Guías de diseño para sistemas adaptativos basados en componentes

Juan F. Inglés-Romero y Cristina Vicente-Chicote
Departamento de Tecnologías de la Información y Comunicaciones (TIC)
Universidad Politécnica de Cartagena
30202 Cartagena (Murcia)
{juanfran.ingles, cristina.vicente}@upct.es

***Resumen**. En la actualidad, uno de los retos en los que más esfuerzos está invirtiendo la comunidad de Ingeniería del Software, consiste en dotar a las aplicaciones de capacidad para modificar dinámicamente (esto es, en tiempo de ejecución) su estructura y comportamiento, de modo que puedan hacer frente, de manera robusta y eficiente, a los cambios que se producen en el contexto. En este artículo, se propone una arquitectura basada en componentes que, a modo de guía, facilite a los diseñadores la incorporación de mecanismos de adaptación en sus aplicaciones. Dado que las necesidades de adaptación pueden variar mucho de una aplicación a otra, se ofrecen diferentes variantes de la arquitectura, así como un criterio con el que seleccionar cual de ellas se ajusta mejor a los requisitos de adaptación de cada sistema.*

1. Introducción

La tecnología actual evoluciona rápida e incesantemente, dando lugar a la aparición de nuevas plataformas, tanto hardware como software. Esto, unido a la creciente demanda de aplicaciones, cada vez más complejas y destinadas a usuarios cada vez más exigentes y con un perfil más variado, hace que el desarrollo de software suponga un reto cada vez mayor [1]. La personalización de las aplicaciones y su adaptación en función de la plataforma o del contexto en el que éstas se ejecutan, se encuentran entre los requisitos más demandados por el mercado actual del software. Para hacer frente a estas nuevas demandas, es necesario dotar a las aplicaciones de mecanismos de adaptación, que les permitan modificar, dinámicamente (en tiempo de ejecución), su estructura, su comportamiento, o ambos. Esto resulta especialmente crítico en determinados dominios (p.ej., robótica, aviónica, sistemas empotrados móviles, etc.), en los que las aplicaciones deben gestionar, de la forma más eficiente posible, determinados recursos (p.ej., energía, memoria, etc.), muy limitados en algunas plataformas.

La incorporación de mecanismos de adaptación en el software no es algo realmente novedoso. Son muchos los trabajos en el ámbito de la Ingeniería del Software que han abordado este problema, si bien, en la mayoría de ellos, se utilizan mecanismos ad-hoc [2] en los que, el código responsable de la adaptación, aparece entretejido (y, por lo tanto, fuertemente acoplado) con el resto del código de la aplicación. Esto no hace sino aumentar la complejidad de las aplicaciones y dificultar sobremanera su reutilización y mantenimiento. En este artículo, se propone una arquitectura basada en componentes que, a modo de guía, sistematice y facilite la incorporación de mecanismos de adaptación, desacoplándolos del resto del código de la aplicación.

2. Arquitectura propuesta

2.1. Descripción general

En la Figura 1 se muestra la estructura general de la arquitectura basada en componentes que se propone en este artículo, a modo de guía, para diseñar sistemas adaptativos. Como se puede observar, el diseño de un *Adaptive System* consta de dos componentes principales: *Reconfigurable Component* y *Adaptation*. El primero de ellos, implementa la funcionalidad principal del sistema, es decir, la lógica de negocio de la aplicación. Como su propio nombre indica, se trata del único componente de la arquitectura cuya estructura interna se puede modificar en tiempo de ejecución. Este componente puede contener cualquier número de componentes internos, tanto de tipo *BaseCO* (componentes fijos en la arquitectura), como de tipo *OptCO* (componentes opcionales, que pueden reconfigurarse en tiempo de ejecución). Es importante tener en cuenta que no se restringe la naturaleza de estos componentes (tanto los *BaseCO* como los *OptCO* pueden ser componentes primitivos o compuestos) ya que, desde el punto de vista de la adaptación, todos ellos son tratados como cajas negras.

El componente *Adaptation* implementa la lógica de adaptación del sistema, de modo que ésta queda totalmente desacoplada de la lógica de negocio de la aplicación. *Adaptation* contiene un componente opcional *Context Ctrl* (sólo incluido en sistemas auto-adaptativos) y un componente obligatorio *Reconf Ctrl*. El primero de ellos (cuando aparece en la arquitectura) aporta al sistema la capacidad de reaccionar ante un entorno cambiante, permitiéndole: (1) adquirir y estructurar el contexto en el que se ejecuta la aplicación, esto es, obtener información de bajo nivel de los sensores y monitores del sistema y, a partir de ella, derivar variables de contexto

