INTRODUCCIÓN PATRONES SOFTWARE

Gustavo Millán García

ORÍGENES

- *The Timeless Way of Building [Alexander79],*

"Cada patrón es una regla de 3 partes, que expresa una relación entre un contexto, un problema y una solución. Como un elemento en el mundo, cada patrón es una relación entre un contexto, un sistema de fuerzas que ocurren repetidamente en ese contexto y una configuración espacial que permite que esas fuerzas se resuelvan entre sí."

En 1977, en el libro A Pattern Language, Alexander presenta junto a otros colegas estas teorías y el uso de la metáfora de patrón, que define de la siguiente manera:

"Cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, para describir después el núcleo de la solución a ese problema, de tal manera que esa solución pueda ser usada más de un millón de veces sin hacerlo ni siquiera dos veces de la misma forma."

HISTORIA. CONCEPTO PATRÓN EN EL ÁMBITO DEL DESARROLLO DE SOFTWARE

- Eric Gamma, Richard Helm, Ralph Johnson y John Vlissides publicaron el primer catálogo de patrones en el ámbito del software
  - *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley, ©1995.
  - Se inspiraron en el trabajo de Christopher Alexander, que había documentado patrones en el ámbito de la arquitectura civil
- Pattern-Oriented Software Architecture: A System of Patterns (POSA)
  - Primer libro en hacer una clasificación de patrones
    - Patrones arquitectónicos
    - Patrones de diseño
    - Patrones elementales (Idioms)

DEFINICIÓN DEL TÉRMINO PATRÓN SOFTWARE

- Es una **solución** a un **problema recurrente** que surge en un **contexto determinado** [POSA]
- Documenta diseños probados y basado en la experiencia
- Especifica una **configuración de elementos y su comportamiento**
- El contexto de un patron describe cuando puede surgir el problema

SOBRE EL USO DE PATRONES Y LA INGENIERÍA SOFTWARE

- A medida que ha aumentado el uso de los patrones se ha tenido certeza de que éstos pueden mejorar la productividad y calidad de las soluciones software gracias a las siguientes cualidades (Chambers et al., 2000):
- **Promueven la reutilización de diseño**. Los patrones de diseño capturan experiencias en la resolución de problemas de diseño cuya utilidad o necesidad se ha evidenciado repetidamente en diferentes proyectos. Por lo tanto, la aplicación de un patrón lleva implícita la reutilización de esos diseños.
- **Forman un vocabulario común de diseño**. Los nombres de los patrones pueden ayudar a los diseñadores a comunicarse de una manera más eficiente, extendiendo su vocabulario y siendo capaces de transmitir el problema y la solución como un todo comprensible mediante el nombre del patrón.
- **Mejoran la documentación**. La utilización de un patrón en el desarrollo de un sistema permite conocer qué solución se ha aplicado y por qué, puesto que el patrón no es una idea del programador sino una solución probada y reconocida en una comunidad de desarrollo.

CONSIDERACIONES GENERALES SOBRE EL USO DE PATRONES(I)

- **Encontrar** el **patrón adecuado no es fácil**.
- La implementación de forma correcta de patrones **requiere experiencia.**
- El uso de patrones **no da automáticamente** como resultado un diseño de alta calidad.
- Normalmente la gente ve los patrones como plantillas o como bloques modulares de construcción. **No son plantillas**!
- Los patrones **no automatizan el proceso de desarrollo** del software.

CONSIDERACIONES GENERALES SOBRE EL USO DE PATRONES(II)

- Los Patrones software no son una solución completa en si misma
- Los Patrones complementan , pero no sustituyen otras técnicas y métodos dentro de la ingeniería de software.
- Los patrones no son un sustituto de la inteligencia ý creatividad del ingeniero software.
- No hay que forzar la aplicación de patrones, hay que usarlos con cuidado

