Inclusión de patrones de diseño en un plan de estudios de Ingeniería Técnica en Informática de Gestión

Carlos López Nozal, Raúl Marticorena Sánchez, Judith Antolín Sendino, Ignacio Cruzado Nuño
Escuela Politécnica Superior
Departamento de Ingeniería Civil
Área de Lenguajes y Sistemas Informáticos
Universidad de Burgos
e-mail: {clopezno, rmartico, jantolin, icruzado}@ubu.es

Resumen

El presente trabajo define una propuesta de la inclusión del concepto de patrón de diseño en la titulación de Ingeniería Técnica en Informática de Gestión (ITIG), estableciendo los niveles de desarrollo a tratar en las asignaturas *tipo* de esta titulación.

La utilización de patrones de diseño tiene dos aspectos claramente diferenciados: uno relacionado con la aplicación del patrones y otro relacionado con la búsqueda y selección de un patrón que resuelva un determinado problema. El trabajo se basa en esta diferenciación de aspectos para distribuir las responsabilidades entre las diferentes asignaturas del plan de estudios.

El objetivo final del mismo, es mostrar la experiencia previa de dicho trabajo en la Universidad de Burgos y concluir con una propuesta para la incorporación inmediata y adecuada de dichos temas.

1. Presentación

La importancia de los patrones de diseño hoy en día es evidente, sin embargo, la actual definición de planes de estudio, en la mayoría del ámbito nacional, no señala su inclusión, provocando un vacío evidente a nuestro juicio.

Por otro lado, la última actualización del *Computing Curricula 2001* [1], incorpora en muchas de sus diferentes propuestas de cursos de programación la unidad *SE1.Software Design*[1], dentro del núcleo de dichas asignaturas. Esta unidad incorpora el concepto de patrones de diseño para elevar el nivel de abstracción en el planteamiento de soluciones.

Siguiendo esta recomendación se pueden asignar responsabilidades concretas a las asignaturas implicadas en la enseñanza de patrones, con el fin de conseguir el objetivo de aprendizaje de *"selección y aplicación de los patrones de diseño adecuado en la construcción de una aplicación software"*[1].

El objetivo es cubrir las deficiencias actuales estableciendo los cursos y nivel de profundidad en el desarrollo docente de los patrones, de tal forma que su inclusión en un plan de estudios no sea un cambio traumático, sino natural.

La estructura del documento, consiste en primer lugar, en el planteamiento del contexto particular en el que hemos llevado a cabo este proceso, en nuestro caso bajo el plan de estudios de ITIG en la Universidad de Burgos, para posteriormente detallar la solución implantada en cada una de las asignaturas correspondientes, indicando la responsabilidad específica y cómo se realiza en cada asignatura. Finalizamos con las conclusiones y experiencias obtenidas de la realización de dicho proceso.

2. Contexto propio

Para comprender la propuesta, es necesario conocer el actual contexto en el que se está llevando a cabo.

En el plan de estudios de ITIG en la Universidad de Burgos, vigente desde el curso 1995/1996, se identifican tres asignaturas que pueden estar implicadas en la docencia de patrones: *Metodología de la Programación*,

[1] SE significa el área del Software Engineering y el número 1 la unidad dentro del área.

Análisis e Ingeniería del Software y *Programación Avanzada*.

Dentro del segundo curso de la titulación se imparten las asignaturas troncales de *Metodología de la Programación* y *Análisis e Ingeniería del Software*. En el tercer curso se imparte la asignatura de *Programación Avanzada* de carácter optativo, con claras dependencias de los contenidos vistos en las asignaturas anteriormente mencionadas.

La propuesta se basa en introducir los aspectos relacionados con la aplicación de patrones de diseño en las asignaturas de segundo curso e incidir en búsqueda y selección de patrones en la asignatura de tercer curso. La causa de esta distribución de aspectos viene dada por la necesidad de un conocimiento exhaustivo de un catálogo de patrones para poder realizar el proceso de búsqueda y selección de los mismos. Este requisito no puede ser considerado en las asignaturas de segundo curso por tener que satisfacer otros múltiples objetivos.

La razón de introducir el concepto de patrón en dos asignaturas del segundo curso es debida a la distintas técnicas utilizadas para presentar los elementos descriptivos de un patrón. En la Tabla 1 se muestran dos plantillas descriptivas de un patrón especificadas en [2], [3]. En ella se han sombreado los elementos relacionados con el conocimiento concreto de un lenguaje de programación orientado a objetos. Estos elementos sólo pueden ser tratados en asignaturas orientadas a programación, *Metodología de la Programación* y *Programación Avanzada*.

