Representación de la arquitectura de software usando UML

Sandra Victoria Hurtado Gil
Universidad Icesi-I2T
shurtado@icesi.edu.co

RESUMEN
En los últimos años se han llevado a cabo una gran cantidad de investigaciones en el área de arquitectura de software, buscando principalmente una forma de representación de un sistema que supere la informalidad de las líneas y cajas pero que a la vez sirva de medio de comunicación con los diferentes interesados en el proyecto, es decir, que no sea demasiado complejo. El desarrollo de lenguajes de descripción de Arquitecturas da a los ingenieros de sistemas una nueva herramienta para la acertada representación de la arquitectura de un sistema; sin embargo, los lenguajes desarrollados actualmente por lo general son muy complejos o solo se adaptan a un tipo particular de sistemas.

En este artículo se presenta una forma de representación de la arquitectura de software basada en UML, aprovechando las ventajas de este lenguaje de modelamiento e incluyendo varias estructuras que facilitan la representación de amplia variedad de sistemas.

PALABRAS CLAVES
Arquitectura de software, estructuras arquitecturales, Lenguaje Unificado de Modelamiento (UML), Lenguajes de descripción de arquitectura.

Clasificación: B

ABSTRACT
A significant amount of research has been conducted in the Software Architecture field in the last few years. The focus of many of these studies is

to find a representation system able to go beyond the informality of the traditional "box-and-line" diagram, but keeping a low complexity level, so it can be used as a communication tool between all the software project stakeholders.

Computer systems engineers now may use Architecture Description Languages as a valuable tool for a more accurate representation of the system's architecture. However, most of these languages are very complex or purpose-specific.

This article presents an UML-based scheme for software architecture representations. This novel scheme benefits from all the advantages of UML, and includes several structures that enable the representation of an ample variety of systems.

KEYWORDS
Software architecture, architectural structures, unified modeling language (UML), architecture description languages.

INTRODUCCIÓN

Uno de los desarrollos más importantes dentro de la construcción del software ha sido el desarrollo de la arquitectura de software, que permite representar la estructura de un sistema a un nivel mayor que el dado por la programación o incluso el diseño [Boa95], [SG96].

Para representar adecuadamente la arquitectura de un sistema es necesario contar con varios diagramas o vistas [BCK98]. Dada la cantidad de características y de elementos que tiene un sistema de software no es posible incluirlos todos en un solo diagrama y que sirva, además, para todas las personas que participan en el desarrollo. Cada una de estas vistas es una estructura de la arquitectura del sistema, que muestran una parte del sistema como un conjunto de componentes, conectores y restricciones sobre sus tipos y relaciones. Además, cada estructura puede relacionarse con las demás para complementar la visión integral del sistema.

La arquitectura, conformada por diferentes visiones del sistema, constituye un modelo de cómo está estructurado dicho sistema, sirviendo de comunicación entre las personas involucradas en el desarrollo y ayudando a realizar diversos análisis que orienten el proceso de toma de decisiones.

Para que la arquitectura se convierta en una herramienta útil dentro del desarrollo y mantenimiento de los sistemas de software es necesario que se cuente con una manera precisa de representarla.

Las herramientas que se han elaborado para representar una arquitectura de software son los Lenguajes de Definición de Arquitecturas o ADL (Architecture Description Language). Sin embargo, los lenguajes desarrollados hasta el momento presentan diferentes problemas para su utilización en una empresa, como:

- Requieren una extensa capacitación.
- No son amigables para presentar la arquitectura a personas ajenas a la construcción del software.
- No tienen herramientas ni metodologías de apoyo.
- Algunos se encuentran especializados solo en un tipo particular de sistemas.
- Sólo tienen en cuenta una sola estructura del sistema.

Las desventajas que se presentan en estos lenguajes pueden ser superadas si se utiliza un lenguaje de modelamiento que sea conocido en la industria y que además esté apoyado por herramientas y metodologías de desarrollo, este lenguaje de modelamiento es UML, que se está convirtiendo en una notación estándar de hecho en las empresas.

