Revista Ingenierías Universidad de Medellín 5(9): 111-122 julio-diciembre de 2006

Reglas de conversión entre el diagrama de clases y los grafos conceptuales de Sowa[1]

Carlos Mario Zapata*, Betsy María Estrada** y Guillermo González***

Recibido: 17/03/2006
Aceptado: 28/08/2006

RESUMEN

La conversión entre modelos de un nivel de abstracción inferior a otro de nivel de abstracción superior facilita la comunicación entre los involucrados en un proceso de desarrollo de software. Los grafos conceptuales son diagramas que presentan la información modelada de una manera semiformal, y pueden llegar a ser comprensibles tanto por el humano como por el computador. El diagrama de clases, en cambio, presenta las clases, atributos, operaciones y relaciones principales de un sistema en un lenguaje propio de los expertos en modelamiento de productos de software. En este artículo se propone un conjunto de reglas de conversión para traducir el diagrama de clases (más detallado y, en consecuencia, de bajo nivel de abstracción) en una forma más comprensible al interesado (y de más alto nivel de abstracción) como lo son los grafos conceptuales de Sowa.

Palabras clave: reglas de conversión, grafos conceptuales de Sowa, diagrama de clases.

ABSTRACT

The conversion from lower-level abstraction models to upper-level abstraction models encourages communication between the parts of the software development process. Conceptual graphs are diagrams for presentation of the modeled information in a semi-formal way, and they can be understandable both for human and computer. Class diagram, in contrast, presents the systems classes, attributes, operations and main relations in an expert language for software product modelers. In this paper, we propose a set of conversion rules for translating class diagram (a more detailed, low-level abstraction model) into a one more understandable form for stakeholders (an upper-level abstraction model) based on Sowa's Conceptual Graphs.

1 Este artículo se realizó en el marco de los siguientes proyectos de investigación: "CONSTRUCCIÓN AUTOMATICA DE ESQUEMAS CONCEPTUALES A PARTIR DE LENGUAJE NATURAL", financiado por la DIME y "DEFINICIÓN DE UN ESQUEMA PRECONCEPTUAL PARA LA OBTENCIÓN AUTOMÁTICA DE ESQUEMAS CONCEPTUALES DE UML", financiado por DINAIN y administrado por la DIME.

* Ph. D. (c) en Ingeniería. Grupo UN-INFO. Profesor asistente de la Escuela de Sistemas. Universidad Nacional de Colombia. Sede Medellín.Calle 80 # 65-223 Bloque M8-112. Tel: 4255374. e-mail: cmzapata@unalmed.edu.co

** Ingeniera de sistemas e informática. Calle 80 # 65-223 Bloque M8-118. Tel: 4255376. E-mail: bmestrad@unalmed.edu.co

*** Ingeniero de sistemas e informática. Calle 80 # 65-223 Bloque M8-105. Tel: 4255350. E-mail: ggonzal@unalmed.edu.co

INTRODUCCIÓN

Durante la fase de definición del proceso de desarrollo de un producto de software, los interesados y los analistas necesitan estar en constante comunicación para realizar una abstracción correcta del sistema modelado, reflejada en los esquemas conceptuales elaborados en las fases siguientes. La mayoría de las veces esa comunicación se ve empañada por la diferencia de conocimiento entre las partes, dando como resultado esquemas conceptuales incompletos o incorrectos respecto del dominio del problema en cuestión (Zapata y Arango, 2005).

La validación, por parte del interesado, de los modelos conceptuales es una forma de capturar la información incompleta y corregir la información incorrecta, para posteriormente realizar el refinamiento de diagramas y disminuir tanto las probabilidades de desbordamientos de costos y tiempos de entrega como también las de pérdidas de requisitos. Para que los procesos de validación y refinamiento tengan buenos resultados es necesario que el interesado entienda la sintaxis de los diagramas que va a validar. El diagrama de clases, por ejemplo, no posee una sintaxis de fácil comprensión para personas que no estén relacionadas con el análisis y diseño de productos de software.

