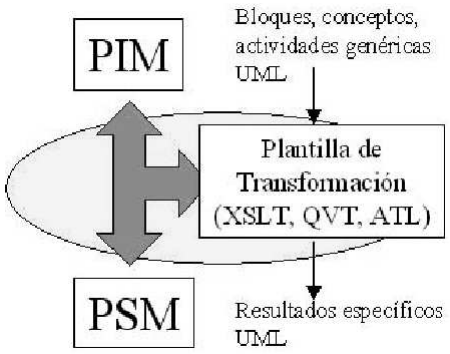

Figura 3: Transformación en MDA: de lo genérico a lo particular

tipo de tecnología particular. Este tipo de detalles sí aparecen en los modelos dependientes de la plataforma (Platform Specific Models o PSM). En MDA existen estándares que permiten describir el proceso de conversión de un modelo PIM a PSM, lo que constituye el procedimiento habitual para transformar diseños de alto nivel en otros más cercanos a la implementación final. En el ámbito del desarrollo de software dirigido por modelos, un modelo PIM será refinado y convertido a uno PSM que contenga elementos específicos para un lenguaje de programación, librerías específicas o situaciones de despliegue (por ejemplo de componentes en nodos de la red, entre otros). El fin último de un modelo PSM en este contexto suele ser el de la generación automática de código.

La fig. 3 presenta un proceso típico de transformación en el ámbito de MDA donde, dado un modelo de alto nivel, éste se refina de forma automática a través de una plantilla de transformación. Para este fin, el OMG ha estandarizado recientemente el lenguaje QVT (Query-View-Transform) [6] que, junto con otras propuestas muy extendidas [1], permite elaborar reglas que guíen la transformación de un modelo en otro, o incluso la generación de código final. El éxito de estos lenguajes se sustenta en que la descripción de los modelos origen y destino es conforme a unos metamodelos (UML es un ejemplo de ello) que a su vez fueron descritos en un metamodelo común, el meta-metamodelo MOF (Meta Object Facility). Otra de las ventajas del uso de modelos derivados de MOF es la disponibilidad de almacenarlos en un formato común XML [11], el denominado XMI (XML Metadata Interchange)[5]. Los modelos en XMI pueden ser leídos por las herramientas UML (para visualización, análisis, generación de código), pero también pueden utilizarse otras herramientas y librerías XML habituales, de forma que puedan manipularse los datos contenidos a más bajo nivel. En la fig. 3, los modelos origen pueden consistir en diagramas como los de las figuras 1 y 2. Las especificaciones MDA preveen mecanismos de transformación en ambos sentidos, aunque dichos mecanismos de ida y vuelta no suelen estar implementados.

3. Aplicación a una asignatura: Software de Comunicaciones

La docencia de Software de Comunicaciones, en Málaga, es responsabilidad del Área de Telemática del Departamento de Lenguajes y Ciencias de la Computación. La impartición de sus contenidos se realiza en una o varias asignaturas y en distintas titulaciones [1].

En este apartado se introduce el diseño de una asignatura de Software de Comunicaciones a través de una metodología que utiliza UML y MDA en dos pasos. En primer lugar se definirán bloques temáticos, conceptos y sus relaciones de forma genérica. También, a nivel general, se planificará la secuencia básica de actividades por bloque. A continuación, se aplicarán los elementos de transformación adecuados para obtener conceptos y tareas específicos, teniendo en cuenta factores como la titulación, curso, créditos u otras restricciones.

3.1. Elaboración de contenidos genéricos

Para aplicar la metodología propuesta a esta materia, se parte de unos objetivos de aprendizaje básicos sobre los que se diseñarán bloques temáticos,

[1] Como optativa de quinto en Ingeniería de Telecomunicación (Softw. de Comunicaciones y Lab. de Softw. de Comunicaciones) e Ingeniería Informática (Softw. de Comunicaciones y POO para Disp. de Teleco.), y en segundo de Ingeniería Técnica de Telecomunicación, en la especialidad de Sistemas de Telecomunicación (Lab. de Softw. de Comunicaciones).

