UN ENTORNO PARA LA REPRESENTACIÓN DE ASPECTOS EN ESQUEMAS PRECONCEPTUALES

Carlos Mario Zapata Jaramillo*
Guillermo González Calderón**
John Jairo Chaverra Mojica***

Recibido: 09/02/2010
Aceptado: 08/10/2010

RESUMEN

Desde mediados de los 90 se viene dando la tendencia a identificar aspectos desde las etapas más tempranas del desarrollo de software. Las herramientas CASE(*Computer-Aided Software Engineering)* estándar permiten representar los aspectos en diagramas UML(*UnifiedModelingLanguage)* pero, para ello, se requiere que el analista interprete de manera subjetiva el dominio, sin que medie una validación de los interesados. Es por esto que en este artículo se proponeun entorno, basado en EMF (*Eclipse® Modelling Framework*), que permite representar los aspectos en esquemas preconceptuales para luego traducirlos a los diagramas de clases y secuencias. El uso de la herramienta se ejemplifica con un caso de estudio.

Palabras clave: EMF, GMF, MOFScript, aspecto, esquemas preconceptuales.

* Ph. D. en Ingeniería, profesor asociado de la Universidad Nacional de Colombia, líder del grupo de investigación en Lenguajes Computacionales. Facultad de Minas, Escuela de Sistemas. Correo electrónico: cmzapata@unal.edu.co.
** Docente de la Facultad de Ingenierías, Universidad de Medellín, Grupo de Investigación ARKADIUS. Correo electrónico: ggonzalezc@udem.edu.co.
*** Estudiante de maestría en Ingeniería de Sistemas, Universidad Nacional de Colombia. Correo electrónico: jjchaverra@unal.edu.co.

AN ENVIRONMENT FOR REPRESENTING ASPECTS IN PRE-CONCEPTUAL SCHEMAS

ABSTRACT

Since the mid-nineties, there has been a trend for identifying aspects from the earliest stages of software development lifecycle. The standard CASE (Computer-Aided Software Engineering) tools are used to represent aspects in UML (Unified Modeling Language) diagrams. However, the analyst makes this task by interpreting the domain in a subjective way, with no validation from stakeholders. For this reason, in this paper we present an EMF-based environment for representing aspects in pre-conceptual schemas, and then translating them into class and sequence diagrams. We exemplify the use of such environment with a case study.

Key words: EMF, GMF, MOFScript, aspect, pre-conceptual schemas.

INTRODUCCIÓN

La programación orientada a aspectos (POA) tiene como propósito permitir una adecuada modularización de los sistemas y facilitar una mejor separación de conceptos o requisitos. Dichos requisitos pueden tener dos orígenes: funcionales y no funcionales. Generalmente, los aspectos agrupan comportamientos comunes de las operaciones o restricciones asociadas con requisitos no funcionales [1].

Últimamente, se viene dando la tendencia a identificar los aspectos desde etapas más tempranas del desarrollo de software, para que, cuando se llegue a la etapa de implementación, se tengan claros todos los aspectos. Así, se busca hacer más rápido y efectivo el proceso de desarrollo de software, logrando mayor consistencia, trazabilidad y facilidad el mantenimiento futuro en los aplicativos finales [2]. De esta manera, si hubiese cambios en los requisitos del interesado, no serían tan traumáticos.

Para representar los aspectos en diagramas UML existen herramientas CASE pero, para ello, se requiere que el analista interprete de manera subjetiva el dominio [2]. Además, estas herramientas utilizan un lenguaje técnico que el interesado no entiende fácilmente. Por ende, no se tiene una verdadera validación de la aplicación que se está construyendo, sino hasta las etapas finales de desarrollo.

Zapata *et al.*[2] especifican una propuesta para la representación de los aspectos a partir de esquemas preconceptuales. Como son de fácil interpretación para los interesados, estos esquemas solucionan parcialmente el análisis subjetivo del dominio y logran una validación con el interesado desde las fases iniciales del desarrollo. De esta manera, el desarrollo depende menos del analista y se disminuyen los errores que se cometen al elaborar manualmente los diagramas. Además, se mejora la interacción con el interesado, logrando que, a la hora de desarrollar el producto, no se tengan inconvenientes por funcionalidades que el analista no interprete adecuadamente. Sin embargo, esta propuesta aún no se incorpora en las herramientas CASE convencionales. Por ello, en este artículo se propone la implementación de un entorno, basado en EMF y GMF*(Graphical Modelling Framework)*, para la representación de aspectos en esquemas preconceptuales y su traducción a los diagramas de clases y secuencias de UML.

