AUTOMATIZACIÓN DE LA ARQUITECTURA DE COMPONENTES GENÉRICOS USANDO UML

Resumen / *Abstract*

La arquitectura de componentes genéricos permite chequear la consistencia interna de sus elementos arquitectónicos (componentes y conectores) a partir de las relaciones internas en sus respectivas estructuras, que pueden ser de inclusión y transformación. También permite verificar la conexión entre componentes y conectores a partir de las relaciones de transformación entre sus interfaces respectivas. Las ideas que se presentan aquí constituyen una propuesta para la automatización de la descripción de esta arquitectura usando el lenguaje de modelado unificado (UML) a partir de la descripción formal de sus diagramas de clases y de secuencia, así como para el chequeo de la consistencia. En este artículo se muestra la aplicación de esta propuesta mediante una extensión de la herramienta Visual Paradigm, por medio de un módulo de software conectable.

The basis of consistency check of architectural elements (components and connectors) in the generic component-based architecture is the definition of embedding and refinement relations for their internal relationships and interconnections. The ideas which are presented here constitute a guide for the automation of this architecture description using UML beginning with classes and sequence diagrams formal definition as well as consistency checking of components and connectors. This paper presents a plugin extension to Visual Paradigm which instruments this proposal.

Palabras clave / *Key words*

Arquitectura, UML, consistencia

Architecture, UML, consistency

Sonia Pérez Lovelle, *Ingeniera en SistemasAutomatizados de Dirección, Máster en Informática Aplicada, Asistente, Centro de Estudios de Ingeniería de Sistemas (CEIS), Instituto Superior Politécnico José Antonio Echeverría Cujae, Ciudad de La Habana, Cuba*
e-mail:sperezl@ceis.cujae.edu.cu

Fernando Orejas Valdés, *Licenciado en Matemática, Doctor en Matemática, Catedrático, Departamento de Lenguajes y Sistemas Informáticos, Universidad Politécnica de Cataluña, España*
e-mail:orejas@lsi.upc.edu

Noel Fuentes Ramírez, *Ingeniero Informático, Instructor, Departamento de Matemática General, Facultad de Ingeniería Industrial, Instituto Superior Politécnico José Antonio Echeverría Cujae, Ciudad de La Habana, Cuba*
e-mail:nfuentes@ind.cujae.edu.cu

Exiquio C. Leyva Pérez, *Ingeniero Químico, Doctor en Ciencias Técnicas, Profesor Titular, CEIS, Instituto Superior Politécnico José Antonio Echeverría Cujae, Ciudad de La Habana, Cuba*
e-mail:exiquio@ceis.cujae.edu.cu

Recibido:octubre del 2005
Aprobado:diciembre del 2005

INTRODUCCIÓN

UML es un lenguaje de modelado universal, por lo que cada vez más, es empleado para la descripción de arquitecturas. En este caso, se usa para la descripción de la Arquitectura de Componentes Genéricos, y en particular, para especificar la estructura interna de los elementos arquitectónicos mediante los diagramas de clases y de secuencia.

La formalización de estos diagramas permite establecer qué es para cada uno de estos tipos de modelos, relaciones de inclusión y refinamiento y a partir de esto probar la consistencia interna de cada uno de los elementos, así como la verificación de las interconexiones entre dichos elementos.

Para ello se ha extendido la herramienta de modelado Visual Paradigm mediante un módulo de software conectable (*plugin*).

LENGUAJE UNIFICADO DE MODELADO

El lenguaje unificado de modelado (UML) es un lenguaje para la especificación, la visualización, la construcción y la documentación de los artefactos de los sistemas de software y también para otros tipos de sistemas. Representa una colección de las mejores prácticas de ingeniería que han

sido probadas con éxito en el modelado de sistemas grandes y complejos.[1] Se convirtió en estándar del object management group (OMG) en 1997, después de tres años de trabajo, como consecuencia de la llamada guerra de las metodologías.[2]

La principal ventaja de UML es que contituye un lenguaje de propósito general, aunque esto en ocasiones se puede convertir en una desventaja, porque no se pueden representar cabalmente las situaciones o características propias de dominios específicos.[3] Es un lenguaje gráfico, que puede ser usado en todas las fases de desarrollo de software y que permite representar los sistemas con varios modelos parciales, lo que facilita su entendimiento y la comunicación.

Por todo lo anterior, un mayor número de proyectos utilizan UML para representar la arquitectura de sus sistemas,[4,5] y por eso se ha decidido usarlo para la especificación de la arquitectura de componentes genéricos.