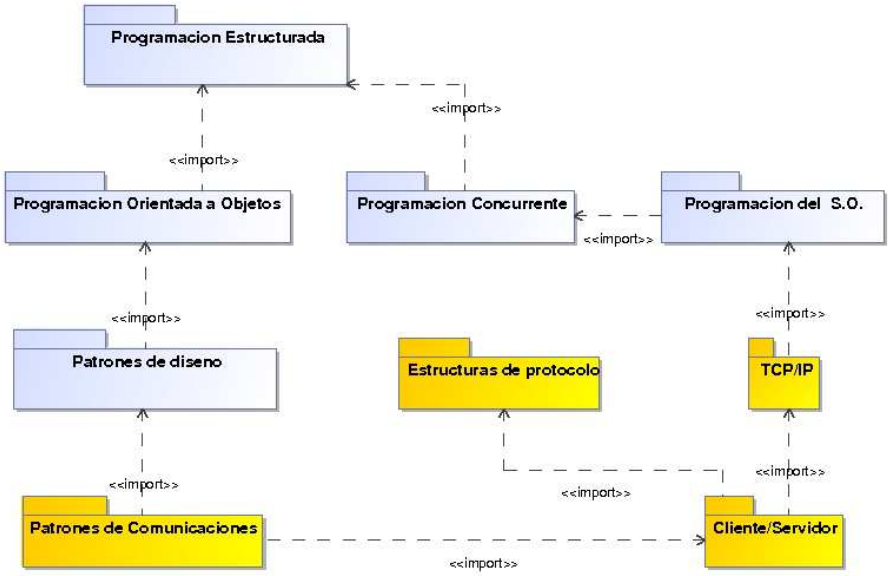

Figura 4: Diagrama de bloques temáticos y sus dependencias

contenidos y actividades de forma general. En primer lugar, se elaborará un diagrama de clases para incorporar los bloques temáticos a impartir, indicando opcionalmente su relación entre ellos y el resto del plan de estudios. La fig. 4 muestra un ejemplo de esta descripción, en la que cada bloque temático constituye un paquete UML. En la figura, los bloques objetivo de la materia abarcan la programación de Sockets TCP/IP, la arquitectura Cliente/Servidor y los patrones de diseño específicos para comunicaciones [9]. El paquete *Estructuras de protocolo* se centrará en la enseñanza de conceptos auxiliares típicos necesarios en su construcción: mensajes, colas, temporizadores o máquinas de estado.

En el diagrama se determinará también la relación entre bloques temáticos, partiendo de la base de la Programación Estructurada que se imparte en primer curso. Mediante las relaciones de tipo `<<import>>` se indica que un bloque requiere del conocimiento previo de los temas contenidos en otros bloques a los que apunta. Esta relación de orden se clarifica dentro de cada bloque, donde se van a enumerar los temas o conceptos a tratar como

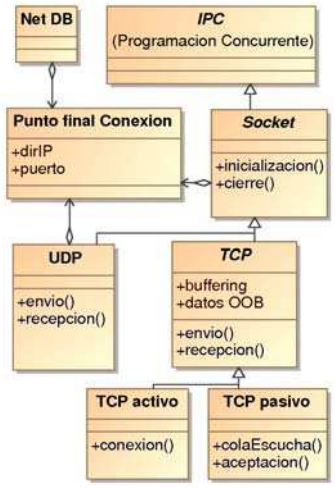

Figura 5: PIM para conceptos de sockets (genérico)

clases UML y su dependencia respecto a conceptos aprendidos con anterioridad. La fig.5 plantea un

posible escenario donde se han diseñado clases que están enfocadas a definir conceptos del bloque de *TCP/IP* [2]. En la figura se aprecia el concepto de *Socket*, un derivado del concepto de *IPC* (comunicación entre procesos), conocido desde el bloque *Programación Concurrente*, y que a su vez se especializa en los sockets *UDP* y *TCP*. En este diagrama se ha decidido incluir en la parte de atributos alguna de las propiedades ligadas a cada concepto. Por ejemplo, en la definición de punto final de conexión se requiere una dirección IP y de un puerto de transporte. En la parte de operaciones, se ha decidido resaltar los métodos más importantes que utilizará el estudiante en los *Sockets* y que, en este caso, también requieren de explicación por parte del docente. El diagrama incorpora algo más de notación UML. Por ejemplo, las clases cuyo nombre aparece en cursiva se denominan abstractas, e indican que el concepto requiere ser ampliado por otras clases. En efecto, en la fig. 5 el concepto *IPC* se concreta en un *Socket* y, a su vez, el socket de tipo *TCP* será concretamente un socket TCP *activo* o *pasivo*. Como se ha comentado, la figura establece un posible modelo de conceptos relacionados con TCP/IP y la interfaz de Sockets. En este ejemplo, se va a considerar que este modelo de clases es el genérico (PIM), y habrá de ser refinado posteriormente.

