resumen

Al utilizar una tecnología particular, los desarrolladores suelen partir de cero en la implementación de los componentes de software de un sistema, incluso de aquellos que son comunes a los proyectos en esa tecnología. La construcción de algunos de esos componentes se podría automatizar, dado que su proceso es mecánico y responde a estándares conocidos. Este artículo trata de la construcción de un framework para el desarrollo de aplicaciones por componentes que permite definir lineamientos estandarizados y claros para el desarrollo de aplicaciones, utilizando las mejores prácticas en construcciones de este estilo, minimizando tiempo y costos de desarrollo.

Así mismo, se ilustran los pasos a seguir de acuerdo con los resultados obtenidos en esta primera etapa de la aplicación y se explica cómo su crecimiento puede llegar a ser una solución óptima frente a los problemas planteados.

Eukopoteros: Framework para el desarrollo de aplicaciones por componentes

Juan Camilo Guerrero P. • Diana Carolina Jaimes F. • María Fernanda Soto P.

introducción

El desarrollo de *software* por componentes ha tenido un auge inusitado en los últimos tiempos como un nuevo paradigma en la construcción de aplicaciones, debido a que pretende resolver grandes problemas, como por ejemplo, la reutilización de código, la escalabilidad y el mantenimiento. En particular, el aumento de la potencia de los computadores, la disminución en el costo del *hardware* y las comunicaciones, y el continuo crecimiento de redes de datos de cobertura global han incrementado el uso de los sistemas abiertos y distribuidos.

Este desarrollo ha contribuido a mejorar el proceso de construcción de aplicaciones y, al mismo tiempo, ha aumentado su complejidad. Las aplicaciones por componentes son cada vez más grandes y más difíciles de entender, porque se pueden utilizar en diferentes sistemas y también ensamblar con otros. En la práctica, los existentes se encuentran soportados por alguna tecnología, cada una de las cuales impone enormes restricciones en su ambiente de implementación, lo que obliga a los desarrolladores a conocer en un nivel detallado la parte técnica de la implantación de la tecnología, haciendo costoso en esfuerzo, tiempo y dinero dicho proceso. Todo esto ha hecho evidente la necesidad de crear nuevos modelos, pues la programación tradicional ha hecho imposible tratarlos de una forma natural.

Problemática y estado del arte

En la actualidad, el desarrollo de aplicaciones por componentes representa un conjunto de buenas prácticas que permite reutilizar piezas de código previamente elaboradas, cuya funcionalidad radica en realizar diversas tareas, con sus consecuentes beneficios.

El uso de este paradigma trae algunas ventajas:

• Reutilización del *software*. Alcanzar un mayor nivel de reutilización de *software* del que se tiene hoy.

• Simplificación de las pruebas. Realización de pruebas para cada uno de los componentes, antes de tener el sistema completo.

• Facilitación del mantenimiento del sistema. Los componentes ofrecen un bajo acoplamiento, por lo cual su modificación y adición se resuelven fácilmente.

• Mayor calidad. La construcción y el posterior mejoramiento de un componente lo llevan en el tiempo a tener una alta calidad.

De igual manera, cuando se reutilizan componentes en lugar de desarrollarlos, se tienen algunas ventajas:

• Ciclos de desarrollo más cortos. Si se trabaja por funcionalidad, se requieren algunos días para su desarrollo y no meses o años.

• Mejor retorno de la inversión. Si hay una buena estrategia, se puede obtener el máximo de utilidad en la adquisición de componentes.

• Funcionalidad mejorada. El uso de un componente consiste en entender su estructura y la forma en la que puede implementarse, para satisfacer las necesidades propias.

Sin embargo, la amplia variedad de tecnologías que han surgido gracias al desarrollo por componentes, hace que la estandarización y la organización de las aplicaciones se conviertan en un caos de información. Pese a que la idea de contar con piezas de *software* listas para su uso puede ser tentadora para el progreso de la industria, se tiene el problema de cuál es mejor y, más que eso, quién impone su sello. Así, los grandes de la industria han abordado este inconveniente de diferentes formas, mostrando la solución por medio de variados enfoques y, de cierta manera, convirtiendo una buena solución en uno más de los tantos problemas que se tienen a la hora de construir *software*.

