# REPRESENTACIÓN DE ASPECTOS CANDIDATO EN ESQUEMAS PRECONCEPTUALES

Carlos Mario Zapata Jaramillo*
Guillermo González Calderón**
Gabriel Eduardo Orozco***

Recibido: 29/09/2009
Aceptado: 07/05/2010

## RESUMEN

Los aspectos se vienen consolidando poco a poco como un nuevo paradigma de desarrollo de *software*. Si bien ya son comunes en programación, donde existen lenguajes orientados a aspectos, su evolución se viene trasladando hacia las etapas iniciales del ciclo de vida del *software*. Existen algunas iniciativas para la identificación de aspectos desde lenguaje natural, pero aún presentan falencias en la manera de traducir los aspectos desde lenguaje natural hasta los esquemas conceptuales. Por ello, en este artículo se presenta una propuesta para la representación de aspectos en los denominados esquemas preconceptuales, que son esquemas previos a la elaboración de esquemas conceptuales, para luego traducirlos en dos de los diagramas más representativos de UML: clases y secuencias. El entorno completo se ejemplifica luego mediante un caso de estudio.

**Palabras clave:** aspectos, ciclo de vida del *software*, representación, esquemas preconceptuales, diagramas de clases y de secuencias de UML.

* Ph. D en Ingeniería, profesor asociado de la Universidad Nacional de Colombia, líder del grupo de investigación en Lenguajes ComputacionalesFacultad de Minas, Escuela de Sistemas. Universidad Nacional. E-mail: cmzapata@unal.edu.co
** M. Sc. Ingeniería de Sistemas, docente de la Facultad de Ingenierías, Universidad de Medellín, Grupo de Investigación ARKADIUS. E-mail: ggonzalezc@udem.edu.co
*** Estudiante de Maestría en Ingeniería de Sistemas. Universidad Nacional. Medellín, Colombia. E-mail: georozco@unalmed.edu.co

# REPRESENTING CANDIDATE ASPECTS BY MEANS OF PRE-CONCEPTUAL SCHEMES

## ABSTRACT

Aspects have been gradually consolidating as a new software development paradigm. Although aspects are already common in programming, as aspect-oriented languages have shown, aspect evolution is currently going to the first phases of software development lifecycle. Some work has been devoted to natural-language-based aspect identification. However, translation from natural language to conceptual schemas still has problems. For this reason, we present, in this paper, a proposal for representing aspects in the so-called pre-conceptual schemes (PS), which are previous diagrams to conceptual scheme generation. PS representation is, then, translated into class and sequence diagram, two of the most representative UML diagrams. The overall environment is, finally, exemplified by means of a case study.

**Key words:** aspects, software development lifecycle, representation, pre-conceptual schemas, UML class and sequence diagrams.

## INTRODUCCIÓN

Kiczales *et al.* [1] presentan los aspectos como un paradigma de programación, diferente a la orientación a objetos, que pretende solucionar los problemas del código disperso y enmarañado. Este nuevo paradigma viene, desde entonces, progresando en paralelo con los demás paradigmas de desarrollo y consolidándose como una de las maneras de programación de aplicaciones.

Los lenguajes de orientación aspectual, tales como AspectJ™, HyperJ™, AspectC™ y AspectC++™, ya se suelen utilizar regularmente en el proceso de desarrollo, de modo que es posible generar código que opere con intereses transversales a las diferentes aplicaciones, representando, en ocasiones, requisitos no funcionales de las mismas [2]. El progreso en programación aspectual se viene complementando con una creciente necesidad de encontrar equivalencias, en diferentes esquemas conceptuales de análisis y diseño, para el código aspectual. Así, se presentan varias propuestas para la representación de aspectos en diferentes diagramas, particularmente los pertenecientes al Lenguaje Unificado de Modelado (UML) [3-6]. En estos proyectos, sin embargo, es el diseñador del sistema quien se debe ingeniar la representación de los aspectos y trazarla en el lenguaje gráfico que utilice.

