Patrones
De Alexander a la Tecnología de Objetos

Francisco José García Peñalvo
Licenciado en Informática. Profesor del Departamento de Informática y Automática de la ***Universidad de Salamanca****.*
fgarcia@usal.es

Los patrones (patterns) pueden considerarse como unas entidades destinadas a soportar, documentar y transmitir la experiencia en la resolución de problemas tipo que pueden aparecer en diferentes contextos. Cuando el contexto de los problemas a estudiar es el desarrollo de software nos encontramos dentro de los denominados patrones software. El objetivo que se persigue con los patrones dentro del mundo del desarrollo del software es establecer un catálogo de referencia para ayudar a los ingenieros de software a solucionar o a afrontar problemas de Ingeniería del Software, dando lugar a un lenguaje común en el que comunicar la experiencia y los trucos entorno a dichos problemas y a su solución

El concepto de patrón como elemento reutilizable de experiencia y conocimiento ha calado profundamente en el área del desarrollo de aplicaciones software, teniendo su caldo de cultivo más activo en la comunidad de orientación a objeto. De este hecho se deriva el término patrón software y más concretamente el de patrón de diseño para hacer referencia al uso de patrones en el Diseño Orientado a Objeto (DOO).

La serie de artículos, que sobre patrones software se inicia con el presente trabajo, tiene como objetivo familiarizar al lector con el uso de patrones en el diseño de sus aplicaciones software, consiguiendo un mayor grado de fiabilidad y flexibilidad en sus diseños al reutilizar estructuras ya probadas en otros contextos, siendo además los sistemas software construidos más sencillos de comprender y mantener al estar documentados sobre la base de un conjunto de patrones ampliamente conocidos y difundidos.

No obstante, antes de empezar a trabajar con los patrones en el campo del DOO, y de cara a evitar mal entendidos, mitos o creencias en panaceas que den soluciones a todos los males del software, se va a realizar una primera aproximación[1] a los patrones en la que se va a explicar su origen, su evolución hacia el mundo del software, su significado, la terminología que introducen, así como un conjunto de referencias (tanto bibliográficas como direcciones web) que permitan a los lectores profundizar más en este interesante campo.

Origen e Historia de los Patrones

El concepto de patrón software, más concretamente el concepto de patrón de diseño, se hizo popular dentro del mundo del desarrollo del software en 1995 con la publicación del libro **Design Patterns: Elements of Reusable Object-Oriented Software** [12] cuyos autores son *Erich Gamma, Richard Helm, Ralph Jonhson y John Vlissides* (*normalmente referenciados como la Banda de los Cuatro – The Gang of Four – o simplemente por GoF*). Este libro presenta un conjunto de 23 patrones de diseño de software bajo el paradigma de la orientación a objeto, que van a suponer la aceptación definitiva del concepto de patrón de diseño en el mundo del software y el inicio de una importante comunidad de personas trabajando e investigando en esta área.

De todas formas cabe destacar que ni los orígenes de los patrones, ni el comienzo de su uso en el mundo del software se encuentran en el famoso libro del GoF.

Realmente el uso del término patrón, con el significado que actualmente se le da en la Ingeniería del Software y más concretamente en el área de la tecnología de objetos, se deriva de los trabajos del

[1] En el número 43 se publicó una introducción a los patrones de diseño en la que se presentaba algunos patrones de diseño [15].

arquitecto Christopher Alexander desde mediados de los años 60 (en el Cuadro 1 se recogen unas notas biográficas sobre Alexander).

El trabajo de Alexander se resume en varios libros y artículos sobre planificación de urbanismo y arquitectura de edificios [1, 2, 3, 4] (en el Cuadro 2 se tiene un resumen de algunas de las ideas de la obra del Dr. Alexander). Sin embargo, aún siendo sus publicaciones propias de arquitectura, sus ideas son aplicables a muchas otras disciplinas, entre las que cabe destacar su aplicación en el desarrollo de software.

Las ideas de Alexander fueron utilizadas por Ward Cunningham y Kent Beck para desarrollar un lenguaje de patrones (compuesto por cinco patrones) como guía de iniciación a la programación en Smalltalk, presentando su trabajo en la OOPSLA'87[2].

Posteriormente, Jim Coplien fue recopilando y creando un catálogo de *idioms*[3] propios de C++ que fue posteriormente publicado en 1992 en su libro *Advanced C++ Programming Styles and Idioms* [7].

Christopher Alexander nació en Viena, Austria, pero se crió en Inglaterra, concretamente en Oxford y en Chichester. Estudió Matemáticas y Arquitectura en la Universidad de Cambridge. Obtuvo el título de Doctor en Arquitectura por la Universidad de Harvard. Su Tesis Doctoral fue premiada con la Medalla de Oro para la Investigación por el American Institute for Architecs, posteriormente su Tesis fue publicada en [1].

