Artículo **Revista de Tecnología e Innovación**
Marzo 2018 Vol.5 No.14 35-40

Implementación del Patrón de Diseño Fábrica Abstracta "Abstract Factory" en dos Módulos diferentes dentro de un mismo Dominio

Implementation of Abstract Factory Design Pattern in two Different Modules within the same Domain

GUTIÉRREZ-TORRES, Ludivina[1]†*, ZÚNIGA-FÉLIX, Ismael[2], SÁNCHEZ-HERNÁNDEZ, Zindi[1] y SAINZ-ALFARO, Francisca[1]

[1]Instituto Tecnológico de Nogales, Departamento de Posgrado e Investigación, Ave. Instituto Tecnológico #911, Colonia Granja, CP. 84065, H. Nogales, Sonora, Mexico DEPI
[2]Instituto Tecnológico de Agua Prieta, Sistemas y Computación, Carretera a Janos y Ave. Tecnológico s/n, Colonia Progreso CP, 84200, Agua Prieta, Sonora, México. Ingeniería en Sistemas Computacionales

ID 1º Autor: *Ludivina Gutiérrez-Torres* /**ORC ID**: 0000-0003-2450-6607, **Researcher ID Thomson**: F-4360-2018, **arXiv ID**: LudyGtz

ID 1º Coautor: *Ismael Armando Zúñiga-Félix* /**ORC ID**: 0000-0003-3085-4253, **Researcher ID Thomson**: F-1872-2018, **arXiv ID**: iazuniga

ID 2º Coautor: *Zindi Sánchez-Hernández* /**ORC ID**: 0000-0002-0211-2378, **Researcher ID Thomson**: F-4328-2018, **arXiv ID**: zindi.sanchez

ID 3º Coautor: *Francisca Irene Saiz-Alfaro* /**ORC ID**: 0000-0003-2728-7542, **Researcher ID Thomson**: F-5366-2018, **arXiv ID**: irenesaiz

Recibido: 20 de Diciembre, 2017; Aceptado 22 de Febrero, 2018

Resumen

Los patrones de diseño están hoy en día, ampliamente difundidos. Y es posible aplicarlos profusamente, sin embargo, mientras que los diseñadores orientados a objetos con experiencia realizan buenos diseños, los diseñadores principiantes por falta de práctica recurren a opciones no orientadas a objetos que han utilizado previamente. Este trabajo muestra la utilización practica del patrón de diseño creacional Fábrica Abstracta el cual es aplicado en dos contextos diferentes (dos módulos diferentes) dentro de un mismo dominio. El principal objetivo de esta propuesta es mostrar que lo que dijo Christopher Alexander acerca de que la solución a un problema con patrones puede ser usada un millón de veces más sin hacerlo dos veces de la misma forma. Obteniéndose soluciones orientadas a objetos más flexibles, entendibles y en última instancia reutilizables

Programación orientada a Objetos, Patrones de diseño, Patrón de diseño Fábrica Abstracta

Abstract

The design patterns are nowadays widely disseminated. And it is possible to apply them greatly, however, while experienced object-oriented designers make good designs, beginning designers feel overwhelmed and use non-object-oriented options that they have previously used. This work shows the practical use of the creational design pattern Abstract Factory, which is applied in two different contexts (two different modules) within the same domain. The main objective of this proposal is to show that what Christopher Alexander said about the solution to a problem with patterns can be used a million times more without even doing it twice in the same way. Obtaining object-oriented solutions more flexible, understandable and ultimately reusable.

Object Oriented Programming, Design Patterns, Abstract Factory

Citación: GUTIÉRREZ-TORRES, Ludivina, ZÚNIGA-FÉLIX, Ismael, SÁNCHEZ-HERNÁNDEZ, Zindi y SAINZ-ALFARO, Francisca. Implementación del Patrón de Diseño Fábrica Abstracta "Abstract Factory" en dos Módulos diferentes dentro de un mismo Dominio. Revista de Tecnología e Innovación 2018, 5-14: 35-40

* Correspondencia al Autor (Correo electrónico: ludygt@depiitn.edu.mx)
†Investigador contribuyendo como primerAutor.