Lo antes expuesto no significa que la notación de UML cubra todas las necesidades para representar elementos arquitectónicos,[5] pero su popularidad y facilidad para la comunicación, así como la posibilidad de usar herramientas para su modelado, lo hacen un buen candidato para usarlo, aunque a veces se tenga que recurrir a las extensiones que propone el propio UML para ello. En este caso se han propuesto extensiones para UML mediante un perfil y también mediante la creación de nuevas metaclases y metaasociaciones.

ARQUITECTURA DE COMPONENTES GENÉRICOS

La arquitectura de componentes genéricos (ACG)[6-8] es un marco conceptual que tiene como elementos arquitectónicos los componentes y conectores, inspirados en las nociones de módulos con especificaciones algebraicas y los conectores de Allen y Garlan,[9] respectivamente. Esta arquitectura es válida para las distintas etapas del ciclo de vida del desarrollo de componentes y no solo de la implementación, como es usual, por ejemplo, en los desarrollos basados en componentes.[10]

El término genérico significa en primera instancia, que no se dice a priori la técnica o lenguaje a emplear para la especificación de sus elementos, sino que basta con escoger un dominio apropiado que posibilite describir cada uno de los elementos y a partir de esto, definir el tipo de conexiones a establecer internamente en componentes y conectores y, por último, definir una relación de transformación o refinamiento adecuada que permita que se cumplan las propiedades de extensión y extensión paralela.

Los elementos de esta arquitectura tienen una estructura interna formada por un cuerpo, donde se encuentra su implementación y un conjunto de interfaces, que en el caso del componente se llaman puertos, y puede ser un conjunto vacío, en el caso de los conectores se llaman roles y este conjunto debe tener al menos dos elementos y así permitir la conexión.

La técnica de especificación o modelado debe permitir representar (especificar) tanto los cuerpos (de componentes y conectores) como los puertos y los roles, así como tener nociones adecuadas de relaciones de inclusión y transformación. Esto está dado porque en esta arquitectura se parte del hecho de que las relaciones que se establecen internamente en los conectores, es decir, entre su cuerpo y los roles deben ser inclusiones, mientras que las relaciones que se establecen internamente en los componentes, así como entre los puertos y los roles, a los efectos de la composición de componentes, deben ser relaciones de transformación.

La inclusión significa que se pueda definir cuándo una especificación es un subconjunto de otra, aunque no en el sentido estricto de la teoría de conjuntos, sino teniendo en cuenta que pueden existir cambios de nombres o alias. Mientras que la transformación define relaciones que permiten pasar de un nivel de abstracción más alto de la especificación, sin detalles de implementación a uno más detallado o concreto, con niveles de implementación, a partir de un proceso de refinamiento. Esto significa de manera general, que todo lo que está permitido a nivel de interfaz tiene que poder hacerse en el cuerpo.

REPRESENTACIÓN DE LA ACG MEDIANTE UML

Representar la arquitectura de componentes genéricos significa que una vez seleccionada la técnica o lenguaje, se deben poder especificar con ella sus elementos arquitectónicos, o sea, los componentes y conectores y en particular, cada cuerpo, cada puerto y cada rol.

Estos elementos deben ser especificados al menos desde los puntos de vista de estructura y comportamiento.

UML tiene trece diagramas diferentes para el modelado, seis para representar estructura y siete para el comportamiento. En este caso se han seleccionado los diagramas de clases y de secuencia respectivamente.

De todos los diagramas estructurales, el de clases es el único que aparece a través de todo el ciclo de vida y tiene un mayor nivel de detalles en el sentido de que se representan los conceptos que se manejan en el dominio o negocio que se esté tratando, así como sus interrelaciones.

En el caso del comportamiento se escogió el diagrama de secuencia porque se puede usar desde las etapas tempranas de desarrollo, teniéndose en cuenta que representa escenarios y que estos pueden verse en diferentes niveles de detalle. Además, es el más común de los diagramas de interacción,[11] porque es anterior a la aparición de UML y tiene su origen en el **Message Sequence Chart** (MSC) que fue estandarizado por la ITU.[12]

Una vez que se han definido los diagramas, estos deben ser formalizados a los efectos de definir las relaciones de inclusión y transformación entre diagramas de igual tipo y probar que se cumplan las propiedades de extensión y extensión paralela,[7] con el fin de garantizar que se pueden formar nuevos componentes a partir de otros relacionados a través de conectores.