Este artículo se organiza de la siguiente manera: en la sección 2, se define el marco teórico que agrupa los conceptos de este dominio; en la sección 3, se resumen algunos trabajos en representación de aspectos; en la sección 4, se plantean los elementos de la propuesta de implementación en el entorno EMF; en la sección 5, se plantea un caso de estudio para ejemplificar el uso de la herramienta. Las conclusiones y el trabajo futuro se incluyen en las secciones 6 y 7, respectivamente.

1. MARCO TEÓRICO

Se definen, en esta sección, algunos términos que serán útiles para entender la solución.

Un aspecto es la agrupación de requisitos funcionales y no funcionales, que se diseminan por todo el código [3].Algunos de sus conceptos básicos son:

- *Aspect:* Funcionalidad que se cruza a lo largo de la aplicación (cross-cutting) de forma modular y separada del resto del sistema.
- *Join point:* Es un punto de ejecución dentro del sistema, donde un aspecto se puede conectar como una llamada a un método, el lanzamiento de una excepción o la modificación de un campo.
- *Pointcut:* Define los métodos que se aplicarán a cada join point. Se especifica mediante nombres (de clases, métodos, campos) o expresiones regulares, e incluso dinámicamente en tiempo de ejecución

Según Zapata *et al.* [4] los esquemas preconceptuales permiten la representación de la terminolo-

gía de un dominio para facilitar su traducción a diferentes esquemas conceptuales. La descripción de los diferentes elementos que conforman los esquemas preconceptuales se puede ver en Zapata *et al.*[2, 4-6]. Para poder emplear los esquemas preconceptuales en la representación de aspectos es necesario construir un metamodelo, que es un modelo de modelos que incluye definiciones que abarcan los elementos que se emplean gráficamente en un determinado diagrama [7]. Como el metamodelo se puede definir en términos de UML, es necesario contar con un lenguaje para la descripción de restricciones que se aplican al metamodelo, para determinar si existe buena formación en el lenguaje derivable del metamodelo[7]. Para las restricciones se emplea en este artículo OCL *(ObjectConstraintLanguaje).*

El entorno para implementar el desarrollo se compone de lo siguiente:

- EMF (Eclipse® Modelling Framework): es una estructura de modelado que facilita la generación de código, para construir herramientas y otras aplicaciones, basadas en un modelo de datos estructurado desde una especificación del modelo descrito en XMI (XML—Extensible Markap Language—Metadata Interchange) [8].
- GMF (Graphical Modelling Framework): proporciona una herramienta para la generación de editores gráficos basados en EMF. Permite asociar cada nodo especificado en el metamodelo con su respectiva componente gráfica [8].
- MOFScript (Meta-object facilitysript): es un plug-in de Eclipse® que permite definir conjuntos de reglas, en las que se indican cómo realizar transformaciones desde un modelo estructurado a texto. Como modelo de entrada es posible utilizar modelos que soporta el plug-in EMF o personalizados. En el contexto de este artículo, este plug-in se utilizará para definir las reglas de transformación a código Java y Aspecto (uno de los principales lenguajes orientados a aspectos) a partir de esquemas preconceptuales [8].

2. ANTECEDENTES

Las herramientas CASE convencionales (por ejemplo, ArgoUML [9] y Eclipse®) y algunas meta-CASE (como AToM3® [10]) se pueden usar para la elaboración de diagramas y su posterior traducción a diferentes lenguajes de programación. Sin embargo, en estos casos la elaboración de los diagramas corre a cargo del analista, sin que los interesados puedan validar los diagramas. Además, en general, estas herramientas no posibilitan la traducción a lenguajes orientados a aspectos, como el AspectJ. Otros trabajos se encargan de la representación de aspectos, tales como:

- Bennett et al. [11] presentan un framework para la generación automática de código orientado a aspectos a partir del diagrama de clases. Los autores proponen la representación de los aspectos como una clase. Para generar el código se analiza el XML que el framework genera.
- Baschy Sánchez [3] presentan una propuesta para la incorporación de aspectos en UML para el diagrama de clases. En la figura 1 se presenta un diagrama de clases con aspectos, en donde incorporan un nuevo elemento donde se representa el aspecto C, siendo C una operación común en la clase A y B.

