Curso de Patrones y Buenas Prácticas .Net

Patrones de Diseño

Plataforma: Visual Studio .Net
Framework: Microsoft .Net Framework

Documento de Referencia y Capacitación
(Este documento está sujeto a cambios)

Fecha de Creación: 03-Jun-2010
Versión: 2.0.0.0
Autor: Julio Cesar Robles Uribe

Tabla de Contenido

Introducción

Cuando construimos una pieza de software, desde un pequeño utilitario, hasta un sistema completo distribuido, estamos encarando un trabajo de análisis, diseño y desarrollo. Esa actividad puede realizarse en grupo o individualmente. En muchas situaciones, hemos notado que la implementación de una solución, en un caso de diseño o de programación, es similar a otra que hemos adoptado en el pasado. También suele suceder que descubrimos, más adelante, que la solución adoptada para un problema en particular, sea opacada por otra solución no contemplada inicialmente. Cuantas veces nos "despertamos" a una nueva alternativa de implementación, al ver el código o el esquema de la solución de otro producto. Es común encontrar también pistas e ideas interesantes en los artículos y libros que vamos consultando.

Siendo la creación de software, una actividad humana que se viene desarrollando desde hace décadas, no es extraño que hayan surgido esquemas de soluciones a problemas planteados anteriormente. Y, ante tanta "crisis del sofware", (el siempre presente problema de generar soluciones en tiempo y costo), esta situación ha incentivado la aparición de metodologías de análisis, de diseño, de implementación, y de manejo de proyectos.

En esta sección vamos a ver una breve introducción acerca de los patrones de diseño y su utilización, como norma estándar para plantear el diseño de soluciones con elementos ya probados y de aplicabilidad en la implementación de aplicaciones.

Se detallaran los patrones más conocidos en la industria del software y su aplicación en el diseño de soluciones.

Patrones de Diseño

Los patrones de diseño son soluciones a los problemas recurrentes de software que encontramos una y otra vez en el desarrollo de aplicaciones del mundo real. Los patrones se centran en el diseño y la interacción de los objetos, así como ofrecen una plataforma de soluciones elegantes y reutilizables a problemas que son comúnmente encontrados en programación.

Se puede decir que los patrones de diseño comprimen el conocimiento de experiencias anteriores y pueden utilizarse en crear nuevas soluciones en contextos similares, los patrones tienen en esencia una base empírica, generalmente no son creados sino detectados, por lo que la principal fuente de patrones será tanto la aportación de expertos como el proceso inductivo de los diseñadores.

Los expertos en cualquier campo normalmente no crean nuevas soluciones en cada problema que se presenta, sino que se basan en su experiencia para adecuar soluciones de problemas anteriores (patrones) y aplicarlos en los nuevos problemas.

Breve reseña histórica

En 1979, el arquitecto **Chistopher Alexander** escribe el libro “The Tímeles Way of Building” sobre el uso de patrones en la construcción de edificios, en él proponía el aprendizaje y uso de una serie de patrones para la construcción de edificios de una mayor calidad, lo que contribuyó a que años más tarde se escribiese otro libro “A pattern Language” que fue el primer intento por formalizar los conocimientos arquitectónicos.

En palabras de este autor, "Cada patrón describe un problema que ocurre infinidad de veces en nuestro entorno, así como la solución al mismo, de tal modo que podemos utilizar esta solución un millón de veces más adelante sin tener que volver a pensarla otra vez."

Los patrones que Christopher Alexander y sus colegas definieron, publicados en un volumen denominado “A Pattern Language”, son un intento de formalizar y plasmar de una forma práctica generaciones de conocimiento arquitectónico. Los patrones no son principios abstractos que requieran su redescubrimiento para obtener una aplicación satisfactoria, ni son específicos a una situación particular o cultural; son algo intermedio. Un patrón define una posible solución correcta para un problema de diseño dentro de un contexto dado, describiendo las cualidades invariantes de todas las soluciones.

Más tarde, en 1987, Ward Cunningham y Kent Beck usaron varias ideas de Alexander para desarrollar cinco patrones de interacción hombre-computador (HCI) y publicaron un artículo en OOPSLA-87 titulado Using Pattern Languages for OO Programs.

No obstante, no fue hasta principios de los 90's cuando los patrones de diseño tuvieron un gran éxito en el mundo de la informática a partir de la publicación del libro “**Design Patterns**” escrito por el grupo Gang of Four (GoF) compuesto por Erich Gamma, Richard Helm, Ralph Johnson y John Vlisides, en el que se recogían 23 patrones de diseño comunes.

Definiciones de Patrones

- Son el esqueleto de las soluciones a problemas comunes en el desarrollo de software.
- Los patrones son un punto de partida no un destino.
- Cada patrón no es una solución en sí misma, es la documentación de la forma en que se construyeron las soluciones a problemas similares.
- Los patrones son un gran mecanismo de comunicación para trasmitir la experiencia de los Ingenieros y diseñadores experimentados.

