Revista EIA, ISSN 1794-1237 Número 10, p. 89-103. Diciembre 2008
Escuela de Ingeniería de Antioquia, Medellín (Colombia)

GENERACIÓN DEL DIAGRAMA DE SECUENCIAS DE UML 2.1.1 DESDE ESQUEMAS PRECONCEPTUALES

Carlos Mario Zapata*
Gilma Liliana Garcés**

RESUMEN

El diagrama de secuencias es un esquema conceptual que permite representar el comportamiento de un sistema, para lo cual emplea la especificación de los objetos que se encuentran en un escenario y la secuencia de mensajes intercambiados entre ellos, con el fin de llevar a cabo una transacción del sistema. Existen diferentes enfoques que buscan la generación automática de modelos conceptuales, como el diagrama de secuencias. Algunos trabajos parten del lenguaje natural, pero generan diagramas diferentes al de secuencias o, si lo hacen igual, dejan de lado elementos como los fragmentos combinados, que describen ciertas condiciones lógicas en el sistema. Otros trabajos parten del código fuente, el cual se suele ubicar en una fase más avanzada del ciclo de vida del software. En este artículo se define un método, basado en reglas heurísticas, que permite identificar los elementos del diagrama de secuencias, incluyendo los fragmentos combinados, tomando como punto de partida los esquemas preconceptuales. Se realiza la implementación de las reglas en la herramienta AToM^3 aplicándolas a un caso de estudio.

PALABRAS CLAVE: modelo conceptual; diagrama de secuencias; fragmento combinado; esquema preconceptual, AToM^3.

GENERATION OF UML 2.1.1 SEQUENCE DIAGRAM FROM PRE-CONCEPTUAL SCHEMES

ABSTRACT

Sequence diagram is a conceptual schema for representing behavior of a system. For performing such a task, it employs the object spec from a scenario and the sequence of messages exchanged among the objects. These elements describe a transaction of the system. Several approaches try the automated generation of conceptual

* Ingeniero Civil, Especialista en Gerencia de Sistemas Informáticos, Magíster en Ingeniería de Sistemas y Doctor en Ingeniería con énfasis en Sistemas, Universidad Nacional de Colombia. Profesor Asociado, Universidad Nacional de Colombia, Sede Medellín. Líder del Grupo de Investigación en Lenguajes Computacionales. cmzapata@unal.edu.co

** Ingeniera de Sistemas e Informática. Magíster en Ingeniería de Sistemas. Asistente de Investigación del Grupo de Investigación en Lenguajes Computacionales, Universidad Nacional de Colombia. glgarces@unalmed.edu.co

Artículo recibido 29-IX-2008. Aprobado 17-XII-2008
Discusión abierta hasta junio de 2009

models (like sequence diagram). Some of them use natural language as a starting point, but they are focused on other diagrams. Some others are focused on sequence diagram, but they do not obtain elements like combined fragments describing several logical constraints of the system. Other approaches use source code as a starting point, but source code can be related to an advanced phase of the software development life cycle. In this paper we define a method based on heuristic rules for obtaining automatically the elements of the sequence diagram (including combined fragments) from pre-conceptual schemas. These heuristic rules are implemented in the AToM^3 tool and applied in a case study.

KEY WORDS: conceptual model; sequence diagram; combined fragment; pre-conceptual schema; AToM^3.

1. INTRODUCCIÓN

En un proceso de desarrollo de software, una de las tareas más relevantes para el éxito de un proyecto de diseño e implementación de un sistema informático es garantizar, de una forma apropiada, la representación y el modelado de los requisitos de usuario. El planteamiento de objetivos confusos, los tiempos de entrega tardíos y las especificaciones y requisitos incompletos (Zapata y Arango, 2005) apuntan a que se debe enfatizar en el desarrollo de la fase de requisitos, de forma tal que el sistema obtenido satisfaga las necesidades del interesado. Durante la fase de análisis, específicamente, los modelos dinámicos adquieren mayor importancia. Estos se utilizan para especificar patrones de interacción entre objetos o instancias de clases (Díaz *et al.*, 2005) y describen la secuencia ordenada de los mensajes que envían y reciben las instancias durante la ejecución del escenario de un caso de uso (que se define como una descripción parcial del comportamiento de un sistema en una transacción particular). Para la representación de dichos patrones, la bibliografía propone, entre otros, los diagramas de secuencias y de comunicación (OMG, 2007).

El diagrama de secuencias, que se define en UML *(Unified Modeling Language),* es uno de los más utilizados para identificar el comportamiento de un sistema (Pressman, 2005), por representar los objetos que se encuentran en el escenario y la secuencia de mensajes intercambiados entre los objetos para llevar a cabo la funcionalidad descrita por una transacción del sistema. Además, se utiliza con frecuencia para validar los casos de uso y apreciar la lógica del diseño de forma dinámica (Ambler, 2005; Fowler y Scott, 1997 y 2003).

Existen diversos proyectos y herramientas que buscan facilitar la extracción de la información necesaria para la generación automática de esquemas conceptuales (incluyendo el diagrama de secuencias), con el fin de agilizar el desarrollo de las aplicaciones de software. Los diferentes enfoques se pueden clasificar en dos puntos de partida: especificaciones textuales en lenguaje natural y código fuente. Estos proyectos y herramientas, sin embargo, aún presentan algunas fallas por mejorar:

- Algunos se enfocan en diagramas diferentes al de secuencias, lo cual deja de lado las particularidades de ciertas restricciones del sistema que sólo se pueden representar en dicho diagrama.
- Los que se enfocan en el diagrama de secuencias sólo obtienen los elementos básicos, dejando de lado importantes elementos que permiten expresar condiciones especiales y apreciar la lógica del diseño de forma dinámica (como es el caso de los fragmentos combinados).
- Los que parten del código fuente se convierten en herramientas interesantes para revisar el diagrama de secuencias *a posteriori,* es decir, en una fase más avanzada del ciclo de vida del software, y como tales permiten la realización de ingeniería inversa. Sin embargo, si se pretende agilizar el desarrollo de software es

preferible identificar el diagrama de secuencias desde las fases iniciales del desarrollo y no esperar a la implementación del código fuente para su obtención.

