Una propuesta para organizar la enseñanza de la Orientación a Objetos

Jesús García Molina, Marcos Menárguez Tortosa y Begoña Moros Valle

Depto. de Informática y Sistemas
Universidad de Murcia
Campus de Espinardo-30071 Murcia
e-mail: (jmolina|marcos|bmoros)@um.es

Resumen

En este trabajo se presenta una propuesta sobre cómo organizar en los planes de estudio de las titulaciones de informática los temas relacionados con la orientación a objetos. Se describen dos asignaturas cuatrimestrales obligatorias de seis créditos: una primera de *Introducción a la Programación Orientada a Objetos* y una segunda sobre *Análisis y Diseño Orientado a Objetos*. Proponemos que la primera sea obligatoria en las tres titulaciones y que la segunda sea obligatoria en Ingeniero en Informática y optativa en las dos titulaciones técnicas. Ambas conforman un curso que proporcionará al alumno una formación básica en el desarrollo de software orientado a objetos. Esta propuesta está en la línea expuesta en el *Computing Curricula 2001* de ACM/IEEE.

1. Introducción

En la década pasada la programación orientada a objetos (OO) se ha consolidado como el paradigma de programación más apropiado para construir software de calidad, sobre todo porque favorece la escritura de código reutilizable y extensible, además de reducir el salto semántico entre los modelos del análisis del problema y el código de la aplicación: los objetos del dominio del problema son objetos de la solución software.

La propuesta curricular *Computing Curricula 2001* de ACM/IEEE [2] reconoce la importancia adquirida por la programación OO desde la publicación de las recomendaciones curriculares de 1991 y añade este área al núcleo de conocimientos básicos de la disciplina, estableciendo que cualquier estudiante de informática debería tener una formación en Programación OO (unidad *PL6* de Lenguajes de Programación), Análisis y Diseño OO y Patrones de Diseño (unidad *SE1* de Ingeniería del Software).

Las directrices generales propias de las titulaciones de informática (B.O.E. de 20 de Noviembre de 1990) no incluyen descriptores relacionados con la OO en ninguna de las materias troncales, de modo que la forma de incorporar a los planes de estudio la enseñanza obligatoria de la OO, es a través de asignaturas obligatorias, tal y como se está haciendo en la mayoría de los planes de estudio aprobados recientemente, como es el caso de los planes de la Universidad Politécnica de Valencia y de la Universidad de Alicante, entre otros. También se puede aprovechar las asignaturas que corresponden a la materia troncal *Ingeniería del Software* para incluir contenidos sobre construcción de software OO.

En nuestra facultad se imparte una asignatura sobre programación OO desde el curso 1991-92, en el que se implantaron los estudios de segundo ciclo. Se trataba de una asignatura de cuarto curso, cuyos descriptores correspondían a la materia troncal *Ingeniería del Software*. En la actualidad, nuestros planes de estudio incluyen la asignatura *Programación Orientada a Objetos* como optativa en las dos titulaciones técnicas (planes de 1994) y como obligatoria en el primer ciclo de Ingeniero en Informática (plan de 1996, con otro nombre). Esta titulación también incluye una asignatura troncal en cuarto curso sobre análisis y diseño OO.

En este trabajo analizamos cómo organizar la enseñanza del paradigma OO en los planes de estudio de informática. Presentamos una propuesta para la titulación Ingeniero en Informática, indicando como trasladarla a las titulaciones técnicas. Esta propuesta es fruto de nuestra experiencia al impartir las asignaturas relacionadas con la OO en nuestra facultad.

El trabajo se organiza del siguiente modo. En la siguiente sección presentamos nuestra propuesta para Ingeniero en Informática. Luego describimos las dos asignaturas que conforman la enseñanza obligatoria sobre OO que proponemos y que actualmente impartimos en nuestra facultad. A continuación comentamos cuál es la situación en el caso de las titulaciones técnicas y en los futuros planes. Finalmente presentamos las conclusiones.