UML permite que se represente de manera semi-formal la estructura general del sistema, con la ventaja de que este mismo lenguaje puede ser usado en todas las etapas de desarrollo del sistema y su representación gráfica puede ser usada para comunicarse con los usuarios.

En la siguiente sección se describe de manera general el lenguaje de modelamiento UML y los mecanismos que presenta para su extensión. La sección tres presenta una propuesta

para representar las estructuras de arquitectura utilizando UML, y en la sección cuatro se muestran trabajos relacionados. Por último se plantearán las conclusiones de esta propuesta de investigación y las posibilidades de trabajos futuros a partir de ella.

UML

Generalidades

UML es un lenguaje gráfico de modelamiento que usa conceptos de orientación por objetos. Este lenguaje tiene una sintaxis y una semántica bien definidas, sirviendo además para todas las etapas de desarrollo [RMR98], [BRJ99].

En UML se utilizan para el modelamiento de un sistema diferentes elementos y relaciones, que tienen una semántica y sintaxis definidas. Estos elementos se agrupan en diagramas preestablecidos que corresponden a diferentes proyecciones del sistema.

Los elementos básicos de UML, aquellos que representan principalmente las partes estáticas del sistema, son:

- Clases
- Casos de uso
- Componentes
- Nodos
- Paquetes

Las relaciones que se utilizan para establecer conexiones entre los elementos son:

- Dependencia
- Asociación
- Generalización
- Realización

Cada uno de estos elementos y relaciones tiene una representación gráfica y puede complementarse su información utilizando lo que se conoce como especificación. La especificación de un elemento o relación generalmente no es visible en la representación gráfica, o sólo lo es parcialmente, y corresponde a los datos o propiedades adicionales que completan o detallan la semántica del elemento o relación, y por lo tanto del sistema en general.

Los elementos y relaciones se agrupan en diagramas que representan diferentes aspectos del sistema. Los diagramas de UML son:

- *Diagrama de clases:* Presenta las clases, junto con sus atributos, operaciones, interfaces y relaciones. También presenta el agrupamiento de clases en paquetes y las relaciones entre ellos.
- *Diagrama de objetos:* Muestra instancias de clases (objetos) con valores en sus atributos y relaciones.
- *Diagrama de casos de uso:* Los escenarios de uso del sistema, incluyendo los roles de los usuarios.
- *Diagramas de interacción:* Comprende los diagramas de secuencia y de colaboración. Presenta objetos y relaciones entre ellos desde el punto de vista dinámico.
- *Diagrama de estado:* Representa los posibles estados, eventos y transiciones entre las clases u objetos.
- *Diagrama de componentes:* Organización y dependencia entre componentes físicos.
- *Diagrama físico (deployment):* La distribución y comunicación de los

componentes en los dispositivos de hardware.

La gran ventaja de UML es el hecho de que poco a poco se ha venido adoptando en diferentes medios empresariales y académicos como el lenguaje "estándar" para el análisis y diseño de los sistemas de software. Gracias a la posibilidad de extender el UML y a la construcción de herramientas y metodologías que apoyan este lenguaje se ha convertido en el estándar de facto en la actualidad para el modelamiento de los sistemas.

Mecanismos de extensión

UML tiene principalmente tres mecanismos de extensión que permiten construir nuevos elementos o modificar la semántica de los ya existentes, para hacer más precisa la representación del sistema. Estos mecanismos son:

1. **Valores Adicionados** (tagged values): Mediante estos valores es posible adicionarle nuevas propiedades o atributos a los elementos del modelo de UML.
2. **Restricciones** (constraints): Las restricciones permiten adicionar nueva semántica o modificar la existente.
3. **Estereotipos**: Los estereotipos permiten crear nuevos elementos en el modelo basados en otros ya existentes. Cada nuevo estereotipo puede reunir propiedades (tagged values) y restricciones particulares.

A través de estas extensiones es posible enriquecer el modelo de UML para representar adecuadamente los diferentes aspectos del sistema.