Si bien en la primera propuesta se genera código orientado a aspectos, en ambas propuestas es el analista quien debe elaborar los diagramas en su lenguaje técnico, basado en UML, lo cual sigue impidiendo la validación que pueden hacer los interesados.

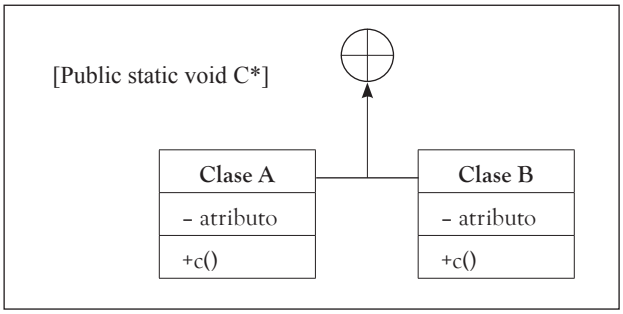


Figura 1. Diagrama de clases con aspectos
Fuente: elaboración propia.

3. ESQUEMAS PRECONCEPTUALES PARA LA REPRESENTACIÓN DE ASPECTOS EN EL ENTORNO EMF

Para construir una herramienta CASE en el entorno EMF y GMF es necesario definir el metamodelo del dominio, en este caso del esquema preconceptual, que se debió construir en el marco de este artículo. En la figura 2 se presenta el metamodelo del esquema preconceptual, en la notación propia del diagrama de clases. En este trabajo se utiliza OCL para garantizarle al usuario que su esquema preconceptual es, sintácticamente, correcto. OCL permite definir restricciones de conexión entre los diferentes elementos del metamodelo. En la figura 3 se ilustra la forma de utilizar OCL en EMF. Inicialmente, se define un *link* de conexión entre dos nodos, luego se limita a que la conexión parta de una "Especificación" y finalice en un "Concepto". De esta manera, se definen todas las conexiones que sean necesarias, siguiendo los lineamientos propios de la sintaxis del esquema preconceptual. Una vez que el usuario grafique su esquema preconceptual, se obtiene un XMI que contiene toda la información de los elementos definidos. En la figura 4 se presenta la estructura en XMI de un ejemplo.