En este artículo se parte de una representación gráfica de las especificaciones textuales de un sistema, mediante los denominados esquemas preconceptuales (Zapata, Gelbukh y Arango, 2006a; Zapata, 2007) y, mediante la definición de un conjunto de reglas heurísticas, se pretende la obtención de los elementos del diagrama de secuencias, incluyendo los fragmentos combinados. El método así definido se implementa en la herramienta AToM^3 *(A Tool for Multi-Formalism Modeling and Meta-Modeling)* y se ejemplifica mediante un caso de estudio cuyo enunciado proviene de la literatura especializada en el tema.

Para lograr este objetivo, en la sección 2 se presenta el cuerpo teórico necesario para este desarrollo, que incluye una descripción de los principales elementos que componen un diagrama de secuencias según el estándar UML 2.1.1, además de los esquemas preconceptuales y la herramienta AToM^3; en la sección 3, se presentan algunas propuestas en torno a la generación automática del diagrama de secuencias y otros esquemas conceptuales, resaltando sus principales aportes y desventajas; en la sección 4, se presentan las reglas de conversión entre el esquema preconceptual y algunos elementos del diagrama de secuencias UML 2.1.1, incluyendo los fragmentos combinados; en la sección 5, se presenta el caso de estudio, y en la sección 6, las principales conclusiones obtenidas y el trabajo futuro que se deriva de esta propuesta.

2. MARCO TEÓRICO

2.1 Principales elementos del diagrama de secuencias

El diagrama de secuencias hace parte de los diagramas de interacción de la especificación UML 2.1.1 que describen los aspectos dinámicos de un sistema y muestran la interacción entre los objetos de un sistema y los mensajes enviados entre ellos, ordenados según su secuencia en el tiempo.

Los diagramas de secuencias son útiles para diversos usos (Ambler, 2005; Fowler y Scott, 1997 y 2003):

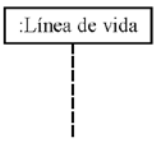
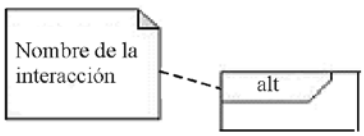
- *El modelado de escenarios de uso*. Un escenario de uso es una descripción de una posible forma en que un sistema se utiliza. La lógica de un escenario de uso puede ser parte de un caso de uso, por ejemplo, una secuencia alternativa o un paso completo a través de un caso de uso, tal como la lógica que describe la secuencia normal de la acción o una parte de ella. Un escenario de uso también puede ser un paso a través de la lógica contenida en varios casos de uso.
- *El modelado de la lógica de los métodos*. Los diagramas de secuencias se pueden utilizar para explorar la lógica de una operación, función o procedimiento complejos, ya que ofrece una forma de observar las invocaciones a las operaciones definidas en las clases.
- *La detección de cuellos de botella en un diseño orientado a objetos*. Al observar los mensajes enviados a un objeto y cuánto se tardan en ejecutar el método invocado, es posible concluir que es necesario cambiar el diseño con el fin de distribuir la carga dentro del sistema.

Los principales elementos del diagrama de secuencias especificados por Fowler y Scott (1997 y 2003) para la versión 2.0 y que siguen siendo válidos en la Superestructura UML 2.1.1 (OMG, 2007) se pueden apreciar en la tabla 1.

La semántica de los fragmentos combinados (FC) depende del operador de interacción, que contiene un cierto número de operandos y un identificador. Mediante los operadores, se pueden definir, entre otros:

- *Alternativa* (denotado "alt"), que modela estructuras if…then…else.

Tabla 1. Principales elementos del diagrama de secuencias

Tipo de Nodo	Notación	Descripción
Marco		Provee un borde visual para el diagrama de secuencias
Línea de Vida		Representa un participante individual en una interacción
Actor		Representa el papel desempeñado por un usuario
Mensaje		Define una comunicación particular entre líneas de vida de una interacción
Fragmento combinado		Describe una interacción reutilizable

- *Opción* (denotado "opt"), que modela estructuras switch. Una opción es semánticamente equivalente a un fragmento combinado alternativo donde hay un operando con contenido vacío y otro no vacío.
- *Quiebre de secuencia* (denotado por "break"), que modela una secuencia alternativa de eventos, que se procesa en lugar de todo el resto del diagrama.
- *Paralelo* (denotado por "par"), que modela procesos concurrentes.
- *Ciclo* (denotado por "loop"), el cual incluye una serie de mensajes que se repiten.

2.2 Esquemas preconceptuales

Un modelo verbal es una representación de los requisitos del interesado y un medio de comunicación con el analista, cuya finalidad es describir las necesidades y problemas de un sistema. Sin embargo, por estar escrita en lenguaje natural, dicha descripción resulta ser compleja, vaga o ambigua. Por esta razón, se hace necesaria la definición de un modelo que permita expresar los requisitos del interesado de una forma clara, sin sacrificar la riqueza que el lenguaje natural ofrece. En este contexto, nacen los EP (esquemas preconceptuales) que, según Zapata, Gelbukh y Arango, (2006b), utilizan una notación gráfica para la expresión de los diferentes elementos del discurso de un interesado y constituyen un paso intermedio en la obtención automática de los diagramas de UML a partir de un lenguaje controlado. La sintaxis básica de los esquemas preconceptuales se muestra en la figura 1 y su explicación es la siguiente: los conceptos son sustantivos o sintagmas nominales del discurso del interesado, las instancias son conjuntos de valores que puede tomar un concepto y que sirven para aclararlo (se unen al concepto que las origina mediante una línea discontinua), las relaciones estructurales son relaciones permanentes entre los conceptos (asociadas con los verbos "es" y

"tiene"), las relaciones dinámicas se asocian con los denominados "verbos de actividad" (que generan relaciones de tipo temporal entre los conceptos), las implicaciones sirven para unir relaciones dinámicas o para unir condicionales con relaciones dinámicas (estableciendo entre ellas una relación causa-efecto), los condicionales son relaciones de causalidad que indican las restricciones o reglas del negocio que se deben cumplir y las conexiones permiten enlazar los conceptos con las relaciones y las relaciones con los conceptos.