Una continuación obligada de estas nuevas formas de representación de aspectos en etapas tempranas del desarrollo la constituyen los trabajos que procuran la identificación de los aspectos desde lenguajes naturales o controlados. Proyectos como *Theme* [7] y *Lexical Chain Viewer* [8] permiten identificar conceptos y relaciones que se podrían considerar aspectos, pero se ocupan poco de su conversión a alguna de las propuestas de representación de aspectos en esquemas conceptuales.

Los problemas antes descritos justifican la propuesta que se presenta en este artículo para emplear los denominados esquemas preconceptuales (un tipo especial de diagramas que compendia las características estructurales, de comportamiento e interacción en un único diagrama), ampliar su sintaxis para dar cabida a nuevos elementos y traducirlos en dos de los diagramas más representativos de UML: clases y secuencias.

Este artículo se estructura así: en la sección 2 se expone el marco teórico con los conceptos iniciales que permiten la comprensión de la propuesta; en la sección 3 se incluyen los antecedentes en representación de aspectos desde las fases tempranas del ciclo de vida del *software*; en la sección 4 se propone la representación mediante esquemas preconceptuales; en la sección 5 se presenta un caso de estudio; finalmente, en la sección 6 se concluye y se presentan líneas de trabajo futuro.

## 1 MARCO TEÓRICO

### 1.1 Aspectos

Losavio *et al.* [9] recopilan algunas de las definiciones más importantes que tienen que ver con los aspectos, de la siguiente manera:

*Crosscutting concern*: una funcionalidad o asunto de interés para el sistema que se encuentra dispersa en toda la aplicación.

Aspecto: es la implementación de un *crosscutting concern*.

*Join point*: es un punto de interés de una aplicación en el cual se pueden componer (ligar elementos que se crearon separadamente) dos aspectos.

*Point cut*: es un predicado que permite identificar y seleccionar un conjunto de *join points*.

### 1.2 Representación de aspectos en diagramas de clases y secuencias de UML

El *Unified Modeling Language* (UML), que es el estándar *de facto* para el desarrollo de aplicaciones de software, posee un lenguaje complementario

denominado *Object Constraint Language* (OCL) para expresar las restricciones de los diferentes diagramas. Este lenguaje, sin embargo, no posee equivalencias con los lenguajes de orientación aspectual. Por ello, Tabares *et al.* [6] proponen una manera de representar los aspectos en diagramas de UML, específicamente en los diagramas de clases y secuencias. En cuanto al primer diagrama, un aspecto candidato se puede representar en términos de una clase con el estereotipo <<*aspect*>>, la cual posee una operación estereotipada <<*point cut*>>, tal como se muestra en la figura 1(a). En el caso del diagrama de secuencias, un aspecto se asimila a una nota UML ligada con el(los) objeto(s) que intervengan en la interacción. En la nota se detalla el aspecto a que se refiere y el *point cut* respectivo, tal como se muestra en la figura 1(b).

### 1.3 Esquemas preconceptuales

Zapata *et al.*[10] proponen una forma de representación del conocimiento mediante esquemas preconceptuales. En estos esquemas se puede representar gráficamente un discurso expresado en UN-Lencep (Universidad Nacional—Lenguaje controlado para la especificación de esquemas preconceptuales, que es un subconjunto del lenguaje natural que los interesados entienden y pueden validar en las fases iniciales del desarrollo de *software*) para luego traducirlo en diferentes diagramas. En la figura 2 se presentan los diferentes elementos que hacen parte de los esquemas preconceptuales, que son: conceptos, que incluyen sustantivos y frases centradas en sustantivos; relaciones estructurales, que son los verbos "es" y "tiene"; relaciones dinámicas, que coinciden con los verbos de operación; notas, que permiten especificar posibles valores de un concepto; condicionales, que establecen restricciones que se deben cumplir para poder realizar operaciones; conexiones, que unen relaciones (estructurales y dinámicas) con conceptos y viceversa; implicaciones, que constituyen relaciones causa-efecto entre relaciones dinámicas o entre condicionales y relaciones dinámicas.