3.2. Planificación de tareas

Una vez que se tienen los diagramas con los bloques temáticos y los temas o conceptos a tratar, el siguiente paso en la metodología de diseño consiste en organizar de forma secuencial todas aquellas actividades conducentes a su aprendizaje. Para esta planificación se va a recurrir a los Diagramas de Actividades UML, que se añadirán al modelo PIM. La fig. 6 es un ejemplo de diagrama preparado para la enseñanza de los conceptos correspondientes al bloque de TCP/IP. Cada acción está perfectamente identificada por su nombre. A su vez, el flujo de control permite observar cómo se avanza en el temario de forma intuitiva. En el caso de la acción *Modo de empleo*, el control se bifurca para indicar que se pueden realizar dos acciones independientes entre sí, por ejemplo un ejercicio con sockets TCP y otro con UDP.

[2] Este diagrama no pretende ser exhaustivo, y se han eliminado relaciones y conceptos secundarios.

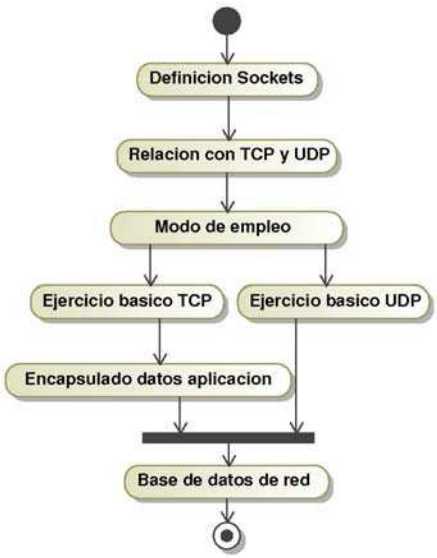

Figura 6: Planificación del bloque temático de software con Sockets TCP/IP

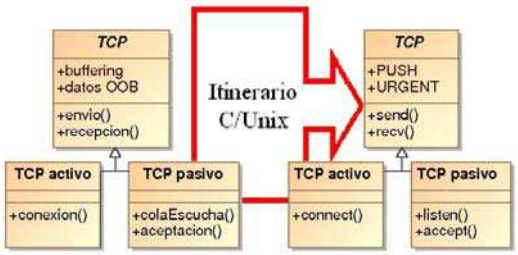

Figura 7: Aplicación de parte del itinerario de C/Unix a los conceptos de Sockets

3.3. Obtención de asignaturas específicas

Definiremos un itinerario formativo como un proceso de transformación parametrizable que se aplica a un diseño de asignatura para obtener otro de un nivel de concreción mayor. El profesor debería preparar distintos itinerarios formativos para po-

der aplicarlos en el caso apropiado. Cada itinerario formativo, por tanto, se traduciría en una plantilla de transformación MDA, que trabajaría sobre modelos conformes al metamodelo UML. El modelo PIM origen de la transformación contendrá todos los diagramas diseñados anteriormente. El modelo PSM destino surgiría tras las modificaciones realizadas en dichos diagramas. El resultado típico de la aplicación de un itinerario formativo incluiría la inserción, modificación o borrado de acciones en los diagramas de actividades, de bloques temáticos, contenidos y explicaciones. En una asignatura de Software de Comunicaciones, por ejemplo, se han previsto itinerarios para C/UNIX, C++/ACE [9] y Java. Por ejemplo, la fig. 7 presenta una vista parcial del proceso de aplicación del itinerario formativo C/UNIX al diagrama de conceptos TCP/IP. En él, se advierte como los conceptos básicos de la parte izquierda se convierten en otros a la derecha, para indicar que las explicaciones se centrarán en las llamadas al sistema UNIX de la librería de Sockets (*send, recv, connect, listen, accept*, ...). De la misma forma, los itinerarios han de reemplazar las actividades básicas por acciones concretas (explicaciones, prácticas orientadas al lenguaje y sistema operativo, etc.), refinando también la planificación temporal, y el número de actividades teóricas y de laboratorio.

3.4. Herramientas de soporte

De cara a obtener un soporte automático para la metodología propuesta, se han evaluado las tecnologías disponibles tanto para UML como para MDA. Aunque para el proceso de modelado en UML es válida casi cualquier herramienta comercial o de código abierto, las posibilidades de transformación reducen este conjunto. Eclipse es una de las herramientas que aglutina más desarrollos en relación con MDA, por lo que servirá como base para la construcción de las herramientas que soporten el diseño y generación de asignaturas.

