# Estándar para documentar el uso de patrones de diseño en un diseño de software

Maximiliano Cristiá
Ingeniería de Software
Facultad de Ciencias Exactas, Ingeniería y Agrimensura
Universidad Nacional de Rosario

2015

**Resumen**

A continuación se presenta una forma de documentar el uso de uno o más patrones de diseño en un diseño de software concreto. Esta forma se considerará un estándar para esta materia. Se asume que los patrones son los descriptos en [1].

## 1. Documentar el uso de un patrón de diseño en un diseño de software concreto

El uso de un patrón de diseño en un diseño concreto se documenta, fundamentalmente, señalando la relación que hay entre los elementos estructurales del patrón (según [1]) y los elementos estructurales del diseño. De esta forma se intenta capturar la *semántica estructural* de cada elemento del diseño concreto al ligarlo a un elemento del patrón de diseño pues en este se explica la semántica de cada término estructural[1]. Estas relaciones se establecen por medio de asignaciones de la forma *elemento_patron* := *elemento_diseno*, donde *elemento* puede ser un módulo, subrutina o variable de estado. La documentación está completa si el conjunto de asignaciones incluye todos los *elementos* estructurales del patrón. Puede incluirse un comentario informal que explique la relación entre los elementos del patrón y los del diseño concreto.

Además es muy conveniente acompañar la documentación con una justificación en términos del análisis de cambio realizado. En otras palabras, se debe justificar el uso del patrón por las

[1]Por *semántica estructural* de un elemento del diseño entendemos la función que este elemento cumple en la estructura del sistema, y no su semántica funcional. Por ejemplo, estructuralmente, la subrutina Ejecutar() de la interfaz del módulo Orden (en el patrón homónimo) es el medio que tiene el cliente para ejecutar cierta porción de código, independientemente de la función que lleve a cabo ese código (poner en negrita, justificar, consultar un sensor, etc.).

posibilidades de cambio que este ofrece y los posibles cambios analizados en el diseño concreto del sistema que se está construyendo.

Concretamente, usaremos la siguiente construcción gramatical para documentar el uso de un patrón en un diseño concreto.

| | |
|---|---|
| **Pattern** | **Nombre que se le da al patrón en este diseño concreto** |
| **based on** | Nombre de uno de los patrones de [1] |
| **because** | Fundamentación de la elección del patrón en términos de<br>▪ Los cambios que este admite y los cambios probables anticipados en el diseño concreto<br>▪ Las necesidades funcionales de alguna parte del sistema<br>▪ Las restricciones de diseño que se deseen imponer |
| **where** | elemento_patrón **is** elemento_diseño<br>elemento_patrón **is** elemento_diseño<br>elemento_patrón **is** elemento_diseño<br>. . . . . **is** . . . . . |
| **comments** | Explicación coloquial de la relación entre los elementos del patrón y los elementos del diseño concreto; otros comentarios adicionales que ayuden a entender cómo se aplica el patrón de diseño. |

En el caso de que la construcción **Pattern** ocupe más de una página dividiremos la caja como se muestra a continuación.

| | |
|---|---|
| **Pattern** | **Patrón muy largo para documentar**<br>*La doble línea inferior indica que la caja continúa en otra página.* |

| |
|---|
| *Las dobles líneas indican que esta porción de la documentación viene de una página anterior y continúa en otra página.* |

| |
|---|
| *La doble línea superior indica que la documentación es continuación de algo que viene de páginas anteriores.* |

Se admite, e incluso en muchos casos es muy necesario e importante, que un mismo elemento_patrón se repita en varias cláusulas **is**. Por ejemplo, en el patrón Abstract Factory el elemento **FabricaConcreta** se deberá usar en tantas cláusulas **is** como fábricas concretas incluya el diseño en donde se aplica el patrón. Más precisamente si seguimos el uso que se hace en Lexi de este patrón tendríamos:

| | |
|---|---|
| **Pattern** | **FabricaAbstractaUtiles** |
| **based on** | Abstract Factory |
| **because** | . . . . . |
| **where** | . . . . . **is** . . . . . |
| | FabricaConcreta **is** FabricaUtilesMotif |
| | FabricaConcreta **is** FabricaUtilesPM |
| | FabricaConcreta **is** FabricaUtilesGnome |
| | . . . . . **is** . . . . . |
| **comments** | . . . . . |

Idealmente todos los elementos mencionados en la descripción del patrón de diseño en [1] deben estar a la izquierda de al menos una cláusula **is**, pues de lo contrario el patrón no estaría debidamente documentado (o peor aun estaría indebidamente utilizado)[2]. Simétricamente todos los elementos mencionados en el lado derecho de una cláusula **is** deberían estar definidos en algún modulo 2MIL o ser un módulo 2MIL.