Introducción

Bosch, Molin, Mattson en el año 2000 dijeron que los conceptos orientados a objetos han existido desde hace ya más de tres décadas, actualmente ya tienen más de cuatro décadas de existir. Son una parte esencial de la programación de computadoras (Bosch, Molin, Mattson, & Bengtsson, 2000). Han proporcionado un gran soporte para comprender la construcción del diseño de aplicaciones (Lajoie & Keller, 1995), (Bosch, Molin, Mattson, & Bengtsson, 2000).

En estas cuatro décadas la orientación a objetos ha tenido grandes avances siendo uno de los más relevantes el uso de los patrones de diseño. Sin embargo a pesar de esto, sigue siendo una realidad las palabras de Gamma, Helm¸ Johnson y Vlissides en 1994 cuando dijeron que diseñar software orientado a objetos es difícil, y diseñar software reutilizable orientado a objetos es aún más difícil. Deben encontrarse los objetos pertinentes, factorizarlos en clases con la granularidad correcta, definir interfaces de clase y jerarquías de herencia, y establecer relaciones clave entre ellos. Su diseño debe ser específico para el problema en cuestión, pero también lo suficientemente general para abordar problemas y requerimientos futuros.

Vijay K Kerji en 2011 en su trabajo Abstract Factory and Singleton Design Patterns to create Decorator Pattern objects in Web application encontró que el uso de los patrones Fábrica Abstracta "Abstract Factory" y Singleton que pertenecen a la categoría de patrones creacionales dio como resultado un mejor rendimiento y más facilidad para mantener el código de la aplicación para futuros cambios de requisitos. (Kerji, 2011).

Huaxin Mu & Shuan Jiang en 2011 en su trabajo "Design patterns in software development" proporciona un ejemplo del uso del patrón de diseño Fábrica Abstracta utilizado en un módulo de encriptamiento para proporcionar una única interfaz con un solo punto de acceso para varios tipos de métodos de encriptación. La implementación de patrones de diseño de software en el desarrollo de los sistemas dará el beneficio de un mejor diseño estructural del sistema y una mayor flexibilidad para las mejoras futuras de los sistemas. (Huaxin & Shuai, 2011)

Patrones de diseño

Christopher Alexander dijo: "Cada patrón describe un problema el cual sucede en nuestro entorno, y luego describe el núcleo de la solución a ese problema, de tal forma que pueda utilizar esa solución un millón de veces más, sin siquiera hacerlo dos veces de la misma forma" (Christopher Alexander, 1977). Como mencionan Gamma Helm¸ Johnson y Vlissides en 1994 a pesar de que Alexander estaba hablando de patrones en edificios y ciudades, lo que dijo es cierto acerca de los patrones de diseño orientados a objetos. Nuestras soluciones se expresan en términos de objetos e interfaces en lugar de paredes y puertas, pero el núcleo de ambos tipos de patrones es una solución a un problema dentro de un contexto.

En la década de los noventa el auge de los patrones de diseño alcanzó más popularidad entre los desarrolladores de software orientado a objetos. Con la publicación del libro *"Design Patterns: Elements of Reusable Object Oriented Sofware"* escrito por Erich Gamma, Richard Helm, Ralp Johnson y John Vlissides conocidos actualmente como "*Gang of Four"* abreviado por las siglas GoF; en éste fueron recopilados 23 patrones de diseño, los cuales fueron clasificados en creacionales, estructurales y de comportamiento. (Gamma Erich, 1994)

Los patrones de diseño tienen 4 elementos esenciales:

- Un nombre de patrón que representa el problema y su solución.
- El problema describirá cuando aplicar el patrón. Explica el problema y su contexto.
- La solución proporciona una descripción abstracta del diseño y como participan los elementos que intervienen.
- Las consecuencias explican los resultados de aplicar el patrón, sus costes y beneficios. (Gamma Erich, 1994)

Un patrón de diseño es una "descripción de objetos comunicándose y clases que son personalizadas para resolver un problema general de diseño dentro de un contexto particular" Los diseñadores pueden incorporar patrones en sus programas para direccionar los problemas generales dentro de la estructura de los diseños de sus programas. (Gamma Erich, 1994)

ISSN-2410-3993

Los patrones se implementan a partir de diseños orientados a objetos, donde se identifican clases e instancias, sus roles y colaboraciones y se describen la distribución de responsabilidades. Usar un lenguaje orientado a objetos nos proveerá el soporte necesario para aplicar e implementar patrones de diseño. (Fco. José García Peñalvo, 1998). Para realizar este trabajo se ha tomado el patrón de diseño creacional Fábrica Abstracta, debido a que se adapta correctamente al dominio con el que se trabajó.