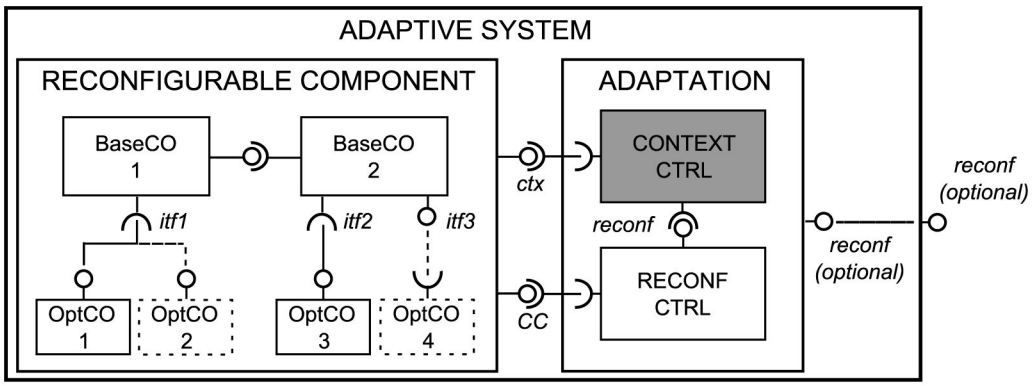

Figura 1. Vista general de la arquitectura basada en componentes propuesta. La interfaz opcional "reconf", ofrecida por los componentes *Adaptive System* y *Adaptation*, se habilita sólo cuando el componente *Context Ctrl* no está presente. En ese caso, el componente *Reconf Ctrl* delega su interfaz "reconf" en dichas interfaces para que la reconfiguración la lleve a cabo un actor (usuario o sistema) externo.

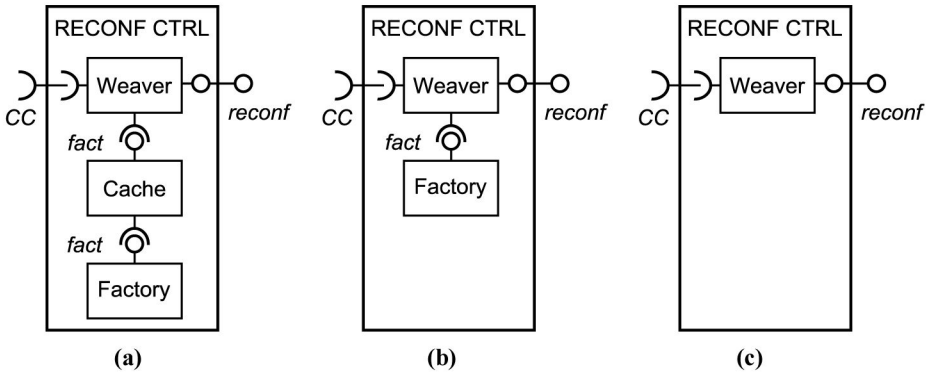

Figura 2. Variantes de diseño disponibles para el componente *Reconf Ctrl*.

significativas, de alto nivel; (2) en base a estas variables, decidir cuándo es necesario reconfigurar el sistema; y (3) cuando sea necesaria una reconfiguración, calcular la mejor arquitectura posible para hacer frente al contexto actual. Es importante señalar que la información relativa al contexto de la aplicación es ofrecida por la interfaz *ctx* de *Reconfigurable Component*. Aunque no se especifica cómo este componente debe implementar dicha interfaz, una solución elegante sería utilizar el patrón de diseño Façade [3]. En este sentido, *Reconfigurable Component* podría delegar, en alguno de sus componentes internos, la responsabilidad de obtener la información relevante del contexto.

Como se ha comentado anteriormente, el componente *Context Ctrl* es opcional y sólo se incluirá cuando se quiera dotar al sistema de la capacidad de auto-adaptación. Cuando este componente se omite en el diseño, se asume que sus funciones las realiza un actor externo (un usuario u otro sistema). En este caso, la arquitectura representada en la Figura 1 varía de la siguiente forma: (1) la interfaz *ctx*, ofrecida por *Reconfigurable Component*, es eliminada o simplemente se deja sin conectar; y (2) los componentes *Adaptive System* y *Adaptation* activan su interfaz opcional *reconf*, permitiendo a *Reconfigurable Component* delegar su interfaz ofrecida en ellos.

Respecto al componente *Reconf Ctrl*, se contemplan tres posibles variantes de diseño que ofrecen al desarrollador, de manera incremental, las siguientes capacidades de reconfiguración dinámica: (a) enlace dinámico de interfaces; (b) instanciación dinámica de componentes; y (c) gestión de la eficiencia. La Figura 2 muestra las tres posibles configuraciones internas del componente *Reconf Ctrl*. Las características principales de estas tres variantes de diseño se detallan a continuación en los siguientes apartados.