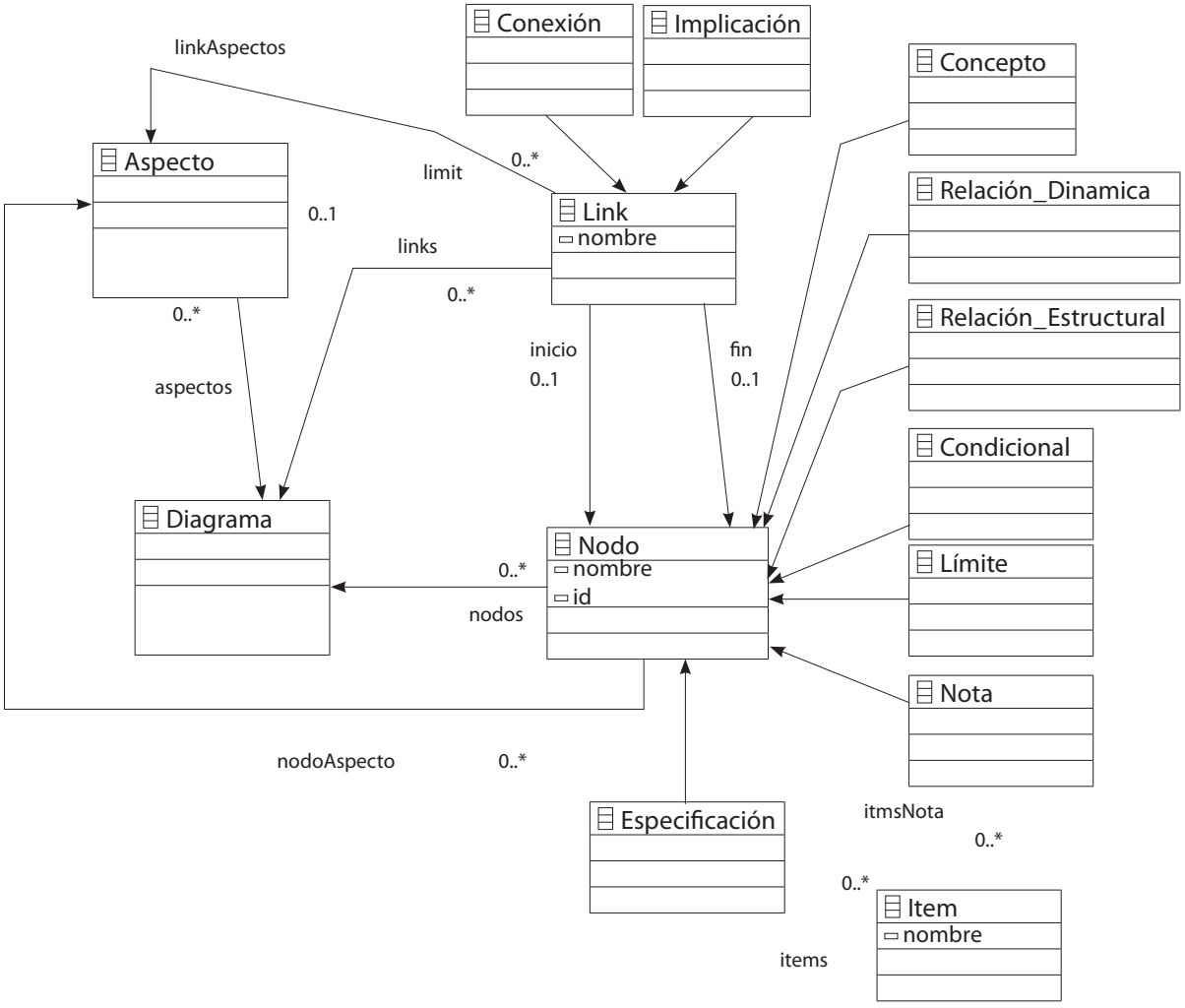


Figura 2. Metamodelo del esquema preconceptual

Fuente: elaboración propia.

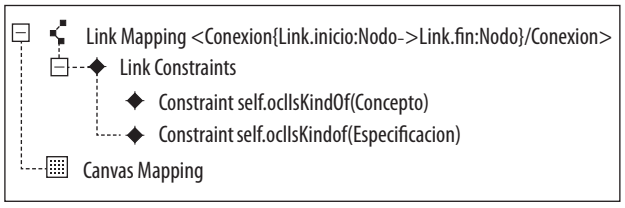


Figura 3. Incorporación de OCL en EMF

Fuente: elaboración propia.

Para realizar la traducción del esquema preconceptual a diagramas UML y código fuente Java y AspectJ se utiliza MOFScript. Éste analiza el XMI que contiene toda la información del esquema preconceptual que el usuario grafique. En la figura 5 se ilustra la manera de recolectar la información de los nodos o conceptos contenidos en el XMI.

Una vez almacenada la información de los conceptos, se ejecuta el proceso para la generación de código o de los diagramas UML que se requieran. Las reglas para obtener los diagramas de clases y secuencias se basan en las que definen Zapata *et al.* [4, 6], para los elementos básicos de estos diagramas y en las reglas que se definen para incorporar los aspectos en dichos diagramas [2]. Las reglas para la traducción a AspectJ y Java se generan a partir de los trabajos previos [3, 11]. En la figura 6 se ilustra la forma de recorrer los "conceptos" almacenados y la manera de escribir los archivos correspondientes.

4. CASO DE ESTUDIO

Con el fin de ejemplificar las reglas definidas en Zapata *et al.* [2], se presenta un caso de estudio de una pizzería, tomado de Arango y Zapata [12] y adaptado para este artículo:"Esta pizzería cuenta con un despachador, tres chefs y siete repartidores que realizan las diferentes funciones necesarias para la satisfacción de las necesidades de los clientes. El cliente llama a la pizzería y realiza el pedido. Una vez se le toma el pedido, el repartidor debe entregar el producto en máximo 30 minutos. Cuando el cliente recibe la pizza, realiza el pago

```
<?xml version="1.0" encoding="UTF-8"?>
<Preconceptual: Diagrama xmi: version="2.0" xmlns: xmi=" http://www.omg.org/XMI"
xmlns:xsi="http://www3.org/2001/XMLSchema-instance"
xmlns:Preconceptual="Preconceptual">
<links xsi:type="Preconceptual:Conexión" inicio=//@nodos.0 fin="//@nodos.1"/>
<links X3i:type="Preconceptual:Conexión" inicio="//@nodos.1" fin="//@nodos.2"/>
<nodos xsi:type="Preconceptual:Concepto" nombre="Persona"/>
<nodos xsi: type=" Preconceptual:Estructural nombre="tiene"/>
<nodos xsi:type="Preconceptual:Concepto" nombre="Telefono"/>
```