La principal ventaja que ofrecen los grafos conceptuales en comparación con otros esquemas conceptuales generados en el proceso de desarrollo de un producto de software es la posibilidad de representar la información del dominio de un problema de forma precisa, comprensible por los interesados y manejable computacionalmente (Sowa, 1984). Esta ventaja hace que, en algunos casos, por ejemplo en el refinamiento de modelos de datos, se deba recurrir a una conversión de esquemas de nivel de abstracción inferior a diagramas más amigables al interesado, como lo son los grafos conceptuales. Por lo anterior, se puede considerar que los grafos conceptuales constituyen un punto intermedio entre los esquemas en un formalismo poco entendido por el humano y muy comprensible por la máquina (de bajo nivel de abstracción) y los lenguajes naturales (de alto nivel de abstracción).

Los grafos conceptuales, además de poseer una sintaxis gráfica, entendible por los interesados, pueden representarse en una notación basada en caracteres (forma lineal, de intercambio de grafos conceptuales y de intercambio de conocimiento, Sowa, 1984), que también puede ser comprensible por la máquina, aunque de manera diferente a como se entiende el diagrama de clases, por ejemplo. Las bondades de los grafos conceptuales han sido explotadas en áreas tales como: recuperación de información, diseño de bases de datos, procesamiento de lenguaje natural y sistemas expertos, entre otras.

El objetivo de este artículo es proponer una serie de reglas para permitir la conversión de un diagrama de clases dado en sus respectivos grafos conceptuales. Algunos autores realizan este proceso con fines de integración de esquemas (Creasy y Ellis, 1993), traducción automática (Snook y Butler, 2001), elaboración de esquemas conceptuales a partir de lenguaje natural (Coad y Yourdon, 1990), (Chen, 1983) y demás aplicaciones que requieran la información de los esquemas conceptuales en una representación más abstracta pero entendible computacionalmente.

DESCRIPCIÓN DE LOS GRAFOS CONCEPTUALES Y DEL DIAGRAMA DE CLASES DE UML

Los grafos conceptuales son diagramas que modelan los conceptos y relaciones conceptuales de un sistema, son de fácil entendimiento para el interesado, y permiten un manejo computacional. La notación gráfica de los grafos conceptuales representa los conceptos identificados en el dominio del problema de un sistema a través de rectángulos; las relaciones conceptuales por medio de círculos u óvalos y los enlaces establecidos entre conceptos

y relaciones conceptuales se dibujan como flechas. (Véase Figura 1).

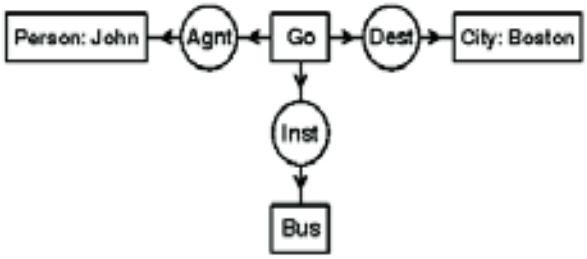

Figura 1. Forma gráfica del grafo conceptual para la oración: John va a Boston en bus. (Tomada de Sowa, 1984)

La forma gráfica de los grafos conceptuales tiene equivalencias en formatos legibles por máquina, tales como el CGIF (Conceptual Graph Interchange Format) y el KIF (Knowledge Interchange Format), los cuales contribuyen a darle utilidad computacional a estos grafos, puesto que existen herramientas actualmente que permiten obtener estas equivalencias desde editores de dichos grafos (Sowa, 1984).

A mediados de la década de los noventa, surgió el Unified Modeling Language (Lenguaje Unificado de Modelamiento o UML), como uno de los principales estándares para el desarrollo de software. El diagrama de clases de UML muestra los objetos relevantes del dominio del problema de un sistema específico y los agrupa en clases (Object Management Group, 2003). Para ello identifica en cada objeto los atributos que le pertenecen, las operaciones que se pueden realizar con él y sus relaciones con otros objetos (asociaciones, generalizaciones, dependencias, agregaciones y composiciones) con sus cardinalidades y el rol que desempeña en cada relación. Desde sus inicios, el diagrama de clases ha permitido el modelamiento de los conceptos del dominio de un problema determinado, pero en una notación que está dirigida a analistas entrenados en su uso, constituyéndose en un diagrama cuyo nivel de abstracción es bajo. La simbología básica del diagrama de clases de UML se muestra en la figura 2.