2. La propuesta

Creemos que un ingeniero en informática debe recibir una formación básica en orientación a objetos a través de dos cursos: uno de *introducción a la programación OO* (nos referiremos a él como IPOO) y otro de *análisis y diseño OO* (nos referiremos a él como ADOO). En IPOO se introducen los conceptos básicos que caracterizan al paradigma de programación OO y en ADOO el alumno conoce y aprende a aplicar un proceso software OO y los patrones de diseño básicos. IPOO se desarrollaría como una asignatura de tercero de seis créditos, una vez que el alumno conoce la programación procedural y modular, mientras que ADOO se cursaría como una asignatura de cuarto curso también de seis créditos. Es recomendable que ambos cursos no se impartan en el mismo año ya que es bueno que el alumno tenga tiempo para madurar esa nueva visión de la programación que le muestra IPOO.

Desde hace cinco años, nuestro departamento sigue este enfoque en los estudios de Ingeniero en Informática, a través de la asignatura obligatoria *Diseño de Programas* en tercero (corresponde a IPOO) y de la troncal *Arquitectura del Software* de cuarto (corresponde a ADOO). En este trabajo utilizaremos los nombres IPOO y ADOO, en vez de los nombres de estas asignaturas, para insistir en el carácter general de nuestra propuesta.

La organización de IPOO presupone un enfoque *procedural-primero* para la introducción a la programación del primer año, por los motivos que se exponen en [5]. En la mencionada propuesta curricular de ACM/IEEE [2] se señala que es posible idear planes en los que la introducción a la programación puede ser abordada tanto a través del paradigma imperativo como del paradigma OO o incluso del funcional, siendo una cuestión a debatir en cada facultad. En [5] se defiende comenzar con el paradigma imperativo, aunque con un enfoque en el que los conceptos de secuencia e inducción juegan un papel central, así como el de tipo abstracto de datos.

Proponemos pues seguir un enfoque que podemos llamar *evolutivo*, en el sentido que pasamos de la programación procedural a la modular y luego aterrizamos en la programación orientada a objetos. En primer año una introducción a la programación tal y como la descrita en [5]; en segundo curso el alumno aplicaría la programación modular dentro una asignatura anual de algoritmos y estructuras de datos, antes de cursar las dos asignaturas mencionadas arriba IPOO y ADOO que en su conjunto constituyen un curso completo sobre construcción de software OO que incluirían todos los contenidos sobre OO considerados como básicos en la recomendación curricular de ACM/IEEE [2].

B. Meyer critica este enfoque evolutivo al considerarlo fruto de una visión equivocada del profesor que cree necesario enseñar la programación siguiendo el orden en que él ha conocido los paradigmas, y se pregunta por qué no empezar con los objetos desde el primer año, dado que todos estamos de acuerdo en que la OO es el medio adecuado para construir software. Aunque reconocemos el papel de B. Meyer en el desarrollo de la OO y sus grandes aportaciones, después de reflexionar sobre esta cuestión y tras una experiencia de más de diez años creemos que nuestro enfoque *procedural-modular-OO* es el camino adecuado para abordar la enseñanza de la programación, ya que el alumno puede valorar mucho mejor las ventajas de la OO y además es más natural el paso *procedural-OO* que al revés. De hecho, en la primera parte del libro [7], B. Meyer justifica las ventajas de la OO partiendo de la experiencia del lector en programación

procedural, en el manejo de rutinas y módulos, y en el conocimiento del diseño descendente.

Lógicamente, el curso IDOO es un prerrequisito para el curso ADOO. En estos dos cursos, el alumno recibe una formación que le capacita para estudiar otras temas relacionados con la tecnología del software OO como es la construcción de aplicaciones basadas en objetos distribuidos (RMI y CORBA) y el desarrollo basado en componentes. Nuestro departamento oferta en quinto curso dos asignaturas relacionadas con estas dos tecnologías.

Conviene señalar que en las titulaciones técnicas se sigue el enfoque evolutivo comentado, pero no en la titulación superior ya que dos de las cuatro asignaturas de programación que hay entre primero y segundo no están adscritas a nuestro departamento y en ellas se utiliza Java para la enseñanza de los tipos abstractos de datos y estructuras de datos. En el siguiente apartado comentaremos este hecho. A continuación describiremos los cursos IDOO y ADOO que proponemos.