Metodología

Este trabajo corresponde al dominio de una Estación Meteorológica Automática, la cual se encuentra ubicada dentro de las instalaciones del Instituto Tecnológico de Nogales (ITN). Este proyecto consiste en desarrollar un sitio Web que permita visualizar los datos extraídos de ella. La estación Meteorológica del ITN genera información de variables Meteorológicas tales como: temperatura del aire, precipitación pluvial, humedad ambiental,

Radiación solar, velocidad del viento, dirección del viento, entre otras. Debido a que no existe una aplicación donde se puedan consultar los datos generados por la estación, se pretende diseñar e implementar un software para consultar tal información. Para realizar este proyecto se obtuvieron los requerimientos funcionales que integran el diseño de la aplicación. En el presente caso se toman dos módulos diferentes de este mismo dominio que son los siguientes: Módulo No. 1. Reportes y Gráficas de variables meteorológicas, Módulo No. 2. Capa de Conexión de Base de Datos.

Patrón de diseño Fábrica Abstracta

En este trabajo se utilizará el patrón de diseño Fabrica Abstracta el cual pertenece a la categoría de los patrones creacionales. El patrón de diseño creacional Fábrica Abstracta, consiste en establecer una interfaz para crear objetos pertenecientes a la misma familia de objetos relacionados o dependientes sin especificar sus clases concretas. (Freeman, 2004) (Gamma Erich, 1994)

El objetivo es usar este mismo patrón en dos módulos diferentes dentro del mismo dominio. Comprobando lo que dice Christopher.

RF	Descripción	Categoría
3.1	El sistema permitirá al usuario obtener los datos en forma de listado, del comportamiento de la variable meteorológica Temperatura.	Evi-dente
3.2	La página permitirá al usuario obtener la gráfica del comportamiento de la variable meteorológica Temperatura.	Evi-dente

Tabla 1 Requerimientos funcionales del Módulo Reportes y Gráficas de Variables Meteorológicas

Alexander, en cuanto a que el patrón de diseño describe el núcleo de la solución a un determinado problema, de tal forma que se pueda utilizar esa solución un millón de veces más, sin hacerlo dos veces de la misma forma (Christopher Alexander, 1977).

Módulo No. 1: Reportes y Gráficas de variables meteorológicas.

Este módulo se utiliza para realizar reportes y graficas de las variables meteorológicas como son temperatura del aire, precipitación pluvial, humedad ambiental, radiación solar, velocidad del viento, dirección del viento, entre otras. En la Tabla No.1 se pueden ver los requerimientos funcionales del Módulo Reportes y Gráficas de variables meteorológicas y se da una muestra de dos de los requerimientos funcionales del mismo. Se crea una fábrica abstracta para generar las gráficas y reportes de los datos meteorológicos utilizando el Patrón de Diseño Fábrica Abstracta, de tal manera que se puedan generar reportes y gráficas de cualquier variable meteorológica.

Lo anterior tiene la gran ventaja de que si se agregan nuevas estaciones meteorológicas, es posible utilizar la misma fábrica, lo que nos proporciona una gran reusabilidad de código. En la Figura No.1. se puede ver el Modelo del patrón de diseño de Fábrica Abstracta del módulo Reportes y Gráficas de variables meteorológicas y se puede observar la implementación del patrón de diseño para este caso particular.