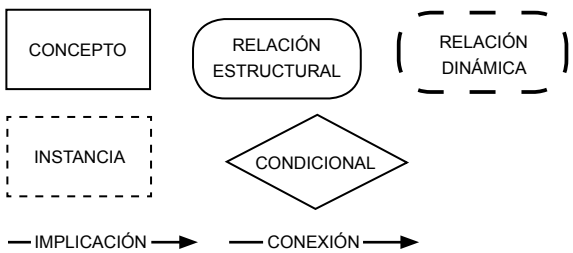


Figura 1. Elementos de los esquemas preconceptuales

2.3 Metamodelador AToM³

Las dos tareas principales de AToM³ son el metamodelado y la transformación de modelos. El primero se refiere a la descripción o modelado de diversas clases de formalismos usados para modelar los sistemas. La transformación de modelos se refiere al proceso automático de convertir, traducir o modificar un modelo, que se encuentra en un formalismo dado en otro modelo, que puede o no estar en el mismo formalismo.

AToM³ tiene una interfaz gráfica que facilita la construcción de las especificaciones de un diagrama de manera similar a como se elaboran las instancias de un diagrama en una herramienta CASE *(Computer-Aided Software Engineering)* convencional. Esta especificación se puede usar posteriormente en la materialización de instancias que representen el dominio de un problema particular (De Lara y Vangheluwe, 2002).

Aparte de las ventajas ofrecidas por AToM³, esta herramienta se seleccionó para elaborar el prototipo del método descrito en este artículo por las siguientes razones (Torres, 1993):

- Es posible construir formalismos de modelos desde cero, lo cual implica la posibilidad de correcciones (adición o borrado de elementos, por ejemplo) en modelos que cambian constantemente de versión o en otros que presentan con frecuencia modificaciones sutiles. Así, para la conversión de que se ocupa el presente artículo, cabe mencionar los fragmentos combinados, adicionados en la versión UML 2.1.1 en el diagrama de secuencias y que pocas herramientas CASE consideran para la elaboración de estos diagramas.
- Es posible expresar gráficamente el formalismo de los diferentes modelos, lo que reduce la complejidad para su comprensión.
- Es posible crear instancias de los modelos y verificar las restricciones planteadas en el metamodelo sobre esas instancias en particular.

Adicionalmente, AToM³ permite expresar ciertas restricciones en términos de gramática de grafos, que puede combinar la expresión gráfica con la textual, en forma de precondiciones y postcondiciones que se pueden establecer en el lenguaje Python (De Lara, Guerra y Vangheluwe, 2004).

La gramática de grafos se define como un conjunto de reglas que poseen un lado izquierdo *(left-hand side* o LHS*)*, que contiene las precondiciones que se deben cumplir para activar una determinada regla, y un lado derecho *(right-hand side* o RHS*)*, que contiene el grafo que reemplazará el que equipare el lado izquierdo de la regla (De Lara, Vangheluwe y Alfonseca, 2003). De esta manera, es posible expresar restricciones que permitan transformar modelos por medio de los metamodelos definidos para estos, las precondiciones y postcondiciones y las reglas que activan las transformaciones. La gramática de grafos de AToM³ posee también un mecanismo que va

reescribiendo el modelo a medida que las diferentes reglas se van activando hasta que no haya reglas que se puedan ejecutar.

3. GENERACIÓN AUTOMÁTICA DE DIAGRAMAS DE SECUENCIAS

En la literatura se presentan diferentes propuestas para abordar la generación automática de esquemas conceptuales. Por un lado, se encuentran propuestas que parten del lenguaje natural o de alguna representación intermedia de él; por otro lado, están aquellas que emplean el código fuente como punto de partida.

3.1 Métodos que parten del lenguaje natural o de una representación intermedia

En este grupo de propuestas existen métodos que permiten generar de manera semiautomática el diagrama de clases de UML a partir de especificaciones textuales de requisitos, como lo hace NL-OOPS *(Natural Language Object-Oriented Product System,* Mich, 1996; Mich y Garigliano, 2002), CM-Builder (*Conceptual Model Builder,* Harmain y Gaizauskas, 2000) y LIDA (*Linguistic Assistant for Domain Analysis,* Overmyer, Lavoie y Rambow, 2001). Todos ellos comparten las mismas limitaciones: el proceso es, por lo general, semiautomático y, a pesar de que podrían identificar algunos de los elementos dinámicos que hacen parte de los diagramas de secuencias, no lo hacen y se limitan a la sintaxis del diagrama de clases.

Otra propuesta que se ocupa de un único diagrama es la de Zapata, Tamayo y Arango (2007), que parte de esquemas preconceptuales para generar los diagramas de casos de uso. Esta propuesta permite identificar algunos de los elementos que se requieren en el presente artículo, pues el diagrama de casos de uso contiene información común con el de secuencias, pero no permite identificar los fragmentos combinados.

Dos propuestas más completas en este grupo son NIBA (sigla en alemán de Análisis de Requisitos de Información en Lenguaje Natural, Fliedl *et al.* (2002), que reconoce diagramas de clases, actividades y máquina de estados, y la de Zapata (2007), que parte de un lenguaje controlado denominado UN-Lencep (acrónimo de Universidad Nacional, Lenguaje Controlado para la Especificación de Esquemas Preconceptuales) para obtener los diagramas de clases, comunicación y máquina de estados. Pese a la completitud que exhiben ambas propuestas, pues identifican elementos correspondientes a diagramas de comportamiento, tales como actividades o comunicación, que comparten ciertos elementos con los diagramas de secuencias, tampoco identifican los fragmentos combinados.

En la propuesta de Díaz *et al.* (2005), se realiza una conversión semiautomática de los pasos correspondientes a un escenario de un caso de uso a una versión preliminar del diagrama de secuencias. En este proceso, que define una sintaxis clara para expresar los pasos del escenario, se pueden identificar algunos inconvenientes: el proceso es semiautomático, pues requiere una alta participación del analista, y no se identifican los fragmentos combinados.

