INFORMÁTICA

PERFIL PARA REPRESENTAR UNA ARQUITECTURA DE COMPONENTES EN UML

Resumen / ***Abstract***
El lenguaje unificado de modelado (Unified Modeling Language, UML) es un lenguaje de modelado estándar para problemas generales, sin embargo, es necesario extenderlo para dominios específicos como puede ser el caso de determinadas arquitecturas. Aquí se presenta una extensión mediante un perfil para una arquitectura de componentes y conectores genéricos.

Unified Modeling Language, (UML) is a standard modeling language for general purposes, however it may be extended in order to represent specific domain such component-based architectures. This paper presents an extension using a profile for generic components and connectors-based architecture.

Palabras clave ***/ Key words***
UML, perfil UML, componentes, conectores

UML, UML profile, components, connectors

INTRODUCCIÓN

El UML es un lenguaje de modelado estándar que debe ser extendido para la representación de dominios específicos, mediante cualquiera de los dos mecanismos permitidos. En este caso, el objetivo es presentar el diseño de un perfil para representar la arquitectura de componentes genéricos. Primeramente se presentará la arquitectura de componentes genéricos, para luego explicar UML y porqué es insuficiente su metamodelo para representar esta arquitectura y, por último, se mostrará la propuesta de perfil que da solución a esta problemática.

ARQUITECTURA DE COMPONENTES GENÉRICOS

La arquitectura de componentes genéricos (ACG)[1-3] 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,[4] 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.[5]

El término genérico es usado en tres sentidos:

- No se dice a priori el lenguaje o técnica a emplear para especificar los elementos de esta arquitectura. La idea es que los mismos conceptos puedan ser usados por diferentes instancias.
- Se deja abierto también, el tipo de conexiones a establecer internamente en los componentes y conectores.
- La noción genérica de transformación de las especificaciones.

Sonia Pérez Lovelle, *Ingeniera en Sistemas Automatizados 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

Recibido:octubre del 2005
Aprobado:diciembre del 2005

Para la definición de componentes y conectores, se parte de un componente general que tiene un cuerpo, un conjunto de puertos y un conjunto de roles, como se puede observar en la figura 1.

A partir de este componente general se obtienen el componente y el conector con sus características propias.

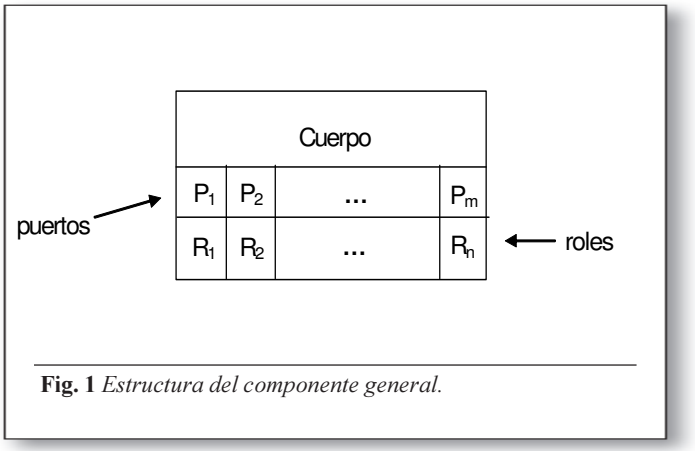

Fig. 1 *Estructura del componente general.*

Un componente está constituido por un cuerpo, cero o más puertos y el conjunto vacío de roles.

Un conector tiene un cuerpo, un conjunto de (al menos dos) de roles que tienen que ser consistentes,[2] y un conjunto vacío de puertos.

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.

LENGUAJE UNIFICADO DE MODELADO

El lenguaje unificado de modelado 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.[6] 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.[7]

La principal ventaja de UML es ser 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.[8] 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.

LIMITACIONES DE UML PARA REPRESENTAR LA ACG

A pesar de la existencia de los lenguajes de descripción de arquitecturas (LDA), cada vez más proyectos utilizan UML para representar la arquitectura de sus sistemas.[9,10]

Lo antes expresado no significa que la notación de UML cubra todas las necesidades para representar elementos arquitectónicos,[10] pero como se ha explicado en el epígrafe anterior, 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 para ello.