Si establecemos una equivalencia con lo que se entiende por programación de nivel[2] II y III, se corresponderían con *Metodología de la Programación* y *Programación Avanzada* respectivamente. Por otro lado nos referiremos a la asignatura de *Análisis e Ingeniería del Software* como ingeniería del software. Señalar que uno de los problemas que surge de este planteamiento es el carácter optativo de la asignatura de programación de nivel III, que dificulta el llevar a cabo el proceso completo para todos los alumnos, como hubiese sido la intención de los autores[3].

Plantilla de GoF[4]	**Plantilla de Grand**
Nombre y clasificación	Nombre
Propósito	Sinopsis
También conocido	
Motivación	Contexto
Aplicabilidad	Fuerzas
Estructura	Solución
Participantes	
Colaboraciones	
Consecuencias	Consecuencias
Implementación	**Implementación**
Código de ejemplo	
Usos Conocidos	**Java API**
Patrones Relacionados	Patrones Relacionados

Tabla 1. Plantilla de patrón de diseño.

Para finalizar el flujo de dependencias, señalar también la existencia de un trabajo final de carrera en el que actualmente en el 90% de los casos, se están utilizando lenguajes orientados a objetos (Java y C++) para la parte de programación. Una vez que nuestra propuesta esté implantada, es en esta asignatura donde se puede observar finalmente el éxito o fracaso de la misma. Desgraciadamente dado lo reciente de la experiencia y el pequeño nº de proyectos en los que se ha empezado a aplicar este planteamiento (aunque el objetivo final es abarcar, en mayor o menor medida, la práctica totalidad de proyectos propuestos por el Área de Lenguajes y Sistemas Informáticos) las observaciones obtenidas no son relevantes todavía.

3. Propuesta realizada

La propuesta que aquí se realiza consiste en presentar al alumno el concepto de patrón de diseño y ejemplos de uso en la asignatura de programación de nivel II y en la asignatura de ingeniería del software, mientras que en la asignatura de programación de nivel III se hace un estudio más completo del catálogo de los mismos,

[2] Se entiende por programación de nivel II y III a las asignaturas de programación vistas en el segundo y tercer año de la titulación técnica.

[3] Este posible vacío esperamos que sea revisado en la próxima elaboración del plan de estudios que se está llevando a cabo actualmente.

[4] Gang of Four

razonando sobre su selección y utilización para la resolución de problemas generales.

La mayor parte de la bibliografía referente a patrones [2], [3], [5], [7], [8], utiliza alguna plantilla descriptiva, como la mostrada en Tabla 1 o equivalente, cuyo enfoque se fundamenta principalmente en satisfacer el proceso de aplicación del mismo.

Sin embargo, y por regla general, la bibliografía se limita a utilizar el patrón en un contexto particular. Un elemento clave de la plantilla que interviene en el proceso de búsqueda es el de patrones relacionados, que se describe de forma textual y necesita del conocimiento de un catálogo de patrones. El método didáctico seguido para poder mostrar este proceso de búsqueda y selección es a través de un caso de estudio tal como se propone en el capítulo 2 de [2]. En particular, lo que tratamos de plantear y resolver es precisamente ese vacío existente, que por regla general nos han transmitido los alumnos en otros trabajos [6], existiendo un cierto desánimo a su utilización posterior.

A continuación vamos a detallar las distintas estrategias adoptadas en cada una de las asignaturas para llevar a cabo este proceso.

4. Patrones de diseño en ingeniería del software

La responsabilidad asignada a esta asignatura en lo referente a patrones de diseño se basa fundamentalmente en la aplicación de los mismos.

En las clases de teoría se expone el proceso de desarrollo propuesto por Larman en [5] compuesto por varias iteraciones de las fases de análisis, diseño y construcción. Es en la fase de diseño donde se introducen los patrones, en concreto se trabaja con los patrones propuestos por el autor y recogidos en [4] como catálogo de patrones GRASP (General Responsibility Assignment Software Patterns). Además se exponen algunos patrones aislados del catálogo de propuesto en [2] (Singleton, Fachada, Observador, Comando, etc.).

Para la exposición teórica del patrón se utiliza una descripción de los elementos esenciales: intención, contexto, solución y consecuencias. Una vez expuesto cada patrón, se aplica a un caso práctico guiado, que sirve como referencia para poder obtener los artefactos propuestos en el proceso.