Otro de los trabajos que cabe mencionar en este enfoque es el realizado por Feijs (2000) y Li (1999 y 2000), que relacionan algunos tipos de oraciones escritas en lenguaje natural con los MSC *(Message Sequence Chart)*. Este trabajo es similar al anterior, pues se parte de los pasos de un caso de uso, expresados en oraciones que deben cumplir con ciertas condiciones que restringen el lenguaje natural. También comparte sus limitaciones: es semiautomático y no identifica los fragmentos combinados.

3.2 Métodos que emplean el código fuente como punto de partida

Las propuestas enmarcadas dentro de este enfoque emplean el código fuente de una aplicación de software ya terminada como punto de partida en la obtención del diagrama de secuencias. Además, en

su mayoría, tienen como objetivo el entendimiento y prueba del software en ejecución, empleando para ello el diagrama de secuencias, que desempeña un rol central en el modelado de la interacción entre objetos (Rountev *et al*., 2005; Zapata, Ochoa y Vélez, 2008). Por ejemplo, Rountev *et al*. (2005) aplican una técnica de ingeniería inversa estática, haciendo un mapeo de objetos de interacción que parte del código fuente (Java) y obtiene su equivalente en los objetos del diagrama de secuencia mediante un esquema de etiquetado de objetos. Zapata, Ochoa y Vélez (2008) obtienen el diagrama de secuencias a partir de código Java empleando reglas heurísticas de conversión. Los trabajos mencionados sí se enfocan en la generación automática del diagrama de secuencias e identifican los fragmentos combinados como parte fundamental de dichos diagramas, pero tienen como principal inconveniente el punto de partida: el código fuente.

La ingeniería de software procura la aplicación de un enfoque sistemático y disciplinado para el desarrollo de software, en el cual se suele definir un ciclo de desarrollo que incluye varias fases. En dichas fases, el diagrama de secuencias se debe elaborar antes de la implementación de la aplicación en un lenguaje de programación particular. Partir del código fuente para obtener el diagrama de secuencias sirve como validación para la correcta elaboración del diagrama de secuencias, pero no contribuye al proceso de automatización en la elaboración de una aplicación de software.

Por ello, en este artículo se propone la generación del diagrama de secuencias de UML, a partir de una representación del sistema expresada en esquemas preconceptuales. Se toman estos esquemas como punto de partida, pues como demuestra Zapata (2007), tienen una representación textual cercana al lenguaje del interesado, que es el lenguaje UN-Lencep. Cabe anotar que el principal aporte de este artículo es la definición de las reglas que permiten la obtención de los elementos del diagrama de secuencias, incluyendo los fragmentos combinados, que hasta ahora no se identificaban en la literatura o se identificaban desde un producto avanzado del ciclo de vida del software como es el código fuente.

4. OBTENCIÓN DE DIAGRAMAS DE SECUENCIAS DE UML 2.1.1 A PARTIR DE ESQUEMAS PRECONCEPTUALES

Zapata (2007) propone el uso de esquemas preconceptuales, dado que compendian las características de algunos de los diagramas representativos de UML, como son clases, comunicación y máquina de estados. Los esquemas preconceptuales utilizan una notación gráfica para la expresión de los diferentes elementos del discurso de un interesado y constituyen un paso intermedio en la obtención automática de los diagramas de UML a partir de un lenguaje controlado. En este artículo se propone la ampliación de la sintaxis básica de los esquemas preconceptuales con un elemento adicional que permita obtener un diagrama de secuencias con la mayoría de elementos propuestos por UML 2.1.1, como los fragmentos combinados. Así, se define un tipo especial de concepto denominado "ventana", que tiene la misma sintaxis de los conceptos de los esquemas preconceptuales, pero anteponiendo la palabra "ventana" al nombre del concepto. Este elemento se requiere porque el discurso del interesado, que se representa en esquemas preconceptuales, es diferente al discurso que representa la solución, el cual se suele describir en los pasos del escenario de un caso de uso. El elemento "ventana" representa una interfaz gráfica de usuario con la cual un usuario de un sistema puede interactuar. Así, cuando en una transacción se menciona el "sistema", se puede reemplazar con la "ventana" con la cual el usuario interactúa.

4.1 Definición de reglas heurísticas

Los elementos del diagrama de secuencias se pueden identificar a partir de un esquema preconceptual mediante las reglas que se proponen en esta sección. Las reglas 1, 6, 7 y 9 provienen de trabajos previos, en tanto que las demás son originales de este artículo.

Regla 1: Elemento Actor

Esta regla se toma de Zapata, Tamayo y Arango (2007), donde se define que el concepto que antecede a una relación dinámica es un actor.

Regla 2: Elemento Línea de vida a partir de relación dinámica

En una relación dinámica que une dos conceptos A y B; el concepto B es un objeto que posee línea de vida, si B no es el concepto destino de una relación estructural.

Regla 3: Elemento Línea de vida a partir de relación estructural

En una relación estructural de tipo "tiene" que une dos conceptos A y B, el concepto A es un objeto con línea de vida.

Regla 4: Elemento Línea de vida a partir de doble relación estructural

En una relación estructural de tipo "tiene" que une dos conceptos A y B; el concepto B es un objeto con línea de vida, si a su vez desempeña el papel de concepto origen de otra relación estructural de tipo "tiene".

Regla 5: Elemento Interfaz de usuario

El elemento "Ventana" representa el sistema (que en realidad el usuario percibe como un conjunto de interfaces gráficas de usuario) a partir del texto de los requisitos del interesado. Dichas ventanas se consideran interfaces de usuario en el diagrama de secuencias, y para obtenerlas se establece que en una relación dinámica que une dos conceptos A y B, si B es una ventana, el concepto B es una interfaz de usuario candidata. También es posible identificar las interfaces de usuario a partir de los elementos del esquema preconceptual que tienen una relación estructural con el concepto "sesión" propio de un sistema de software terminado.