HERRAMIENTAS PARA EL MODELADO DE UML

Una de las causas de que UML tenga tan amplia aceptación es el hecho de no tener competidores, al ser el único lenguaje de modelado que es estándar, y por lo tanto, su uso puede ser una garantía de establecer comunicación con una gran cantidad de personas. Sin embargo, sin lugar a dudas, su proliferación también se debe, en gran medida, a la existencia de una gran diversidad de herramientas que permiten la realización de sus diagramas de forma automatizada, ya que realizarlos manualmente implica mucho tiempo y puede constituir una fuente importante de errores.

Si los desarrolladores tuvieran que especificar grandes proyectos con UML sin la ayuda de una herramienta y quisieran diagramas con calidad, desde el punto de vista sintáctico y de consistencia entre las diferentes vistas que se modelan, tendrían que dominar su metamodelo, el que se ha ido haciendo cada vez más complejo.[13,14] La complejidad no es solo por la extensión, que en la versión actual son más de 700 páginas, sino también porque la jerarquía de clases es de "grano fino" y esto puede implicar tener que atravesar una gran cantidad de clases ancestras para tener acceso a todas las características de una de ellas.[14]

Las herramientas para el modelado de diagramas de UML también han evolucionado, no solo en el sentido de ir renovando las versiones a las que dan sustento, sino constituyendo un verdadero apoyo en las tareas de desarrollo de software por medio de la mejora cada vez más de la interfaz con el usuario e introduciendo facilidades como chequeos sintácticos de los diagramas, chequeos de consistencia, facilidades de importación y exportación, la generación de código en diferentes lenguajes, uso de ingeniería inversa para obtener diagramas a partir de código, hasta la posibilidad de definir perfiles para el diseño en un dominio específico.

MÓDULO CONECTABLE PARA VISUAL PARADIGM

Para la aplicación práctica de los conceptos de la arquitectura de componentes genéricos (ACG) usando UML, se necesita una aplicación que permita representar, tanto la arquitectura, como las instancias de sus elementos, así como que posibilite la especificación, mediante diagramas, de la estructura interna de los componentes.

A partir de los elementos antes indicados, se requiere tomar la decisión de si se hace una herramienta desde cero o si se utiliza una de las múltiples herramientas que se encuentran disponibles para el modelado de UML.

La primera opción permite desarrollar una aplicación que responda 100 % a las necesidades propias del problema o dominio, pero implica no solo que se puedan dibujar los diagramas, sino también que requiere de todos los chequeos sintácticos asociados y chequeos de consistencia entre los diferentes diagramas, para luego hacer los chequeos de consistencia de los elementos y las conexiones, que es lo que no tienen en cuenta estas herramientas.

Como no se concibe una herramienta de modelado de diagramas de UML que no realice estos chequeos y el modelar los diagramas no es el centro de la ACG, se decidió partir de una herramienta existente y añadirle las cuestiones necesarias.

A pesar de la gran diversidad de herramientas para el modelado de UML que existen y que de manera general son fáciles de usar, había que acotar la búsqueda a las que cumplieran al menos los siguientes requisitos:

- Implementaran la versión 2.0 de UML.
- Fueran posible de extender (ya sea por ser de código abierto o a través de módulos softwares conectables -*plugin*-).
- Fueran gratuita.

Otros requisitos, aunque en un segundo nivel de importancia, serían las facilidades para importar y exportar, sobre todo usando el formato de *Rational Rose,* por ser uno de los más extendidos; la calidad de los chequeos de los diferentes diagramas y de la consistencia entre ellos; el formato de almacenamiento de la información, etcétera.

Lo ideal sería encontrar una herramienta que, además de implementar la versión 2.0 de UML, fuera software libre, no desde el punto de vista de que solo sea gratuito, sino en el concepto que se maneja actualmente de software libre (*libre software*, en inglés). Es decir, software gratuito y de código abierto,[15-17] para poder incluir con mayor libertad el código que responda a las necesidades del proyecto, pues no todas las herramientas comerciales tienen en cuenta todas las posibilidades de UML,[18] por ejemplo, la definición de perfiles.

Después de analizar 17 herramientas diferentes, fue seleccionada la edición Community de Visual Paradigm (VP) (http://www.visual-paradigm.com) que es gratuita, soporta la versión 2.0 de UML y permite su extensión mediante la conexión de módulos conectables (*plugin*) o usando plantillas (*templates*).