Desde 1963 es profesor de Arquitectura en la Universidad de California en Berkeley y Director del Institute for Environmental Structure. En 1980 fue elegido miembro de la Real Academia Sueca; en 1996 fue elegido miembro del American Academy of Arts ans Sciencies.

El Dr. Alexander es autor de varios libros y artículos. Ha iniciado un nuevo movimiento en la forma de pensar en Arquitectura, de forma que las mismas leyes determinan la estructura de una ciudad, de un edificio o de una simple habitación. Su último libro es *The Nature of Order*, que está actualmente en preparación para su publicación. Esta obra de cuatro volúmenes recoge sus investigaciones sobre la abstracción del proceso de evolución de las formas orgánicas e inorgánicas, que es el mismo proceso que gobierna el crecimiento de una ciudad.

Sin duda alguna la obra de Alexander ha cruzado las fronteras de la Arquitectura, influyendo en otras disciplinas, y calando especialmente en la Informática, más concretamente en la comunidad de la orientación a objeto (quizás con un mayor impacto que en la Arquitectura).

Cuadro 1. Notas biográficas de Christopher Alexander.

Durante los años comprendidos entre 1990 y 1994 diversos workshops sobre el tema tuvieron lugar, especialmente en las diversas ediciones de la OOPSLA, hasta que en Abril de 1994 se decidió crear una conferencia internacional especializada en patterns: el PloP (**Pattern Languages of Program Design**), que desde 1994 tiene lugar dos veces al año: PLoP USA y EuroPLoP.

Durante este tiempo la Banda de los Cuatro había recopilado su trabajo, que sería publicado en el afamado libro del GoF [12], siendo presentado en la OOPSLA'94. Este libro fue considerado como el mejor libro de orientación a objeto de 1995 y el mejor libro de orientación al objeto de todos los tiempos por la revista JOOP (Journal of Object Oriented Programming) en su número de septiembre de 1995.

Para conocer con un mayor grado de detalle la historia de los patrones software se recomienda visitar la página HistoryOfPatterns en el WikiWiki Web de Cunningham [10].

[2] OOPSLA (**Object-Oriented Programming Systems and Languages**) es uno de los congresos internacionales sobre orientación a objeto más prestigiosos.

[3] Un idiom es un tipo de patrón como se discutirá más adelante en este mismo artículo.

El trabajo del Dr. Alexander está orientado al mundo arquitectónico, sin embargo sus ideas han tenido mucha más repercusión en el área de la Informática que en el de la Arquitectura, acoplándose especialmente bien en la orientación a objeto. A continuación se va a presentar un pequeño resumen de las ideas que el Dr. Alexander expone en algunas de sus obras, y que realizando una abstracción de su significado pueden perfectamente encajar en el mundo de la construcción de aplicaciones software.

En *Notes on the Synthesis of Form* [1] argumenta que los métodos arquitectónicos de la época dan como resultado productos que fallan en el cumplimiento de la quintaesencia de cualquier diseño o esfuerzo de ingeniería: *la mejora de la condición humana*. Persigue la creación de estructuras que sean buenas para las personas de forma que tengan influencias positivas en ellas mejorando su confort y su calidad de vida. Concluye que los arquitectos deben esforzarse por construir *los productos* que mejor se ajusten a las necesidades de las personas que los van a habitar.

En *A Pattern Language* [3] describe algunas ideas de diseño independientes del tiempo para llevar a cabo sus objetivos arquitectónicos. En este libro pretendía capacitar a cualquier ciudadano para diseñar y construir su hogar. Presentaba un catálogo de 253 procesos de solución de problemas que abordaban cuestiones desde la escala de las ciudades y la cultura hasta la construcción y la decoración.

En *The Timeless Way of Building* [4] propone un paradigma para la Arquitectura basado en tres conceptos: *la calidad, la puerta* y *la forma*.

- **La calidad** (también denominada "*la calidad sin nombre*"): Es la esencia de todas las cosas útiles, y las confiere cualidades como: libertad, integridad, compleción, confort, harmonía, habitabilidad, durabilidad, claridad, elasticidad, variabilidad y adaptabilidad. Es lo que hace sentir a las personas vivas, les da satisfacción y en su último extremo mejora la condición humana. La calidad sin nombre es aquella que imparte una belleza incomunicable y un valor inmensurable a una estructura.
- **La puerta:** Es el mecanismo que permite alcanzar la calidad. Se manifiesta como un lenguaje de patrones vivo que permite crear diseños con multitud formas que satisfacen necesidades con varias facetas. Es el conducto para la calidad.
- **La forma** (también conocida como "*la forma independiente del tiempo*"): Usando la forma, los patrones de la puerta son aplicados utilizando una técnica de espacio diferencial en una secuencia ordenada de crecimiento por partes – progresivamente se hace evolucionar una arquitectura inicial que dará lugar a un diseño *vivo* que tiene calidad -.

