# EXTENSIÓN DEL METAMODELO DE UML PARA UNA ARQUITECTURA DE COMPONENTES

**Resumen / *Abstract***

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. En este caso se presenta una extensión de su metamodelo mediante la creación de nuevas metaclases y metaasociaciones para una arquitectura de componentes y conectores genéricos.

*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 UML metamodel extension based on new metaclasses and metaassociations for generic components and connectors-based architecture.*

**Palabras clave / *Key words***

UML, extensión, metamodelo UML, componentes, conectores

*UML, extension, UML metamodel, components, connectors*

## INTRODUCCIÓN

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 una extensión a su metamodelo, mediante la creación de nuevas metaclases y metaasociaciones para representar la arquitectura de componentes genéricos. Primeramente se presentará la arquitectura de componentes genéricos, para posteriormente explicar UML y porque es insuficiente su metamodelo para representar esta arquitectura; por último se mostrará la propuesta de extensión al metamodelo de UML 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 Aplicada, Doctor en Matemática, Catedrático, Departamento de Lenguajes y Sistemas Informáticos, Universidad Politécnica de Cataluña, España*
e-mail:orejas@lsi.upc.edu

*Recibido:mayo del 2006*
*Aprobado:junio del 2006*

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 ver en la figura 1.

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

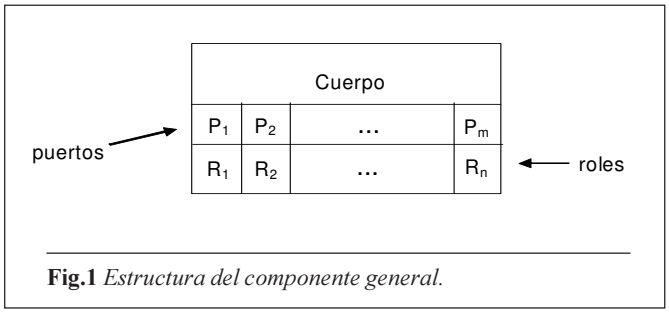


**Fig.1** *Estructura del componente general.*

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

Un conector tiene un cuerpo, un conjunto de al menos dos 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 puede 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 (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.[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 señalado no significa que la notación de UML cubra todas las necesidades para representar elementos arquitectónicos,[10] pero como se ha explicado, 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.

## EXTENSIÓN DE UML

Todas las cuestiones tratadas con anterioridad son válidas a los efectos de justificar que no deben ser usadas las metaclases y metasociaciones del metamodelo de UML para representar la ACG. Esto obliga a hacer una definición propia basada en este metamodelo.[11,14]

Es por esto, que se decide crear nuevas metaclases y metaasociaciones entre estas, en un nuevo paquete de nombre ACG, en el que se puedan representar los requerimientos antes expuestos, como se muestra en la figura 2.

Este metamodelo (paquete ACG) es conservativo[15] en el sentido de que no se modifican los elementos existentes en el metamodelo de UML, sino que las nuevas clases se obtienen por herencia, y además, puede ser usado para crear un perfil a partir de él siguiendo determinadas reglas de correspondencia entre las metaclases y los estereotipos, entre los metaatributos y los valores etiquetados, etc., según Koch y Kraus.[15]

Las metaclases que aparecen en este paquete se describen a continuación en orden alfabético, con sus principales características de acuerdo con la descripción del metamodelo de UML 2.0.

Todos los paquetes que se mencionan pertenecen al metamodelo de UML 2.0 y las clases en las que no se indica el paquete, es porque son propias de este paquete ACG.

Hay que 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.

**1. Clase Body.** Hereda de la metaclase Elemento (*Element*). Tiene asociaciones con las clases *Port* y *Role* y como atributo el conjunto de elementos requeridos para especificar su implementación. Tiene que pertenecer a un componente general o a una de sus subclases (Componente o Conector) y solo puede tener un tipo de relaciones, es decir, todas con puertos o todas con roles.

**2. Clase Component**. Hereda de la metaclase *GeneralComponent* y tiene asociaciones con las clases *Connector* y *Port*.

**3. Clase Connector.** Hereda de la metaclase *GeneralComponent* y tiene asociaciones con las clases *Component y Role*.

**4. Clase GeneralComponent.** Es una clase abstracta que hereda de la metaclase *Class* del paquete *StructuredClasses* de UML y que es superclase para las clases *Component y Connector*. Tiene como atributo un conjunto (que puede ser vacío) de funciones de cambios de nombre, las que permiten relacionar pares de elementos con diferentes nombres o alias en distintos contextos dentro del componente general.