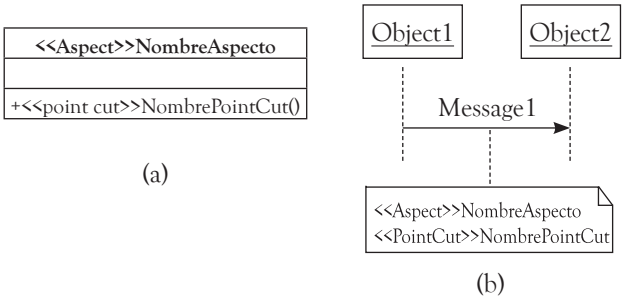


**Figura 1.** Representación propuesta por Tabares *et al.* [6] para los aspectos en: (a) diagrama de clases y (b) diagrama de secuencias.
Fuente: Elaboración propia.

## 2 ANTECEDENTES

### 2.1 Propuestas de representación de aspectos en UML

La propuesta de Tabares *et al.* [6], discutida en la sección 2.2, no es la única que emplea UML y su mecanismo de extensibilidad para la representación de aspectos. Por ejemplo, Anjum [3] propone una manera de modelar algunos requisitos no funcionales como aspectos en el diagrama de clases. Suzuki y Yamamoto [4] proponen un trabajo similar, que incluye los diagramas de clases y paquetes. Finalmente, Krechetov *et al.* [5] presentan una recopilación de algunas de las principales represen-

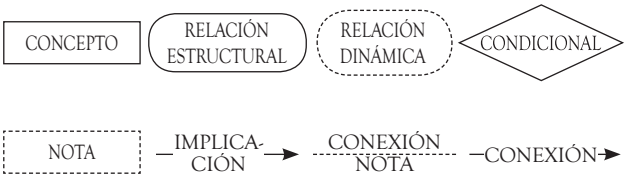


**Figura 2.** Elementos de los esquemas preconceptuales.
Fuente: Elaboración propia

taciones de aspectos en UML para proponer una visión integrada en los diagramas de clases y de paquetes. Todos estos enfoques tienen en común el hecho de que el analista o diseñador, según sea el caso, es quien debe capturar la información y elaborar el diagrama UML que represente los aspectos, sin que medie para ello ningún recurso computacional. Así, la interpretación de lo que se debe colocar como aspecto corre a cargo del analista o diseñador, sin que pueda haber validación de parte del interesado.

### 2.2 Aspectos candidato desde lenguaje natural

En busca de ayudas para la identificación de aspectos desde lenguaje natural o controlado, algunos proyectos pretenden detectar ciertos patrones en el lenguaje para establecer qué se podría definir como "aspecto candidato". En esta línea de trabajo, el proyecto *Theme* [7] es uno de los más completos, pues permite establecer, a partir de un documento de requisitos, algunos verbos que se podrían considerar aspectos candidato, para luego intentar una representación en algún lenguaje gráfico de desarrollo de *software*. En esta misma línea, el *Lexical Chain Viewer* [8] posee un entorno que únicamente identifica los aspectos candidato pero no sugiere su forma de representación. En ambos casos, el documento de partida corresponde a los requisitos identificados para la potencial aplicación de *software*. Este documento, sin embargo, es el resultado concreto de un largo proceso de captura de requisitos, por lo cual se ubica mucho más adelante en el ciclo de vida del *software* que las descripciones iniciales que realizan los interesados. Además, el hecho de no apuntar a una representación concreta aleja estos procesos de identificación de la codificación de la aplicación de *software*, algo que ya los trabajos iniciales en programación orientada a aspectos estructuraban en sus lenguajes.

