ENCAPSULACION DEL PATRÓN OBSERVER EN ECAESARJ

Gabriela Cointa Pantoja Aráoz *
Ulises Juárez Martínez *
Celia Romero Torres *
Hilarión Muñoz Contreras *
Ma. Antonieta Abud Figueroa *
* Instituto Tecnológico de Orizaba

Resumen

El desarrollo de componentes de software proporciona un alto nivel de reutilización además de ayudar en el mantenimiento de sistemas de software. Encapsular en forma de componente de software el patrón de diseño Observer aumenta el grado de reutilización debido a que la reutilización de una solución a nivel de diseño pasa al nivel de implementación en forma directa. El patrón de diseño Observer está asociado al patrón arquitectónico MVC y dada su capacidad modular de notificación ante los eventos que un sistema genera, su encapsulación como componente tiene un fuerte impacto en la arquitectura del software.

Introducción

La reutilización en la ingeniería de software es considerada como un atributo de calidad en los productos de software debido a que ofrece beneficios en cuestiones de tiempo, costos y esfuerzo. La identificación de problemas recurrentes y sus soluciones plasmadas en los patrones de diseño catalogados por Gamma proporcionan un ejemplo de reutilización. Sin embargo esta reutilización se realiza solo a nivel de diseño.

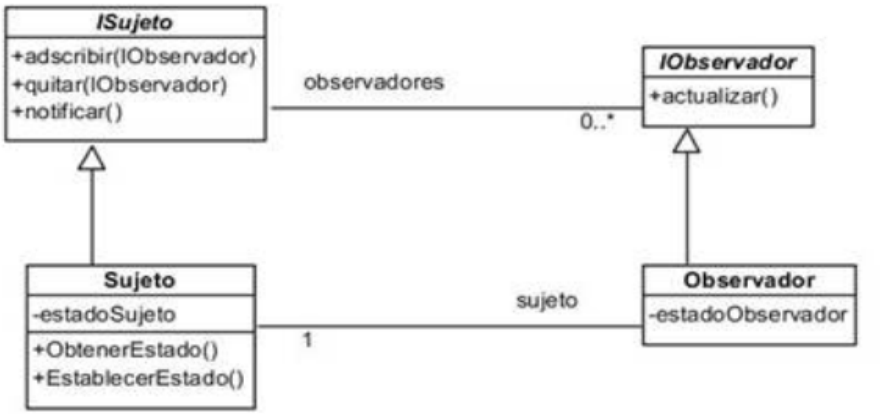
En cuanto a reutilización a nivel de código existen componentes que proporcionan partes de código que se utilizan en sistemas de software de acuerdo a las necesidades que se presenten. Además los componentes proporcionan beneficios desde la construcción del software debido a que proveen piezas ya fabricadas y listas para su utilización, y mejoran el mantenimiento y evolución de software debido a que los componentes al ser piezas encapsuladas permiten aislar la información y realizar modificaciones sin afectar el software que lo utilice.

Actualmente los lenguajes orientados a aspectos ofrecen un alto nivel de modularidad, lo cual permite desarrollar componentes. Entre estos lenguajes destaca ECaesarJ el cual es un lenguaje orientado a aspectos basado en eventos que facilita la modularidad y el desarrollo de componentes reutilizables. El desarrollo de componentes impacta ampliamente el aspecto arquitectónico en la construcción de sistemas viéndose reflejado esencialmente en la reutilización. ECaesarJ es un lenguaje enfocado a la construcción de las líneas de productos de software, las cuales proponen un desarrollo ingenieril en el cual todos los productos están estandarizados y resuelven diferentes necesidades dentro del mismo contexto.

En este artículo se presenta la encapsulación del patrón Observer debido a la importancia que tiene desde el punto de vista arquitectónico al formar parte del patrón arquitectónico MVC y la característica de realizar una acción al cambio de estado. Los eventos y cambios de estado se ven beneficiados con el uso de eventos que ECaesarJ proporciona junto con una mejor manera de administrar dichos cambios. A su vez se destaca la reutilización del patrón como componente de software en la construcción de sistemas.

Estructura e importancia del patrón Observer

El patrón Observer también conocido como Publicación – Subscriptor, notifica a los objetos que dependen de él si un objeto cambia de estado, de esta forma, los objetos se actualizan automáticamente de acuerdo a los cambios realizados. En el patrón Observer se definen dos roles importantes, el rol del Observador y el rol del Sujeto a ser observado. A continuación se muestra la estructura del patrón Observer.