Para un mayor estudio de la influencia del trabajo del Dr. Alexander en la orientación a objeto se recomienda la consulta de los siguientes artículos: [14] y [19].

Cuadro 2. Notas sobre el trabajo de Christopher Alexander.

Concepto Intuitivo de Patrón

Antes de entrar a definir de una forma más concreta lo qué es un patrón, puede ser interesante realizar una primera aproximación informal a este concepto con el fin de clarificar más su significado en el mundo del software.

Según el diccionario de la Real Academia Española un patrón es: "*Dechado que sirve de muestra para sacar otra cosa igual*", estando esta definición muy cercana a la idea que se persigue con los patrones software, donde los patrones, en lugar de servir como muestra física, sirven para almacenar y documentar soluciones recurrentes a problemas tipo.

Como ejemplo ilustrativo supóngase un problema típico que se repite en el diseño de bases de datos de bibliotecas, al intentar modelar la relación existente entre los libros y los ejemplares que existen

de cada libro. Para solucionar este problema de modelado se puede establecer un sencillo patrón, como se muestra en la Figura 1. Este patrón puede ser reutilizado en todo diseño en el que intervengan ambas entidades.

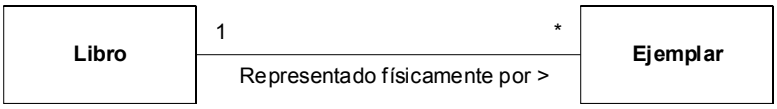

Figura 1. Relación entre Libro y Ejemplar.

Este patrón de diseño tiene el inconveniente de que sólo se puede reutilizar en aplicaciones en las que intervengan estas entidades concretas. Por lo tanto es interesante generalizar este diseño para que pueda ser reutilizado en todos aquellos casos en los que exista una entidad que define unas propiedades y se relaciona con un conjunto de ejemplares que representan físicamente dichas propiedades, como se muestra en la Figura 2.

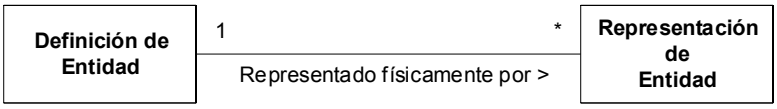

Figura 2. Diseño que relaciona una definición con su representación.

El diseño que muestra la Figura 2 es una clara muestra de reutilización de la experiencia de diseño, de forma que no se corresponde con ningún problema concreto, pero a su vez representa un conjunto bien definido de problemas.

Con este ejemplo se ha buscado clarificar el concepto de patrón software más que primar la calidad del diseño, aunque completándolo con los componentes propios de un patrón podía constituir un claro ejemplo de patrón de diseño para utilizar en el diseño de bases de datos relacionales, especialmente en ambientes docentes de iniciación al diseño de bases de datos.

Por tanto, se tiene que los patrones software representan soluciones a problemas que aparecen en el desarrollo del software en un determinado contexto. Los patrones más difundidos son los patrones de diseño, pues soportan la reutilización de la arquitectura software y del diseño, capturando las estructuras estáticas y dinámicas que dan soporte a las soluciones de problemas que surgen en la construcción de aplicaciones software en un dominio concreto.

Para finalizar esta primera aproximación a los patrones cabe recurrir a un párrafo del Dr. Alexander: "*Cada patrón describe un problema que ocurre una y otra vez en nuestro entorno y entonces se describe el núcleo de la solución a dicho problema de tal forma que se puede usar esta solución un millón de veces sin hacerlo dos veces de la misma forma*" [4].

Definición de Patrón Software

Una vez que se ha realizado una primera aproximación al concepto de patrón software, es más sencillo comprender las diversas definiciones que de patrón aparecen en la literatura. Como sucede con la mayoría de los términos propios de la Ingeniería del Software, no existe una definición estándar para el término patrón, así se pueden encontrar varias en la literatura, de las cuales se van a recoger algunas de las más representativas, aunque parafraseando a James Coplien se puede decir que: "*Alexander podría haber escrito una frase de definición de lo qué es un patrón, pero en su lugar él escribió un libro de 550 páginas para hacerlo porque el concepto es difícil*".

Así, para Trigve Reenskaug, padre del método OOram, un patrón es *una descripción en un formato fijo de cómo solucionar un cierto tipo de problemas* [17]. Para Brad Appleton un patrón es *una unidad de información instructiva con nombre que captura la estructura esencial y la comprensión de una familia de soluciones exitosas probadas para un problema recurrente que ocurre dentro de un cierto contexto y de un sistema de fuerzas* [5]. Sin embargo, para James Coplien cada patrón es *una regla constituida por tres partes, la cual expresa una relación entre un cierto contexto, un cierto sistema de fuerzas que ocurren repetidamente en ese contexto, y una cierta configuración software que permite a estas fuerzas resolverse así mismas* [8].