Regla 6: Elemento Mensaje a partir de relación dinámica

Esta regla se toma de Zapata (2007), para el diagrama de comunicación, pero es válida también para el diagrama de secuencias. Cuando se tiene una relación dinámica que une dos conceptos A y B, la relación dinámica es un mensaje del objeto B, si B no participa en una relación estructural como concepto destino.

Regla 7: Elemento Mensaje con atributos de clase

Al igual que la anterior, esta regla se toma de Zapata (2007). En una relación dinámica que une dos conceptos A y B, si B participa en una relación estructural como concepto destino, entonces dicha relación se define como un mensaje desde el objeto A cuyo nombre se compone del nombre de la relación dinámica y el nombre del concepto B.

Regla 8: Elemento Mensaje reflexivo

En una relación dinámica que une dos conceptos A y B; si B participa en una relación estructural en la que el concepto A es el concepto origen, entonces la relación dinámica es un mensaje de A enviado a sí mismo.

Regla 9: Secuencia

También esta regla proviene de la definición del diagrama de comunicación que realiza Zapata (2007), y se puede aplicar directamente al diagrama de secuencias. Las relaciones de implicación entre relaciones dinámicas se suponen como la secuencia de mensajes enviados entre los objetos identificados.

Regla 10: Elemento Fragmento combinado "opt"

Un condicional en un esquema preconceptual equivale a un fragmento combinado del diagrama de secuencias con operador "opt". El contenido del condicional se escribe como condición de guarda en el fragmento combinado.

Regla 11: Elemento Fragmento combinado "alt"

Dos o más condicionales sobre relaciones dinámicas que apuntan a un mismo concepto en un esquema preconceptual equivalen a un fragmento combinado del diagrama de secuencias con operador "alt". El contenido de los condicionales se escribe como condiciones de guarda en el fragmento combinado.

Regla 12: Elemento Fragmento combinado "par"

Dadas dos relaciones dinámicas 1 y 2 cuyos conceptos origen son iguales, si de la relación dinámica 1 no se inicia una relación de implicación a la relación dinámica 2, esto da origen a un elemento "par".

4.2 Implementación de las reglas en AToM³

En la presente propuesta se aprovechan las ventajas mencionadas de la herramienta AToM³ para la elaboración de modelos y metamodelos y la transformación entre esquemas preconceptuales y diagramas de secuencias.

4.2.1 Definición de los metamodelos

En AToM³ la definición de metamodelos se basa en el modelo entidad-relación extendido con restricciones, en el cual las entidades se representan como rectángulos y las relaciones como rombos. En las figuras 2 y 3 se pueden apreciar los metamodelos correspondientes al esquema preconceptual y diagrama de secuencias. Cabe anotar que el modelo entidad-relación es un diagrama estructural, por lo cual su lectura se puede realizar en cualquier orden, pues se trata sólo de establecer las relaciones entre las entidades del mundo. Por ejemplo, en la figura 2 la entidad "RelacionEstructural" y la entidad "Ventana" se unen mediante una relación denominada "REVentana". Los elementos del segundo cajón de cada entidad son características que la entidad posee y se denominan atributos. Además, se registra la información correspondiente al tipo de atributo. Por ejemplo, el atributo "Nombre" de la entidad "RelacionEstructural" es de tipo string. De forma análoga se puede realizar la lectura de los demás elementos correspondientes a las figuras 2 y 3.