ISSN-2410-3993

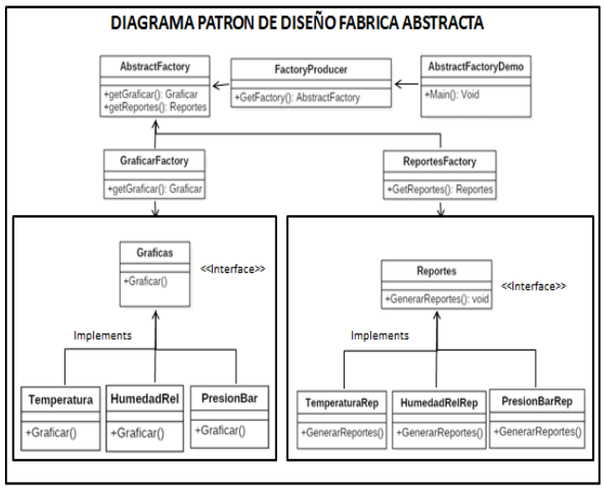

Figura 1 Modelo del patrón de diseño de Fábrica Abstracta del módulo de Gráficas y Reportes

Implementación No. 2:

Descripción del módulo ***Capa de Conexión de Base de Datos.***

Este módulo llamado "***Capa de Conexión de Base de Datos***" consiste en el desarrollo de un patrón de diseño de fábrica abstracta en la plataforma de **Microsoft .NET** basado en lenguaje C# que controla el manejo de clases de conexión entre el sistema o cliente y el gestor de base de datos. Con el cual se pretende agilizar y reutilizar código a la hora de aplicar una nueva conexión de base de datos ya sea en un mismo manejador de base de datos o en un nuevo entorno utilizando este mismo módulo.

Gracias a esto se agrega una propiedad de flexibilidad al sistema el cual nos permite cambiar de una aplicación desarrollada para Microsoft SQL Server a Oracle o MySQL sin necesidad de reescribir código agilizando así cualquier desarrollo. (Patel, 2012)

Para realizar este módulo se partió de la problemática que se puede llegar a generar al desarrollar un sistema bajo solo un gestor de base de datos ya que al desarrollar un sistema muchas veces no se tiene en cuenta que los requerimientos del sistema pueden llegar a cambiar al igual que los manejadores de base de datos.

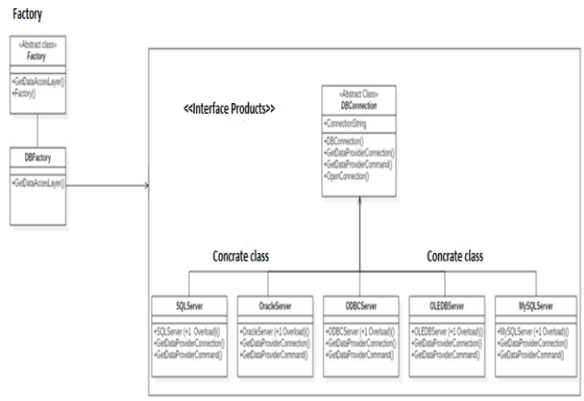

Figura 2 Modelo del patrón de diseño de Fábrica Abstracta

Con el objetivo de no reescribir código es muy importante haber considerado diferentes capas de conexión. Como se muestra con notación UML en el modelo de la Figura No. 2.

- **Products**: interfaz para definir los productos dentro de la clase abstracta ***DBConnection***.
- **Concrete Class** : implementa el producto para definir el objeto concreto.
- **Factory** : clase abstracta utilizada para crear la fábrica.
- **DBFactory** : implementa los métodos declarados dentro de la fábrica abstracta.

A continuación se presenta el código para este patrón de diseño:

Paso 1: crear la interface Products

```
public  interface Product
 {
    IDbConnection                    OpenConnection(string
Connetion_String);
    IDbCommand GeDataProviderCommand();
    IDbConnection GetDataProviderConnection();
  }
```

Paso 2: crear la clase abstracta DBConnection heredando de la interface Products

```
 public abstract class DBConnection : Product
  {
      private IDbConnection connection;
      public DBConnection() { }
      public                abstract                IDbConnection
GetDataProviderConnection();
      public                abstract                 IDbCommand
GetDataProviderCommand();
      public      IDbConnection      OpenConnection(string
Connetion_String)
      {
         try
         {
          connection = GetDataProviderConnection();
          connection.ConnectionString = Connetion_String;
          connection.Open();
         }
```