3. Introducción a la Programación Orientada a Objetos

El objetivo de esta asignatura es introducir al alumno en el paradigma de programación OO, siendo sus objetivos: i) describir los conceptos que lo caracterizan: clase, objeto, herencia, polimorfismo y ligadura dinámica, ii) contrastar cómo esos conceptos se reflejan en los cuatro lenguajes OO más extendidos: C++, Java, Smalltalk y Eiffel, iii) enseñar un lenguaje OO y un entorno de programación, iv) introducir algunas técnicas y heurísticas básicas de programación OO y v) valorar en qué medida las técnicas OO mejoran la calidad del software.

La asignatura se plantea como dos cursos que se imparten en paralelo desde la primera semana: una parte teórica de treinta horas y una parte práctica con quince sesiones de hora y media en el laboratorio. En las clases prácticas el alumno recibe un curso de programación OO en Java. Este curso se imparte de forma simultánea a las clases de teoría, ya que no es viable llevar las clases prácticas al ritmo de las clases teóricas. Esto no crea confusión en el alumno, ya que las clases teóricas sirven para proporcionar una visión más general de los conceptos que previamente ha visto en las clases prácticas en el contexto de Java. Al final del curso, las dos partes han debido servir para conseguir los objetivos y que el alumno tenga unos buenos conocimientos sobre programación OO, dos partes separadas que forman una unidad coherente.

En las clases teóricas describimos con detalle todos los conceptos que caracterizan al paradigma OO. Primero se estudian todos los conceptos relacionados con clases y objetos: ocultación de información, mensajes, semántica referencia, creación y copia de objetos, igualdad e identidad, y genericidad. Después se estudia un capítulo sobre el *diseño por contrato* que sirve para contrastar los mecanismos de asertos y excepciones de Java y Eiffel. A continuación se estudian todos los aspectos que tienen que ver con la herencia: principio de sustitución, polimorfismo y ligadura dinámica, intento de asignación, genericidad en Java, clases abstractas, código genérico, herencia y ocultación de información, herencia múltiple, herencia repetida, diferentes tipos de herencia, e implementación de la ligadura dinámica. En el estudio de la herencia también se incluye la descripción de algunos patrones básicos como son: *Template Method* (para ver el papel de las clases parcialmente diferidas para escribir código genérico), *Iterador* (distinguiendo entre iteradores externos e internos, analizando cómo resolver con herencia la imposibilidad de pasar funciones como argumentos de un método), *Observer* (para ver el papel que pueden jugar las interfaces Java) y *Composite* (como ejemplo de herencia múltiple). Finalmente se valora cómo la OO ayuda a mejorar la calidad del software y se introducen heurísticas básicas para la creación de software OO: algunas de las heurísticas de Riel [8] y los patrones *Controlador* y *Experto* [6].

En un curso de estas características es importante que el alumno no adquiera la visión de la OO ofrecida por un lenguaje particular, sino que comprenda los conceptos que subyacen a la OO de una forma independiente del lenguaje. Para conseguir este objetivo, en la parte teórica primero se explican los conceptos sin recurrir a un lenguaje y luego se muestra el contraste en la forma en que son reflejados en los cuatro lenguajes OO que consideramos. Con este enfoque no se encorseta la visión del alumno a la ofrecida por un lenguaje concreto, sino que

adquiere una compresión de los conceptos de más alto nivel. No cabe duda que el contraste enriquece la visión del alumno, ya que se favorece su actitud crítica y se le prepara para poder programar, sin grandes dificultades, en cada uno de los cuatro lenguajes, ya que conoce los aspectos clave relacionados con la OO en cada uno de ellos. Además, está capacitado para valorar cómo se introducen los conceptos OO en otros lenguajes existentes o que puedan definirse en un futuro (por ejemplo, podría dominar sin dificultad C#).