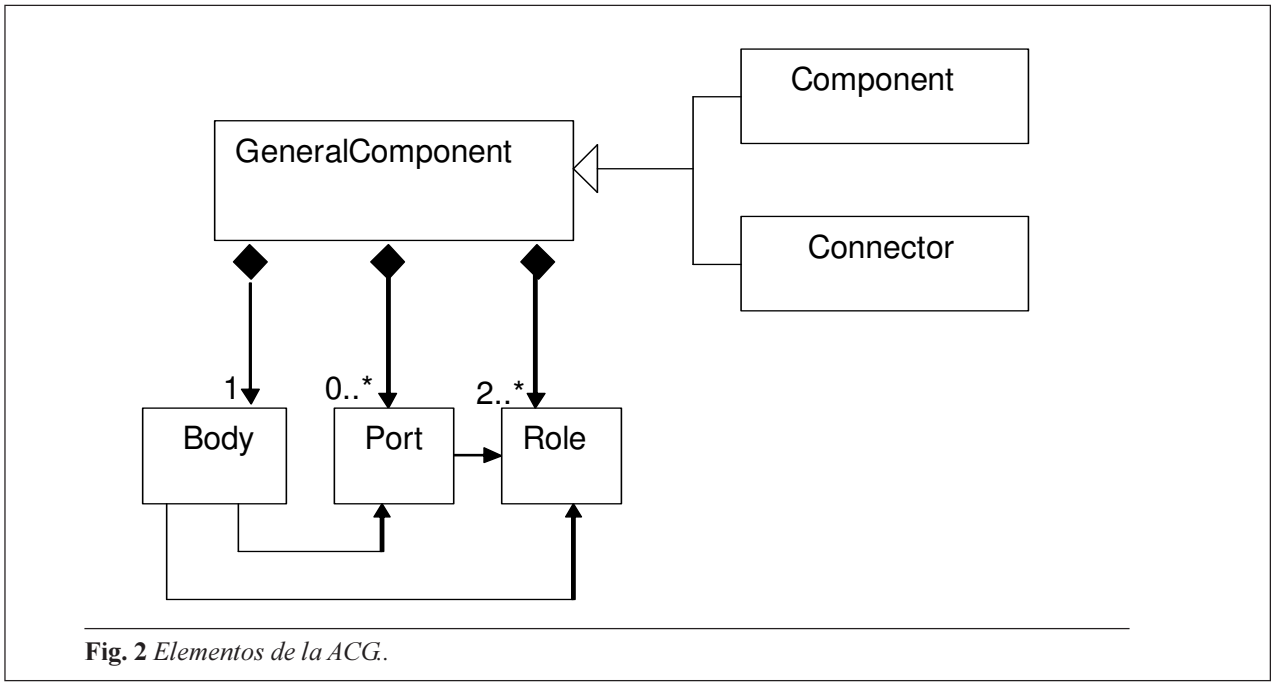


**Fig. 2** *Elementos de la ACG.*

Tiene asociaciones con las clases *Body*, *Port* y *Role*. El cuerpo no es visible desde el exterior y el conjunto de roles debe tener como mínimo dos elementos.

**5. Clase inclusión.** Se trata de un tipo de asociación (hereda de la metaclase *DirectedRelationship* del paquete Kernel de UML) para describir las relaciones de subconjunto que se establecen entre el cuerpo y los roles de un conector. Cada rol solo puede participar en una relación con el cuerpo.

**6. Clase Port.** Hereda de la metaclase *Element* del paquete Kernel de UML. Además de su asociación con el componente del que forma parte, se relaciona con los roles. Tiene como atributo los elementos de la especificación que se estén utilizando, diagramas de UML, redes de Petri, etcétera.

**7. Clase Refinement.** Es un tipo de asociación para describir mediante funciones de transformación, las relaciones que se establecen entre el cuerpo y los puertos de un componente o entre puertos y roles y por esta razón hereda de la metaclase *DirectedRelationship* del paquete Kernel de UML. Cada puerto solo puede participar en una relación con el cuerpo.

**8. Clase Role.** Hereda de la metaclase *Element* del paquete Kernel de UML. Su atributo es una lista de elementos de especificación para el rol, que pueden ser diagramas de UML, redes de Petri, etc. Tiene asociaciones con las clases *Connector y Port*.

## CONCLUSIONES

Se puede concluir que es posible extender UML para representar la arquitectura de componentes genéricos mediante la definición de un nuevo metamodelo que contiene nuevas metaclases y metaasociaciones. Este mecanismo de extensión es más potente que la creación de un perfil con este mismo objetivo, pero tiene como inconveniente que actualmente su integración con herramientas de modelado es prácticamente imposible.

## REFERENCIAS


1. **OREJAS, FERNANDO AND EHRIG HARTMUT:** "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, Oktober, 2000.
6. **JACOBSON, IVAR; GRADY BOOCH AND JAMES RUMBAUGH:** *El proceso unificado de desarrollo de software*, Ed. Félix Varela, 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 JOSÉ LUIZ FIADEIRO:** *A Graph Based Architectural (Re)configuration Language*, ESEC/FSE 2001, Vienna, Austria. ACM, 2001.
10. **MEDVIDOVIC, NENAD; DAVID ROSENBLUM; DAVID REDMILES AND JASON E. ROBBINS:** "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. Object Management Group. UML 2.0 Infrastructure. OMG document ptc/04-10-14. Disponible en http://www.omg.org/cgi-bin/doc?ptc/04-10-14
15. **KOCH, NORA AND ANDREAS KRAUS:** *Toward a Common Metamodel for the Development of Web Applications*. 3rd International Conference on Web Engineering, ICWE´03. LNCS 2722, Springer Verlag, July, 2003.