Figura 4. Estructura XMI de un esquema preconceptual

Fuente: elaboración propia.

```
texttransformation Preconceptual(in modelo:"Preconceptual") {
    property conceptos:Hashtable = "
    modelo.Diagrama::main() {
        self.nodos->forEach(c:modelo.Concepto) {
            conceptos.put(c.id,c.nombre)
        }
    }
}
```

Figura 5. Código MOFScript para recolectar la información de los nodos contenidos en un esquema preconceptual.

Fuente: elaboración propia.

```
var keyss:List = clases.keys()
keyss->forEach(id){
    var nombre:String = conceptos.get(id)
    file (nombre '.java')
    "package " + package_nombre + ";"
    nl(l)
    'public class '   nombre '{'
        nl(l)
    '} '
}
```

Figura 6. Código MOFScript para la generación de clases Java a partir de esquemas preconceptuales

Fuente: elaboración propia.

correspondiente. Luego, el repartidor regresa a la pizzería y el despachador registra el pago. La pizzería maneja un control sobre sus empleados, para ellos tiene todos los datos personales de cada uno, como son: nombre, dirección, teléfono y cédula".

La figura 7 representa el esquema preconceptual para este caso de estudio.

En las tablas 1 y 2 se muestra la forma de aplicar las reglas definidas en Zapata *et al.* [2] para la identificación de aspectos y su traducción a los diagramas UML, código Java y AspectJ. Nótese que los diagramas y el código resultante son consistentes entre sí; por ejemplo, en el diagrama de clases existe una clase llamada "*Persona*" con atributos: *nombre, dirección, teléfono y cédula*. Esta misma información se ve reflejada en el código fuente. De esta manera, el analista no tiene que preocuparse por la consistencia y la trazabilidad entre los diferentes diagramas y el código fuente.

5. CONCLUSIONES

En este artículo se presentó una herramienta CASE elaborada en el entorno EMF, que emplea GMF y que permite graficar el esquema preconcep-