Características de un buen patrón

- Resuelve un problema.
- Es un concepto probado (Debe tener al menos tres implementaciones reales).
- La solución no es obvia.
- Describe una relación.
- Tiene un componente humano significante.

Objetivos de los Patrones

Los patrones de diseño pretenden:

- Proporcionar catálogos de elementos reusables en el diseño de sistemas software.
- Evitar la reiteración en la búsqueda de soluciones a problemas ya conocidos y solucionados anteriormente.
- Formalizar un vocabulario común entre diseñadores.
- Estandarizar el modo en que se realiza el diseño.
- Facilitar el aprendizaje de las nuevas generaciones de diseñadores condensando conocimiento ya existente.
- Reducción de tiempo.
- Disminución del esfuerzo de mantenimiento.
- Aumentar la eficiencia.
- Asegurar la consistencia.
- Aumentar la fiabilidad.
- Proteger la inversión en desarrollos.

Asimismo, no pretenden:

- Imponer ciertas alternativas de diseño frente a otras.
- Eliminar la creatividad inherente al proceso de diseño.

No es obligatorio utilizar los patrones, solo es aconsejable en el caso de tener el mismo problema o similar que soluciona el patrón, siempre teniendo en cuenta que en un caso particular puede no ser aplicable. **Abusar o forzar el uso de los patrones puede ser un error**.

Pattern Happy

- Calificativo aplicable a los programadores, que se refiere al uso indiscriminado de patrones.
- El uso indiscriminado, aún en situaciones donde no aportan ningún beneficio puede ser un **antipatron**.
- Un programador **Pattern Happy** aprende un patrón e “inmediatamente encuentra un lugar para abusar de él”.
- Los patrones no son siempre la solución adecuada o mejor para un problema. Si bien añaden flexibilidad, también añaden complejidad.

Retroalimentación

1. Los patrones son la única solución a los problemas de diseño

   ○ Verdadero ○ Falso

   R/Falso: Los patrones son una forma ya probada de resolver un problema, pero no es la única forma de solucionarlo.

2. Al ser los patrones diseños realizados por seres humanos, pueden tener errores.

   ○ Verdadero ○ Falso

   R/Falso: Los patrones son soluciones ya probadas, aunque no se catalogan como la implementación final no significa que sea un error implementarla o no.

3. Un patrón facilita el desarrollo de soluciones lo que reduce el tiempo de desarrollo.

   ○ Verdadero ○ Falso

   R/Verdadero: Un patrón se puede reutilizar lo que reduce el tiempo de desarrollo ya que la solución ya está desarrollada y probada.

4. Al ser un patrón una solución de fácil implementación se puede utilizar en todas las soluciones y puede repetirse en todo el sistema sin que se vea afectado

   ○ Verdadero ○ Falso

   R/Falso: Abusar del uso un patrón puede ser un error que en lugar de mejorar el rendimiento de la aplicación puede deteriorarlo.

5. El **Pattern Happy** es una metodología que permite utilizar cualquier patrón en la implementación de una solución.

   ○ Verdadero ◉ Falso

   R/Falso: El uso indiscriminado, aún en situaciones donde no aportan ningún beneficio puede ser un **antipatron**.

Patrones de Diseño en el Desarrollo de Software

En el desarrollo de software el patrón de diseño describe las **clases** y **objetos** que se comunicarán entre sí de manera que puedan resolver un problema general de diseño en un contexto particular.

En un contexto informático un patrón de diseño es similar a conceptos como biblioteca de clases, frameworks, técnicas y/o herramientas de refactorización o programación extrema.

Los patrones tiene niveles de aplicación según el contexto en el que se desarrollen, este es el caso que a nivel de la definición de la arquitectura de un sistema, se detallen **Patrones de Arquitectura**, de igual forma en el diseño se detallan **Patrones de Diseño**, que en este caso son el tema central de esta sesión, por último y a más bajo nivel están los **Patrones de Implementación** que ya toman en cuenta la forma en cómo se construye el código.

Patrones Según el Nivel de Detalle

- **Patrones de Arquitectura:** Proveen un conjunto de subsistemas predefinidos que especifican sus responsabilidades e incluyen reglas y guías para organizar las relaciones entre ellos.

- **Patrones de Diseño:** Describen una estructura de componentes que se comunican para resolver un problema general de diseño en un contexto particular. Son muy útiles para crear un diseño orientado a objetos reutilizable.

- **Patrones de Implementación:** Especifican de forma detallada la implementación de un componente o subsistema, que aplica para una plataforma o lenguaje en particular.

Tipos de Patrones de Diseño

Los patrones son considerados como el fundamento de todos los otros diseños y se clasifican en tres grupos: Creacionales, Estructural y de Comportamiento.

- **Creacionales:** Abstraen el proceso de instanciación. Nos ayudan a independizar a un sistema, de cómo sus objetos son creados. En general, tratan de ocultar las clases y métodos concretos de creación, de tal forma que al variar su implementación, no se vea afectado el resto del sistema

- **Estructura:** Se ocupan de cómo clases y objetos se agrupan, para formar estructuras más grandes. Podemos nombrar al clásico patrón Composite, que permite agrupar varios objetos como si fueran uno solo, y tratar al objeto compuesto de una forma similar al simple.