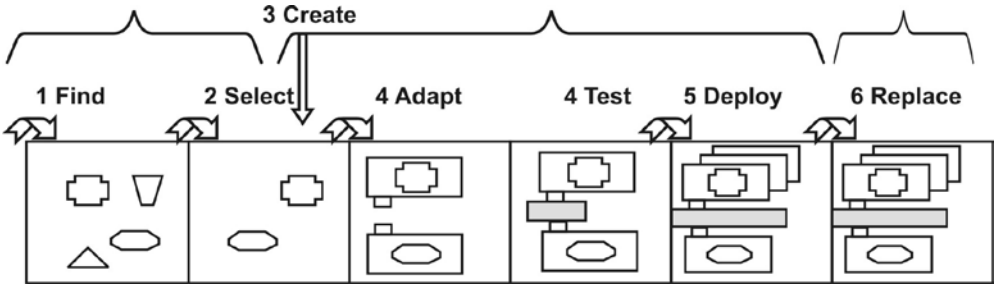

Figura 1. El ciclo de desarrollo orientado por componentes.

Por otro lado, los desarrolladores pasan gran parte del tiempo escribiendo líneas de código que aunque son importantes, muchas veces se han convertido en la base o en estándar de lo que debería contener el código, dada una tecnología en especial. Por estas razones, la generación de código base para una tecnología ahorraría a los desarrolladores tiempo de implementación que se podría emplear en otras fases del desarrollo o inclusive acortar el tiempo de producción de la aplicación.

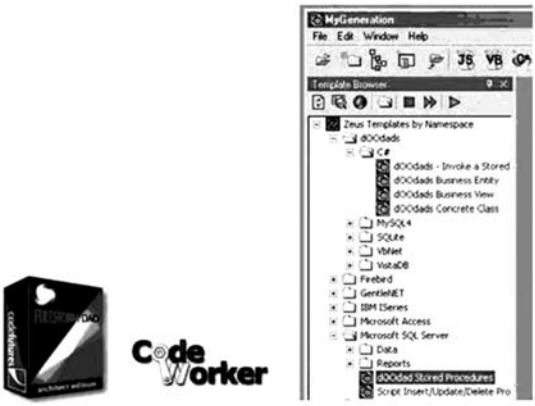

Figura 2. Generadores de código en el mercado.

Aunque en el mercado hay herramientas que minimizan el esfuerzo de implementación de una tecnología (generación de archivos de configuración, de clases huecas, entre otras), lo que cuenta es tener una base lo suficientemente robusta que soporte todos sus componentes. Por ejemplo, *FireStorm/DAO* es un generador de código Java para la capa de persistencia, por medio de la carga del modelo existente en una base de datos, que soporta seis posibles tecnologías. Se basa en el patrón DAO y en toda la arquitectura descrita para este en el núcleo de J2EE. Además de lo anterior crea una interfaz gráfica que da acceso a las acciones básicas para ejecutar sobre el modelo: adicionar, consultar por diferentes parámetros, modificar y eliminar. Así, en el mercado se encuentran cientos de aplicaciones que ofrecen este servicio, ya sea gratis o con algún tipo de licencia. Sin embargo, todas se fundamentan en una generación de código específica, centrándose en una sola funcionalidad, en una o algunas tecnologías; son herramientas independientes, que funcionan en forma diferente y que carecen de un punto de encuentro entre ellas. Una vez que se tiene el código generado no se cuenta con más que un montón de archivos dispersos, sin ninguna relación. Otra desventaja de este tipo de herramientas es que, en general, no interactúan con un entorno de desarrollo, por lo que también se deben ubicar los archivos, de tal modo que sean fáciles de usar en un IDE cualquiera.

En conclusión, actualmente el uso de herramientas de *software* se ha extendido a varias ramas de negocios, industria, administración e investigación. Estas herramientas dejaron de ser exclusivas de los expertos en desarrollo de *software*, y se convirtieron en el factor principal de grandes campos de la industria en el mundo. Los sistemas basados en la funcionalidad, por encima de cualquier otra particularidad, son los más competitivos y solicitados en el mercado; esto ha hecho que

el número de usuarios aumente y cada día sean más las personas del común con poco conocimiento acerca de herramientas de *software* o su funcionamiento. Las anteriores razones plantean nuevos retos en desarrollo de *software* y exigen características que deben estar siempre presentes: funcionalidad, robustez, facilidad de instalación e integración, entre las que deben describir una pieza de *software*. Como consecuencia del amplio uso que se les ha dado a estas herramientas, la integración se constituye en la característica más importante a en el desarrollo. En este punto, el paradigma por componentes entra a cumplir un papel clave, de modo que dichas características se puedan obtener.