2.2. Enlazado dinámico de interfaces

De las tres posibles variantes de diseño disponibles para el componente *Reconf Ctrl*, la mostrada en la Figura 2a representa la más simple. En esta variante, el único mecanismo de adaptación que se ofrece al diseñador es el de enlazar (o desenlazar) interfaces dinámicamente. El componente *Weaver* responde a los comandos de reconfiguración, recibidos a través de su interfaz *reconf*, cambiando la arquitectura interna del componente *Reconfiguration Component*. Para ello, el *Weaver* accede a los servicios proporcionados por la interfaz *CC* para enlazar o desenlazar las interfaces de los componentes de tipo *OptCO*, de acuerdo a una reconfiguración establecida. Es importante señalar que, dado que esta variante no contempla la instanciación dinámica de componentes, todos los componentes de tipo *OptCO* deben ser instanciados en tiempo de compilación y

mantenidos en memoria durante toda la ejecución del sistema, independientemente de si finalmente son utilizados o no en alguna configuración. También cabe señalar que la decisión sobre la granularidad de los comandos de reconfiguración que puede procesar el *Weaver* recae enteramente sobre el diseñador, de modo que éste puede optar por el uso de comandos de reconfiguración primitivos (cada comando realiza una única acción de conexión o desconexión), o tan complejos como desee (cada comando puede realizar cualquier número de conexiones y desconexiones).

2.3. Instanciación dinámica

Esta variante de diseño, ilustrada en la Figura 2b, permite una estrategia de uso de memoria más eficiente que la anterior, gracias a la inclusión del componente *Factory* que, siguiendo lo especificado en el patrón de diseño del mismo nombre [3], permite la creación dinámica de instancias de componentes del tipo deseado.

Esta variante asume que el *Weaver* lleva a cabo las siguientes acciones cuando recibe un nuevo comando de reconfiguración: (1) desenlaza y elimina de la memoria los componentes de tipo *OptCO*, presentes en *Reconfigurable Component*, que ya no son necesarios tras la reconfiguración; y (2) si la nueva configuración requiere nuevos componentes *OptCO*, le solicita al componente *Factory* que cree y añada al *Reconfigurable Component* las instancias necesarias de estos componentes, para después conectarlas como corresponda.

2.4. Gestión de la eficiencia

La gestión eficiente de los recursos de un sistema es una cuestión que debe abordarse cuidadosamente, especialmente, en aquellos sistemas con recursos limitados. Por lo general, resulta imposible optimizar simultáneamente el uso de todos los recursos de un sistema, siendo necesario establecer prioridades y alcanzar compromisos para lograr soluciones equilibradas. Las dos variantes de diseño, descritas en los apartados anteriores, presentan determinadas ventajas e inconvenientes, según en qué términos (parámetros) se mida la eficiencia. Por ejemplo, si consideramos el uso de la CPU, el primer esquema resulta más eficiente que el segundo (requiere menos operaciones) pero, si consideramos la ocupación de la memoria, el segundo resulta más apropiado (sólo se mantienen en memoria los componentes estrictamente necesarios en cada momento). Con el fin de ofrecer a los diseñadores un mecanismo más flexible para gestionar los recursos del sistema, se propone una tercera variante de diseño en la que, como se muestra en la Figura 2c, se ha incluido el componente *Cache*. Este componente proporciona (al *Weaver*) y requiere (al *Factory*) la misma interfaz *fact*, de forma que actúa como una capa transparente entre estos dos componentes.

La *Cache* almacena un conjunto de instancias de componentes, seleccionadas de acuerdo a una política predefinida (p.ej., las más recientemente o frecuentemente utilizadas), poniéndolas a disposición del *Weaver*. Dependiendo del número de instancias disponibles en la *Cache*, el sistema será más o menos eficiente en términos de ocupación de memoria y tiempo de CPU. Cuanto mayor sea la *Cache*, mayor será la ocupación de memoria y más rápidas las reconfiguraciones y, viceversa. De hecho, un uso degenerado de esta variante permitiría simular las otras dos. Por ejemplo, la primera de ellas (Figura 2a) se podría simular con una *Cache* lo suficientemente grande como para almacenar todas las posibles instancias de componentes, necesarias en cualquier reconfiguración. Del mismo modo, la segunda variante (Figura 2b) se podría simular con una *Cache* lo suficientemente pequeña como para que, al no poder almacenar ninguna instancia, tuviera que solicitarlas el *Factory* en cada operación de reconfiguración.

3. Conclusiones

En este artículo se describe una guía de diseño para sistemas adaptativos, que: (1) aboga por una separación clara entre la lógica de negocio y la lógica de adaptación; (2) no se limita a un modelo de componentes concreto ni a una tecnología específica; y (3) es totalmente independiente del dominio de aplicación. Además, el modelo propuesto, permite un cierto grado de flexibilidad, proporcionando a los desarrolladores tres posibles alternativas de diseño. Con todo ello, esta propuesta pretende ofrecer una guía de diseño para sistemas adaptativos que simplifique su desarrollo y mejore su reutilización y mantenimiento.

Agradecimientos

Este trabajo ha sido financiado por los proyectos EXPLORE (MICINN, TIN2009-08572) y MISSION (F. Séneca, 15374/PI/10). Juan F. Inglés agradece a la F. Séneca su beca FPI (Exp. 15561/FPI/10).

Referencias

[1] David, P., Ledoux, T.: Towards a Framework for Self-adaptive Component Based Applications. Proceedings of DAIS'03. LNCS, vol 2893, pp. 1-14 (2003).

[2] Ramirez, J.A., Cheng, B.H.C.: Design patterns for developing dynamically adaptive systems. Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), vol 120, pp.49-58 (2010).

[3] Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented. Addison Wesley Professional (1994).