En la parte práctica de la asignatura se propone un colección de requisitos de un sistema a partir de los cuales se deben obtener los diferentes artefactos presentados en teoría. Para poder obtener los artefactos de diseño y las posibles alternativas se deben aplicar los patrones expuestos. Aunque existe un pequeño proceso de búsqueda sobre el catálogo GRASP, no se puede considerar suficiente por el escaso número de patrones que contiene este catálogo. Además los patrones utilizados son encontrados casi sistemáticamente por el hecho de proponer contextos muy similares al caso práctico mostrado en teoría.

5. Patrones de diseño en programación de nivel II

La asignatura de programación de nivel II, es la primera asignatura en el contexto particular de nuestro plan de estudios en que se introduce al alumno en el paradigma de la programación orientada a objeto (POO).

Aunque dicha inclusión puede llevar a discusiones, en nuestro caso particular y dadas las dependencias existentes en el plan de estudios actual, este hecho es necesario para completar los objetivos de la formación del alumnado.

Además esto se ve confirmado por otras fuentes como la obligatoriedad de un módulo de POO en la formación obligatoria por parte de *Computing Curricula Computer Science* [1].

El lenguaje elegido para la realización de prácticas es Java por cumplir en su mayoría, las características propias un lenguaje orientado a objeto puro[5], con una gran disponibilidad de bibliotecas, entornos y documentación. Además la facilidad de comprensión del lenguaje permite un rápido desarrollo de las prácticas.

El planteamiento en la parte de teoría es presentar al alumno el uso particular de algún patrón de diseño en contextos muy particulares. Siempre explicando el nombre, problema, solución y consecuencias de la aplicación del patrón. En el desarrollo teórico de la asignatura se aprovechan las múltiples circunstancias en las que

[5] Siempre criticable esta afirmación por la inclusión de tipos primitivos en el lenguaje.

el empleo de patrones de diseño resuelve nuestros problemas. Como ejemplo tenemos la utilización de patrones en la categoría de patrones esenciales como Interface y Delegación [3] para la simulación de herencia múltiple con lenguajes que no la soportan. El planteamiento docente es primero mostrar y usar, para después razonar más detenidamente sobre el sentido de reutilización de diseño que nos están aportando.

Como parte del temario, finalizamos en los últimos temas definiendo formalmente la idea de patrón y recapitulamos sobre el empleo "inconsciente" (pero siempre guiado por el profesor) de los patrones para la resolución de un problema general en un contexto particular. Una vez presentado el concepto se puede finalizar con el estudio más detallado de algún patrón concreto, con fuerte uso de las características propias de la orientación a objeto, de esta forma, el alumno confirma y refuerza los conceptos teóricos, viendo una aplicación a posteriori más allá del simple ámbito de la asignatura.

5.1. Práctica planteada

El enfoque en prácticas para la asignatura de programación de nivel II consiste en la resolución de un problema general a través de la utilización de uno o varios patrones de diseño. La finalidad es, de manera transparente, comenzar a trabajar con patrones con un lenguaje orientado a objeto.

En particular, la práctica realizada en el presente curso, plantea la búsqueda de un elemento concreto dentro de estructuras compuestas, que a su vez pueden contener a nuevas estructuras compuestas u objetos finales. Ésto forma árboles en los que la raíz y nodos intermedios son composiciones y los elementos finales son hojas.

Dichas estructuras se encuentran en un gran número de ejemplos de la vida real, como las estructuras de directorios y ficheros, las interfaces gráficas con paneles y componentes gráficos, los paquetes y las clases en Java, etc.

El patrón que se adapta de manera natural a la resolución de dicho problema es el patrón Compuesto[6] [2], [3]. La jerarquía de herencia se establece entre lo que denominamos clases Elemento y dos descendientes directos, Composición y Hoja.

En la Figura 1 podemos ver la solución planteada, señalando que el diseño se proporciona a los alumnos.

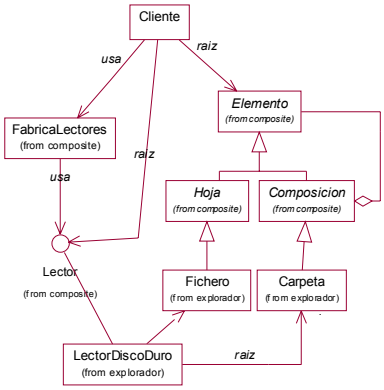

Figura 1. Diagrama de clases del marco de acceso a composiciones de elementos.