Por ejemplo, si se desea mostrar un nuevo elemento en el diagrama de componentes que represente a un sistema externo (como una librería, un programa servidor, etc.) puede crearse un estereotipo basado en el elemento componente. Este estereotipo puede tener una representación gráfica diferente a la de los componentes para que sus elementos sean identificados más fácilmente en el diagrama. Además pueden adicionarse al estereotipo otras propiedades que no tenga un componente, como el nombre del responsable de dicho sistema.

En la Figura 1 se muestra un ejemplo de un diagrama de componentes que incluye un elemento con el nuevo estereotipo externo. En el diagrama aparecen dos componentes (A y B) y un sistema externo (Ext). En este caso el estereotipo tiene una representación gráfica que lo diferencia de los componentes, pero también puede tener el mismo icono de los componentes y llevar el nombre del estereotipo para diferenciarlo.

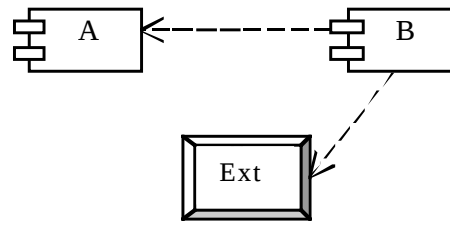


Figura 1. Diagrama de componentes adicionando un elemento con el estereotipo externo

ESTRUCTURAS DE ARQUITECTURA EN UML

La primera fase del proyecto es la identificación de los componentes que participan en la descripción de la arquitectura de un sistema, y luego sus relaciones en las diferentes estructuras. Para cada componente y conector se determinan los elementos de UML que los representan, con su sintaxis y semántica. Algunos componentes o estructuras no tendrán una representación directa y en este caso se utilizarán los mecanismos de extensión que provee UML, como estereotipos o restricciones.

Como parte integral de cada estructura se deben incluir restricciones adicionales que determinan las relaciones y los tipos de componentes y conectores que pueden aparecer en dicha estructura.

Por último en el proyecto se presentan las relaciones existentes entre las diferentes estructuras y la manera de verificar dichas relaciones, lo que ayudará a la persona que modela la arquitectura del sistema a validar la consistencia de esta última.

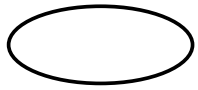
COMPONENTES

1. Casos de uso (componentes conceptuales)

Un caso de uso representa un requerimiento funcional del sistema o un proceso del negocio que se implementa en el sistema de software.

Representación:

Se usa el mismo elemento *Caso de Uso* de UML



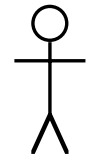
Nombre

2. Actores

Un actor es una persona, sistema o dispositivo que interactúa con el sistema, iniciando, recibiendo los resultados o participando en alguna de las acciones de un caso de uso. Por lo general representa un rol, por ejemplo: jefe de contabilidad, profesor, etc.

Representación:

Se usa el elemento *Actor* de UML.



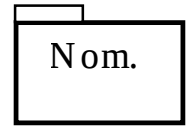
Nombre

3. Módulos

Un módulo es una división conceptual del sistema que puede ser visto como una agrupación de funciones que tengan alguna relación entre ellas y, por lo tanto, puede presentar un servicio completo al exterior una vez se ha desarrollado.

Representación:

Un modulo se representa con el elemento *Paquete* de UML.

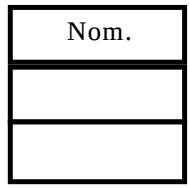


4. Clases

Una clase es la representación abstracta de un conjunto de objetos o artefactos que debe modelar el sistema. Cada clase incluye las características y el comportamiento de los objetos que representa. Una clase puede ser de tipo interfaz (interactúa con el exterior), control (realiza operaciones y controla otras clases) u entidad (hace persistentes los datos).

Representación:

Es el mismo elemento *Clase* de UML

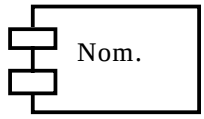


5. Unidades de Software

Conjunto de funciones (en programas o procedimientos) que realizan las acciones del sistema y que se implementan en archivos físicos. Las unidades de software tienen asociado un tipo (valor adicionado), que puede ser: filtro, procedimental, objetos, repositorio de datos activo u otro.

Representación:

Se usa el elemento *Componente* de UML.