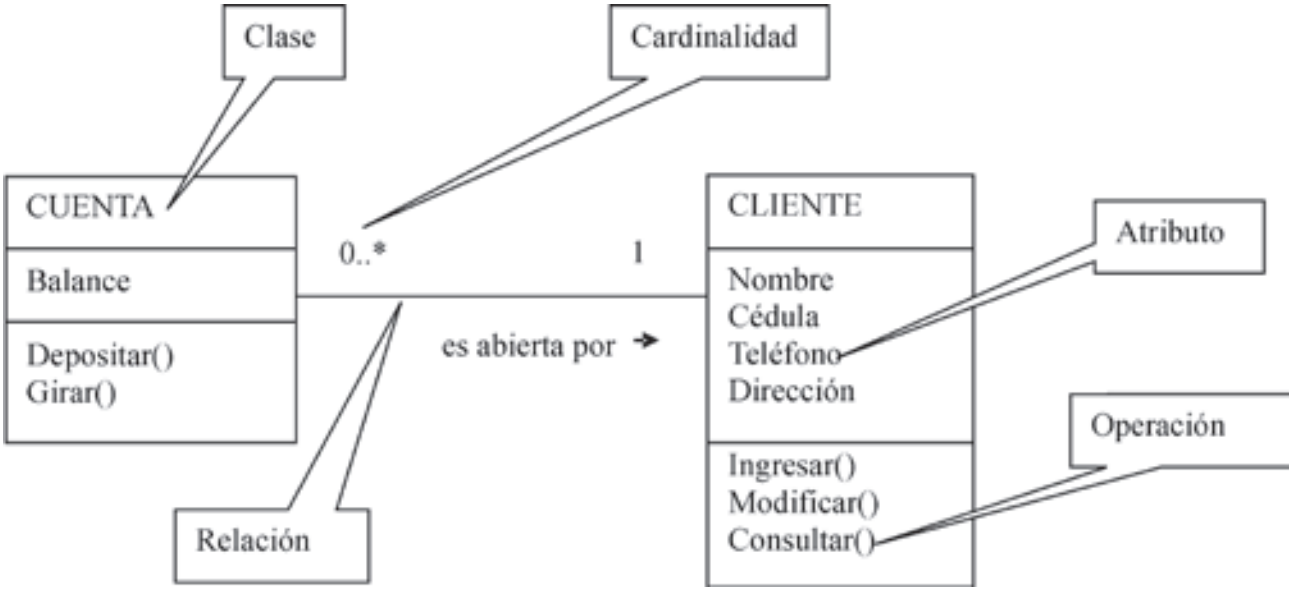

Figura 2. Simbología del diagrama de clases de UML

En la especificación del diagrama de clases de UML (Object Management Group, 2003) se describen cada uno de los elementos del diagrama de clases, los elementos con los cuales es posible su relación, su notación y sus representaciones opcionales.

En la siguiente sección se realiza un análisis crítico de algunos trabajos realizados para establecer conversiones entre esquemas conceptuales generados en el proceso de diseño de sistemas informáticos.

ANTECEDENTES

Entre los principales trabajos realizados para lograr conversión entre modelos conceptuales se pueden mencionar:

Snook y Butler (2001) proponen una adaptación de una notación de diseño gráfica (UML) para la especificación formal y la construcción de una herramienta de traducción automática, agregando detalles semánticos y definiendo el significado de cada elemento del diagrama. Implementan, además, un prototipo para trasladar las características diagramáticas de UML al lenguaje B (Abrial, 1996). B es un lenguaje de especificación formal, y el objetivo de este trabajo es utilizar algunas de las características de los diagramas de clases para realizar especificaciones formales de forma fácil. Las restricciones de los componentes del diagrama de clases son descritas en una forma restringida de la notación del lenguaje B. Esta traducción no agrega información a la especificación de diagramas en UML, sino que proporciona una forma textual alternativa. La iniciativa de agregar detalles semánticos a los modelos conceptuales puede conducir a una forma de encontrar deficiencias en tal diagrama y realizar refinamientos al modelo. La conversión que se plantea en Snook y Butler (2001) no contribuye a mejorar la comprensión del diagrama de clases por parte del interesado; esto se produce porque una especificación en un lenguaje formal como B requiere de un entrenamiento mucho mayor para su comprensión, puesto que el nivel de abstracción de este lenguaje es incluso mucho más bajo que el diagrama de clases de UML.