Para ilustrar la idea sirva el ejemplo de la *ocultación de información*. Al alumno se le muestra su significado y se le justifican sus beneficios, entonces se revisan y se contraponen las variaciones que encontramos en los diferentes lenguajes: atributos públicos en C++ y Java, clases amigas en C++, visibilidad *friendly* en Java, clases anidadas en Java y C++, exportación selectiva en Eiffel, todos los métodos son públicos en Smalltalk ocultación a descendientes en Java y C++, …

En la parte práctica nos hemos tenido que enfrentar al problema de la elección de un lenguaje OO. El lenguaje ideal para la docencia sería aquel que tuviese buenas propiedades, existiesen buenos entornos y abundante documentación, y que además su uso estuviese muy extendido entre las empresas de desarrollo. Desde nuestro punto de vista, que un lenguaje tenga buenas propiedades supone que debería: i) ser OO puro para obligar al alumno a programar dentro del paradigma, ii) reflejar con claridad los conceptos OO, por ejemplo, no parece conveniente poder declarar que un atributo se exporta en modo escritura o tener que declarar los métodos en los que se aplicará ligadura dinámica o no poder declarar métodos privados, iii) ser legible, iv) ser pequeño y con gran potencia expresiva, v) ser tipado estáticamente, para favorecer la legibilidad y fiabilidad. Además, sería conveniente que permitiese escribir asertos para posibilitar la programación por contrato [7] y que incluyese un mecanismo de genericidad. En cuanto a la exigencia de un buen entorno entendemos la existencia de entornos robustos, gratuitos, fáciles de usar y que no necesitasen muchos recursos.

Como es lógico, no existe un lenguaje que cumpla todos los requisitos anteriores. Creemos que Eiffel o Eiffel# podría ser la mejor elección considerando sus buenas propiedades (OO puro, tipado, legible, incluye genericidad, mecanismo de asertos y un entorno robusto), sin embargo está muy poco extendido y no hay buenos entornos gratuitos. Por ello hemos elegido Java que podemos considerar *casi OO puro* ya que el alumno no puede salirse del paradigma OO al programar (aunque no es OO puro ya que hay una distinción entre tipos básicos y clases, un método *main* en las clases y otras cuestiones), es tipado estáticamente, refleja con sencillez los conceptos OO si lo comparamos con C++, existe abundante documentación y sobre todo está muy extendido en la industria, habiendo surgido alrededor de él una importante tecnología para el desarrollo de aplicaciones OO: Servlet, RMI, Beans, EJB, JINI,.. Con Java tenemos un lenguaje que no tiene tan buenas propiedades como Eiffel (por ejemplo, no tiene genericidad, ni herencia múltiple) pero a cambio se trata de un lenguaje muy extendido que es el lenguaje más utilizado en el desarrollo de aplicaciones web. Además, el alumno seguirá trabajando con Java en asignaturas posteriores que traten de tecnología del software, por ejemplo las relacionadas con componentes y objetos distribuidos.

Hasta hace dos años, Smalltalk fue el lenguaje elegido por tratarse de un lenguaje OO puro, muy simple pero de gran potencia expresiva y con buenos entornos de libre distribución que consumen pocos recursos. Nunca nos planteamos C++ por tratarse de un lenguaje híbrido, muy complicado y poco legible. Al aparecer Java, y a pesar de su rápido éxito, no apostamos por él ya que no era un lenguaje estable, no se disponían de buenos entornos y por la resistencia al cambio. Entendíamos que Smalltalk permitía cumplir los objetivos y teníamos mucha experiencia y documentación. Sin embargo, ahora no dudamos que Java es la mejor elección.