## 3 PROPUESTA PARA LA REPRESENTACIÓN DE ASPECTOS CANDIDATO EN ESQUEMAS PRECONCEPTUALES

La sección 3 muestra una problemática de la realidad a la que se enfrentan los trabajos orientados a aspectos en la actualidad, que se puede sintetizar de la siguiente manera:

Cuando se parte de documentos de requisitos se desperdicia información valiosa que puede entregar el interesado en las entrevistas de captura de requisitos. Además, no se cuenta, generalmente, con elementos de enlace para una representación en lenguajes de diseño.

Cuando se parte de lenguajes gráficos de representación, como el UML, la responsabilidad sobre la definición de los aspectos recae directamente sobre el analista o el diseñador, puesto que el interesado no entiende lo suficiente esos lenguajes de modelado y sus extensiones como para validar los resultados.

Cuando se parte de lenguajes de programación orientada a aspectos, la responsabilidad es del programador y cualquier error detectado en esta etapa podría tener repercusiones desastrosas en los tiempos de entrega o la calidad de la aplicación de *software*.

Así, se requiere una solución que parta de una representación del discurso del interesado en lugar de un documento de requisitos. Se requiere, además, que los interesados puedan entender y validar esa representación del discurso, de modo que se puedan detectar errores desde fases muy tempranas del desarrollo. Finalmente, se requiere que esa representación se pueda traducir automáticamente a un lenguaje de modelado, con el fin de continuar con el proceso de elaboración de la aplicación de *software*.

La solución que se propone en este artículo integra esos elementos requeridos de la siguiente manera:

El punto de partida es una representación del discurso en esquemas preconceptuales [10], que los interesados pueden entender y validar.

El mecanismo de traducción se basa en reglas heurísticas de transformación hacia la representación en UML planteada por Tabares et al. [6]. De esta manera, se garantiza la trazabilidad hacia etapas posteriores del desarrollo de la aplicación y se pueden evitar errores de transformación de la información en esas etapas.

Para realizar la representación de aspectos en esquemas preconceptuales, en este artículo se propone un elemento adicional que no estaba presente hasta ahora en la definición de dichos esquemas. Este elemento, que se denomina "marco", se puede apreciar en la figura 3(a). Los marcos se requieren para agrupar diferentes elementos de los esquemas preconceptuales o, incluso, para guardar esquemas preconceptuales completos en su interior. Además, es necesario modificar el uso de las notas, con el fin de que se puedan unir a los marcos, y modificar levemente su sintaxis, para permitir la representación de restricciones en las notas. Para ello se utilizan los símbolos "{" y "}" en las notas, para representar que el contenido entre llaves se trata de una restricción, como se muestra en la figura 3(b). Con estos nuevos elementos definidos se pueden definir las reglas que permitirán la traducción a UML.

Los aspectos son comportamientos transversales a una aplicación de software y pueden provenir de requisitos funcionales y no funcionales. En relación con los requisitos funcionales, que son aquellos que se ligan con los procesos de la organización, la regla de transformación se define así: las relaciones dinámicas del esquema preconceptual que posean el mismo nombre se pueden agrupar bajo un mismo aspecto, generando en el diagrama de clases una clase con el estereotipo <<Aspect>>, cuyo nombre es el mismo de la relación dinámica, y con una operación estereotipada <<Point Cut>> con el mismo nombre; además, se generan asociaciones estereotipadas <<crosscut>> con cada una de las clases que posean la operación con el nombre de la relación dinámica; finalmente, se genera una nota UML en el diagrama de secuencias, con el mismo contenido de la clase y con un vínculo estereotipado <<crosscut>> con cada una de las apariciones del mensaje con el nombre de la relación dinámica. Las relaciones dinámicas que no se repiten se convierten en operaciones del diagrama de clases y mensajes del diagrama de secuencias, tal como se presenta en Zapata et al. [10], pero no se generan los elementos ligados con los aspectos.