La herramienta de conversión descrita en Timm y Gannod (2005) fue creada para tomar una especificación UML y convertirla en una representación equivalente en OWL-S (que es un lenguaje enfocado a agentes de software, creado para describir conceptos y relaciones semánticas de servicios Web usando ontologías). Para ello, utiliza XMI (un estándar de comunicación entre aplicaciones basado en el lenguaje de marcado extendido XML) y sobre él aplica las transformaciones con un lenguaje basado en este estándar. Estos autores buscan la forma de generar un modelo de servicios OWL-S tomando una representación XML basada en un diagrama de actividades de UML. El mecanismo utilizado para validar qué tan correcta es la conversión realizada es la herramienta de especificación de ontologías Protégé. Este tipo de conversión se ocupa de la legibilidad por parte de la máquina y poco o nada contribuye a la legibilidad por parte de seres humanos; además, el nivel de abstracción se conserva bajo, lo cual no permite la validación por parte de los interesados.

Creasy y Ellis (1993) presentan una herramienta informática (basada en grafos conceptuales) para solucionar el problema de integración de esquemas conceptuales generados en el diseño de sistemas de información complejos. Los lineamientos teóricos de esta propuesta están orientados al diagrama entidad-relación, no al diagrama de clases que es completamente objetual y además es reconocido como el estándar aceptado por el OMG (Object Management Group, 2003). Pese a trabajar con grafos conceptuales, esta propuesta no eleva de manera sustancial el nivel de abstracción, pues se limita a elaborar una descripción del diagrama entidad-relación con la terminología de este diagrama, la cual es poco entendible para los interesados. Por otra parte, esta metodología no utiliza los roles semánticos para especificar las relaciones entre los conceptos del

dominio, haciendo más compleja la traducción de esquemas conceptuales a lenguaje natural.

Coad y Yourdon (1990) establecieron un método que parte del análisis del modelo verbal y a partir de tal análisis identifican los principales elementos del diagrama de clases, diagrama principal en el modelamiento y desarrollo de un producto de software orientado por objetos. Esta metodología identifica los principales sustantivos presentes en el modelo verbal, información valiosa para la detección de las "clases" y "objetos" del modelo, y los principales verbos, que pueden suministrar información sobre las relaciones entre las diferentes entidades del modelo. Esta conversión parte de un modelo más comprensible por el interesado hacia uno de notación más elaborada y compleja, que es un proceso inverso al planteado en este artículo. Como la entrada de esta conversión es el lenguaje natural, es necesario llevar a cabo procesos de desambiguación antes de generar el diagrama de salida. En esta misma línea de trabajo, Harmain y Gaizauskas (2000) proponen el CM-Builder, una herramienta CASE para la obtención del diagrama de clases a partir de las especificaciones de requisitos en lenguaje natural. En este caso, dada la existencia de un diagrama de clases, no es posible realizar el proceso contrario, con el fin de realizar una validación del mismo por parte del interesado.

NL-OOPS (Natural Language Object – Oriented Product System), es un sistema propuesto por Mich (1996) e incorpora un sistema de procesamiento del lenguaje natural denominado LOLITA (Large-scale Object-based Language Interactor, Translator and Analyser). LOLITA contiene un conjunto de funciones para el análisis del lenguaje natural y a través de este análisis detecta ambigüedades en el texto. Como resultado final NL-OOPS entrega, en una estructura de árbol, un conjunto de las clases candidatas y sus posibles instancias, atributos y métodos (Mich y Garigliano, 2002). Esta solución es sólo una contribución a la complementación del diagrama de clases, ya que no posee la representación de las relaciones entre las clases del modelo convertido.