- **Comportamiento:** Más que describir objetos o clases, sino la comunicación entre ellos. Frecuentemente, describen las colaboraciones entre distintos elementos, para conseguir un objetivo.

Patrones Reconocidos

Dentro de los patrones más reconocidos están:

Patrones de Comportamiento	Patrones de Creación	Patrones Estructurales
1. Chain of Responsibility 2. **Command*** 3. Interpreter 4. Iterator 5. Mediator **6. Memento*** 7. **Observer*** 8. State **9.** Strategy 10. Template Method 11. Visitor	12. **Abstract Factory*** 13. Builder 14. **Factory*** 15. Prototype 16. **Singleton***	17. **Adapter*** 18. Bridge 19. Composite 20. Decorator 21. **Facade*** 22. Flyweight 23. **Proxy***

Loa patrones marcados con asterisco (*) serán desarrollados en esta sesión por ejemplos de cada uno de los tipos.

Cuando (no) utilizar patrones de diseño

La primera regla de los patrones de diseño coincide con la primera regla de la optimización: retrasar. Del mismo modo que no es aconsejable optimizar prematuramente, no se deben utilizar patrones de diseño antes de tiempo. Seguramente sea mejor implementar algo primero y asegurarse de que funciona, para luego utilizar el patrón de diseño para mejorar las flaquezas; esto es cierto, sobre todo, cuando aún no ha identificado todos los detalles del proyecto (si comprende totalmente el dominio y el problema, tal vez sea razonable utilizar patrones desde el principio, de igual modo que tiene sentido utilizar los algoritmos más eficientes desde el comienzo en algunas aplicaciones).

Los patrones de diseño pueden incrementar o disminuir la capacidad de comprensión de un diseño o de una implementación, disminuirla al añadir accesos indirectos o aumentar la cantidad de código, disminuirla al regular la modularidad, separar mejor los conceptos y simplificar la descripción. Una vez que aprenda el vocabulario de los patrones de diseño le será más fácil y más rápido comunicarse con otros individuos que también lo conozcan. Por ejemplo, es más fácil decir “ésta es una instancia del patrón Visitor” que “éste es un código que atraviesa una estructura y realiza llamadas de retorno, en tanto que algunos métodos deben estar presentes y son llamados de este modo y en este orden”.

La mayoría de las personas utiliza patrones de diseño cuando perciben un problema en su proyecto —algo que debería resultar sencillo no lo es — o su implementación— como por ejemplo, el rendimiento. Examine un código o un proyecto de esa naturaleza. ¿Cuáles son sus problemas, cuáles son sus compromisos? ¿Qué le gustaría realizar que, en la actualidad, es muy difícil lograr? A continuación, compruebe una referencia de patrón de diseño y busque los patrones que abordan los temas que le preocupan.

La referencia más utilizada en el tema de los patrones de diseño es el llamado libro de la “banda de los cuatro”, Design Patterns: Elements of Reusable Object-Oriented Software por Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides, Addison-Wesley, 1995. Los patrones de diseño son muy populares en la actualidad, por lo que no dejan de aparecer nuevos libros.

¿Por qué preocuparse?

Si es usted un programador o un diseñador brillante, o dispone de mucho tiempo para acumular experiencia, tal vez pueda hallar o inventar muchos patrones de diseño. Sin embargo, esta no es una manera eficaz de utilizar su tiempo. Un patrón de diseño es el trabajo de una persona que ya se encontró con el problema anteriormente, intentó muchas soluciones posibles, y escogió y describió una de las mejores. Y esto es algo de lo que debería aprovecharse.

Los patrones de diseño pueden parecerle abstractos a primera vista o tal vez no tenga la seguridad de que se ocupan del problema que le interesa. Comenzará a apreciarlos a medida que construya y modifique sistemas más grandes.

Práctica

1. Para el ejercicio de capas de la sesión anterior, aplique lo visto en patrones de diseño para garantizar que se tenga una sola conexión hacia la base de datos.
2. Debe tener en cuenta que los cambios que se realicen a los proyectos sean mínimos y que los métodos de las clases no deben cambiar.
3. Al finalizar debe eliminar la utilización directa, desde los proyectos, de la librería de **EandT.Framework.Data.Base**.
4. Recuerde que la conexión a la base de datos debe abrirse y cerrarse en el mismo punto o componente.

Recomendaciones

1. Cree una nueva librería de acceso a datos llamada **DirTel.Utilities** y en ella aplique lo visto en patrones.
2. Revise los patrones de **Singleton**, **Proxy** y **Bridge**.
3. Desde esta nueva librería puede referenciar o utilizar la librería de **EanT.Framework.Data.Base**.
4. Tome como referencia el formulario principal **MDIMain** para abrir y cerrar la conexión.