La definición de James Coplien nos va a servir para introducir una serie de términos que configuran la terminología propia de los patrones. En primer lugar nos encontramos con el término ***sistema de fuerzas***, el cual hace referencia a que un problema se refiere a un conjunto de fuerzas. La noción de fuerza generaliza el tipo de criterios que los ingenieros del software utilizan para justificar los diseños y las implementaciones. En el caso de los patrones, éstos casan con conjuntos de objetivos y restricciones que se encuentran en el desarrollo de cada elemento que se vaya a crear. Estas fuerzas son grandes, difíciles de medir y conflictivas. También es interesante resaltar el término ***configuración software*** en el sentido de un diseño en una forma canónica o un conjunto de reglas de diseño que alguien puede aplicar para solucionar las fuerzas del problema.

Como se desprende de las definiciones anteriores, un patrón involucra una descripción general de una solución recurrente para un problema recurrente repleto de diferentes objetivos y restricciones. Pero un patrón hace algo más que identificar una solución, explica el porqué se necesita la solución. Debe tenerse presente que no cualquier solución, algoritmo, máxima de diseño o heurística constituye un patrón. Es más, aunque algo parezca tener todos los ingredientes necesarios para ser un patrón, no será considerado como tal hasta que verifique ser un fenómeno recurrente (preferiblemente en al menos tres sistemas existentes; esto es lo que se denomina la **regla de los tres**). Un patrón que se encuentra a la espera de ser considerado como tal se denomina **proto-patrón**. Es un buen criterio no denominar a nada patrón hasta que no haya sido sometido a algún tipo de examen o revisión por parte de otras personas.

Para James Coplien [8] un buen patrón debe cumplir los siguientes requisitos:

- **Debe solucionar un problema:** Los patrones capturan soluciones, no sólo principios abstractos o estrategias.
- **Son conceptos probados:** Los patrones capturan soluciones que han sido probadas, no teorías o especulaciones.
- **La solución no es obvia:** La mayoría de las técnicas de resolución de problemas (tales como métodos de diseño) intentan derivar soluciones partiendo de principios básicos. Los mejores patrones generan una solución para un problema indirectamente, una aproximación necesaria para los problemas de diseño más complejos.
- **Describe una relación:** Los patrones no deben describir módulos, sino que deben describir sistemas, estructuras o mecanismos más profundos.
- **El patrón debe tener un componente humano importante:** Todo software sirve para el confort humano o para la calidad de vida; los mejores patrones recurren explícitamente a la estética y a la utilidad.

Componentes Esenciales de un Patrón Software

Existen diferentes formatos para describir los patrones, sin embargo hay una serie de elementos que se consideran esenciales y que deben compartir todas las representaciones de los patrones, indicados por Christopher Alexander en [4]: un patrón es una regla que establece una relación entre un **contexto**, un **sistema de fuerzas que aparecen en el contexto** y una **configuración** que permite que las fuerzas se resuelvan dentro del contexto. Después la literatura especializada en patrones software ofrece diferentes variantes de los elementos considerados esenciales, pero todas ellas contienen los tres apartados indicados por el Dr. Alexander, que en un lenguaje más coloquial podían denominarse: **contexto, problema** y **solución.**

Así, en el libro de GoF [12] se indica que un patrón debe contar con cuatro elementos esenciales:

- *El nombre del patrón: Un descriptor manejable del problema. Debe ser corto (una palabra o dos). Amplía el vocabulario de diseño.*
- *El problema: Describe cuando aplicar el patrón. Explica el problema y su contexto.*
- *La solución: Describe los elementos que forman el diseño, sus relaciones, responsabilidades y colaboraciones. No describe un diseño o una implementación concreta, sino que ofrece una descripción abstracta de un problema.*

- *Las consecuencias: Son los resultados de aplicar un patrón. Son necesarias para evaluar alternativas de diseño, así como para evaluar los costes y beneficios de su aplicación.*

Otros autores [14, 18] difieren en cuanto al número de elementos propios de un patrón, pero en esencia coinciden con las ideas de Alexander y al final vienen a representar la misma información que los patrones del libro de GoF.

Descripción de los Patrones Software

Los patrones se describen utilizando formatos consistentes. Un formato es una plantilla dividida en secciones. La plantilla ofrece una uniformidad en la estructura de los patrones que hace que éstos sean más sencillos de aprender, de comparar y de utilizar.

Existen diferentes formatos de descripción de patrones, entre los que cabe destacar el formato utilizado por Christopher Alexander en su trabajo que es denominado *formato de Alexander*, el formato utilizado en el libro del GoF [12] que se denomina *formato del GoF*, y que se muestra en la Tabla 1, la plantilla recomendada por Doug Lea (http://hillside.net/patterns/Writing/Lea.html) que se recoge en la Tabla 3, el formato propio del *Portland Pattern Repository* (http://www.c2.com:80/ppr/), o el *formato canónico* que es el utilizado en [6][4] (ver Tabla 2).