ISSN-2410-3993

```
        catch ( Exception )
        {
           connection.Close();
        }
        return connection; ;
     }
  }
```

Paso 3: crear las clases concretas para las diferentes conexiones heredando de la clase abstracta

SQL:

```
using System.Data.SqlClient;
public class SQLServer : DBConnection
  {
     public SQLServer() { }
     public override IDbConnection
GetDataProviderConnection()
     {
        return new SqlConnection();
     }
     public override IDbCommand
GeDataProviderCommand()
     {
        return new SqlCommand();
     }
  }
```

Oracle:`using System.Data.OracleClient;`
ODBC:`using System.Data.Odbc;`
OLE:`using System.Data.OleDb;`
MySQL: `using MySql.Data.MySqlClient;`

Se generan las clases concretas con los métodos de GetDataProviderConnection y GetDataProviderCommand regresando el tipo de las clases concreta.

Paso 4: Crear la clase Factory

```
  public enum DataProviderType
  {
     Access, Odbc, OleDb, Mysql, Oracle, Sql
  }
  public abstract class Factory
  {
     public abstract Product
GetDataAccessLayer(DataProviderType
dataProviderType);
  }
```

El código anterior se muestra en la Figura No. 3.

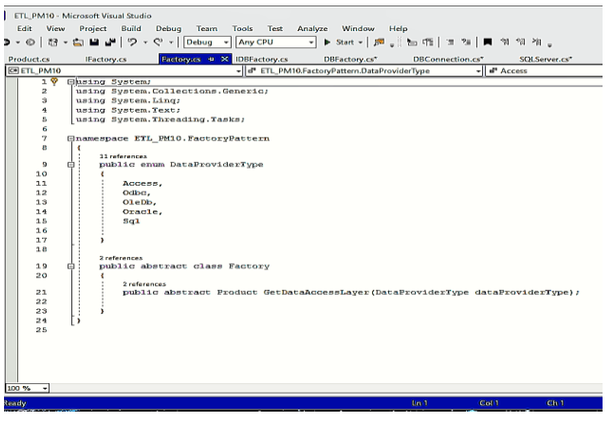

Figura 3 Muestra el código de la clase Factory

Paso 5: crear la clase DBFactory heredando de la clase Factory

```
public class DBFactory : Factory
  {
     public override Product
GetDataAccessLayer(DataProviderType
dataProviderType)
     {
        switch (dataProviderType)
        {
           case DataProviderType.Access:
           case DataProviderType.OleDb:
              return new OLEDBServer();
           case DataProviderType.Mysql:
              return new MySQLServer();
           case DataProviderType.Odbc:
              return new ODBCServer();
           case DataProviderType.Oracle:
              return new OracleServer();
           case DataProviderType.Sql:
              return new SQLServer();
           default:
              throw new ArgumentException("Acceso
invalido.");
        }
     }
  }
```

Paso 6: implementación en el software.

```
Factory Factory = new DBFactory();
 var Layer =
Factory.GetDataAccessLayer(DataProviderType.Mysql);
 IDbConnection con = Layer.OpenConnection("Connection
String");
```

En la Figura No. 4 se muestra el resultado de la ejecución del código, que da como resultado una conexión exitosa.

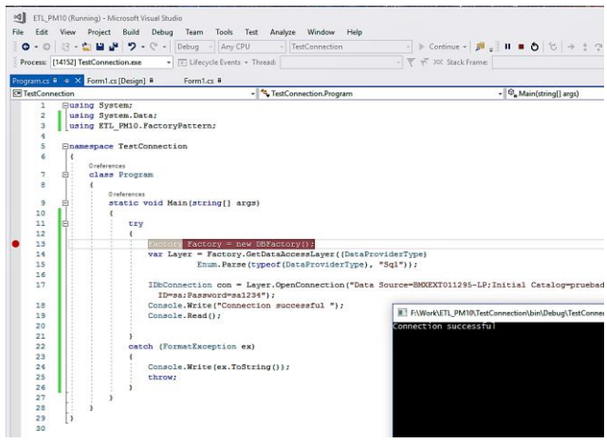

Figura 4 Conexión exitosa

Conclusiones

En las secciones anteriores de este artículo se puede observar cómo es posible utilizar un mismo patrón de diseño, en este caso el patrón de Fábrica Abstracta, en dos módulos diferentes dentro de un mismo proyecto de desarrollo de software.