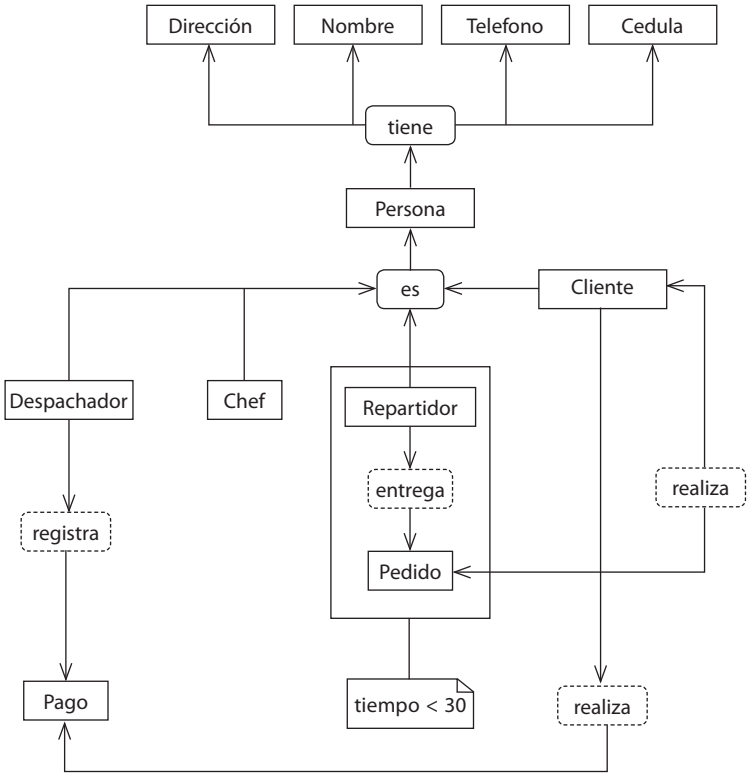
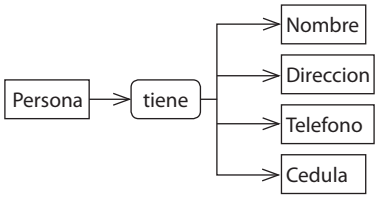
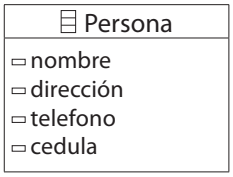
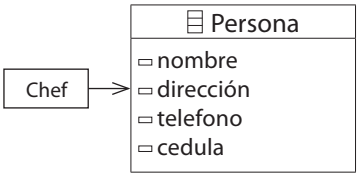
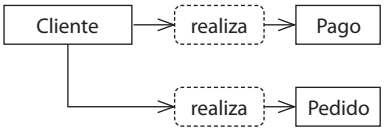
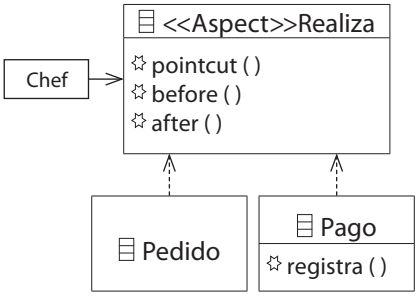
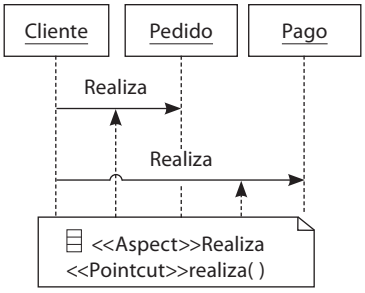
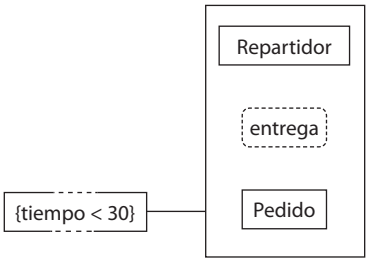
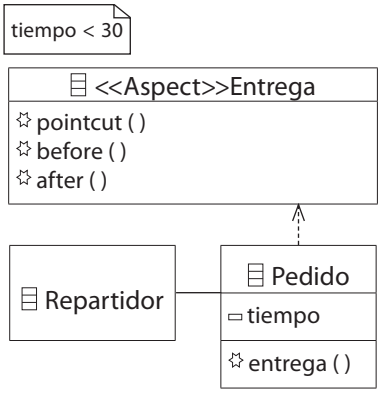
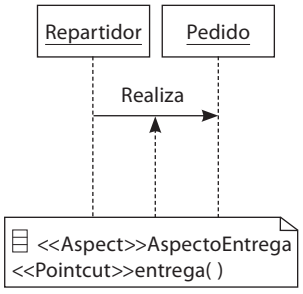


Figura 7. Esquema preconceptual para el caso de estudio

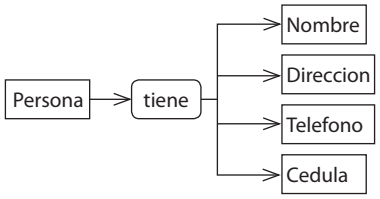
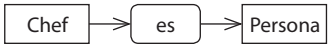
Fuente: elaboración propia.

Tabla 1. Esquema preconceptual a diagrama de clases y secuencias

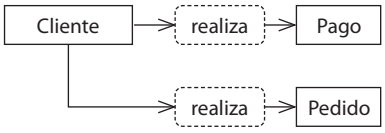
Precondición	Diagrama de clases	D. de secuencias
		No tiene representación asociada
		No tiene representación asociada
		
		
		

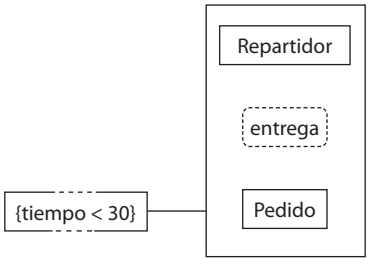
Fuente: elaboración propia.