Como trabajo práctico, el alumno debe realizar un boletín de ejercicios, de complejidad creciente, en los que debe demostrar su conocimiento de aspectos propios de la programación en Java, (entrada/salida, excepciones, concurrencia, serialización e interfaces gráficos), así como de los conceptos de la OO y el manejo de las clases básicas de la jerarquía como son las colecciones. Estos ejercicios obligan al alumno a diseñar e

implementar sus primeras clases para resolver problemas concretos, a ver los programas como una colección estructurada de clases. Finalmente debe desarrollar un pequeño proyecto de programación que abarque de una forma global todos los conceptos estudiados en los ejercicios, en el que se puedan aplicar algunos patrones y heurísticas como la *separación modelo-vista*, el patrón *Observer* y los patrones de reparto de responsabilidades *Controlador* y *Experto* [6]. En los dos últimos cursos hemos encontrado adecuado como proyecto de programación la implementación de un juego en el que varios personajes interactuan en un escenario que impone unas reglas. La implementación exige diseñar las clases que representan el escenario y una jerarquía interesante para representar los personajes del juego (con clases abstractas, código genérico, interfaces,...), y utilizar unos patrones sencillos de colaboración entre objetos.

Conviene señalar que en este curso se cubren todos los temas incluidos en PL6 (*Object-Oriented Programming*) dentro del *Computing Curricula 2001*, así como los objetivos de aprendizaje que allí se plantean.

Las encuestas y comentarios de los alumnos muestran su rechazo a la utilización de Java en dos asignaturas de primer y segundo curso, dedicadas fundamentalmente a trabajar con estructuras de datos y el concepto de tipo abstracto de datos. Los alumnos se quejan de que deben utilizar los conceptos OO sin conocerlos, actuando a base de recetas, "aquí hay que poner *Object*", "aquí debes hacer una conversión de tipos", "las interfaces son un tipo de clases". Reconocen que el conocimiento que tienen de Java no les ha sido de mucha ayuda a la hora de cursar IPOO, ya que tenían pocos conceptos claros. Estas opiniones refuerzan nuestra idea de empezar con el paradigma procedural en el primer curso.

4. Análisis y Diseño OO

Este curso es una continuación del anterior y se dedica a proporcionar al alumno una formación que le permita abordar de una forma sistemática el desarrollo de aplicaciones OO, aplicando un proceso software. Se supone una parte teórica y otra práctica de tres créditos cada una. En las clases teóricas, primero se describe el *Lenguaje Unificado de Modelado*, *UML* como notación estándar para el modelado OO, luego se describe un proceso basado en UML y finalmente se presentan y discuten los patrones de diseño de Gamma et al. [3].

No cabe duda que actualmente el modelado OO está centrado en UML que se ha convertido en un estándar "de facto". En este curso se realiza una descripción detallada del lenguaje UML, analizando con detalle el modelado de casos de uso, modelado estructural, modelado del comportamiento y modelado dinámico.

Tras la presentación de UML, se describe un proceso para UML orientado al desarrollo de aplicaciones de gestión (*business applications*). El proceso presentado es simple y está destinado a aplicaciones cuyo desarrollo no involucra equipos de programadores ni tiempos de desarrollo muy grandes. Se trata de un proceso iterativo, incremental y guiado por los casos de uso, que es resultado de añadir una etapa de modelado de negocio al proceso descrito por Craig Larman [6]. El modelado de casos de usos y el modelado conceptual se realiza a partir de los diagramas de actividad del modelado del negocio siguiendo las técnicas descritas en [4] para identificar casos de uso, vocabulario del sistema y reglas de negocio.

Dentro de una formación básica en orientación a objetos es obligado incluir los *patrones de diseño* o soluciones a problemas recurrentes en el desarrollo de software OO. Los patrones han adquirido bastante importancia en la comunidad software, siendo fundamentales para obtener una arquitectura software de calidad. En la actualidad, existe un consenso al considerar que los patrones de diseño descritos en el libro *Design Patterns* [3] son parte del núcleo básico de conocimientos de informática que un estudiante debe conocer, como se señala en el *Computing Curricula 2001*. Casi la mitad de la parte teórica de la asignatura se dedicará al estudio detallado de estos patrones, planteándose ejemplos que implican identificar qué patrones conviene utilizar.

Dentro de esta parte teórica también se introduce el problema de la persistencia en aplicaciones OO, analizándose con detalle el almacenamiento de objetos en bases de datos relacionales y presentándose el diseño de un nivel de persistencia [1,6]. Este diseño también sirve para ilustrar el uso de patrones.