Una vez determinada la solución de alto nivel, se solicita que utilizando los mecanismos propios de herencia en Java, se reutilice parte de la implementación realizada (concentrada en la implementación del patrón Compuesto) para resolver problemas concretos en un ambiente más real como la búsqueda de ficheros en directorios.

Para facilitar la labor al alumno, la lectura y carga de directorios y ficheros se simula con una clase *LectorDiscoDuro*, con lo que la realización de la práctica se centra en la resolución del problema inicial, evitando el desviarnos con el uso avanzado de alguna API concreta.

El objetivo final de la práctica es que el alumno, además de afianzar los conceptos propios de la POO (herencia, polimorfismo, ligadura dinámica, etc.), enfoque el resultado final a la reutilización mediante la construcción de un marco[7] basándose en uno o varios patrones.

[6] Traducción del inglés Composite

[7] Traducción del término inglés framework

6. Patrones de diseño en programación de nivel III

La experiencia obtenida con la aplicación del plan propuesto en [6], donde se elegía únicamente un patrón, vislumbró que los alumnos eran capaces de aplicar, sin mucha dificultad, el patrón propuesto de forma aislada, pero se encontraban con graves problemas a la hora de crear un sistema utilizando varios patrones relacionados.

Para poder superar esta deficiencia es necesario el conocimiento de un catálogo de patrones que es expuesto en las clases de teoría; en concreto se trabaja con el catálogo propuesto en [2]. El enfoque utilizado en esta exposición se basa fundamentalmente en la aplicación de los patrones.

En la parte práctica es donde realmente se incide en el proceso de búsqueda y selección. Para ello se propone un enunciado donde se especifican unos requisitos de diseño que los alumnos deberán resolver. La metodología didáctica empleada se basa en talleres donde los alumnos discuten y proponen sus soluciones hasta llegar a un diseño por consenso que será utilizado para la implementación del sistema propuesto. En estos talleres el profesor actúa como moderador guiando este proceso de búsqueda y selección de los patrones.

A continuación se describe la materialización práctica de la estrategia mencionada proporcionando el enunciado de la práctica y las soluciones de diseño empleadas.

6.1. Práctica planteada

La práctica consiste en realizar una aplicación Java que permita visualizar un conjunto de imágenes cuyos datos (autor, descripción y nombre de fichero) se encuentran almacenados en un sistema de almacenamiento persistente. La Figura 2 muestra un posible diseño de la interface de usuario para interactuar con los datos. Para la resolución de la práctica se tienen en cuenta una serie de restricciones de diseño:

1. En previsión del crecimiento de la aplicación y para mejorar la reutilización y mantenimiento de la misma, se quiere estructurar su desarrollo con una arquitectura de tres capas (Presentación – Dominio – Servicios) respetando las dependencias mostradas en la Figura 3.
2. En previsión de futuros cambios del sistema de almacenamiento persistente se quiere mejorar la reutilización manteniendo una independencia del mismo.
3. Intentar minimizar el acoplamiento existente entre las clases del paquete presentación para prever una futura reutilización de los mismos.
4. Encapsular las diferentes peticiones que se pueden efectuar desde la interfaz gráfica como objetos.

Figura 2. Interfaz gráfica de la aplicación.

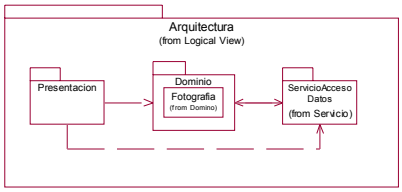

Figura 3. Diseño arquitectónico.

6.2. Soluciones finales de diseño

En este apartado se recogen las soluciones de diseño obtenidas a través de los distintos talleres. El enfoque seguido es ir tratando de forma independiente cada solución seleccionando y buscando patrones que las resuelvan.

- Solución del problema de diseño 1.

El patrón fachada facilita la división de un sistema en capas proporcionando una única clase que permite acceder al resto clases contenidas

dentro del paquete correspondiente. De esta forma se minimiza el acoplamiento de los clientes ya que únicamente tienen que conocer una clase, la fachada, para acceder a los servicios que le pueden proporcionar las clases contenidas en el paquete. Se podría añadir una fachada para el paquete de Dominio y otra para el paquete de ServicioAccesoDatos. Si se analizan las clases del dominio de la aplicación únicamente se identifica la clase Fotografía cuyo tipo se especifica en la Figura 4. Añadir una indirección a través de la fachada para poder acceder a los servicios proporcionados por la clase Fotografía no reporta ningún beneficio, por tanto no se añadirá una Fachada para poder acceder a las clases de dominio.