Formato del GoF	
Sección	**Descripción**
Nombre del Patrón y Clasificación	Comunica la esencia del patrón.
Propósito	Una corta introducción respondiendo a las preguntas siguientes: ¿Qué hace el patrón?, ¿Cuál es su razón? ¿De qué problema particular de diseño se ocupa?
También Conocido Como	Otros nombres para el patrón.
Motivación	Un escenario que ilustra un problema y cómo las clases y los objetos se estructuran en un patrón para solucionar el problema.
Aplicabilidad	¿Cuáles son las situaciones en las que el patrón puede aplicarse? Ejemplos de casos mal planteados en los que el patrón puede aplicarse. ¿Cómo pueden reconocerse estas situaciones?
Estructura	Representación gráfica de las clases en el patrón (diagramas de clase, diagramas de colaboración…).
Participantes	Las clases y/u objetos que participan en el patrón de diseño y sus responsabilidades.
Colaboraciones	La forma en que los participantes colaboran para llevar a cabo sus responsabilidades.
Consecuencias	¿Cómo el patrón soporta sus objetivos? ¿Cuáles son las concesiones y resultados de usar un patrón?
Implementación	¿Qué trucos o técnicas deben tenerse en cuenta para implementar el patrón? ¿Hay temas propios de un lenguaje específico?
Código de Ejemplo	Fragmentos de código que ilustren como se puede implementar el patrón en C++ u otro lenguaje.
Usos Conocidos	Ejemplos del patrón encontrados en sistemas reales.
Patrones relacionados	¿Qué patrones están relacionados con el patrón que se está describiendo? ¿Cuáles son las principales diferencias? ¿Con qué otros patrones debe usarse?

Tabla 1. Formato del GoF.

[4] Este libro "Pattern Oriented Software Architecture: A System of Patterns" es también conocido como POSA, y a sus autores como the Gang of Five (GoV).

Formato Canónico	
Sección	**Descripción**
Nombre	Nombre significativo. Puede ser una única palabra o una frase corta para hacer referencia al patrón. Puede tener alias.
Problema	Descripción del problema indicando su propósito: objetivos dentro de una contexto dado.
Contexto	Precondiciones bajo las cuales el problema y su solución parece recurrir, y para el cual la solución es deseable. Describe la aplicabilidad del patrón.
Fuerzas	Descripción de las fuerzas y restricciones relevantes y cómo interaccionan o entran en conflicto las unas con las otras, así con los objetivos que se quieren perseguir. Las fuerzas revelan las interioridades de un problema y definen las concesiones que se deben considerar.
Solución	Relaciones estáticas y reglas dinámicas que describen como conseguir el objetivo marcado. Puede acompañarse de imágenes, diagramas y texto para identificar la estructura del patrón, sus participantes y sus colaboradores, para mostrar como se resuelve el problema.
Ejemplos	Uno o más ejemplos de la aplicación del patrón.
Contexto Resultante	El estado o la configuración del sistema después de haber aplicado el patrón, incluyendo las consecuencias (buenas o malas) de su aplicación, así como otros problemas y patrones que pueden derivarse en el nuevo contexto. Describe por tanto las postcondiciones y los efectos laterales de los patrones.
Razonamiento	Explicación justificada de los pasos o reglas del patrón.
Patrones Relacionados	Relaciones estáticas y dinámicas entre el patrón que se está describiendo y otros dentro del mismo lenguaje de patrones o dentro del mismo sistema.
Usos Conocidos	Descripción de ocurrencias conocidas del patrón y su aplicación dentro de sistemas existentes. Ayuda a validar el patrón verificando su uso en soluciones probadas para un problema recurrente.

Tabla 2. Formato POSA.

Plantilla de Doug Lea	
[PATTERN-NAME]	**Design**
Author	**[PARAG-1]**
[YOUR-NAME] ([YOU@YOUR.ADDR]).	**[PARAG-2]**
Last updated on [TODAY'S-DATE]	**An Implementation**
Context	**[SOME-CODE]**
[PARAG-1]	**Examples**
[PARAG-2]	**Variants**
Problem	**[VARIANT]**
[ONE-ASPECT]	**[ANOTHER-VARIANT]**
[ANOTHER-ASPECT]	**See Also**
Examples	**Patterns Home Page.**
Forces	**[ANOTHER-REF]**
1.[FORCE-1]	**[SOME-AUTHOR, SOME-ARTICLE]**
2.[FORCE-2]	**[SOME-PATTERNS-LIST-POSTING]**

Tabla 3. Plantilla de Doug Lea para la descripción de patrones.

Ejemplos de Patrones

Numerosos ejemplos de patrones software pueden encontrarse disponibles libremente en Internet, por ejemplo en las páginas del **Portland Pattern Repository** (http://www.c2.com:80/ppr/) o en las páginas web del **Patterns Home Page** (http://hillside.net/patterns/patterns.html). Sin olvidar los ya clásicos patrones de diseño que se encuentran en el libro del GoF [12] o en el libro POSA [6]. Otra importante fuente donde consultar ejemplos de patrones la constituyen las actas de las conferencias PLoP [9, 16, 20].