Para los requisitos no funcionales, ligados con características de calidad como el rendimiento y la seguridad, la regla de transformación se define así: La restricción ligada con un marco en el esquema preconceptual genera, en el diagrama de clases, una clase con el estereotipo <<Aspect>>, con el nombre de la primera relación dinámica del marco, y una operación estereotipada <<Point cut>>, con ese mismo nombre; además, genera una nota

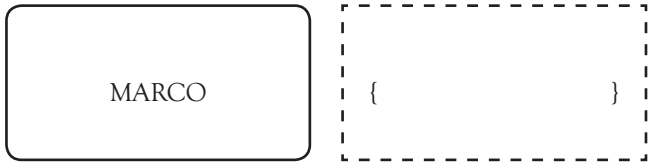


**Figura 3:** Elementos nuevos en los esquemas preconceptuales: (a) Marcos y (b) Llaves en las notas para indicar restricciones.

Fuente: Elaboración propia.

UML ligada con la clase anterior que incluye el texto de la restricción del esquema preconceptual y asociaciones estereotipadas <<crosscut>> (entre la clase <<Aspect>> y las clases que se producen a partir de conceptos de llegada de las relaciones dinámicas incluidas en el marco); finalmente, se genera una nota UML en el diagrama de secuencias, con el mismo contenido de la clase <<Aspect>> (incluyendo el contenido de la restricción) y con un vínculo estereotipado <<crosscut>> con cada uno de los mensajes, que poseen el nombre de las relaciones dinámicas incluidas en el marco. La clase <<Aspect>> y la nota UML generadas serán únicas, así la restricción se ligue con varios marcos a la vez.

## 4 CASO DE ESTUDIO

En una compañía, el interesado reveló durante la entrevista alguna información básica para la construcción de una aplicación de nómina. La información fue la siguiente: la asistente de personal es la persona encargada de liquidar la nómina de la compañía; este proceso no debería tardar más de 15 minutos en total; el jefe de producción debe reportar las novedades en la planta, mientras que el jefe de personal reporta las cuotas que el trabajador paga para reducir el saldo de los préstamos y registra al trabajador.

Por efectos de espacio se omitió una mayor cantidad de información del esquema preconceptual, con el fin de enfocar el problema en lo relevante para la representación de aspectos. Esta información en UN-Lencep tiene la siguiente apariencia: asistente de personal liquida nómina, jefe de producción reporta novedad, jefe de personal reporta cuota; préstamo tiene cuota y saldo, trabajador tiene préstamo, nómina y novedad.

Por ser elementos nuevos, la información ligada con los marcos y las restricciones todavía no posee equivalencias en UN-Lencep. A modo de propuesta, podría ser algo como: la restricción "tiempo liquidación < 15 minutos" afecta el proceso "asistente de personal liquida nómina". Así, el proceso que la restricción afecta es el que debe aparecer dentro del marco en el esquema preconceptual. La figura 4 muestra el esquema preconceptual resultante de este discurso en UN-Lencep. Las figuras 5 y 6 muestran el diagrama de clases y el diagrama de secuencias resultantes del esquema preconceptual de la figura 4.

## 5 CONCLUSIONES Y TRABAJO FUTURO

En este artículo se propuso una manera de representar, en esquemas preconceptuales, la información del dominio que puede derivar aspectos para el desarrollo de *software*. Para elaborar esta propuesta fue necesario crear un