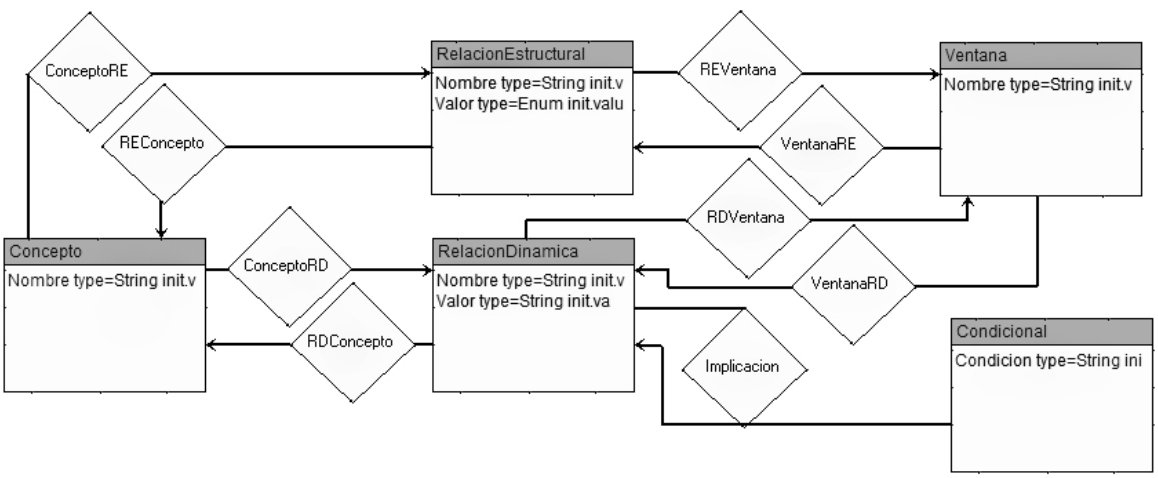


Figura 2. Metamodelo del esquema preconceptual en AToM³

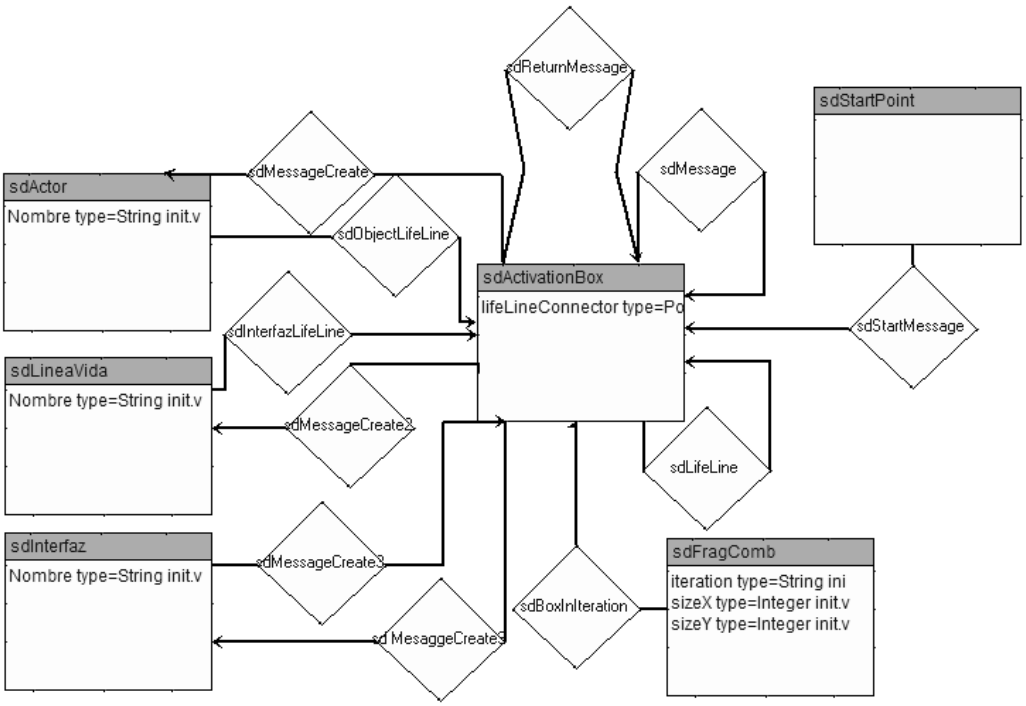


Figura 3. Metamodelo del diagrama de secuencias en ATOM^3

4.2.2 Implementación de las reglas de transformación en AToM^3

La escritura de las reglas de transformación entre los diferentes metamodelos en AToM^3 se realiza empleando la gramática de grafos y el lenguaje Phyton. Mediante este mecanismo se definieron las reglas para el presente artículo.

5. CASO DE ESTUDIO

Con el fin de ejemplificar las reglas heurísticas que se definieron en la sección 4.1, se presenta un caso de estudio relacionado con el despacho de un pedido. El enunciado de este caso de estudio se presentó originalmente en inglés en el proyecto NIBA (Fliedl *et al.*, 2002) y lo adaptó Zapata, (2007) al lenguaje UN-Lencep en español. Por razones de espacio, sólo se presenta una fracción del discurso:

Un artículo es parte de un pedido; cuando el asistente recibe el pedido, entonces el departamento_pedidos verifica la cantidad_existente del artículo; luego el asistente consulta el pago; si pago.estado='autorizado', entonces el departamento_pedidos tramita el pedido; si pago.estado='no_autorizado', entonces el departamento_pedidos rechaza el pedido; un pago contiene un pedido; el pago posee un estado; el departamento de pedidos posee un inventario.

Con el discurso presentado se obtiene el esquema preconceptual que se presenta en la figura 4. El esquema preconceptual combina información de tipo estático con información de tipo dinámico. Los elementos que conforman el primer tipo se pueden leer de manera similar a como se hace en el modelo entidad-relación; por ejemplo, se pueden leer los conceptos "departamento_pedidos" e "inventario" vinculados por la relación estructural "tiene"