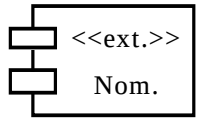


6. Sistemas externos

Un sistema externo representa un sistema de la organización que interactúa con el sistema que se está desarrollando. Por ejemplo, el sistema de contabilidad (si se está desarrollando el de recursos humanos).

Representación:

Se creará un estereotipo para representar a este componente. El estereotipo, llamado *Externo*, está basado en el elemento componente de UML.

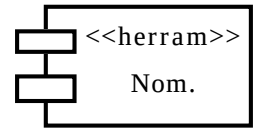


7. Herramientas de software

Sistemas o programas que contribuyen al adecuado funcionamiento del sistema. Por ejemplo, el sistema operativo, un programa navegador de Internet, la máquina virtual de java, etc.

Representación:

Se crea el estereotipo *Herram*, basado en el elemento componente de UML.

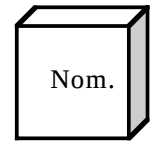


8. Procesador

Este componente representa un computador (procesador y memoria) donde se localizan programas o datos y donde, por lo general, se corren dichos programas. Este procesador puede tener roles como servidor, cliente, terminal, etc. Además, puede establecerse su ubicación física (ciudad o área de la organización) para complementar la información de este componente.

Representación:

Se usa el elemento *Nodo* de UML.

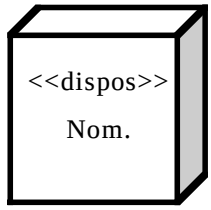


9. Dispositivo

El dispositivo es un componente o elemento de hardware que presenta una interacción con el sistema. Por ejemplo, un medidor de presión, un terminal de computadores o un módem.

Representación:

Se crea un estereotipo *Dispos* basado en elemento nodo de UML.



Estructuras

Para el proyecto de investigación se ha desarrollado la forma de representación de ocho estructuras. Para cada una se identifican los componentes, los conectores y las restricciones que deben cumplirse. A continuación, a manera de ejemplo, se presentan dos estructuras, la estructura funcional y la estructura de llamados.

1. Estructura funcional

Esta estructura representa las funciones que ofrece el sistema a los usuarios finales, a otros sistemas o dispositivos, es decir, lo que representa el sistema para los que interactúan con él. Puede verse esta estructura como la visión conceptual del sistema, permitiendo determinar los aspectos del negocio que se desean implementar en el sistema.

Esta es una de las estructuras más importantes en un sistema, ya que ayuda a la captura de los requerimientos y se convierte en un medio de comunicación útil con los usuarios, permitiendo ver gráficamente las relaciones de los usuarios con el sistema y los procesos del negocio identificados por ellos mismos.

La estructura funcional corresponde al diagrama de casos de uso de UML, pero pueden adicionarse nuevos tipos de conectores o restricciones, que muestren relaciones de alto nivel entre los casos de uso y/o actores.

A. *Componentes:*

- Casos de uso
- Actores

B. *Conectores:*

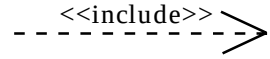
Generalización (herencia): Indica que un componente hereda el comportamiento y atributos del otro.

- Participación: Permite relacionar a los actores con los casos de uso, indicando así que se presenta algún tipo de interacción entre el actor y el sistema a través del caso de uso.

  {tipo participación}

- Inclusión: Representa que las acciones del caso de uso que recibe la relación se adicionan a las acciones del caso de uso que la inicia.

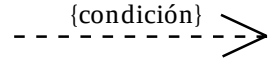
  <<include>>

- Inclusión condicionada: Se adicionan las acciones del caso de uso que recibe la relación, pero sólo cuando se cumple una condición dada.

  <<condic>>

  {condición}

C. *Restricciones:*

- Entre dos casos de uso no puede presentarse simultáneamente la relación inclusión e inclusión Condicionada, sólo una de ellas.
- Entre dos actores sólo puede utilizarse el conector generalización.