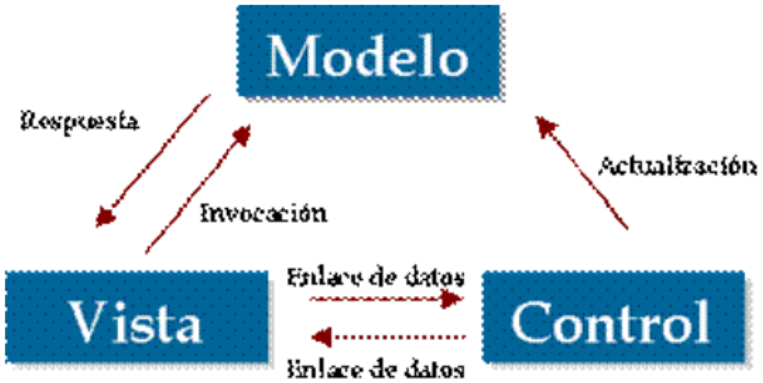


El patrón Observer es uno de los patrones más conocidos que al implementarlo en forma de componente se obtienen beneficios importantes. Es necesario tener clara la idea de lo que significa un componente: una unidad de composición con interfaces especificadas en forma contractual. Estas interfaces establecen condiciones entre el proveedor del servicio y el usuario, y son el punto de acceso de los usuarios para acceder al servicio que se presta.

Desde el punto de vista arquitectónico el conjunto de componentes conectados entre sí forman una arquitectura, la cual permite definir la estructura de un sistema de software. Cada uno de los componentes ofrece funcionalidad que ayuda a cumplir el objetivo por el cual se decide crear el sistema. La arquitectura es quien determina la independencia que tendrá cada componente en la arquitectura para solucionar las funciones que tiene a su cargo y también determina la cooperación que cada componente tendrá con otros componentes. Por lo tanto es importante que cada componente cuente con los requisitos que la arquitectura propone, ya que son la parte medular de las arquitecturas de software.

Por lo tanto al encapsular el patrón Observer en forma de componente de software se ofrecen beneficios a nivel de reutilización arquitectónico, ya que este componente es capaz de formar parte de diferentes arquitecturas en donde se identifique que es necesaria la utilización de dicho patrón.

El patrón Observer se incorpora patrón arquitectónico MVC (Model –View - Controller, Modelo - Vista - Controlador). El modelo arquitectónico MVC propone la separación de una implementación, lo cual ayuda al mantenimiento de un sistema al estar definidos claramente los componentes que son parte de la vista, del control y del modelo. Con el patrón Observer es posible definir la interacción entre el modelo y la vista, lo cual se muestra en la siguiente figura. Debido a que la conexión entre el modelo y la vista es de manera dinámica y se produce a tiempo de ejecución, es necesario reflejar los cambios que se produzcan para mantener un estado de consistencia.



Implementación en ECaesarJ

Para la encapsulación de este patrón de diseño se utilizó el lenguaje ECaesarJ, lenguaje que integra características de su antecesor CaesarJ y el paradigma orientado a aspectos dirigido por eventos.

ECaesarJ es un lenguaje que se enfoca en el desarrollo de líneas de productos de software debido a la capacidad de modularizar la variabilidad de una línea de productos de software mediante características como lo son: clases virtuales, eventos y máquinas de estados.

ECaesarJ permite la identificación de eventos, es decir sucesos de interés para los observadores, los eventos que se permiten en este lenguaje son implícitos y explícitos. Los eventos explícitos, como su nombre lo dice, se especifican explícitamente por el desarrollador (se generan desde un origen), estos eventos sirven para que los destinatarios reaccionen según el evento que ocurrió en la fuente origen. Por otro lado los eventos implícitos son comparados con elementos que expongan otros eventos o valores de campos o variables que estén definidas y que le sea posible acceder, es decir, cualquier acceso a variables, campos o ejecuciones de métodos y constructores. Además es posible utilizar operadores lógicos como lo son: && (and), || (or), y ! (not). Estos eventos se consideran implícitos porque no requieren notación adicional (explícita) por parte del desarrollador.

Adicionalmente ECaesarJ implementa el mecanismo de máquinas de estado el cual esencialmente es un manejador de eventos. Este mecanismo permite controlar de manera lógica los estados de un objeto en forma organizada, mejorando la flexibilidad y estabilidad de características basadas en la descomposición de comportamiento (no está disponible en lenguajes orientados a objetos como Java). Las máquinas de estados en ECaesarJ se integran definiendo dentro de clases el estado inicial y las transiciones a cada estado.

Las características presentes en ECaesarJ para la implementación del patrón Observer impactan de manera importante debido a que ECaesarJ por naturaleza ofrece el soporte de eventos explícitos.

Para la implementación del patrón Observer se define una interfaz para el Observador con el método actualizar que permite actualizar el estado de los sujetos observados. Por el lado del sujeto se definen métodos para agregar, eliminar, contar el número de observadores y notificar. El método notificar es el encargado de disparar el evento definido en ECaesarJ para notificar a los observadores el cambio de estado que acaba de tener el sujeto. A continuación se muestra el fragmento de código que muestra en el evento de notificación para el patrón Observer en ECaesarJ.