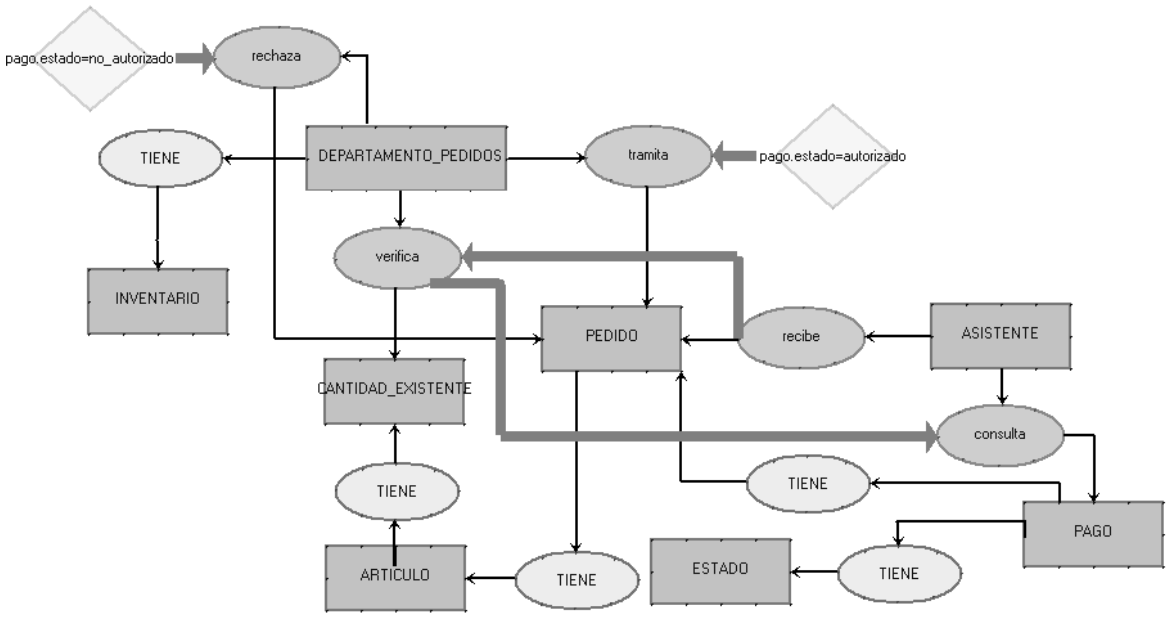


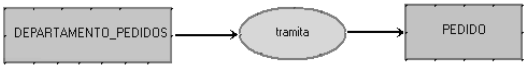
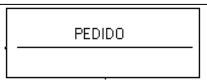
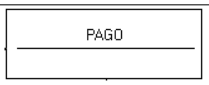
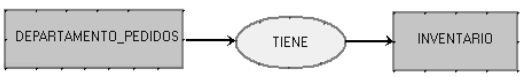
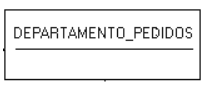
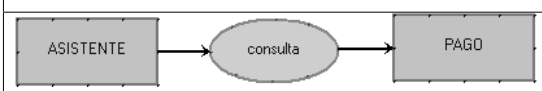
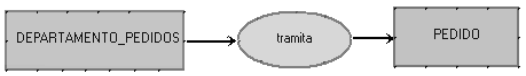
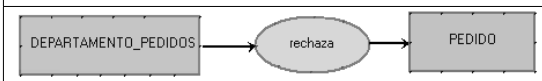
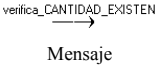
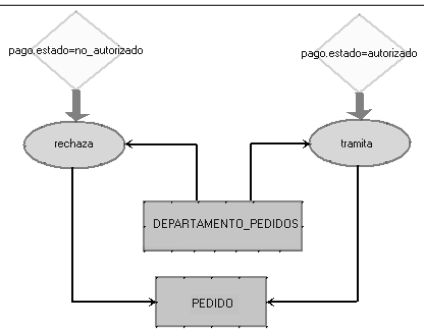
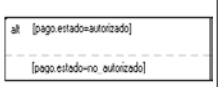
Figura 4. Esquema preconceptual del caso de estudio

como "departamento_pedidos tiene inventario". Los elementos que constituyen la parte dinámica son aquellos que poseen implicaciones (flechas gruesas); en este caso, la lectura se realiza desde el primer concepto unido a una relación dinámica que a su vez es origen de una implicación y se continúa con los demás elementos. Por ejemplo, en el diagrama se puede leer: "cuando asistente recibe pedido, entonces departamento_pedidos verifica cantidad_existente y luego asistente consulta pago". Los condicionales (rombos) se leen como relaciones causa-efecto y son de tipo dinámico; por ejemplo, se puede leer "si pago.estado=no_autorizado, entonces departamento_pedidos rechaza pedido".

En la tabla 2 se muestra paso a paso la aplicación de las reglas de transformación para el caso de estudio; la tabla 2 consta de la porción del esquema preconceptual, los elementos identificados del diagrama de secuencias y las reglas aplicadas. Además, de las reglas aplicadas en la tabla 2, la regla 9 permite identificar la secuencia de los mensajes recibe(), verifica_CANTIDAD_EXISTENTE() y consulta().

Finalmente, en la figura 5 se puede observar el diagrama de secuencias obtenido al ejecutar la transformación. En este diagrama "asistente" es un actor, "departamento_pedidos", "pedido", "articulo" y "pago" son clases de objetos y las flechas representan los diferentes mensajes que se envían los diferentes objetos para realizar la transacción representada por la secuencia.

Tabla 2. Proceso de transformación entre esquema preconceptual y diagrama de secuencias

Esquema Preconceptual	Elemento identificado	Regla aplicada
	 Actor	1
	 Línea de Vida	2
	 Línea de Vida	3
	 Línea de Vida	3
	 Línea de Vida	4
	 Mensaje	6
	 Mensaje	6
	 Mensaje	6
	 Mensaje	6
	 Mensaje	7
	 alt	11

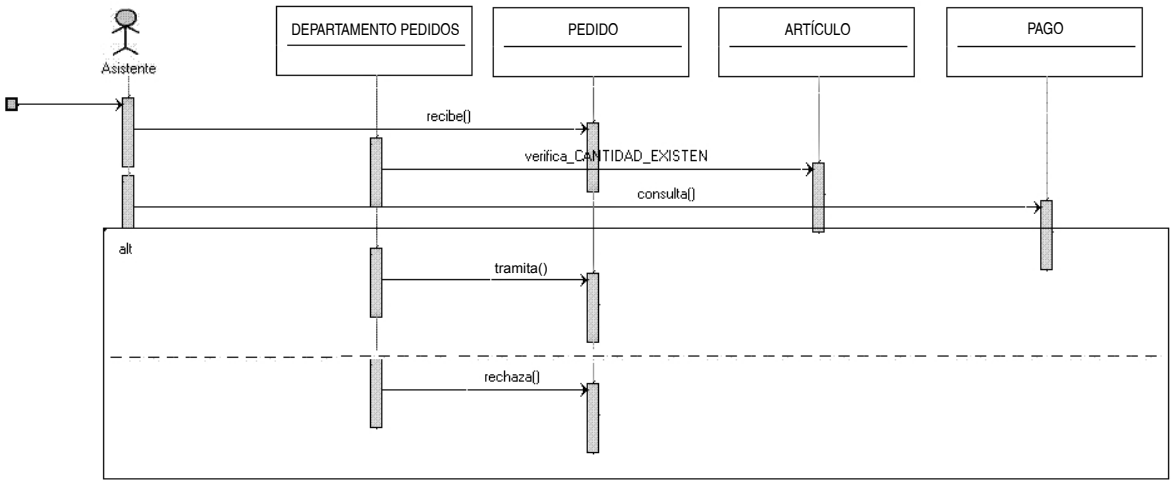


Figura 5. Diagrama de secuencias obtenido

6. CONCLUSIONES Y TRABAJO FUTURO

En este artículo se presentó un método que, además de facilitar la interacción con el interesado en el proceso de desarrollo empleando los esquemas preconceptuales, define las reglas de transformación automática entre dichos esquemas y el diagrama de secuencias, que permite representar la interacción entre los diferentes objetos de un sistema.

Los principales aportes para destacar son los siguientes:

- Se parte de una representación del dominio del interesado (esquema preconceptual), que se puede ver como un intermediario en la relación analista-interesado. Además, con la automatización introducida en la obtención del diagrama de secuencias se reduce la ambigüedad que generan interesados y analistas, quienes sólo participan en la construcción del esquema preconceptual, validándolo o corrigiéndolo en cualquier momento. Una vez generado el esquema preconceptual, ni el analista ni el interesado intervienen en la obtención del diagrama de secuencias.
- Se definió un conjunto de reglas heurísticas de transformación entre los esquemas preconceptuales y el diagrama de secuencia UML 2.1.1. Algunas reglas se adaptaron de trabajos previos para el diagrama de comunicación (que presenta algunas similitudes con el de secuencias), aunque la mayoría de ellas se proponen en este artículo. Esas reglas identifican los principales elementos del diagrama de secuencias definidos en el estándar UML 2.1.1, incluyendo los fragmentos combinados, de gran utilidad en el modelado dinámico del sistema.
- Se realizó la inclusión de un nuevo elemento ("ventana") en la especificación de los esquemas preconceptuales, con el fin de facilitar la conversión expuesta en este artículo.