- El conector de participación sólo puede relacionar un actor y un caso de uso.
- Un componente (caso de uso o actor) no puede ser simultáneamente hijo y padre de otro componente usando el conector generalización
- El conector generalización no puede establecerse entre un caso de uso y un actor

2. Estructura de llamados

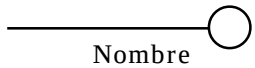
En esta estructura se presentan los servicios que ofrece y los eventos que genera cada unidad de software, señalando además las relaciones de dependencia que se presentan entre estas unidades por llamar un servicio o escuchar un evento de otra unidad.

A. *Componentes:*

- Unidades de software
- Sistemas externos

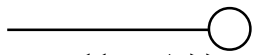
B. *Conectores:*

- Servicio: Este conector representa los servicios que ofrece una unidad de software o sistema externo, y que pueden ser utilizados por otros componentes o por el usuario final mediante un llamado explícito.



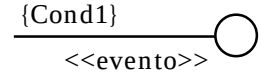
Nombre

- Evento: Representa los eventos que dispara una unidad de software o sistema externo, y que pueden ser escuchados por otros componentes.



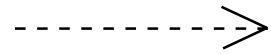
<<evento>>
Nombre

- Si la generación del evento se realiza de manera condicional puede incluirse la condición en la representación gráfica



{Cond1}
<<evento>>
Evento 1

- Uso: Este conector permite que se relacionen los componentes con los conectores servicio y evento, que presentan los otros componentes, indicando así la interacción o dependencia entre ellos. El tipo del conector indica la forma por la cual se realiza la interacción. Este tipo puede ser pipe, llamado remoto, llamado directo, escucha trigger, escucha evento u otro.



C. *Restricciones:*

- Todo conector uso debe estar relacionado con uno y solo un conector servicio o evento.
- Todo sistema externo debe tener por lo menos un conector, ya sea servicio, evento o uso.
- Si un sistema externo tiene un conector servicio o evento, éste no puede encontrarse sin relación con un conector uso de otro componente.
- Los conectores servicio o evento pueden relacionarse con múltiples conectores uso.
- Un conector uso, que se relacione con un conector servicio sólo puede ser de tipo pipe, llamado remoto, llamado directo u otro.

– Un conector que se relaciona con un conector evento sólo puede ser de tipo escucha trigger, escucha evento u otro.

3. Otras estructuras

Las otras estructuras identificadas para representar la arquitectura de un sistema son:

– *Estructura modular:* Presenta una división del sistema en módulos más pequeños o subsistemas.

– *Estructura de uso:* Permite mostrar las relaciones de dependencia que se presentan entre los componentes del sistema.

– *Estructura de clases:* Esta estructura es similar al diagrama de clases presentado en UML, y representa el modelo lógico de los datos necesarios en el sistema.

– *Estructura de flujo de datos:* Modela el intercambio de datos e información que relaciona los diferentes componentes de software.

– *Estructura de sincronización:* Corresponde a las relaciones de sincronización y de concurrencia que se dan entre los componentes, indicando las restricciones que son necesarias para el correcto funcionamiento del sistema.

– *Estructura física:* Permite modelar la distribución de los componentes del sistema en los diferentes dispositivos físicos o de hardware de que se dispone.

Relaciones entre estructuras

Una parte fundamental de la arquitectura de software es la relación entre los diferentes componentes de las estructuras, lo que permite que las diversas visiones que se tienen del sistema se complementen adecuadamente.