Tabla 2. Esquema Preconceptual a código Java y AspectJ

Precondición	Código Java o Aspecto
	public class Persona' private String nombre; private String direccion; private String telefono; private String cedula;
	public class Chef extends Persona } public Chef( ) { } }

	public class Pago {
	public Pago( ) { }
	public void registra( ) { }
	}

	public aspect Realiza {
	private Pago pago = new Pago( );
	private Pedido pedido = new Pedido( );
	after (Pedido p) : target(p)
	&& call(void pedido.realiza( )) {
	}
	after (Pago p): target(p)
	&& call(void pago.realiza( )) {
	}
	after (Pedido c) : endTiming (c) { }
	after (Pago c) : endTiming (c) { }

	public aspect Entrega {
	private Pedido pedido = new Pedido( );
	after (Pedido p) : target (p) &&
	call(void pedido.entrega( )) {
	}
	after (Pedido c) : endTiming (c) {
	pedido.tiempo <= 30
	}
	}

Fuente: elaboración propia.

tual de un dominio, generar automáticamente dos diagramas UML (clases y secuencias) que incluyen la representación de aspectos y generar código fuente en Java y AspectJ. Los principales aportes son:

- El interesado no necesita tener previo conocimiento de las reglas para obtener los diagramas UML y el código en Java y AspectJ.
- Se aumenta la participación del interesado en el proceso del desarrollo de Software. De esta manera se tiene una validación en fases tempranas del ciclo de vida de una aplicación.
- Se reduce el trabajo del analista y se disminuye el tiempo de desarrollo.

- Se mantiene consistencia entre los diagramas resultantes.

6. TRABAJO FUTURO

Las líneas de futuro desarrollo que se pueden generar de este trabajo son:

- Lograr que el prototipo permita la edición de los diagramas UML resultantes.
- Definir tipos de datos en las relaciones dinámicas y estructurales para mayor completitud cuando se genere el código.
- Identificar reglas para la generación automática de casos de uso con aspectos.
- Definir nuevas reglas en la generación de aspectos para requisitos no funcionales.

AGRADECIMIENTO

Este trabajo se realizó en el marco del proyecto de investigación "Transformación semiautomática de los esquemas conceptuales, generados en uncdiagramador, en prototipos funcionales", financiado por la Vicerrectoría de Investigación de la Universidad Nacional de Colombia.

REFERENCIAS

[1] M. Tabares *et al.*, "La ingeniería de requisitos orientada a aspectos: una experiencia de aplicación en un sistema de ayuda en línea," *Dyna*, vol. 74, no. 153, pp. 285-289, 2008.

[2] C. Zapata *et al.*, "Representación de aspectos candidatos en esquemas preconceptuales," *Revista Ingenierías Universidad de Medellín*, vol. 9, no. 16, pp. 123-131, 2010.

[3] M. Basch, y A. Sánchez, "Inorporating aspects into the UML," presentado a 3rd Workshop on Aspect-Oriented modeling with UML, Boston, 2003.

[4] C. Zapata *et al.*, "Pre-conceptual schema: a conceptual-graph-like knowledge representation for requierements elicitation," *Lecture Notes in Computer Science*, vol. 4293, pp. 17-27, 2006.

[5] C. Zapata, y D. Cardona, "Implementación en C# de las reglas heurísticas de conversión de esquemas preconceptuales a diagramas UML 2.0," *Revista Facultad de Ingeniería Universidad de Antioquia*, no. 44, pp. 119-136, 2008.

[6] C. Zapata, y G. Garcés, "Generación del diagrama de secuencias de UML 2.1.1 desde esquemas preconceptuales," *Revista EIA*, no. 10, pp. 89-103, 2008.

[7] OMG. "Unified modeling language specification. Version 2.1.1," septiembre 01, 2009; http://www.omg.org/uml/.

[8] B. Moore *et al.*, "Eclipse development using the Graphical Editing Framework and the Eclipse Modeling Framework," IBM International Technical Support Organization, 2004.

[9] ArgoUML. agosto 25, 2009; http://argouml.tigris.org/documentation.

[10] J. De Lara, y H. Vangheluwe, "AToM3: a tool for multi-formalism and meta-modeling," presentado a Proceedings of the Fifth International Conference on fundamental approaches to software engineering, Londres, 2002, pp. 174-188.

[11] J. Bennet *et al.*, "Aspect-oriented model-driven skeleton code generation: A graph-based transformation approach," *Science of Computer Programming*, vol. 75, no. 8, pp. 689-725, 2009.

[12] F. Arango, y C. Zapata, *Un método para la elicitación de requisitos de software*, Medellín: Universidad Nacional de Colombia, 2006, pp. 113.