Desarrollo de la propuesta

Dadas las problemáticas expuestas, la propuesta consiste en elaborar un marco de referencia para la generación automática de código, orientado a aplicaciones JEE, que permita definir lineamientos estandarizados y claros para el desarrollo, utilizando las mejores prácticas en la construcción de aplicaciones de este estilo, según estándares mundialmente aceptados.

Para sustentar dichas características se manejará una arquitectura multicapa, que asegura la producción de componentes con un bajo acoplamiento; esto le permitirá al usuario generar el código de la capa que necesite, en forma independiente y conjunta. Así mismo, este marco de referencia soportará diferentes tecnologías para cada una de las capas de la aplicación, de modo que se tengan diferentes propuestas a la hora de generar el código de una aplicación. La arquitectura base es, en general, el esqueleto que sostiene toda la estructura, que comprenderá una aplicación multicapa producida por la herramienta; esta une las funcionalidades de cada componente para obtener así una aplicación completa y funcional, después que se genera el código de la forma más conveniente. La arquitectura base propuesta soportará las capas de presentación, lógica y de persistencia; en cada una de ellas se contará inicialmente con una tecnología, que será el piloto del proyecto.

¿Cómo es posible la generación de dicho código?

El núcleo de la aplicación se desarrolló pensando en la futura funcionalidad de la aplicación y en los requerimientos planteados para esta primera fase. De acuerdo con esto, el núcleo de la aplicación se construyó lo suficientemente genérico y robusto para soportar la generación de las líneas de código y la posterior adición de tecnologías, en cada una de las capas. Para realizar el proceso de generación de

código se parte de un modelo de negocio descrito en un archivo XML, diseñado únicamente para este propósito. Adicional a esto, en el archivo está la información correspondiente al modelo de datos que se utilizará en la aplicación. Dicha descripción contiene la suficiente información para generar código en cada una de las capas soportadas, algo propio de cada una de las tecnologías implementadas.

Para la primera funcionalidad, generación de líneas de código, la aplicación trabaja de la mano con un motor de plantillas, *FreeMarker*, para generar los archivos de salida que contienen el código producido. FreeMarker es un generador de texto de cualquier tipo, el cual recibe un mapa de datos que traduce según se le indique en una plantilla. La aplicación traduce la información que le entrega el usuario a una estructura diseñada para soportar modelos de datos complejos, en la que se contemplan objetos, atributos, relaciones y tipos de relaciones; básicamente se habla de entidades abstractas, de atributos simples y de atributos de entidad. La aplicación entrega dicha información a *FreeMarker* de manera tal, que este la entiende y procede a generar los archivos de salida.

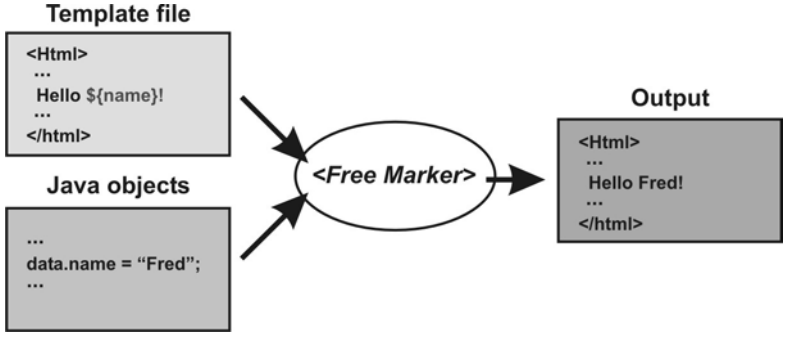

Figura 3. Funcionamiento de *FreeMarker*.

Para la segunda funcionalidad, adición posterior de tecnologías en las diferentes capas, el marco cuenta con ciertas interfaces que deben implementarse, dependiendo de la capa que se quiera trabajar. Por tal razón, para cada tecnología que se desee adicionar es necesario implementar dichas interfaces, además de escribir las plantillas de cada uno de los archivos a generar. Esto resulta bastante sencillo, porque el núcleo es capaz de manejar varias tecnologías sin problema y no existe ningún tipo de acoplamiento entre cada uno de sus componentes. Así mismo, la forma como se lee el modelo de datos y como se generan los archivos es independiente del resto de la lógica del núcleo y puede remplazarse fácilmente. Una vez que se tengan los elementos que se deben implementar para una tecnología, esta se ubica sobre la arquitectura base en la capa que corresponda, manteniendo así

un bajo acoplamiento sobre la aplicación generada y obteniendo un desarrollo de calidad y con buenas prácticas.