Como trabajo futuro se pueden plantear los siguientes ítems:

- Establecer reglas de transformación adicionales que permitan obtener los elementos faltantes del diagrama de secuencias del estándar UML 2.1.1. Particularmente, se requieren reglas para otros elementos como la muerte de los objetos o los fragmentos combinados *break* y *loop*.

- Combinar las reglas aquí definidas con reglas de ingeniería inversa para tratar de definir un método para obtener código fuente desde lenguaje natural.
- Avanzar en el estudio de la obtención del diagrama de secuencias a partir del lenguaje natural y no de un lenguaje controlado.
- Poner a disposición de analistas expertos los casos de estudio y método planteados, con el fin de realizar comparaciones de los resultados obtenidos.

REFERENCIAS

Ambler, S. (2005). The elements of UML 2.0 style. Cambridge University Press, New York.

De Lara, J. and Vangheluwe, H. (2002). AToM^3: A tool for multi-formalism and meta-modelling. Proceedings of the Fifth International Conference on Fundamental Approaches to Software Engineering, London (England). pp. 174-188.

De Lara, J.; Vangheluwe, H. and Alfonseca, M. (2003). Using meta-modelling and graph-grammars to create modelling environments. Electronic Notes in Theoretical Computer Science, 72(3): 36-50.

De Lara, J.; Guerra, E. and Vangheluwe, H. (2004). Meta-modelling, graph transformation and model checking for the analysis of hybrid systems. Lecture Notes in Computer Science, 3062:292-298.

Díaz, I.; Moreno, L.; Fuentes, I. and Pastor, O. (2005). Integrating natural language techniques in OO-method. Lecture Notes in Computer Science, 3406:560-571.

Feijs, L.M.G. (2000). Natural language and message sequence chart representation of use cases. Information and Software Technology, 42(9):633-647.

Fliedl, G.; Kop, Ch.; Mayerthaler, W., Mayr, H. C. and Winkler, Ch. (2002). The NIBA workflow: From textual requirements specifications to UML-schemata. Proceedings of the ICSSEA'2002 International Conference "Software & Systems Engineering and their Applications", Paris (France).

Fowler, M. and Scott, K. (1997). UML distilled: applying the standard object modeling language. Addison-Wesley Series Editors, Essex.

Fowler, M. and Scott, K. (2003). UML distilled: a brief guide to the standard object modeling language, 3rd ed. Addison-Wesley Professional, Essex.

Harmain, H. M. and Gaizauskas, R. (2000). CM-Builder: an automated NL-based CASE tool. Proceedings of the Fifteenth IEEE International Conference on Automated Software Engineering. Grenoble (France). pp. 45-53.

Li, L. (1999). A semi-automatic approach to translating use cases to sequence diagrams. Technology of Object-Oriented Languages and Systems, Jul.1999:184-193.

Li, L. (2000). Translating use cases to sequence diagrams. Proceedings of the Fifteenth IEEE International Conference on Automated Software Engineering. Grenoble (France). pp. 293-296.

Mich, L. (1996). NL-OOPS: From natural language to object oriented requirements using the natural language processing system LOLITA. Journal of Natural Language Engineering, 2(2):161-187.

Mich, L. and Garigliano, R. (2002). NL-OOPS: A requirements analysis tool based on natural language processing. Proceedings of the 3rd International Conference on Data Mining. Bologna (Italy). pp. 321-330.

Objetc Management Group (OMG). (2007). Unified modeling language specification. Version 2.1.1. http://www.omg.org/uml /20-09-08.

Overmyer, S.; Lavoie, B. and Rambow, O. (2001). Conceptual modeling analysis using LIDA. Proceedings of the 23rd International Conference on Software Engineering, Toronto (Canada). pp. 401-410.

Pressman, R. (2005). Software Engineering: a practitioner's approach, 6th ed. McGraw-Hill, NewYork.

Rountev, A.; Kagan, S. and Sawin, J. (2005). Coverage criteria for testing of object interactions in sequence diagrams. Lecture Notes in Computer Sciences 3442:282-297.

Torres, A. (1993). Modelamiento y metamodelamiento de la incertidumbre en problemas de ingeniería. Revista de Ingeniería Universidad de los Andes 4:29-36.

Zapata, C. and Arango, F. (2005). Los modelos verbales y su utilización en la elaboración de esquemas conceptuales: una revisión crítica. Revista EAFIT. 41(137):77-95.

Zapata, C.; Gelbukh, A. and Arango, F. (2006a). Pre-conceptual schemas: a conceptual-graph-like knowledge representation for requirements elicitation. Lecture Notes in Computer Sciences, 4293:27-37.

Zapata, C.; Gelbukh, A. y Arango, F. (2006b). UN-Lencep: obtención automática de diagramas UML a partir de un lenguaje controlado. Memorias del 3er Taller de Tecnologías del Lenguaje Humano, Encuentro Nacional de Ciencias de la Computación, San Luis Potosí (México).

Zapata, C. (2007). Definición de un esquema preconceptual para la obtención automática de esquemas conceptuales de UML. Tesis doctoral, Universidad Nacional de Colombia. Medellín.

Zapata, C.; Tamayo, P. and Arango, F. (2007). Conversion of pre-conceptual schema into use case diagrams by using AToM^3. Revista DYNA, 74(153):237-251.

Zapata, C.; Ochoa, O. y Vélez, C. (2008). Un método de ingeniería inversa de código Java hacia diagramas de secuencias de UML 2.0. Revista Escuela de Ingeniería de Antioquia, 9:31-42.