TIPOS DE PATRONES SOFTWARE

- **Patrones de Arquitectura**
  - Los patrones de arquitectura expresan el esquema fundamental de organización de sistemas software. Proveen un conjunto de subsistemas predefinidos, especifican sus responsabilidades e incluyen reglas y guías para organizar las relaciones entre ellos. [POSA96]
- **Patrones de Diseño**
  - Un patrón de diseño describe una estructura recurrente de componentes que se comunican para resolver un problema general de diseño en un contexto particular [GoF95].
  - Tipos : De creación , estructurales y de comportamiento
- **Idioms**
  - Es un patrón considerado especificamente en un lenguaje de programación. Describe como implementar aspectos de componentes y sus relaciones en un lenguaje de programación

- La diferencia entre estas clases de patrones esta en su nivel de abstracción y su nivel de detalle (los patrones de aquitectura son mucho más abstractos que idioms)

DESCRIPCIÓN DE UN PATRÓN SOFTWARE

- **Nombre**: Un nombre descriptivo y único que ayude a identificar y referenciar al patrón.
- **Problema:** Descripción resumida en una o dos frases que describe la intención del patrón, es decir, las metas y objetivos que se quieren alcanzar.
- **Contexto:** Problema recurrente en el que es aplicable el patrón.
- **Fuerzas:** Descripción de las fuerzas, los objetivos y restricciones relevantes para ese patrón, y de cómo éstas interaccionan entre ellas o con las metas que deseamos alcanzar. Además se puede incluir un escenario concreto que sirva de motivación para el patrón. La noción de fuerza generaliza los tipos de criterios que justifican al patrón.
- **Solución**: Suele ser un conjunto de instrucciones que describen cómo construir el producto resultante. La descripción puede ir acompañada de dibujos, diagramas o esquemas de la solución. Es el corazón del patrón. El patrón *North Face* nos indica las siguientes instrucciones: "Haz de la cara norte del edificio una cascada que descienda hacia el suelo para que el sol que normalmente proyecta una larga sombra hacia el norte alcance el suelo inmediatamente al lado del edificio".
- **Ejemplos**: Ejemplos, que pueden ser visuales, que ayudan al lector a entender el uso y la aplicabilidad del patrón (el contexto inicial, cómo el patrón transforma el contexto, y el contexto resultante). En el patrón *North Face* se presenta un ejemplo visual de la aplicación del patrón sobre el contexto inicial, como se puede ver en la parte derecha de la figura 2.2.b.
- **Contexto resultante**: Indica el estado del sistema después de aplicar el patrón, incluyendo sus consecuencias (positivas y negativas).
- **Exposición razonada**: Expone cómo funciona el patrón y por qué es útil. Mientras que la solución muestra la estructura visible del patrón, la exposición explica sus mecanismos subyacentes.
- **Patrones relacionados:** Patrones que se pueden combinar con este, o es posible aplicar a partir del contexto resultante.

¿COMO SE DECIDE QUE ES UN PATRÓN?

- ¿Como se decide que es lo que hace que algo sea un buen patrón?: Las conferencias PoLP (Pattern Language of Programs) han establecido varios críterios para aceptar publicaciones o consideraciones de nuevos patrones. Estos criterios estan resumidos en Buschmann et. al. in **Pattern-Oriented Software Architecture**
  - **Enfoque práctico:** Los patrones deben describir soluciones probadas para problemas recurrentes más que el último avance cientifico
  - **No busqueda de originalidad:** Las personas que descubren patrones no necesitan ser inventores o descubridores
  - **Revisión pública:**
  - **Talleres en lugar de presentaciones**: Más que ser presentados por su autor los patrones son discutidos en talleres, que son foros abiertos donde se busca la mejora continua.
  - **Minuciosamente publicados**

CARACTERISTICAS

- Un buen patrón debe tener las siguientes características (Coplien, 1998):
- Resolver un problema: Los patrones capturan soluciones, no sólo principios abstractos o estrategias.
- Ser un concepto probado: Los patrones capturan soluciones con un registro de los pasos a llevar a cabo, no teorías o especulaciones.
- La solución propuesta no es obvia: Muchas técnicas de resolución de problemas intentan derivar la solución a partir de principios. Los mejores patrones generan una solución a un problema de manera indirecta, una aproximación necesaria para los problemas más difíciles de diseño.
- Describir una relación: Los patrones no sólo describen módulos, sino estructuras y mecanismos del sistema en detalle.
- Tener un componente humano significativo, minimizando así su intervención: El software sirve a la comodidad humana o a la calidad de vida; los mejores patrones explícitamente apelan a la estética y a la utilidad.