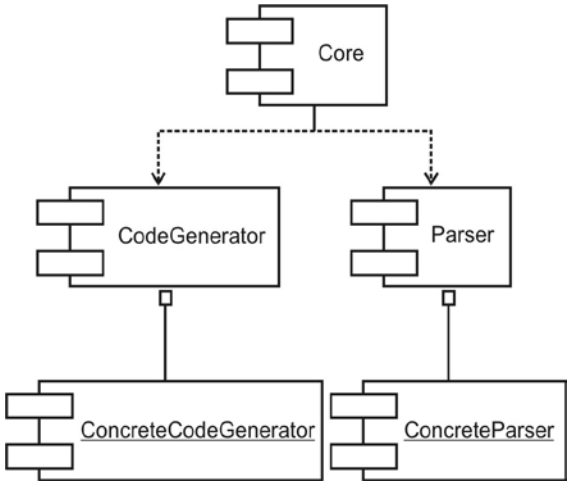

Figura 4. Componentes de la aplicación.

La aplicación se incluye en el entorno de desarrollo Eclipse como un *plug-in,* haciéndola diferente de otras aplicaciones de este estilo. Se acomoda a la arquitectura de Eclipse como un elemento tipo vista, el cual proporciona un entorno agradable y amplio para el usuario, al igual que para posteriores desarrollos de nuevas funcionalidades.

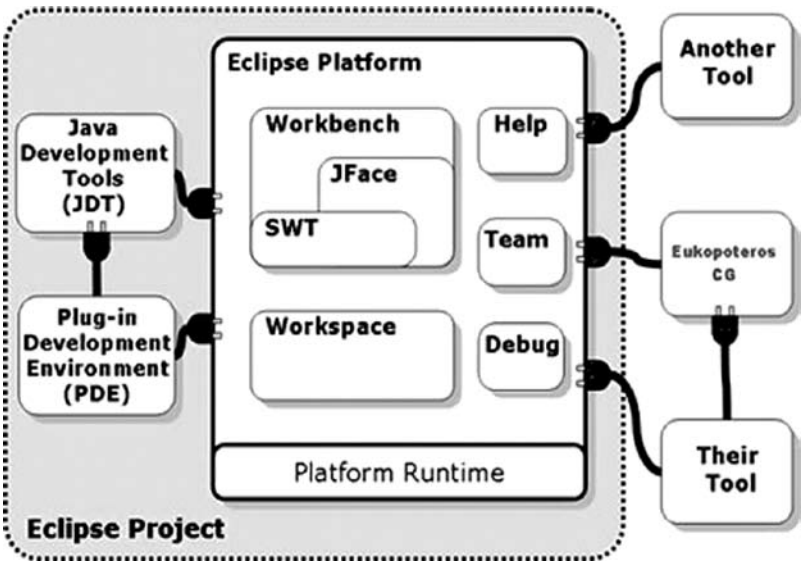

Figura 5. Adición de la aplicación al entorno de desarrollo Eclipse.

Una vez desplegado sobre el entorno, el *plug-in* ofrece en pocos pasos la generación de código para cada una de las capas por separado y la opción de trabajar directamente sobre Eclipse, después de que el proceso termine. Para esta primera fase el *plug-in* soportará tres tecnologías, una por cada capa; para la capa de persistencia se generará el patrón DAO; para la capa de lógica se utilizará EJ3; y, para la vista se generarán archivos para *Ajax Yahoo*.

Figura 6. Vista del *plug-in* sobre el entorno de desarrollo Eclipse.

Mediante su uso se desarrollarán soluciones de *software* con reducción de esfuerzo, tiempo y dinero en la elaboración de aplicaciones distribuidas con un nivel de complejidad alto, y se obtendrán componentes que cumplan con las características antes mencionadas. Así mismo, se espera que sea adoptado para el desarrollo de las aplicaciones internas de la institución.