Fliedl (1999, 1999a, 2002, 2002a y 2003) acompañado de un grupo de ingenieros de sistemas (Kop y Mayr, 2002), (Mayr y Kop, 2002) desarrollaron el KCPM (Klagenfurt Conceptual Predesing Model) como parte del proyecto NIBA. El objetivo del proyecto es facilitar el desarrollo de productos de software a través de la traducción automática del lenguaje natural a esquemas conceptuales. Para lograr este objetivo cuenta con los siguientes elementos:

- Un analizador sintáctico.
- Un intérprete semántico basado en la teoría de roles Theta (Agente, Experimentador, Tema, Meta, Fuente, Locación) (Gruber, 1965), y en lineamientos teóricos para determinar las funciones de las palabras que acompañan a un verbo (Fillmore, 1968).
- Una herramienta KCPM para la traducción y análisis (sintáctico y semántico) de oraciones.
- Una herramienta para el cálculo de puntos de función.

El proceso de conversión descrito en el proyecto NIBA es completamente contrario al planteado en este artículo, ya que va de lenguaje natural a esquemas conceptuales y requiere desambiguación.

En la sección siguiente se generan las reglas de transformación entre diagramas de clases y grafos conceptuales de Sowa, partiendo de las ventajas y desventajas de las propuestas existentes y de los aspectos comunes de los diagramas involucrados en la conversión.

BREVE DESCRIPCIÓN DE LA PROPUESTA

Los antecedentes analizados en la sección anterior no suministran una solución que posibilite a los interesados lograr una comprensión del diagrama de clases, que le permita definir si efectivamente

ese diagrama representa lo que está tratando de expresar. Sin embargo, en la mayoría de esos trabajos se emplean reglas de conversión para lograr la transformación de un modelo a otro.

Entre las limitaciones de los trabajos expuestos en la sección anterior se pueden nombrar los siguientes:

- En la mayoría de los casos la conversión se presenta a partir de lenguaje natural, pero no se establece qué ocurre cuando el diagrama de clases ya existe y se requiere que el interesado realice una validación del mismo. Esta es una situación típica durante la captura de los requisitos.
- En los casos en que se parte de diagramas se llega a una especificación formal o a un diagrama de igual o incluso menor nivel de abstracción que, por ende, requiere mayor entrenamiento para el entendimiento humano.

Para aprovechar las bondades de las propuestas existentes y dejar de lado las limitaciones expuestas, en este artículo se propone un método para realizar la conversión entre el diagrama de clases y los grafos conceptuales de Sowa. Esta conversión se realizará directamente sobre el diagrama de clases generado por el analista en la fase de análisis del ciclo de vida de un producto de software. El objetivo de esta conversión es encontrar una forma de expresar una sintaxis tan compleja como lo es el diagrama de clases de UML en una notación más comprensible por los interesados en la construcción de un sistema informático; con ello se busca involucrar al interesado en la validación del diagrama de clases, especialmente tratando de establecer si dicho diagrama realmente representa de manera adecuada el dominio del problema que se pretende solucionar con la pieza de software.

El primer paso para lograr la conversión será encontrar las partes de ambos diagramas que expresan aspectos comunes. Por ejemplo, las clases del diagrama de clases muestran los principales conceptos de un sistema informático modelado al igual que los conceptos de los grafos conceptuales. De igual manera, las relaciones entre las clases de un diagrama de clases suministran cierta información para identificar las relaciones conceptuales de los grafos de Sowa y los enlaces establecidos entre conceptos y relaciones conceptuales.

Para llevar a cabo la conversión de diagramas se definen los siguientes principios:

- Toda clase, atributo y operación del diagrama de clases debe ser un concepto del grafo conceptual de Sowa.
- Las relaciones entre clases, atributos y operaciones del diagrama de clases se pueden expresar en términos de otros conceptos y los roles theta o casos semánticos del grafo conceptual de Sowa (agente, experimentador, locación, destino, instrumento, etc.).

Adicionalmente, se definen las siguientes reglas de transformación del diagrama de clases a grafos conceptuales de Sowa:

Regla 1: clase-atributo

Un atributo de una clase genera los siguientes elementos en el grafo conceptual:

- Un concepto dado por el nombre del atributo.
- El concepto "tener".
- Un concepto dado por el nombre de la clase.
- Una relación "beneficiario" entre los conceptos "tener" y el nombre de la clase.
- Una relación "tema" entre los conceptos "tener" y el nombre del atributo.

Una frase que se puede extraer de la definición de la clase cliente, que se aprecia en la figura 3 es: "El cliente es el beneficiario del concepto tener, cuyo

tema es código" o, más concretamente, "El cliente tiene código". El grafo conceptual correspondiente a esta frase se muestra en la figura 4.