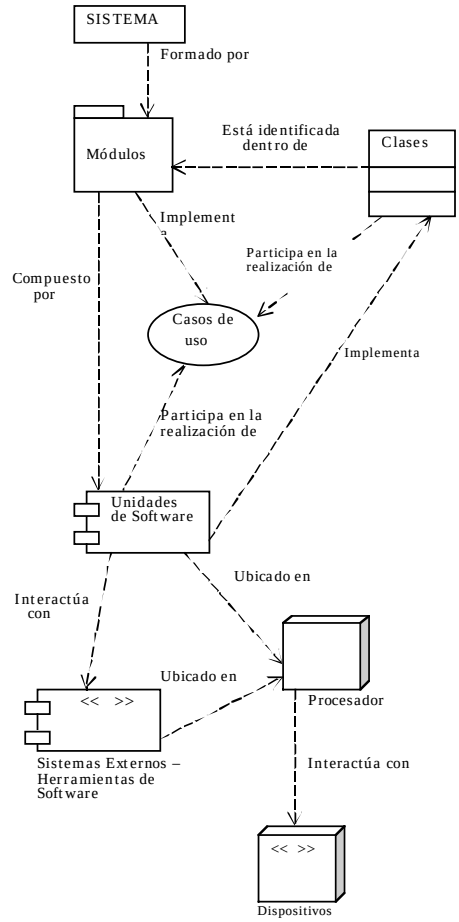


Figura 2. Relaciones entre los componentes de la arquitectura de software

En la Figura 2 se resumen las principales relaciones entre los componentes de la arquitectura del sistema. Además de estas relaciones, las estructuras presentan restricciones entre sí que permiten integrar toda

la visión del sistema. Ejemplos de estas restricciones son:

- Todo caso de uso debe ser implementado en un módulo, es decir, no pueden encontrarse casos de uso que no se relacionen con algún módulo.
- Toda clase debe participar en la realización de por lo menos un caso de uso.
- Toda relación establecida entre una unidad de software y un caso de uso debe corresponder a una relación que exista entre un caso de uso y una clase y entre ésta y una unidad de software.
- Si una clase se identifica con un módulo debe participar en la realización de por lo menos un caso de uso que implemente el módulo.
- ...

Esta relación entre las diferentes componentes de las estructuras no sólo permite que se tenga una visión integral del sistema sino que se puedan realizar comprobaciones relacionadas con la completitud del sistema. Por ejemplo, cuando se modelan las unidades de software cada una debe relacionarse con una o más clases, y si hay alguna unidad de software que no se relaciona con ninguna clase puede significar que ha sido omitido algún objeto o artefacto en el modelo lógico de datos.

Las verificaciones pueden realizarse de forma manual por las personas que elaboran las diferentes estructuras de la arquitectura del sistema, pero también pueden ser asistidas por herramientas automáticas, de manera que sólo se presenten mensajes de error cuando se encuentren inconsistencias.

Por último es importante resaltar que algunas relaciones entre las estructuras pueden servir de base para la elaboración de otras estructuras o diagramas que no corresponden a la arquitectura del sistema, por lo que no se detallan en este proyecto, pero son importantes para su desarrollo. Por ejemplo, puede elaborarse un diagrama de interacción entre las unidades de software para mostrar en detalle cómo implementan las acciones de un caso de uso, pero esto haría parte del diseño detallado del sistema.

TRABAJOS RELACIONADOS

A medida que el concepto de arquitectura de software evolucionaba, diferentes grupos empezaron a desarrollar lenguajes de descripción de arquitecturas u otras formas para representar la arquitectura de un sistema. De estos desarrollos pueden resaltarse algunos aspectos importantes y su relación con el presente trabajo.

Algunos trabajos presentan grafos o gramáticas de grafos para representar una arquitectura [Met96]. Este tipo de representación se asemeja mucho al tradicional esquema de cajas y flechas con el cual se modela la "arquitectura" del sistema, con la ventaja de que le adiciona mayor formalismo. Sin embargo, esta representación no incluye la semántica de los diferentes componentes ni la relación entre las diversas estructuras.

La mayoría de las investigaciones se han orientado al desarrollo de lenguajes de definición de arquitecturas, en-

tre los cuales se encuentran Unicon, Rapide, C2, Sadl, Wrigh, y otros [Ves93]. Estos lenguajes presentan ventajas para representar sistemas específicos o para realizar diferentes tipos de análisis. Su desventaja es que por lo general sólo incluyen una estructura y su representación textual es muy compleja, lo que hace difícil su utilización en las organizaciones.

Por último la utilización de UML para representar arquitecturas de software, presentada en [Kru95] y [JBR99], no adiciona realmente un nuevo nivel de abstracción, pues se basa en agrupar los elementos más importantes de los diferentes diagramas de UML.