Las clases prácticas se dedican al principio a resolver problemas de modelado de casos de uso, modelado estructural y el diseño de colaboraciones entre objetos (un total de 5 sesiones de una hora). Es bien sabido que la parte más difícil del modelado OO es diseñar cómo deben colaborar un grupo de objetos para realizar una determinada actividad [3, 6], por lo que se le presta especial atención. Más adelante se describe como aplicar el proceso software estudiado en clase sobre un ejemplo concreto; por ejemplo, se parte de la especificación de la práctica del curso anterior (tres sesiones de una hora). Luego se describe la herramienta que el alumno utilizará para el modelado, *Rational Rose* en nuestro caso (esto conlleva tres sesiones de hora y media). Finalmente se le plantea al alumno una especificación de un problema que debe modelar hasta llegar a una implementación en Java.

El trabajo práctico ideal para esta asignatura sería un proyecto de desarrollo de una aplicación para una empresa, desde el modelado del negocio hasta la implementación. El profesor entregaría al alumno una especificación de un sistema y el alumno debería primero hacer el modelado del negocio y luego aplicar el proceso descrito en [6]. Sin embargo, esto no tiene cabida en una asignatura cuya carga práctica son tres créditos por lo que la implementación se limita a unos pocos casos de uso y no se presta importancia a la interfaz gráfica y a la persistencia en bases de datos relacionales.

Los alumnos forman grupos de dos y el profesor mantiene dos reuniones con los alumnos para seguir el proyecto, una al acabar el modelado de casos de uso y modelado conceptual, y otra al modelar una serie de colaboraciones que son elegidas por el profesor. El objetivo de la primera reunión es asegurarnos que el alumno ha comprendido bien los requisitos, ha identificado una colección de casos de uso apropiada y los ha descrito bien, haciendo uso de la plantilla utilizada en [6]. El objetivo de la segunda entrevista es comprobar que los alumnos han comprendido bien cómo diseñar colaboraciones entre objetos. Esta actividad es la que les resulta más complicada y a la que dedican mayores esfuerzos.

El modelado del negocio es opcional para los alumnos, ya que involucra un tiempo considerable y preferimos llegar a la implementación de alguna parte del sistema, aunque son bastantes los grupos que lo desarrollan ya que lo encuentran útil para identificar los casos de uso. El alumno entrega una documentación que refleja cada etapa de aplicación del proceso y cada grupo mantiene una entrevista final con el profesor en la que se evalúa el trabajo práctico en su conjunto.

Conviene señalar que este curso cubre los temas y objetivos de aprendizaje incluidos en SE1 (*Software Design*) del *Computing Curricula 2001* que tienen que ver con el análisis y diseño OO y patrones de diseño: "seleccionar y aplicar patrones de diseño apropiados en la construcción de una aplicación", "crear y especificar el diseño software para un producto software de tamaño medio, usando un método de desarrollo de software y notación apropiada.

En nuestro caso, este curso también permite que los alumnos puedan contrastar la aplicación de un proceso software OO con el análisis y diseño estructurado que ya conocen de la asignatura *Fundamentos de Ingeniería del Software* de tercer curso. Un aspecto que los alumnos valoran en gran medida es la trazabilidad que ofrecen los casos de uso dentro de los procesos OO: se diseñan colaboraciones que satisfacen un objetivo que corresponde a un caso de uso y se implementan clases teniendo en cuenta las colaboraciones, de modo que es fácil llegar del código al requisito que implementa. Los alumnos también valoran positivamente la continuidad entre las diferentes etapas del proceso, además notan como todo el trabajo es de utilidad y está encaminado a escribir código que es lo que hay que acabar haciendo.

5. La propuesta en las titulaciones técnicas y los futuros planes

En las titulaciones técnicas creemos necesaria una asignatura obligatoria que corresponda a IPOO en el primer cuatrimestre del tercer curso. Se podría incluir una optativa en el segundo cuatrimestre de este tercer año con los contenidos de ADOO, aunque nos parece que el proyecto de la parte práctica y el estudio de los patrones no puede ser tan exigente como el caso de la asignatura de cuarto curso, ya que el alumno tiene menos tiempo para madurar los conceptos.