PATRONES DE ARQUITECTURA

- Definen un estructura global de organización del sistema software
  - MVC (Model- View-Controller)
  - Layers
  - Pipes and Filtres
  - Blackboard
  - Broker
  - Presentation-Abstraction-Control
  - Microkernel
  - Reflection

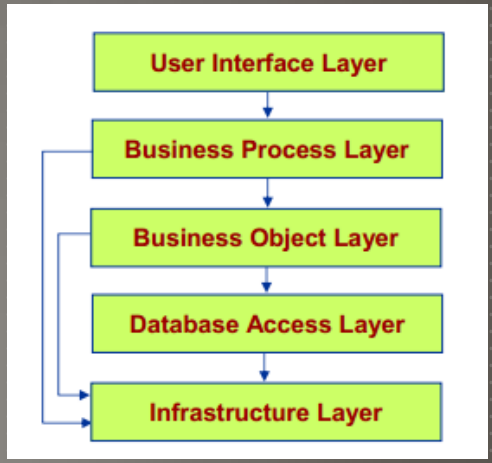
PATRONES DE DISEÑO

- Patrones de diseño [GoF 95]

Patrones de diseño				
		Propósito		
		De Creación	Estructurales	De Comportamiento
Ámbito	Clase	• Factory Method	• Adapter (de clases)	• Interpreter • Template Method
	Objeto	• Abstract Factory • Builder • Prototype • Singleton	• Adapter (de objetos) • Bridge • Composite • Decorator • Façade • Flyweight • Proxy	• Chain of Responsibility • Command • Iterator • Mediator • Memento • Observer • State • Strategy • Visitor

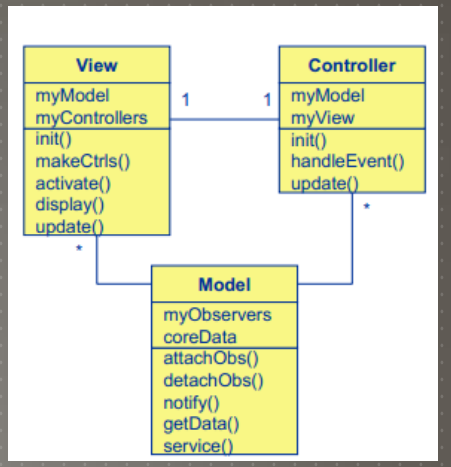
EJEMPLO DE PATRÓN ARQUITECTURA

- ¿Cómo se puede dividir la funcionalidad del sistema ?de forma que:
  - La funcionalidad de un nivel de abstracción este tan desacoplada como sea posible del resto
  - La funcionalidad de un determinado nivel de abstracción evolucione de forma independiente a las demás sin afectarlas
- Solución
  - Definir una o más capas , donde cada capa tenga un única responsabilidad clara y bien definida de acuerdo a su nivel de abstracción Y grado de variación
  - **Patrón Layers**