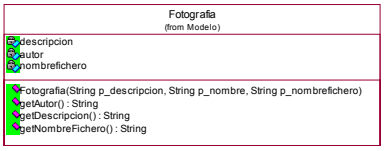

Figura 4. Clase Fotografía.

Para poder acceder a los datos de un sistema de almacenamiento persistente (fichero o bases de datos), se tiene que conocer un conjunto de clases del API (java.io o java.sql), cómo se relacionan y cómo se usan. En este caso sí parece interesante incluir una clase Fachada que permita acceder al sistema de almacenamiento persistente. De esta forma se libera al cliente del conocimiento de las clases del API concreta. En el diagrama de clases de la Figura 5 se muestra un posible diseño de una fachada para acceder a los datos del sistema de almacenamiento persistente, en este caso una base de datos. Hay que observar que la creación de datos del modelo (Fotografía) puede recaer sobre la Fachada o sobre el cliente. Con esta segunda opción se elimina la dependencia de la capa de servicios sobre la capa de modelo a costa de añadir más complejidad en las clases cliente.

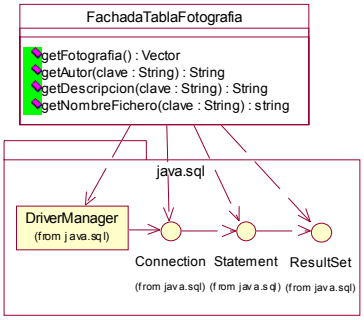

Figura 5. Fachada del servicio de acceso a datos.

Una vez que los datos del modelo están cargados, los objetos de la capa de presentación utilizaran los objetos del modelo para acceder a los datos.

- Solución del problema de diseño 2

Si se quiere soportar diferentes sistemas de almacenamiento persistente se debe establecer una relación de herencia respecto a las fachadas de acceso a datos, donde la superclase define las operaciones abstractas que permiten acceder a los datos y las subclases concretas definen estas operaciones para el sistema de almacenamiento concreto. De esta forma los clientes pueden utilizar objetos de las subclases concretas mediante referencias de la superclase (polimorfismo). En este enfoque la única dependencia que tiene el cliente sobre el sistema de almacenamiento concreto está en el momento de instanciar u obtener la fachada que da acceso al mismo. Para poder eliminar esta dependencia se puede utilizar el patrón de diseño método de fabricación. La Figura 6 muestra la solución estructural de este problema.

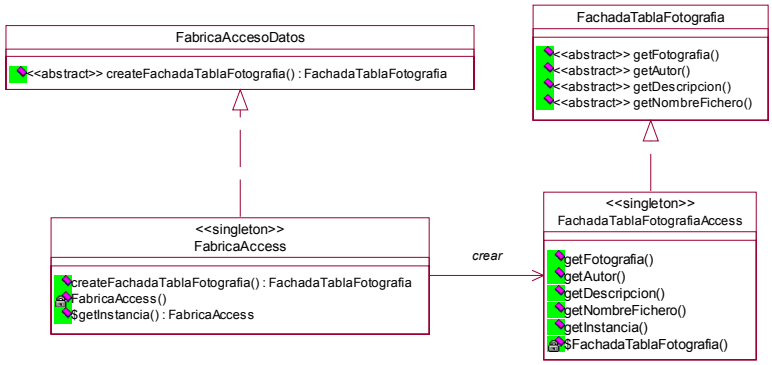

Figura 6. Independencia del sistema de almacenamiento

- Solución del problema de diseño 3.

Cuando se diseñan interfaces gráficas suelen aparecer muchas dependencias entre las clases que contienen componentes gráficos. Estas dependencias hacen que dichas clases tengan un alto acoplamiento dificultando su posterior reutilización. El patrón mediador minimiza el número de dependencias incorporando en el diseño una nueva clase Mediador que se encarga de tratarlas.

En el diseño gráfico presentado en la Figura 2 se identifican los siguientes componentes gráficos:

Figura 7. Panel barra de desplazamiento (PBarraDesplazamiento).

Figura 8. Panel de datos textuales (PDatos).