CONCLUSIONES Y TRABAJOS FUTUROS

Para poder aprovechar los beneficios que presenta la arquitectura de software es importante que se involucre como una nueva disciplina dentro del desarrollo de un sistema en las organizaciones. Este objetivo se logra no sólo con el aprendizaje de los nuevos conceptos, sino también con el conocimiento y utilización de herramientas que apoyen el desarrollo de la arquitectura del sistema.

La propuesta que se ha presentado de representar la arquitectura de un sistema utilizando el lenguaje de modelamiento UML apoya la difusión de los conceptos de la arquitectura de software, ya que facilita su apropiación en las empresas por estar basado en un lenguaje de modelamiento muy conocido.

Esta propuesta también presenta la ventaja de incluir diferentes estructuras de la arquitectura que se relacionan entre sí, permitiendo que todas las personas interesadas en el sistema tengan varias visiones del mismo y puedan analizar los componentes y sus relaciones en cada una de ellas.

Para apoyar esta forma de representación pueden desarrollarse herramientas que automaticen en parte el proceso de elaboración, verificación y análisis de la arquitectura. Por ejemplo, pueden verificarse las restricciones generales establecidas para cada estructura o para las relaciones entre las estructuras.

Para involucrar los estilos de software en la representación pueden elaborarse restricciones para los componentes, conectores y estructuras para que representen a un estilo de arquitectura en particular. También es posible realizar análisis posteriores a la arquitectura del sistema, para determinar si pertenece a algún estilo de software.

Otro trabajo que se puede desarrollar a partir de la forma de representación mostrada en esta propuesta es la construcción de metodologías (o modificación de existentes) que incluyan la arquitectura del sistema en el proceso de desarrollo.

REFERENCIAS

[BRJ99] Grady Booch, James Rumbaugh e Ivar Jacobson. *The Unified Modeling Language User Guide.* Rational Software Corporation. Addison-Wesley, 1999.

[BCK98] Len Bass, Paul Clements y Rick Kazman. *Software Architecture in Practice.* Sei Series In Software Architectures. Addison Websley, 1998.

[Boa95] Maarten Boasson. *The Artistry of Software Architecture.* IEEE Software, Vol. 12, No. 6, November 1995 (pp. 13-16).

[GP95] David Garlan, Dewayne E. Perry. *Introduction to the Special Issue on Software Architecture.* IEEE Transactions on Software Engineering, Vol. 21, No. 4, Abril 1995 (pp. 269-274).

[JBR99] Ivar Jacobson, Grady Booch y James Rumbaugh. *The Unified Software Development Process.* Rational Software Corporation. Addison-Wesley, 1999.

[Kru95] Philippe Kruchten. *The "4+1" View Model of Software Architecture.* IEEE Software, Vol. 12, No. 6, November 1995 (pp. 42-50).

[Met96] Daniel Le Métayer. *Software Arquitecture Styles as Graph Grammars.* Irisa/Inria. 1996.

[RMR98] Jason E. Robbins, Nenad Medvidovic, David F. Redmiles y David S. Rosenblum. *Integrating Architecture Description Languages with a Standard Design Method.* 20th International Conference on Software Engineering, 1998.

[SDK95] Mary Shaw, Robert DeLine, Daniel V. Klein, Theodore L. Ross, David M. Young y Gregory Zelesnik. *Abstractions for Software Architecture and Tools to Support Them.* IEEE Transactions on Software Engineering, Vol. 21. No. 4, Abril 1995.

[SG96] Mary Shaw y David Garlan. *Software Architecture Perspectives on an Emerging Discipline.* Prentice Hall, 1996.

[Ves93] S. Vestal. *Acursory Overview and Comparison of Four Architecture Description Languages.* Tecnical Report, Honeywell Tecnology Center, Febrero 1993.

CURRILUM

Sandra Victoria Hurtado Gil. Ingeniera de Sistemas de la Universidad Icesi. Realizó la maestría de ingeniería de Sistemas y computación en la Universidad de los Andes, con concentración en construcción de software. Trabajó en el grupo de desarrollo de Sistemas de la Universidad Icesi y actualmente es jefe del Departamento de Sistemas en esta misma institución.