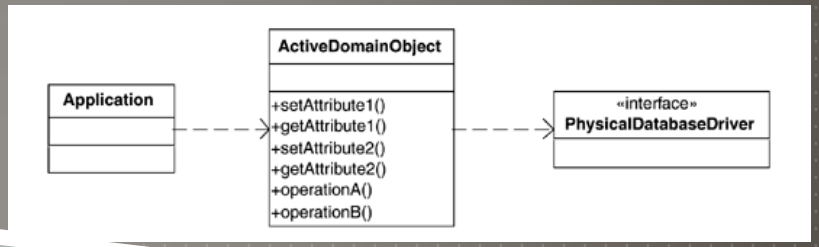
PATRÓN DE ARQUITECTURA MVC

- Problema
  - Como se puede desarrollar sistemás interactivos con interfaces gráficas de usuario de forma que:
    - Soporten adaptaciones y cambios sin afectar a la funcionalidad nucleo del sistema
    - La interfaz gráfica muestre el estado del proceso y que también se actualize ante cambios de estado interno
- Solución
  - Dividir la aplicación interactiva en tres componentes debilmente acoplados
    - Componente Model : Encapsula la funcionalidad nucleo de la aplicación y es independiente de su representación
    - Componente View: Encargado de presentar la información de forma gráfica al usuario
    - Componente Controller: Están asociados con las vistas y el modelo. Permiten manipular el modelo y seleccionar la vista a adecuada



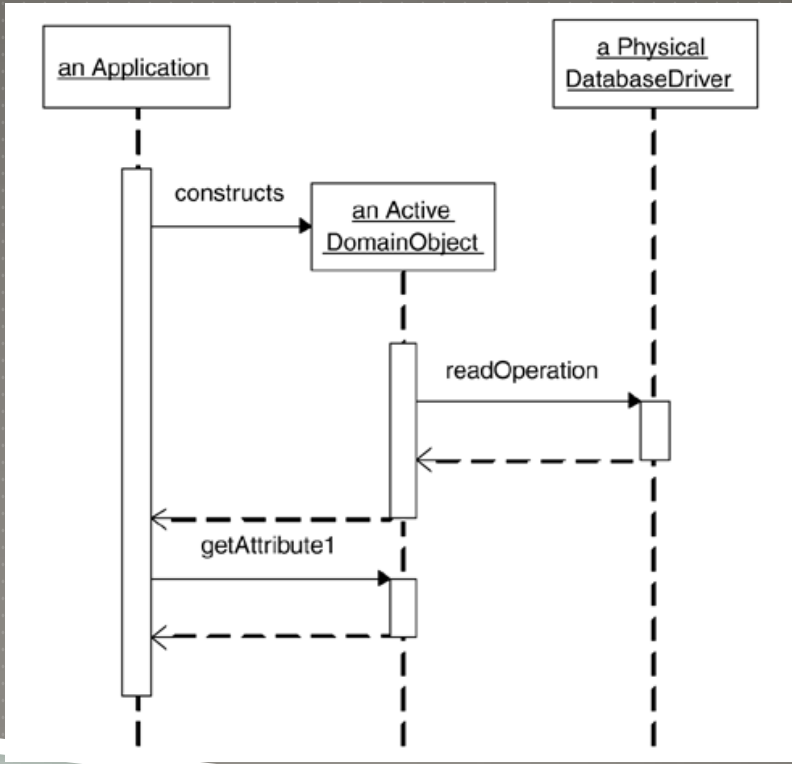
EJEMPLO DE PATRONES DE DISEÑO

- Patrón Active Domain Object (Data Access patterns addison Wedsley 2003)
- Contexto
  - Aplicaciones que manipulan directamente filas de tablas relacionales
- Aplicabilidad
  - Se desea ocultar el modelo de datos físico y la complejidad del acceso a estos datos
  - Se desea encapsular toda la lógica de acceso en un único componente conceptual
  - Se dese enmascarar las posibles inconsistencia en en modelo de datos fisico
- Solución
  - El patrón Active Domain object aborda esta cuestión encapculando el modelo de datos dentro de un conjunto de objetos de dominio
- Estructura del patrón



EJEMPLO DE PATRONES DE DISEÑO (II)

- Comportamiento



EJEMPLO DE PATRONES DE DISEÑO (III)