¿p() * 0.5000 ¿p() * 0.5000@

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14  cclass Subject implements ISubject{
...
notifyAllObservers(ISubject subject) => {
        IObserver[] observers = theObservers.toArray(EMPTY_ARRAY);
        for(IObservermyObserver: observers){
                try{
```

```
                    myObserver.update(subject);
                }
                catch(Exception e){}
            }
        }
        eventnotifyAllObservers(ISubjectsubject);
...
}
```

Para implementar un evento explícito en ECaesarJ se necesita declarar el evento mediante la palabra reservada event lo cual se muestra en la línea 12. La implementación de un evento en ECaesarJ es de manera similar a un método, lo cual se muestra de las líneas 3 al 11 del código anterior.

El resultado de esta encapsulación es una interfaz que ofrece los servicios que el patrón Observer proporciona. Como se muestra en el siguiente diagrama de componentes.

Dicha encapsulación puede ser utilizada solamente definiendo quiénes fungirán en los roles de sujeto y observador, además de definir cuál es el estado a observar.

Esta implementación es mucho más flexible y con un nivel de abstracción mayor y más semejante al patrón debido a la incorporación de eventos que son lanzados conforme un cambio aparece.

Además esta encapsulación permite su incorporación en forma transparente en la construcción de sistemas por ser un componente de software y con el beneficio adicional de trabajar a nivel arquitectónico con el patrón MVC.

Conclusiones

Los patrones de diseño permiten reutilizar diseños de soluciones a problemas frecuentes. Contar con reutilizaciones en forma de componentes simplifica el desarrollo de software, no solo desde la etapa de diseño, sino además en las etapas de mantenimiento. Por otro lado, desde el punto de vista arquitectónico un componente aporta rapidez, consistencia y mejora la calidad en el desarrollo de software, también los componentes dentro de un sistema mejoran la evolución de los sistemas de manera más simple y estandarizada.

Finalmente es importante destacar la importancia en utilizar eventos ya que facilita la administración de eventos, permiten mayor flexibilidad en obtener modularidad no solo como componentes estructurales, sino como componentes capaces de modularizar el comportamiento. Por supuesto que estas capacidades son altamente deseables en la implementación de otros patrones de diseño y patrones arquitectónicos.

Referencias

- Bass L; Clements P; Kazman R. Software architecture in practice. Addison Wesley (2003).
- Gamma, E; Helm, R; Johnson, R; and Vlissides, J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison Wesley(1995).
- Metsker, S. J. Design pattern Java Workbook. Addison Wesley(2002).
- Nuñez A; Noyé J; Gasiunas V.Declarative definition of contexts with polymorphic events. In International Workshop on Context-Oriented Programming (COP '09) (2009).
- Nuñez A; Gasiunas V; ECaesarJ User's Guide http://ample.holos.pt/gest_cnt_upload/editor/File/public/ECaesarJ-manual.pdf, Julio (2012).
- applications (OOPSLA) (2008).
- Pohl K; Günter B; Linden F. Software Product Line Engineering Foundations, Principles, and Techniques. Springer (1998). · Spersky, C; Dominik, G; and Stephan, M. Component Software, 2a ed. AddisonWesley, (2002).

0.0.1. Noticias

- Noticias - número 110
  - 2do. congreso interpolitécncio de investigación
  - Programa de desarrollo institucional
- Noticias - número 109
  - Congreso Interpolitécnico de Investigación para Alumnos de Posgrado
- Noticias - número 106
  - Brazos biónicos con tecnología politécnica
- Noticias - número 105
  - Festival gamer polígono
  - Competencias docentes para combatir la desinformación digital
  - 2do. foro de semiconductores
- Noticias - número 104
  - GISLATAM
  - HACK Mex capture de flag

0.0.2. Números anteriores

- Números anteriores
- Ciencia y Tecnología
- Cultura
- Egresados
- Acontecimientos
- Eventos
- Identidad.
- Deportes
- Avisos y noticias
- Jajaja

0.0.3. Artículos nuevos

- Procesamiento de Imágenes en Escala de Grises mediante Evaluación Local por Bloques
- Lectura correcta y eficiente de ENCODERS de cuadratura en ESP32: análisis comparativo entre el periférico pcnt e interrupciones externas
- Tecnologías para la protección infantil por contexto de operación: revisión comparativa de enfoques y mecanismos de alerta
- Inteligencia artificial explicable (xai) con lime y shap: interpretación del modelo del TITANIC
- Principales usos de los Algoritmos de Control en las diferentes campos disciplinares e industrias en la era digital: Una Revisión de la literatura (Parte I)
- Cálculo de varianza para el análisis visual de texturas textiles
- Introducción al patronaje digital con Seamly 2D: una herramienta accesible para la industria textil contemporánea

Login

Close

Username

Password

Remember Me

- Forgot your password?
- Forgot your username?

Av. Instituto Politécnico Nacional 2580, Barrio La Laguna Ticomán, Gustavo A. Madero, 07340, México .D.F.

Tels.: (55) 57296000, ext. 56807, 56870

JavaScript debe ser habilitado para poder utilizar Mapas de Google.
Sin embargo, parece que JavaScript está deshabilitado o no soportado por su navegador.
Para ver Google Maps, habilite JavaScript cambiando las opciones de su navegador y vuelva a intentarlo.