Las plantillas se usan para el diseño de reportes y los módulos conectables, brindan la posibilidad de incluir nuevas funcionalidades, las que se pueden añadir a la herramienta mediante su invocación desde elementos de menú o por botones y pueden tener acceso a los elementos y diagramas modelados a través de la biblioteca OpenAPI, que VP pone a disposición de los usuarios para llevar a cabo la comunicación con este.

ACG-Plugin es la herramienta desarrollada para automatizar todo lo correspondiente a la arquitectura de componentes genéricos para UML y constituye una extensión de Visual Paradigm.

Se desarrolló siguiendo la metodología RUP[1] y usando el lenguaje de programación Java, no solo por sus funcionalidades, sino también porque es una exigencia de la conexión con la herramienta Visual Paradigm.

Este desarrollo, se concibió como un módulo conectable que actúa como un elemento externo y que se sirve de VP para recuperar la información correspondiente a los diferentes

diagramas (de clases y de secuencia) desarrollados y al diagrama de la arquitectura.

Con esta información recuperada, se establecen las relaciones de los diagramas de clases y de secuencia con los correspondientes componentes y conectores, y dentro de estos, con cada una de sus partes.

El módulo ACG-Plugin fue concebido en los cuatro subsistemas que se describen a continuación:

1. Subsistema diagramas de especificación: Este subsistema es el que permite la comunicación con Visual Paradigm para obtener los diagramas de clases y de secuencia que se hayan desarrollado para modelar los componentes y conectores, así como el diagrama de la arquitectura, es decir, cuáles son los componentes y conectores, sus cuerpos, puertos y roles y cómo se interconectan. Toda esta información se almacena en la estructura de datos para su posterior análisis.

2. Subsistema de arquitectura: En este subsistema es donde se definen, para cada uno de los elementos de la arquitectura que llevan especificaciones en UML, qué diagramas de clases y de secuencia están asociados, así como las funciones de cambio de nombre y de transformación. Toda esta información complementa la estructura de datos.

3. Subsistema Análisis de la ACG: Este es el subsistema fundamental desde el punto de vista del procesamiento. A partir de la información almacenada en la estructura de datos y que se corresponde con la estructura del sistema, es decir, todos los componentes y conectores con sus respectivas estructuras internas, la especificación mediante los diagramas de UML y su forma de conexión y las respectivas funciones, chequea que dicha información esté completa y que la arquitectura sea consistente, registrando los resultados de estos análisis. Está dividido a su vez en dos subsistemas:

- Un subsistema que se encarga de verificar que todos los elementos a chequear estén completos para que puedan ser analizados. En primer lugar, se chequea que cada componente y conector tenga definido su cuerpo y sus interfaces (puertos y roles, respectivamente), luego que para cada uno de ellos existan especificaciones completas asociadas (un diagrama de clases y al menos un diagrama de secuencia) y que existan las funciones de transformación.
- Un subsistema que se encarga, si toda la información está completa, de chequear la consistencia en cada uno de los elementos (inclusiones para los conectores y transformaciones para los componentes) y la consistencia a nivel de la arquitectura, relaciones de transformación entre los puertos y los roles.

4. Subsistema Reportes: Como su nombre indica, es el que permite visualizar los resultados del análisis realizado en el subsistema de Análisis de la ACG.

De la explicación anterior, se deduce que la interrelación entre los diferentes subsistemas es a través de la estructura de datos, que fue concebida como una multilista en la que se almacena toda la información de la arquitectura en primera instancia, es decir, todas las ocurrencias de componentes y conectores, sus interrelaciones y para cada uno, su estructura interna.

A su vez, en cada elemento de la estructura interna, o sea, cuerpos, puertos y roles, se indican los diagramas que los especifican.

Tanto la estructura de datos como los resultados de los análisis, se almacenan en ficheros de formato XML.

En estos momentos se han realizado pruebas con un desarrollo hipotético que describe el funcionamiento de un elevador y con la biblioteca de componentes desarrollada por los integrantes del proyecto multidisciplinario en donde participan el CEIS y el Centro Nacional para la Producción de Animales de Laboratorio (CENPALAB), relacionado con la producción de software para Sistemas de Información Geográfica.

CONCLUSIONES

A partir de esta experiencia se puede concluir que es factible la representación de la ACG usando UML y que esta mediante extensiones es capaz de adecuarse a cualquier dominio específico. Por otro lado, se comprobó mediante su automatización las reglas de transformación propuestas para el chequeo de la consistencia interna en la ACG.