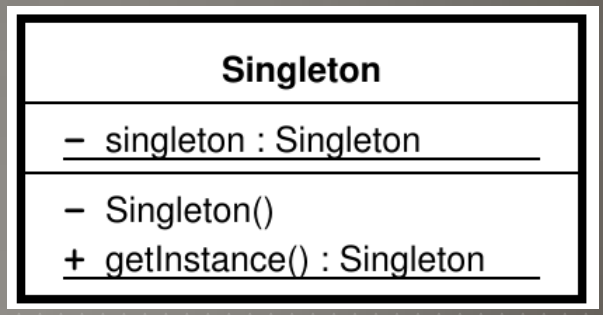
- **Beneficios**
  - Proporciona un codigo de aplicación más limpio y facil de mantener
  - Desacopla el código de aplicación del modelo fisico relacional de datos
- **Inconvenientes**
  - Esparce el código de acceso a datos en múltiple objetos de dominio. Cada active domain object es responsible de su propieo implementación de acceso a datos en la base de datos relacional
  - No es una estructura adecuada para politicas generales de acceso a datos (ej cache, pool de conexiones,etc)
  - Limita el control de acceso a datos.

PATRÓN DE DISEÑO SINGLETON (I)

- Propósito:
  - Asegurar que una clase sólo tiene un ejemplar, y
  - proporcionar un punto de acceso global a éste
- Motivación
  - Algunas clases sólo necesitan exactamente un ejemplar
  - Un *spooler* de impresión en un sistema, aunque haya varias impresoras
  - Un sólo sistema de archivos
  - Un sólo gestor de ventanas
  - ...
  - En vez de tener una variable global para acceder a ese ejemplar único, la clase se encarga de proporcionar unmétodo de acceso

PATRÓN DE DISEÑO SINGLETON (II)

- Aplicación
  - Cuando sólo puede haber un ejemplar de una clase, y debe ser accesible a los clientes desde un punto de acceso bien conocido
  - Cuando el único ejemplar pudiera ser extensible por herencia, y los clientes deberían usar el ejemplar de una subclase sin modificar su código
- Esquema, participantes y colaboraciones
  - Los clientes acceden al ejemplar de Singleton únicamente através del método Instance de la clase Singleton



PATRÓN DE DISEÑO SINGLETON (III)

- **Consecuencias**
  - Acceso controlado a un ejemplar único
  - Espacio de nombres reducido
  - Evita la necesidad de utilizar variables globales
- Se puede heredar de la clase Singleton para configurar el ejemplar para una aplicación concreta
  - Permite un número de ejemplares variable
  - **Es posible tener un conjunto de ejemplares en vez de uno sólo:**
    - **Object Pool**
- Más flexible que las operaciones de clase (static)
- Con static no es posible considerar que hubiera más de un solo ejemplar
- En C++ las funciones static no pueden ser virtuales, y por tanto las subclases no pueden redefinirlas polimórficamente

PATRÓN DE DISEÑO SINGLETON (IV)

- Implementación

```
class Singleton {
private static Singleton instancia = null;
public static Singleton getInstancia() {
                    if ( instancia == null )
                                instancia = new Singleton();
                    return instancia
}
protected Singleton() {
                    // creación de la instancia
}
public void metodo() {...}
}
```

PATRÓN DE DISEÑO SINGLETON (IV)

- **Aplicabilidad**
  - Deba haber exactamente una instancia de una clase y ésta deba ser accesible a los clientes desde un punto de acceso conocido.
  - La única instancia debería ser extensible mediante herencia y los clientes deberían ser capaces de utilizar una instancia extendida sin modificar su código.
- **Consecuencias**
  - Acceso controlado a la única instancia. Puede tener un control estricto sobre cómo y cuando acceden los clientes a la instancia.
  - Espacio de nombres reducido. El patrón Singleton es una mejora sobre las variables globales.
  - Permite el refinamiento de operaciones y la representación. Se puede crear una subclase de Singleton.
  - Permite un número variable de instancias. El patrón hace que sea fácil cambiar de opinión y permitir más de una instancia de la clase Singleton.
  - Más flexible que las operaciones de clase (static en C#, Shared en VB .NET).

EJEMPLOS PATRONES IDIOMS

- **http://c2.com/ppr/wiki/JavaIdioms/JavaIdioms.html**

COMUNIDAD DE INVESTIGACIÓN Y ESTUDIO DE PATRONES

- Comunidad de trabajo sobre patrones
  - http://hillside.net/patterns/
- Pattern Conferences
- – PLoP®since 1994 at AllertonHouse, Monticello, IL*