Actualmente, en las titulaciones de Sistemas y Gestión de nuestra facultad, nuestro departamento

oferta una asignatura optativa denominada *Programación Orientada a Objetos* (6 créditos) que coincide con IPOO, aunque con algunas diferencias en el planteamiento de las prácticas. Se exige un menor número de ejercicios individuales y se propone como proyecto de programación una aplicación de gestión, guiándoles en la implementación, aunque sin aplicar un proceso. Este enfoque es motivado por el hecho que la asignatura no tiene continuidad en otra de la naturaleza de ADOO. Siempre se elige una especificación que al menos exija la definición de una jerarquía de clases y la existencia de clases parcialmente diferidas con código genérico (patrón *Template Method*). Se introducen los diagramas de clase de UML y se insiste en la aplicación de heurísticas básicas de diseño OO como no incluir código de las clases del modelo en las interfaces y el reparto de responsabilidades entre clases.

En los futuros planes, que se implantarán el próximo curso, en Ingeniero en Informática se mantiene la organización propuesta en este trabajo, con dos asignaturas que corresponden a IPOO y ADOO en tercer y cuarto curso, respectivamente; además se mantienen las optativas de quinto curso sobre objetos distribuidos y componentes. En cuanto a las titulaciones técnicas, la asignatura *Programación Orientada a Objetos* será obligatoria en tercer curso de Sistemas y Gestión. Además, se ha incluido en la oferta de optativas una asignatura que corresponde a la descripción de ADOO. La organización refleja la propuesta presentada por los autores al departamento sobre la enseñanza de las materias relacionadas con la OO.

6. Conclusión

Hemos presentado una propuesta de organización de la enseñanza de las materias relacionadas con la OO: una asignatura obligatoria de seis créditos en tercer curso (titulaciones técnicas e Ingeniero en Informática), destinada a ofrecer una introducción a los conceptos que configuran el paradigma OO y una asignatura, también de seis créditos en cuarto curso (como optativa de tercero en las técnicas) para describir un proceso basado en UML y los patrones de diseño básicos. Ambas asignaturas conforman un curso de doce créditos sobre OO, con el que los alumnos conseguirán una buena formación en el desarrollo de software OO y que coincide con la propuesta curricular de *Computing Curricula 2001*. Además, estos dos cursos preparan al alumno para estudiar otros aspectos de la tecnología del software como son las aplicaciones basadas en objetos distribuidos y el desarrollo basado en componentes.

Este enfoque de la enseñanza de la OO, es fruto de la colaboración, en los últimos cuatro años, de los autores de este trabajo al encargarse de la enseñanza de las asignaturas relacionadas con IPOO y ADOO, partiendo de la experiencia del primer autor que hace once años comenzó a impartir la docencia relacionada con la OO en nuestra facultad.

Referencias

[1] S. Ambler, *Mapping Objects to Relational Databases,* http://www.ambysoft.com/mappingObjects.html

[2] *Computing Curricula 2001*, Final Report, December 2001, ACM e IEEE, http://www.computer.org/education/cc2001/final/index.htm

[3] E. Gamma et al., *Design Patterns, Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[4] J. García Molina et al. *Towards Use Case and Conceptual Models Through Business Modeling*, ER2000: 19th International Conference on Conceptual Modeling, Utah, USA, 2000. Versión en castellano *Un proceso basado en UML para aplicaciones de gestión* en http://dis.um.es/~jmolina/as.html

[5] J. García Molina, *¿Es conveniente la orientación a objetos en un primer curso de programación?*, VII Jornadas de Enseñanza Universitaria de la Informática (JENUI'2001), Palma de Mallorca, 16-18 de Julio, 2001. Puede descargarse en http://dis.um.es/~jmolina/publi.html

[6] C. Larman, *Applying UML and patterns*, 2ª edición,Prentice-Hall, 2002

[7] B. Meyer, *Construcción de Software Orientado a Objetos,* 2ª edición,Prentice-Hall, 1998.

[8] A. Riel, *Object-Oriented Design Heuristics,* Addison-Wesley, 1996.