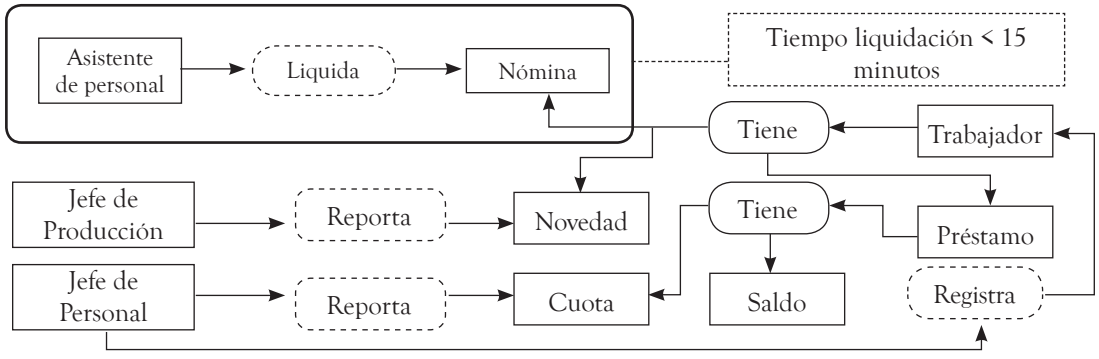


**Figura 4:** Esquema preconceptual para el caso de estudio.

Fuente: Elaboración propia.