Sin embargo, desde aquí se recomienda consultar un patrón que nada tiene que ver con el software, ya que es un patrón que indica como comer pizza lo más rápidamente posible, el patrón *Pizza Inversion* (http://www.enteract.com/~bradapp/docs/pizza-inv.html). Este patrón resulta sumamente didáctico para comprender la esencia de lo que es patrón y sobretodo para recalcar su estructura.

Tipos de Patrones Software

Debido al éxito alcanzado por el libro del Gof [12], existe una cierta tendencia a que cada vez que se menciona el término patrón se relaciona con los patrones presentados en este libro. Dichos patrones son patrones de diseño orientado a objeto, pero existen muchos otros tipos de patrones aparte de los patrones de diseño. Se pueden citar como ejemplos de patrones no de diseño los patrones de análisis [11], o los patrones de organización[5] de los que existe un website especializado administrado por James Coplien (http://www.bell-labs.com/cgi-user/OrgPatterns/OrgPatterns). Además, los patrones enviados a las ediciones del PLoP cubren la mayoría de los campos de la Ingeniería del Software.

Así, en [6] se distinguen tres tipos de patrones, marcando de una forma muy clara tres niveles conceptuales. Estos tipos son:

- **Patrones arquitectónicos:** Expresan una organización estructural fundamental para un sistema software. Normalmente expresa un conjunto de subsistemas predefinidos, especifica sus responsabilidades e incluye reglas y guías para organizar las relaciones entre ellos.
- **Patrones de Diseño:** Ofrecen esquemas para refinar subsistemas y componentes de un sistema software, o las relaciones entre ellos. Describe normalmente una estructura de comunicación recurrente entre componentes, que sirve para resolver un problema general de diseño dentro de un contexto particular.
- **Idioms:** Un idiom es patrón de bajo nivel, específico de un determinado lenguaje de programación. Describen como implementar aspectos particulares de los componentes, o de las relaciones entre ellos, utilizando las características de un determinado lenguaje. Como ejemplo de un idiom se tiene la siguiente construcción en C++ para copiar cadenas de caracteres: `while (*destino++ = *src++);`

Centrando de nuevo la atención en los patrones de diseño, en el libro de GoF [12] se distinguen tres tipos de patrones de diseño: **patrones de creación, patrones estructurales** y **patrones de comportamiento**. Además, en cada una de las categorías se distinguen dos subtipos, patrones que trabajan con clases y patrones que trabajan con objetos.

- **Patrones de Creación:** Los patrones de creación abstraen el proceso de instanciación de objetos. Tienen la misión de permitir construir sistemas independientes de la forma en que se crean, se componen o se representan los objetos. Un patrón de creación de clases utiliza la herencia para variar la clase que es instanciada, mientras que un patrón de creación de objetos delega la instanciación en otro objeto. Como ejemplo de patrón de creación de clases se tiene el ***Factory Method***, y como ejemplo de patrón de creación de objetos se puede citar al patrón ***Builder***.
- **Patrones Estructurales:** Se cuidan de cómo las clases y los objetos se componen para formar estructuras mayores. Un patrón estructural de clases utiliza la herencia para componer interfaces o implementaciones, por ejemplo el patrón ***Adapter***. Un patrón estructural de objetos describe la forma en que se componen objetos para obtener nueva funcionalidad, además se añade la flexibilidad de cambiar la composición en tiempo de ejecución, lo cual no es posible con la composición de clases estáticas, como ejemplo de este tipo de patrones se puede mencionar al patrón ***Composite***.
- **Patrones de Comportamiento:** Este tipo de patrones está relacionado con algoritmos y la asignación de responsabilidades entre objetos. Describen, además de los patrones de objetos y clases, los patrones de comunicación entre ellos. Los patrones de comportamiento de clases utilizan la herencia para distribuir el comportamiento entre las clases, como ejemplo se puede citar al patrón **Template Method**. Por su parte los patrones de comportamiento de objetos utilizan la composición de objetos en lugar de la herencia, por ejemplo como un grupo de

[5] Describen las estructuras y las prácticas de las organizaciones humanas.

objetos interconectados cooperan para realizar una tarea que un solo objeto no puede hacer por sí mismo, como representante de este tipo de patrones se puede mencionar al patrón ***Observer***.

Patrones y Orientación a Objeto

Hasta el momento se ha intentado dejar patente en este artículo que un patrón es algo independiente de un determinado paradigma, de una metodología concreta o de un lenguaje particular. Aunque también debe señalarse que la mayor actividad en la comunidad de los patrones software ha venido de la mano de la tecnología de objetos.