Uno de sus principales problemas es que UML es orientado a objetos y no a componentes.[9] Una de las críticas que se han hecho tradicionalmente a UML, es justamente la ausencia o tratamiento inadecuado de los elementos arquitectónicos,[10] en especial debido a su estrecha relación con el proceso unificado de desarrollo (RUP), en el que la arquitectura es un concepto central.[9]

Esta situación ha originado una evolución en el tratamiento de estos conceptos a través de sus diferentes versiones. Pero a pesar de las nuevas definiciones en el metamodelo de los elementos arquitectónicos, estos no se adecuan a las necesidades de la arquitectura de componentes genéricos, como se explica a continuación.

La metaclase *component* de UML[11] representa una parte modular de un sistema que encapsula su contenido y que es remplazable.

Las diferencias con el componente de la ACG, se ven en primer lugar en la estructura, pues a diferencia del componente de la ACG que obligatoriamente tiene que tener un cuerpo, en este caso es opcional que tenga estructura interna y un conjunto de puertos como puntos de interacción. Además, su comportamiento se define en términos de interfaces requeridas y suministradas y su forma de conectarse, puede ser a través de las interfaces o usando conectores.

El conector (metaclase *Connector* de UML[11]) ha sido concebido como un enlace primitivo que no tiene estructura interna y ni siquiera tiene nombre.[9] Parece que todavía no representa el concepto de conector que necesita la comunidad de arquitectura de software.[12] Además de lo anterior, tampoco es adecuado para la representación del conector de la ACG, pues

tiene un único atributo para indicar si es un conector de delegación o de ensamblaje.

De acuerdo con lo explicado para las metaclases *Component* y *Connector*, no existe un cuerpo y tampoco el concepto de rol.

En UML también aparece el concepto de puerto (metaclase *Port*), pero se trata de una propiedad (por lo tanto, tampoco tiene estructura interna) de los clasificadores (metaclase *Classifier*), que especifica un punto de interacción entre estos y sus partes internas. Funciona como una puerta en el encapsulamiento de la clase, a través de la cual entran o salen mensajes a la clase, en dependencia del tipo de interfaz (suministrada o requerida).[13]

Sin embargo, esto no es un gran problema porque UML permite su extensión a través de dos mecanismos: la creación de perfiles y la extensión del metamodelo mediante la creación de nuevas metaclases y metaasociaciones. La primera alternativa es más fácil de implementar, y se puede integrar más fácilmente a las herramientas de modelado, pero es menos potente. Mientras, la extensión del metamodelo permite un modelado más cercano a la realidad que se desea representar, pero implica un mayor conocimiento del metamodelo y su integración con herramientas en estos momentos es casi imposible.

DESCRIPCIÓN DEL PERFIL

Para tener un entorno real de trabajo con la ACG es necesario no solamente que se puedan comprobar sus elementos de acuerdo con lo explicado, sino que se necesita también poder representar el marco conceptual como un todo para lograr definir los elementos base, así como sus relaciones, o lo que es lo mismo, es necesario describir la arquitectura como tal.

Sin embargo, los elementos arquitectónicos que aparecen en UML no permiten una correcta modelación de arquitecturas particulares y tampoco son suficientes para representar todos los elementos estructurales y la semántica de la ACG.

Para la creación de un perfil de UML con el objetivo de representar un dominio específico se pueden usar estereotipos, restricciones y valores etiquetados.

Los estereotipos se corresponden con una clasificación de un elemento de modelado de UML que se asocia a una metaclase y que tienen sus propias características expresadas a través de valores etiquetados (parámetros o información asociada) y las restricciones, los que permiten diferenciarlos de los definidos en UML y asociarle una semántica. Además, se pueden asociar iconos gráficos a estos estereotipos.[14]

A partir de la definición hecha de la arquitectura de componentes genéricos, se deben representar en este perfil los elementos Componente General, Componente, Conector, Cuerpo, Puerto y Rol, así como las interrelaciones entre estos. En la figura 2 se representa el diagrama de clases de UML donde aparecen estos elementos.