En el caso de que un elemento del patrón (elemento_patrón) se repita en varios lugares (por ejemplo, InterfazAlgoritmo() en el patrón Strategy se utiliza en cada una de las EstrategiasConcretas) y se lo asocie con el mismo nombre (elemento_diseño) cuando el patrón es aplicado a un diseño concreto, entonces no será necesario calificar elemento_patron. Por ejemplo la aplicación de Strategy en Lexi se documenta (parcialmente) así:

| | |
|---|---|
| **Pattern** | **AlgoritmosComposicion** |
| **based on** | Strategy |
| **because** | . . . . . |
| **where** | . . . . . **is** . . . . . |
| | EstrategiaConcreta **is** ComponedorMatriz |
| | EstrategiaConcreta **is** ComponedorTeX |
| | EstrategiaConcreta **is** ComponedorSimple |
| | . . . . . **is** . . . . . |
| | InterfazAlgortimo() **is** Componer() |
| | . . . . . **is** . . . . . |
| **comments** | . . . . . |

[2]Un editor 2MIL podría configurarse con una lista de patrones, cada uno de los cuales incluiría la lista de elementos del patrón lo que le permitiría verificar la consistencia del uso del patrón

a pesar de que exiten tres usos diferentes de InterfazAlgoritmo() en Lexi (uno en cada una de las EstrategiasConcretas) pues Componer() es el nombre que se usa para InterfazAlgoritmo() en **todas** las EstrategiasConcretas.

Tener en cuenta que en [1] muchas veces ciertos elementos del patrón no se mencionan explícitamente (por ejemplo, no suelen incluir, sino hasta donde se consignan porciones de código, los constructures de los distintos módulos los cuales muchas veces ayudan a comprender ciertos aspectos complicados de los patrones de diseño). También es usual que un patrón presente variantes las cuales impactan en las interfaces de los módulos que comprenden la estructura del patrón (pero esta información suele estar dispersa en la documentación del patrón). Por lo tanto, se deberían incluir todos estos elementos en la construcción **Pattern**. El problema es que al no tener un elemento_patron documentado en [1] no es simple incluir esta información en nuestra documentación (en las sección Ejemplo la subrutina obtenerPadre() es un ejemplo de esta situación donde también se muestra cómo solucionamos el problema).

# 2. Ejemplo

En esta sección documentaremos el uso del patrón Composite en Lexi [1, página 151]. Asumimos que la versión completa de Lexi solo comprende los elementos mencionados en las páginas 32 a 36 del libro ya citado (por ejemplo la interfaz de Glifo comprende solo las operaciones de la Tabla 2.1 de la página 35).

| | |
|---|---|
| **Pattern** | **EstructuraDocumento** |
| **based on** | Composite |
| **because** | |
| | **Cambios previstos:** implementación de los distintos elementos que componen un documento, estructura del documento, aparición de nuevos elementos, desaparición de elementos existentes. |
| | **Funcionalidad:** representar la estructura del documento preservando la disposición del texto y los gráficos según los ingresó el usuario, generar y representar visualmente el documento, mantener la relación entre poisiciones en la pantalla y elementos de la estructura. |
| | **Restricciones de diseño:** el resto de la aplicación debe acceder a los elementos compuestos (como fila o dibujo) de la misma forma que accede a los elementos simples (como caracter o línea) y lo mismo para los elementos visibles como no visibles. |

| | |
|---|---|
| **where** | Componente **is** Glifo<br>Compuesto **is** Fila<br>Hoja **is** Caracter<br>Hoja **is** Rectangulo<br>Hoja **is** Poligono<br>operacion() **is** dibujar()<br>operacion() **is** interseca()<br>operacion() **is** limites()<br>anadir() **is** insertar()<br>eliminar() **is** borrar()<br>obtenerHijo() **is** hijo()<br>obtenerPadre() **is** padre()<br>hijos **is** hijos |
| **comments** | ▪ obtenerPadre() se menciona implícitamente en la página 154 en el punto 1 de la sección Implementación. |

# 3. La construcción Pattern y la estructura de módulos

Por un lado, un patrón de diseño involucra un conjunto de módulos y por el otro, un módulo lógico también es un conjunto de módulos. Por lo tanto, cabe preguntarse qué papel juega la construcción **Pattern** en relación a la estructura de módulos y en particular al sintagma **comprises**.

En nuestra opinión un **Pattern** puede interpretarse como un módulo lógico que comprende a todos los módulos que se mencionan a la derecha de las cláusulas **is**. Por lo tanto en la estructura de módulos podrán incluirse "módulos"documentados con **Pattern**.

# Referencias

[1] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Patrones de diseño.* Addison Wesley, 2003.