La extensión de UML a los efectos de esta propuesta solo es posible mediante un perfil, definiendo en Visual Paradigm los estereotipos correspondientes, porque no se puede acceder al metamodelo para la inclusión de nuevas metaclases y metaasociaciones y al añadirle un módulo conectable a Visual Paradigm, se obtiene una nueva herramienta con nuevos valores añadidos.

REFERENCIAS

1. **JACOBSON, IVAR; GRADY BOOCH Y JAMES RUMBAUGH:** *El proceso unificado de desarrollo de software*, Ed. Félix Varela, Ciudad de La Habana, Cuba, 2004.
2. **BOOCH, GRADY:** "UML in Action", *Communications of the ACM*, Vol. 42, No. 10, October 1999.
3. **ENGELS, GREGOR; REIKO HECKEL AND STEFAN SAUER:** *UML-A Universal Language*? ICATPN 2000, LNCS 1825, Berlin, Heidelberg, Springer-Verlag, 2000.
4. **WERMELINGER, MICHEL; ANTÓNIA LOPES AND JOSÉ LUIZFIADEIRO:** *A Graph Based Architectural (Re)configuration Language*, ESEC/FSE 2001, Vienna, Austria, ACM 2001.
5. **MEDVIDOVIC, NENAD; DAVID S. ROSENBLUM; DAVID F. REDMILES, AND E. JASON ROBBINS:** *Modeling Software Architectures in the Unified Modeling Language*, ACM Transactions on Software Engineering and Methodology, Vol. 11, No. 1, January, 2002.
6. **OREJAS, FERNANDO AND HARTMUT EHRIG:** "Components for Algebra Transformation Systems, *Electronic Notes in Theoretical Computer Science 82,* No. 7 Elsevier Science B V, 2003.

7. **EHRIG, H.** *et al*: *A Generic Framework for Connector Architectures Based with Components and Transformations,* Formal Foundations of Embedded Software and Component-Based Software Architectures (FESCA´04), 2004.
8. **EHRIG, H.** *et al.*: *Object-Oriented Connector-Component Architectures*. Formal Foundations of Embedded Software and Component-Based Software Architectures (FESCA´05), 2005.
9. **ALLEN, ROBERT AND DAVID GARLAN:** *A Formal Basis for Architectural Connection*, ACM TOSEM´97.
10. **MANN, STEFAN**, *et al.*: *Towards a Component Concept for Continuous Software Engineering*, Fraunhofer Institute Software Und-Systemtechnik, Bericht 55/00, Oktober, 2000.
11. *Object Management Group. UML 2.0 Superstructure*. OMG document ptc/05-07-14. Disponible en http://www.omg.org/cgi-bin/doc?ptc/05-07-14
12. *ITU-TS. Recommendation Z.120: Message Sequence Chart (MSC)*, ITU-TS, Geneva, 1996.
13. **KANDÉ, MOHAMED MANCONA:** "A Concern-Oriented Approach to Software Architecture", Tesis para optar por el grado de Doctor en Ciencias. Faculté Informatique et Communications. École Polytechnique Fédérale de Lausanne. 2003.
14. **KOBRYN, CRIS:** *UML 3.0 and the Future of Modeling,* Soft System Model (2004) 3: 4-8 / Digital Object Identifier (DOI) 10.1007/s10270-004-0051-4. Published Online 24 February 2004. Springer-Verlag 2004.
15. **AMOR-IGLESIAS, JUAN JOSÉ:** *Measuring Libre Software Using Debian 3.1 (Sarge) as A Case Study: Preliminary Results*, UPGRADE Vol. VI, No. 3, June, 2005.
16. **HERRÁIZ, ISRAEL; GREGORIO ROBLES AND JESÚS M. GONZÁLEZ-BARAHONA:** *Towards Predictor Models for large Libre Software Projects*. PROMISE´05. St. Louis, Missouri, USA. May 15, 2005.
17. **ROBLES, GREGORIO; JESÚS M. GONZÁLEZ-BARAHONA AND MARTÍN MICHLMAYR:** *Evolution of Volunteer Participation in Libre Software Projects: Evidence from Debian*, Proceedings of the First International Conference on Open Source Systems, 2005.
18. **PETRIU, DORINA C. AND HUI SHEN:** *Applying the UML Performance Profile: Graph Grammar-Based Derivation of LQN Models from UML Specifications*, In T. Field, P. G. Harrison, J. Bradley, and U. Harder, editors, Computer Performance Evaluation / TOOLS, volume 2324 of Lecture Notes in Computer Science. Springer, 2002.