Se debe destacar que tanto en la figura 2 como en las definiciones asociadas a las extensiones, estos elementos aparecen en idioma inglés a partir de un criterio estético, teniendo en cuenta que aparecerán junto a las definiciones propias de UML que están en este idioma y porque su uso dentro de una herramienta comercial de modelado obligaría a una traducción.

Para desarrollar el perfil se siguieron los pasos recomendados por Kandé y Lidia Fuentes, con Antonio Vallecillo.[15,16] Una primera decisión es definir la o las metaclases de UML que serán extendidas mediante el uso de estereotipos para representar los elementos de la ACG.

Las primeras metaclases candidatas son las que representan un componente y un conector (metaclases *Component* y *Connector*, respectivamente). Sin embargo, a pesar de tener el nombre que se necesita se desechan porque no cubren todas las necesidades, como se explicó anteriormente.

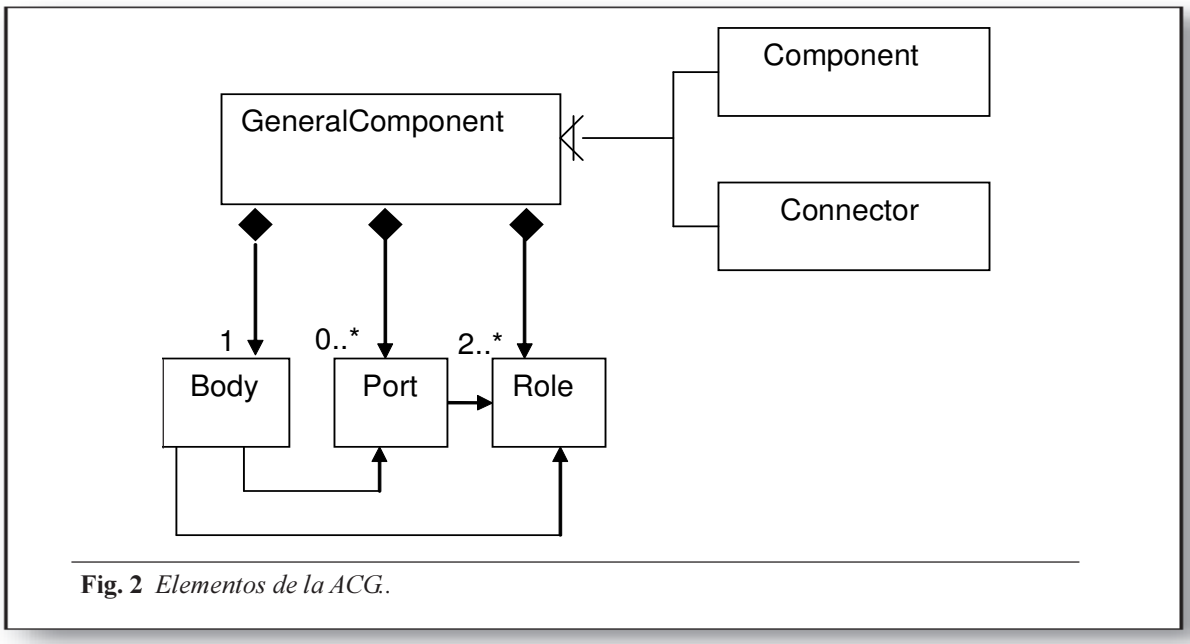

Fig. 2 *Elementos de la ACG.*

Los componentes y conectores de la ACG pueden ser vistos como clases desde el punto de vista de la programación orientada a objetos, sin embargo, tampoco la metaclase clase (*Class*) es adecuada porque, aunque su definición en el paquete *StructuredClasses* del metamodelo de UML, puede tener elementos internos (necesarios para la especificación de diagramas según ACG), también tiene puertos y estos son incompatibles con los puertos de la ACG, como se explicó.

Como consecuencia de que no se usen las metaclases mencionadas anteriormente para representar los componentes y conectores de la ACG, tampoco puede ser usada la metaclase puerto (*Port*) pues estaría fuera de contexto y aunque quizás pasara lo mismo con el caso del cuerpo y el rol, para estos, ni siquiera aparecen metaclases con estos nombres o similares.