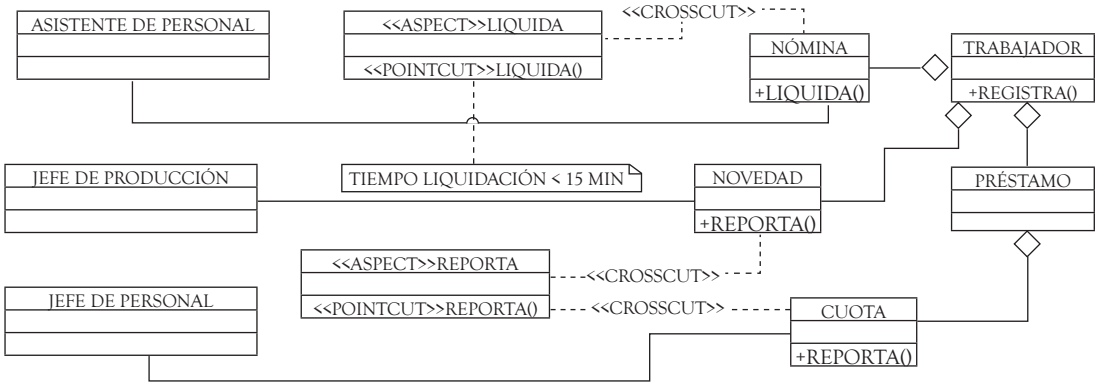


**Figura 5:** Diagrama de clases incluyendo aspectos para el caso de estudio.
Fuente: Elaboración propia.

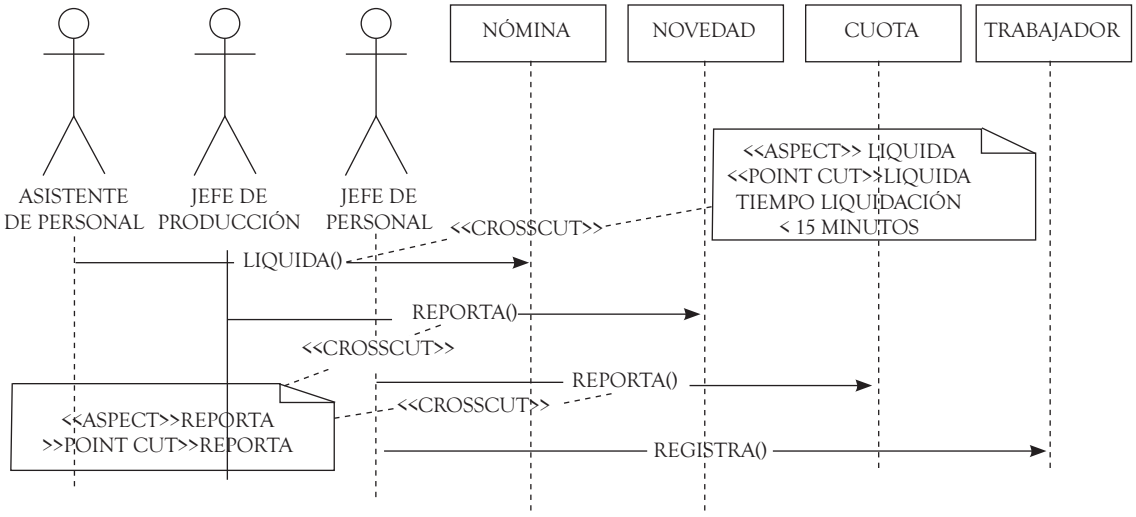


**Figura 6:** Diagrama de secuencias incluyendo aspectos para el caso de estudio.
Fuente: Elaboración propia.

elemento nuevo para los esquemas preconceptuales (el marco) y modificar la sintaxis del elemento nota para que pudiera aceptar restricciones. Las reglas de transformación que poseen los esquemas preconceptuales hacia los diagramas de UML garantizan la consistencia de los diferentes diagramas. Además, las reglas definidas para la transformación de la información aspectual desde esquemas preconceptuales hasta UML permiten seguir manejando adecuadamente la consistencia y se constituyen en una herramienta de trazabilidad, que permita ubicar la información en los diferentes diagramas que se generan.

Como líneas de trabajo futuro, se cuenta con las siguientes:

Explotar aún más los mecanismos de extensibilidad de UML para lograr representaciones más cercanas al código. En este sentido, es importante involucrar los conceptos de MDA (*Model-driven Architecture*) para lograr transformaciones que se vayan acercando paulatinamente al código fuente.

Determinar las reglas heurísticas de transfor-

mación a otros diagramas de UML que permitan el diseño de la aplicación. Diagramas como despliegue, componentes, paquetes y estructura compuesta, que son los que revelan la estructura del diseño y se encuentran cercanos a la codificación deberían ser los primeros en estudio.

Definir una posible transformación desde UN-Lencep hasta código fuente escrito en lenguajes de orientación aspectual, como AspectJ e HyperJ.

## REFERENCIAS


[1] G. Kiczales *et al.*, "Aspect-Oriented Programming," en Proceedings of the European Conference on Object-Oriented Programming (ECOOP), Finlandia, 1997, pp. 220-242.

[2] L. Chung *et al.*, *Non-Funcional Requirements in software Engineering*, Dordrecht: Kluwer Academic Publisher, 2000, 476 p.

[3] S. Anjum, "Incorporating Non-Functional requirements with UML Models," University of Waterloo, Waterloo, Canadá, 2006.

[4] J. Suzuki, y Y. Yamamoto, "Extending UML with Aspects: aspect support in the design phase," en Lecture Notes Computer Science: Proceedings of the Workshop on Object-Oriented Technology, 1999, pp. 299-300.

[5] I. Krechetov *et al.*, "Towards Integrated Aspect-Oriented Modeling Approach for Software Architecture Design."

[6] M. Tabares *et al.*, "El desarrollo de software orientado a aspectos: un caso práctico para un sistema de ayuda en línea," *Avances en Sistemas e Informatica*, vol. 5, no. 2, pp. 61-68, 2008.

[7] E. Baniassad, y S. Clark, «Theme: An Approach for Aspect-Oriented Analysis and Design.»

[8] D. Sheperd *et al.*, "Using Language Clues to discover Crosscutting Concerns."

[9] F. Losavio *et al.*, "UML Extensions for Aspect Oriented Software Development," *Journal of Object Technology*, vol. 8, no. 5, pp. 105-132, 2009.

[10] C. Zapata *et al.*, "Pre-conceptual Schema: A Conceptual-Graph-Like Knowledge Representation for Requirements Elicitation," en *Advances in Artificial Intelligence*, Berlin: Springer Berlin/Heidelberg, 2006.