Figura 9. Canvas para visualizar las imágenes (VisorImagen).

En el diagrama de clases de la Figura 10 se puede observar la aplicación del patrón mediador en este contexto.

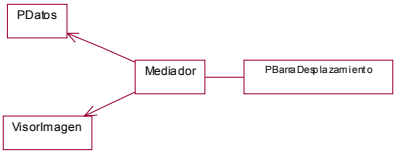

Figura 10. Aplicación del patrón mediador para minimizar dependencias.

- Solución del problema de diseño 4

Las posibles peticiones del usuario al sistema están relacionadas con la forma de obtener los datos. El patrón comando permite encapsular cada

petición como una clase. La Figura 11 muestra un diseño estructural para encapsular estas peticiones.

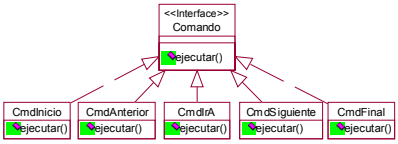

Figura 11. Encapsulamiento de peticiones de la interfaz.

7. Conclusiones

A la vista de los resultados obtenidos planteamos la integración de nuestra propuesta en la titulación de ITIG. Pese a las dificultades de la implantación de dichos temas y a la realización de unas prácticas orientadas, desde un principio, a llevar al alumno a un automatismo en el uso de patrones, los positivos resultados obtenidos nos animan a seguir desarrollando estas ideas, incluso con una labor de coordinación más fuerte entre las asignaturas.

El planteamiento final que realizamos es el mostrado en Tabla 2.

Asignatura	Teoría	Práctica
Ingeniería del software	Presentación del concepto y proceso de búsqueda de patrones en análisis y diseño (6 horas).	Obtención de artefactos de diseño a partir de un conjunto de casos propuestos (6 horas).
Programación de nivel II	Presentación del concepto de patrón y uso de la POO para su implementación (3 horas).	Programación de uno o varios patrones para la resolución de las prácticas planteadas (8 horas).
Programación de nivel III	Presentación de varios catálogos (Gamma & Concurrencia) y ejemplos de uso en contextos determinados (30 horas).	Planteamiento de problemas generales y talleres de búsqueda y selección de patrones para la implementación de la solución del problema (12 horas).

Tabla 2. Propuesta de la planificación de inclusión de patrones de diseño.

La ampliación de esta propuesta inicial puede extenderse introduciendo los patrones en otras muchas asignaturas del plan de estudios. Los patrones homogeneizan la forma de presentar los problemas y sus soluciones. Existen muchos catálogos que pueden ser aplicados en los contenidos de otras asignaturas: catálogos de concurrencia [3] en asignaturas dedicadas a la programación concurrente, catálogos de interfaces [4] en asignaturas dedicadas al diseño de interfaces gráficas de usuario y catálogos de J2EE [8] para asignaturas dedicadas a componentes distribuidos, en este caso especial con Java.

Esta propuesta, teniendo en cuenta las enormes posibilidades abiertas, se puede extender a las asignaturas de un segundo ciclo en Ingeniería Informática, en el caso de que los resultados obtenidos sean positivos.

Como conclusión final creemos cubrir de esta forma el vacío existente en cuanto al tratamiento correcto de los patrones de diseño dentro de una titulación técnica en informática de gestión.

Referencias

[1] ACM & IEEE. *Computing Curricula 2001. Computer Science* (2001). The Joint Task Force on Computing Curricula IEEE Computer Society. Association for Computing Machinery. Ed IEEE-CS y ACM.

[2] Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design Patterns. Elements of Reusable Object-Oriented Software.* Addison Wesley.

[3] Grand, M. (1998). *Patterns in Java. Volume 1*. Wiley Computer Publishing.

[4] Grand, M. (1999). *Patterns in Java. Volumen 2*. Wiley Computer Publishing.

[5] Larman, C. (2002). *UML y Patrones. Una introducción al análisis y diseño orientado a objetos y al proceso unificado*. Ed. Prentice Hall.

[6] Marticorena, R., López, C., García, C. I, Pardo C. (2002). *Aprendizaje práctico de patrones de diseño en asignaturas de programación de nivel III*. Actas JENUI 2002, pág 471.

[7] Metsker, S. J. (2002). *Design Patterns Java WorkBook*. Ed Addison Wesley.

[8] Stelting, S. & Maassen, O. (2002). *Applied Java Patterns*. Ed The Sun Microsystems Press. Java Series.