ISSN-2410-3993

En el primer módulo se implementó una fábrica de Graficas y Reportes y en el segundo una fábrica para la Capa de Conexiones de Bases de Datos, como se pudo ver son dos contextos completamente diferentes aún cuando pertenezca al mismo dominio. En el Modelado UML de los módulos se utilizan interfaces, clases abstractas y clases concretas con sus respectivas relaciones.

Las principales ventajas de utilizar patrones de diseño son:

- Lograr mayor flexibilidad en el código.
- Para los programadores el código es más fácil de comprender, debido a que son soluciones usadas comúnmente.
- Agilizar el desarrollo del software.
- El mantenimiento es más fácil ya que cualquier ingeniero puede entender fácilmente como se realizó el diseño.
- Los diseños son más flexibles.
- Es fácil incorporar nuevos requerimientos, realizando solo pequeños cambios en el diseño.

Con la implementación de este patrón dentro de la aplicación se demostró que los patrones son tan versátiles que es posible usarlos en prácticamente todos los módulos presentes en un mismo dominio en particular que presenten las condiciones para hacerlo. Como dijeron Gamma, Helm¸ Johnson y Vlissides en 1994 los desarrolladores expertos saben que no deben resolver todos los problemas desde cero. Más bien reutilizan soluciones que les funcionaron en el pasado. Cuando encuentran una buena solución, la usan una y otra vez.

Tal experiencia es lo que los hace ser expertos. En consecuencia se pueden encontrar patrones recurrentes, de clases y objetos de comunicación, en muchos sistemas orientados a objetos. Estos patrones resuelven problemas de diseño específicos y hacen que los diseños orientados a objetos sean más flexibles, elegantes y reutilizables.

Lo cual ayuda a los diseñadores a reutilizar diseños exitosos al basar sus nuevos diseños en experiencias anteriores. Un diseñador que esté familiarizado con dichos patrones puede aplicarlos inmediatamente a problemas de diseño sin tener que redescubrirlos. Por lo que se puede concluir que conocer y usar los patrones de diseño, ayuda a todo diseñador a establecer mejores prácticas de desarrollo de software.

Agradecimiento

Al TecNM/Instituto Tecnológico de Nogales (ITN) por haber otorgado a la Ing. Francisca Irene Sainz Alfaro una beca del 70% del costo en los créditos de la Maestría en Sistemas Computacionales.

A los trabajadores del ITN quienes cooperaron económicamente para la publicación de este artículo; así como también a la Delegación Sindical D-V-99 del ITN por complementar el monto requerido y gestionar los recursos.

Al Sr. Jorge Frey por su donativo para hacer posible esta publicación.

Referencias

Bosch, J., Molin, P., Mattson, M., & Bengtsson, P. (2000). Object-Oriented framework-based software development: Problems and experiences. *ACM Comput. Surv 32* .

Christopher Alexander. (1977). A Pattern Language.

Fco. José García Peñalvo. (1998, Nov). Patrones De Alexander a la Tecnología de Objetos. *Universidad de Salama*.

Freeman, E. (2004). Head First Design Patterns. *O'Reilly Media*.

Gamma Erich, H. R. (1994). Design Patterns Elements of reusable Object-Oriented Software.

Huaxin , M., & Shuai, J. (2011). Design patterns in software development. *Software Engineering and Service Science, 2011 IEEE 2nd International Conference*.

Kerji, V. K. (2011). ABSTRACT FACTORY AND SINGLETON DESIGN PATTERNS TO CREATE DECORATOR PATTERN OBJECTS IN WEB APPLICATION. *Interntional Journal of Advanced Information Technology*, 1-9.

Lajoie, R., & Keller, R. K. (1995). *Design and Reuse in Object Oriented frameworks: Patterns, contracts and motifs in concert.* ISBN 9789812831163.

Patel, A. (2012). *Design Patterns & Practices*. Retrieved from www.c-sharpcorner.com.

ISSN-2410-3993