La razón de por la que los patrones han tenido tan buena acogida en el seno de la orientación a objeto se tiene en que la programación orientada a objeto se adapta muy bien a las metáforas arquitectónicas propuestas por el Dr. Alexander. Los objetos pueden ensamblarse en estructuras más complejas, tal y como hace la Arquitectura, añadiendo además una dimensión temporal que recoge el intercambio de mensajes entre objetos.

Por otra parte, la orientación a objeto soporta muy bien el modelado del mundo real, teniendo una importancia capital el diseño de las aplicaciones software.

Pero, aunque las teorías de Christopher Alexander encajan muy bien en la orientación a objeto, debe reconocerse la importancia capital del libro del GoF [12] para la difusión del concepto de patrón de diseño dentro de la orientación a objeto, influencia que por otra parte puede haber influido a pensar que los patrones son un concepto exclusivo de la orientación a objeto.

Relación entre los Patrones Software y la Reutilización

La inmadurez de la Ingeniería del Software como disciplina ingenieril queda plasmada cada vez que se *reinventa la rueda* para afrontar un nuevo proyecto software, en lugar de recurrir a la guía de soluciones ya existentes como sucede en otras ingenierías.

Este problema de falta de madurez se hace más patente cuanto mayor es la organización, donde los proyectos caminan independientes resolviendo una y otra vez problemas similares (por no decir idénticos en un alto porcentaje de los casos) a los ya resueltos en otros proyectos.

RECURSOS SOBRE PATRONES EN INTERNET		
Nombre	**URL**	**Descripción**
Cetus Links: Links on Objects and Components/Patterns	http://www.infosys.tuwien.ac.at/cetus/oo_patterns.html	Importante colección de direcciones relacionadas con patrones.
Patterns Home Page	http://hillside.net/patterns/patterns.html	Repositorio con información y direcciones sobre patrones.
Pattern Definitions	http://hillside.net/patterns/definition.html	Definiciones de patrón.
Pattern Mailing Lists	http://hillside.net/patterns/Lists.html	Listas de correo sobre patrones.
Writing Patterns	http://hillside.net/patterns/Writing/Lea.html	Recomendaciones para crear nuevos patrones.
Patterns-discussion FAQ	http://g.oswego.edu/dl/pd-FAQ/pd-FAQ.html	FAQ sobre patrones, administrado por Doug Lea, un reconocido experto en orientación a objeto.
Portland Pattern Repository	http://www.c2.com:80/ppr/	Dirección WWW donde además de información relacionada con los patrones se pueden encontrar ejemplos.
Book review: Design Patterns	http://iamwww.unibe.ch/CHOOSE/Articles/95-1/DP-book-review.html	Revisión del famoso libro Design Patterns de Gamma et al.
Brad Appleton's Software Patterns Links	http://www.enteract.com/~bradapp/links/sw-pats.html#Sw_Pats	Colección de enlaces a sitios relacionados con patrones.
Brad Appleton's Documents	http://www.enteract.com/~bradapp/docs/	Enlace a la página de documentos de Brad Appleton.
Patterns and Software: Essential Concepts and Terminology	http://www.enteract.com/~bradapp/docs/patterns-intro.html	Excelente documento sobre los conceptos esenciales de los patrones software.
Introducción a Patrones	http://agamenon.uniandes.edu.co/~pfiguero/soo/Magister_Patrones/intropatrones.html	Introducción a los patrones muy escueta pero en castellano.
Design Patterns Tutorial	http://www.csc.calpoly.edu/~dbutler/tutorials/winter96/patterns/	Tutorial sobre patrones de diseño bastante recomendable
Software Patterns -- IAP'97	http://www-white.media.mit.edu/~tpminka/patterns/	Tutorial sobre patrones
AntiPatterns	http://www.antipatterns.com/	Lugar dedicado a los antipatterns. Los antipatterns describen las lecciones aprendidas en el uso de patrones.
Organizational Patterns	http://www.bell-labs.com/cgi-user/OrgPatterns/OrgPatterns	Patrones sobre organizaciones. Otra variante de los patrones, dedicada a las formas de organización humanas. Está administrado por James Coplien.

Tabla 4. Recursos sobre patrones en Internet.

Los patrones software, y en general la noción de patrón, constituyen una forma flexible y adecuada que favorece la reutilización de la experiencia embebida en los procesos que representan soluciones a problemas software, con independencia de su nivel de abstracción (análisis, diseño o implementación).

Dentro de los patrones software, los más útiles desde el punto de vista de aportación a la reutilización del software son los patrones que se denominan generativos, esto es, aquellos que son activos y dinámicos, que no se limitan a mostrar las características de los sistemas que se consideran adecuados, sino que enseñan a construirlos.