El resultado es que se usará, como metaclase para extender mediante estereotipos, la clase Paquete (*Package*) que se usa para agrupar elementos. En particular, esta se usa para particionar grandes modelos y puede contener elementos estructurales, de comportamiento y otras clases agrupables y además, no existen guías formales para su uso.[17] La semántica del paquete está dada por la agrupación de elementos, por lo que es válido asignarle una en función de lo que contiene.

Se representarán mediante paquetes el componente general, el componente y el conector. El cuerpo, el puerto y el rol también serán paquetes, aunque siempre estarán contenidos dentro de los paquetes anteriores. En el caso de las relaciones entre estos elementos, es decir, las funciones de inclusión y de transformación, estas serán representadas únicamente como saetas, extendiendo la metaclase Dependencia (*Dependency*), que se usa para indicar que un conjunto de elementos depende de otro para su especificación o implementación.

En el caso de las inclusiones, esto es más comprensible porque solo hay que indicar en qué sentido existe la inclusión o relación de subconjunto y su concepto está dado por los contenidos que se están relacionando, es decir, no hay otra información adicional. Para las transformaciones es distinto, pues no solo hay que indicar el sentido de la relación, sino que hay que suministrar las funciones. Esto sería un motivo suficiente para representarlas también mediante paquetes, sin embargo, aunque esto pudiera ser más potente, hace la solución más compleja y probablemente poco interesante, debido a que no siempre tiene que tratarse de funciones de transformación, pues puede darse el caso de que resulten de la transformación de un diagrama o conjunto de diagramas al código equivalente o que dichas funciones de transformación puedan ser obtenidas a partir de los dos extremos, sin que estas estén dadas de manera explícita. Todas las cuestiones planteadas se pueden apreciar en la figura 3.

Para definir un perfil, no basta con dar un nombre a los estereotipos y especificar las metaclases extendidas, sino que se debe documentar,[14,17] lo que implica especificar nombre, metaclase base, una descripción, un icono, restricciones y valores etiquetados.

A continuación se describen los diferentes estereotipos usados. No se presentan las descripciones, porque estas ya aparecieron anteriormente, y las metaclases que se extienden pueden apreciarse en la figura 3. Tampoco aparecen los iconos asociados, porque no se definen nuevos sino que se usa el propio de los paquetes.

GeneralComponent

Restricciones	Valores etiquetados
Tiene un único cuerpo	-

Component

Restricciones	Valores etiquetados
El conjunto de roles es vacío	

Conector

Restricciones	Valores etiquetados
El conjunto de puertos es vacío El número de roles es como mínimo 2	

Body

Restricciones	Valores etiquetados
No puede ser accedido desde fuera del paquete contenedor (*Component o Connector)* Como mínimo un diagrama de clases y uno de secuencia	Diagramas asociados

Port

Restricciones	Valores etiquetados
Como mínimo un diagrama de clases y uno de secuencia	Diagramas asociados

Role

Restricciones	Valores etiquetados
Como mínimo un diagrama de clases y uno de secuencia	

Inclusión

Restricciones	Valores etiquetados
Solo ocurren en conectores. El origen siempre es un rol y el destino un cuerpo	Conjunto de diagramas origen, conjunto de diagramas destino

Refine

Restricciones	Valores etiquetados
Siempre están asociadas con puertos. Del puerto hacia el cuerpo o del puerto hacia un rol	Conjunto de diagramas origen, conjunto de diagramas destino

Este perfil puede ser implementado en cualquier herramienta de modelado que permita la definición de perfiles. Particularmente, el *Visual Paradigm* en su edición *Community* admite la definición de estereotipos y valores etiquetados asociados.