Trabajo futuro

El producto final, como aplicación sobre el ambiente de desarrollo Eclipse, y concretamente como un *plug-in* del mismo, brinda la facilidad del desarrollo de aplicaciones JEE sobre diferentes tecnologías, para cada capa de una aplicación. Así mismo, su funcionalidad es fácilmente extensible, su núcleo es genérico y modular para asegurar su desarrollo y, por supuesto, la supervivencia de la misma aplicación en el medio. Por estas razones, se puede utilizar en el área empresarial, para agilizar el desarrollo de *software* en la compañía, y también en el área de investigación, en el desarrollo de futuras etapas de la aplicación.

En este orden de ideas, se cree que será posible extender la funcionalidad de la aplicación a nuevas tecnologías que resulten fácilmente acoplables, dada la arquitectura definida. Igualmente, se pueden hacer unas mejoras sobre la solución planteada, además de pensar en adicionar una mejor forma de ingresar el modelo de negocio y datos a la aplicación, de modo que resulte más cómoda y didáctica para el usuario, como por ejemplo una interfaz gráfica. También se puede pensar en la carga del modelo de datos directamente de cualquier motor de base de datos y hacer una correspondencia con un modelo de negocio. Las posibilidades son muy variadas, pero depende de la continuidad y vida de la aplicación.

Conclusiones

Se ha desarrollado Eukopoteros, un marco de referencia para el desarrollo de aplicaciones por componentes de una aplicación multinivel, que apoya su desarrollo por medio de generación automática de código en diferentes tecnologías soportadas en las capas de la aplicación a desarrollar.

El marco se desarrolló en una sola arquitectura por componentes e implementa una sola tecnología para cada capa, con el objeto de validar el modelo realizado.

Se plantea una solución a las problemáticas expuestas obteniendo así una herramienta con las siguientes ventajas:

- Ahorro en tiempo y dinero para las personas que van a utilizar la aplicación.
- Fácil mantenimiento, de modo que se le puedan adicionar módulos a la aplicación y soporte los cambios.
- Facilidad de uso debido a la interfaz amigable con el usuario.
- Reutilización de código.
- Portabilidad, pues se puede reutilizar la aplicación en diferentes plataformas.

El futuro de la aplicación puede llevarla a la evolución por diferentes caminos, razón por la cual se espera que el proyecto continúe para próximas fases de desarrollo.

Referencias

[1] IEEE, www.ieee.org/

[2] Association for Computing Machinery, http://www.acm.org/.

[3] Kirby McInnis, Castek, Component-based Design and Reuse. http://www.cbd-hq.com/articles/1999/990715_cbdandreuse.asp.

[4] Bruce W. Weide. Component-Based Systems.

[5] Crnkovic, I. Component-based *Software* Engineering - New Challenges in *Software* Development. http://www.mrtc.mdh.se/publications/0328.pdf.

[6] Cai, X.; Lyu, M. & Wong, K. Component-Based *Software* Engineering: Technologies. http://www.cse.cuhk.edu.hk/~lyu/paper_pdf/apsec.pdf.

[7] Grundy, J. C.; Mugridge, W. B. & Hosking, J. G. (2000). Constructing component-based *software* engineering environments: issues and experiences. Information & *Software* Technology 42(2), 103-114.

[8] Microsystems, S. Java EE 5. http://java.sun.com/javaee/technologies/javaee5.jsp.

[9] Terreros, J. C. Desarrollo de *software* basado en componentes. http://www.microsoft.com/spanish/msdn/comunidad/mtj.net/voices/MTJ_3985/default.aspx.

[10] Fuentes, Lidia, J. M. T. y. A. V. Desarrollo de *Software* Basado en Componentes. http://www.lcc.uma.es/~av/Docencia/Doctorado/tema1.pdf.

[11] Balarta, V. Generadores de código, http://www.wikilearning.com/curso_gratis/generadores_de_codigo/10114.

[12] CodeFutures. *FireStorm/DAO*. http://www.codefutures.com/.

[13] Eclipse Org. Eclipse Documentation. http://help.eclipse.org/help33/index.jsp.

Juan Camilo Guerrero Pinilla. *Estudiante de último semestre del Programa de Ingeniería de Sistemas de la Escuela Colombiana de Ingeniería.*

Diana Carolina Jaimes Fajardo. *Estudiante de último semestre del Programa de Ingeniería de Sistemas de la Escuela Colombiana de Ingeniería.*

María Fernanda Soto Perdomo. *Estudiante de último semestre del Programa de Ingeniería de Sistemas de la Escuela Colombiana de Ingeniería.*