Más Información sobre Patrones

A lo largo del presente artículo han ido apareciendo las principales fuentes bibliográficas sobre patrones en general y patrones software en particular. Dichas referencias se encuentran recogidas en el apartado de referencias al final de este artículo. Sin embargo, cabe destacar de una forma especial dos libros imprescindibles para cualquier lector que quiera profundizar en los patrones de diseño: el libro de GoF [12] y el libro POSA [6].

Además, en Internet se puede encontrar una gran cantidad de información relacionada con los patrones. En la Tabla 4 se muestran una destacada selección de direcciones web en las que encontrar abundante material relacionado con los patrones.

Conclusiones

En este artículo se ha pretendido introducir al lector en el mundo de los patrones software, pero partiendo desde la perspectiva inicial de este concepto, para nada relacionada con el mundo del software, para acabar entrando de lleno en el área de la tecnología de objetos y de la reutilización.

Se ha buscado familiarizar al lector con el concepto de patrón, con sus características y sus peculiaridades, más que con su utilización en el ámbito del Diseño Orientado a Objeto [13] para construir aplicaciones software, con el fin de evitar caer en los falsos mitos y errores, causados por el desconocimiento y la falta de comprensión, propios de un acercamiento poco exhaustivo a una nueva área de conocimiento.

Se ha querido recoger en este artículo una amplia lista de referencias, a las que cualquier lector interesado pueda acceder para profundizar más en este interesante campo.

En los próximos artículos, ya se le supondrá al lector un conocimiento suficiente de las bases, y se pasará a la parte práctica de la aplicación de patrones de diseño en la construcción de aplicaciones software, para lo cual nos serviremos de algunos patrones del libro de Gamma, así como de otras conocidas arquitecturas software.

Referencias

[1] **Alexander, Christopher**. "*Notes on the Synthesis of Form*". Harvard University Press. 1964.

[2] **Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M. and Angel, S.,** "*The Oregon Experiment*". Oxford University Press. 1975.

[3] **Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I. and Angel, S.** "*A Pattern Language*". Oxford University Press. 1977.

[4] **Alexander, C.** "*The Timeless Way of Building*". The Oxford University Press. 1979.

[5] **Appleton, Brad.** "*Patterns and Software: Essential Concepts and Terminology*". http://www.enteract.com/~bradapp/docs/patterns-intro.html. November 1997.

[6] **Buschmann, Frank, Meunier, Regine, Rohnert Hans, Sommerlad Peter and Stal Michael.** "*Pattern Oriented Software Architecture: A System of Patterns*". John Wiley & Sons. 1996.

[7] **Coplien, James O.** "*Advanced C++ Programming Styles and Idioms*". Addison-Wesley. 1992.

[8] **Coplien, James O.** "*A Pattern Definition - Software Patterns*". http://hillside.net/patterns/definition.html. 1998.

[9] **Coplien, James O. and Schmidt, Douglas (editors).** "*Pattern Languages of Program Design*". Addison-Wesley. 1995.

[10] **Cunningham, Ward.** "*The History of Patterns*". WikiWiki Web. http://c2.com/cgi-bin/wiki?HistoryOfPatterns. 1998.

[11] **Fowler, Martin.** "Analysis Patterns: Reusable Object Models". Addison-Wesley Object Technology Series. 1996.

[12] **Gamma, Erich, Helm, Richard, Johnson, Ralph and Vlissides, John**. "*Design Patterns. Elements of Reusable Object-Oriented Software*". Addison-Wesley, 1995.

[13] **García, F. J., Marqués, J. M., Maudes, J. M.** "*Análisis y Diseño Orientado al Objeto para Reutilización*". Technical Report (TR-GIRO-01-97V2.1.1), Universidad de Valladolid (España). Octubre 1997.

[14] **Lea, Doug.** "*Christopher Alexander: An Introduction for Object-Oriented Designers*". Software Engineering Notes, ACM SIGSOFT, Vol. 19, N. 1, pp. 39-46, January 1994 (Online version: http://g.oswego.edu/dl/ca/ca/ca.html).

[15] **López Tallón, Alberto**. "*Patrones de Diseño. Reutilización de Ideas*". Revista Profesional para Programadores (RPP). Nº43: 54-58. Septiembre, 1998.

[16] Martin, Robert C., Riehle, Dirk, Buschmann, Frank (editors). "*Pattern Languages of Program Design 3 (Software Patterns Series)*". Addison-Wesley. 1997.

[17] Reenskaug, Trygve, Wold, Per and Lehne, Odd Arild. "*Working with Objects. The OOram Software Engineering Method*". Manning/Prentice Hall. 1996.

[18] Rising, Linda. "*Design Patterns: Elements of Reusable Architectures*". Annual Review of Communications. Vol. 49: 907-909. 1996.

[19] Rising, Linda. "*The Road, Christopher Alexander, and Good Software Design*". Object Magazine, pp. 47-50, March, 1997.

[20] Vlissides, John M., Coplien, James O. and Kerth, Norman L. (editors). "*Pattern Languages of Program Design 2*". Addison-Wesley. 1996.