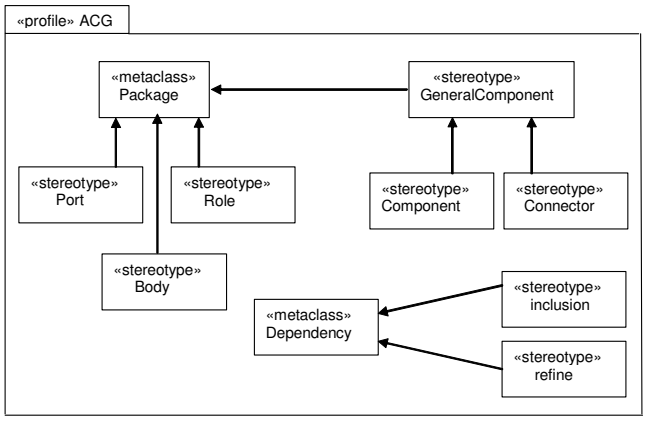

Fig. 3 *Estereotipos en el perfil ACG.*

CONCLUSIONES

Se puede concluir que es posible extender UML para representar la arquitectura de componentes genéricos mediante la definición de un perfil; que aunque es menos potente que la extensión del metamodelo es posible implementarlo en algunas de las herramientas de modelado que existen actualmente.

REFERENCIAS

1. **OREJAS, FERNANDO AND HARTMUT EHRIG:** "Components for Algebra Transformation Systems", *Electronic Notes in Theoretical Computer Science 82,* No. 7, Elsevier Science B V, 2003.
2. **HARTMUT, EHRIG,** *et al.*: *Generic Framework for Connector Architectures Based with Components and Transformations*. Formal Foundations of Embedded Software and Component-Based Software Architectures (FESCA´04), 2004.
3. **HARTMUT, EHRIG, *et al.*:** *Object-Oriented Connector-Component Architectures*, Formal Foundations of Embedded Software and Component-Based Software Architectures (FESCA´05), 2005.
4. **ALLEN, ROBERT AND DAVID GARLAN:** *A Formal Basis for Architectural Connection.* ACM TOSEM´97.
5. **MANN, STEFAN,** *et al.*: *Towards a Component Concept for Continuous Software Engineering*, Fraunhofer Institute Software und-Systemtechnik, Bericht 55/00, October, 2000.
6. **JACOBSON, IVAR; GRADY BOOCH Y JAMES RUMBAUGH:** *El proceso unificado de desarrollo de software* Ed. Félix Varela, Ciudad de La Habana, 2004.
7. **BOOCH, GRADY:** "UML in Action", *Communications of the ACM*, Vol. 42, No. 10, October, 1999.
8. **ENGELS, GREGOR; REIKO HECKEL AND STEFAN SAUER:** *UML - A Universal Language*? ICATPN 2000, LNCS 1825. Berlin Heidelberg, Springer-Verlag, 2000.
9. **WERMELINGER, MICHEL; ANTÓNIA LOPES AND LUIZ JOSÉ FIADEIRO:** *A Graph Based Architectural (Re)configuration Language*. ESEC/FSE 2001, Vienna, Austria, ACM, 2001.
10. **MEDVIDOVIC, NENAD** *et al.*: "Modeling Software Architectures in the Unified Modeling Language", *ACM Transactions on Software Engineering and Methodology,* Vol. 11, No. 1, January, 2002.
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. **PÉREZ-MARTÍNEZ, JORGE ENRIQUE AND ALMUDENA SIERRA-ALONSO:** *UML 1.4 versus UML 2.0 as Language to Describe Software Architectures*. EWSA 2004, LNCS 3047, Berlin, Heidelberg, Springer-Verlag, 2004.
13. **BJÖRKANDER, MORGAN AND CHRIS KOBRYN:** "Architecting Systems with UML 2.0", *IEEE Software*, July/August, 2003.
14. **MIGUEL, MIGUEL DE:** *UML Extensions for the Specification and Evaluation of Latency Constraints in Architectural Model*, WOSP´2000, Ontario, Canada. ACM 2001.
15. **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.
16. **FUENTES, LIDIA AND ANTONIO VALLECILLO:** *An Introduction to UML Profiles. UPGRADE*, The European Journal for the Informatics Professional, Vol. 5, issue No. 2, April, 2004.
17. **LUJÁN-MORA, SERGIO; JUAN TRUJILLO AND IL-YEOL SONG:** *21st International Conference on Conceptual Modeling (ER 2002)*, LNCS 2503. Tampere, Finland. October, 2002.