Hay que hacer notar que, de cara a facilitar el trabajo de detección automática de los elementos UML recomendados en este trabajo, se está planteando la posibilidad de implementar un perfil UML. Un perfil está asociado a la idea de *lenguaje específico para un dominio* (DSL), y consiste en marcar elementos UML con un estereotipo, incorporando la semántica adecuada a un dominio de aplicación, en este caso el diseño de asignaturas. El uso de un perfil resultaría muy útil para las herramientas de manipulación de modelos, al incorporar puntos de referencia evidentes sobre los que aplicar, por ejemplo, reglas de transformación de una forma precisa. Para los estereotipos se barajan nombres como *bloque*, *concepto*, *acción docente* o *clase presencial*, entre otros. El uso de perfiles también ha sido propuesto en [3] para contextos de *e-Learning*.

Las herramientas de soporte deberán contar con interfaces para cargar, editar y salvar diseños de asignaturas (en XMI), así como las capacidades para crear itinerarios formativos, donde de forma cómoda se puedan especificar elementos y relaciones a transformar, y sus resultados. Estos itinerarios se deben crear de forma visual, quedando oculto al profesor el proceso de generación de la plantilla de transformación en alguno de los lenguajes admitidos por Eclipse.

4. Conclusiones

Este artículo ha presentado una metodología visual para el diseño curricular de asignaturas relacionadas con el Software de Comunicaciones. El uso de lenguajes e iniciativas bien conocidas dentro de la comunidad software como UML y MDA, permiten trabajar con herramientas populares de soporte para alcanzar los objetivos propuestos en el trabajo. En primer lugar, se elabora el diseño de contenidos generales de una asignatura y su relación con otros bloques temáticos del plan de estudios, en UML. Dichos contenidos son independientes de aspectos tales como el lenguaje de programación o la plataforma de desarrollo y sistema operativo. En una segunda fase, aplicando técnicas de transformación de modelos MDA, se deben obtener los contenidos particulares adaptados, lo que permitirá obtener la planificación de la asignatura y las tareas seleccionadas como actividades (ejercicios para C, C++, Java, u otros). La metodología propuesta genera documentos XMI portables que pueden servir como entrada a futuras herramientas automáticas de visualización y manipulación de asignaturas. El formato visual

resultante es también atractivo para el estudiante, al presentar contenidos y relaciones como mapas de conceptos, así como la planificación del curso. En el artículo se ha introducido el concepto de itinerario formativo, dotado de la flexibilidad suficiente para que tanto el docente como el propio estudiante puedan seleccionar aquel o aquellos itinerarios que mejor cumplan los requisitos o expectativas, en un contexto de aprendizaje activo. El objetivo a corto plazo consiste en elaborar aplicaciones para dar soporte automático al diseño de asignaturas e itinerarios formativos. Como trabajo futuro inminente, se plantea validar los itinerarios con la retroalimentación de los estudiantes al final de curso.

Agradecimientos

El autor agradece el apoyo de la ETSI Informática de la Universidad de Málaga, donde se realiza este trabajo en el contexto del Proyecto de Innovación Educativa PIE07-089, así como las sugerencias de los revisores para la mejora del artículo.

Referencias

[1] Jouault, F., Kurtev, I., *Transforming Models with ATL*, Proc. of the Model Transformations in Practice (MoDELS), 2005.

[2] Marlowe, T., Ku, C., Benham, J., *Design patterns for database pedagogy: a proposal*, ACM SIGCSE Bulletin, vol-37(1), 2005.

[3] Nodenot, T., Marquesuzaa, C., Laforcade, P., Sallaberry, C., *Model based engineering of learning situations for adaptive web based educational systems*, Proc. 13th WWW Alternate, 2004.

[4] Object Management Group, *MDA Guide Version 1.0.1*, 2003.

[5] Object Management Group, *MOF 2.0/XMI Mapping, v2.1.1*, 2007.

[6] Object Management Group, *MOF Query/Views/Transformations (FTF)*, 2008.

[7] Object Management Group, *MOF Version 2.0*, available at www.omg.org/spec/MOF/2.0/, 2006.

[8] Object Management Group, *UML Version 2.1.2*, available at www.omg.org/spec/UML/2.1.2/, 2007.

[9] Schmidt, D.C., Huston, S.,*C++ Network Programming Volume 2: Systematic Reuse with ACE and Frameworks*, Addison-Wesley, 2003.

[10] The Eclipse Foundation, *Eclipse*, available at www.eclipse.org/, 2008.

[11] W3 Consortium, *Extensible Markup Language (XML) 1.0 (Fourth Edition)*, 2006.