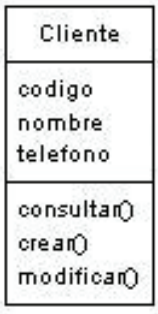

Figura 3. Definición de la clase cliente

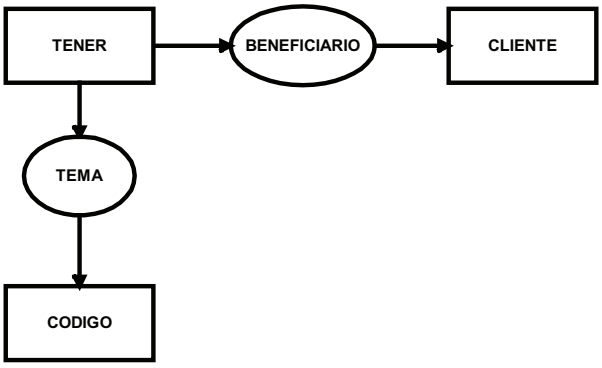

Figura 4: Grafo conceptual de la frase "El cliente tiene código"

Regla 2: clase-operación

Una operación de una clase genera los siguientes elementos en el grafo conceptual:

- Un concepto dado por el nombre de la clase.
- Un concepto dado por el nombre de la operación.
- El concepto "usuario", sugerido al interesado para que interactivamente defina si ese concepto existe o si tiene otro nombre.
- Una relación "agente" entre los conceptos "usuario" y el nombre de la operación.
- Una relación "tema" entre los conceptos dados por el nombre de la operación y el nombre de la clase.

De la definición de la clase cliente del ejemplo anterior se puede afirmar que "el usuario es el agente del concepto consultar, cuyo tema es el cliente", o simplemente que "el usuario consulta al cliente". El grafo conceptual para representar lo anterior sería el que aparece en la figura 5.

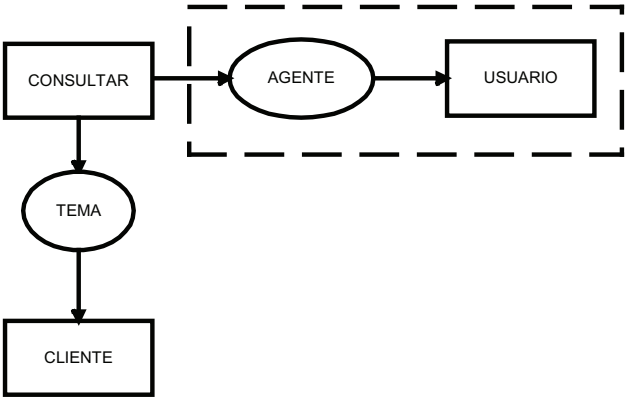

Figura 5. Grafo conceptual de la frase "El usuario consulta al cliente"

Regla 3: clase-relación-clase

Dos clases relacionadas como en el caso de la Figura 6, generan los siguientes elementos:

- Dos conceptos dados por el nombre de las clases.
- Un concepto dado por el nombre de la relación.
- Una relación "agente" entre el nombre de la relación y el nombre de la clase que tiene cardinalidad 1.
- Una relación "tema" entre el nombre de la relación y el nombre de la clase que tiene cardinalidad *.

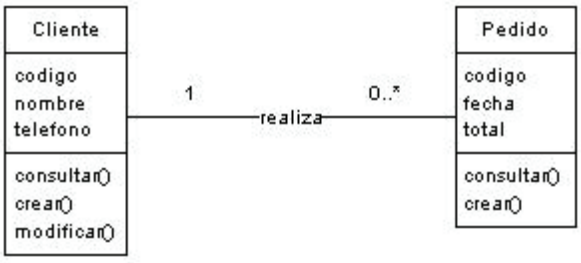

Figura 6. Relación entre las clases cliente y pedido

El grafo conceptual correspondiente se muestra en la figura 7. Es de notar que, si la relación es 1 a 1, entonces el sentido de la relación definirá la relación conceptual de cada una de las clases. Si la relación es una agregación o composición, como en la figura 8, entonces la clase de agregación y la clase de composición no son agentes sino beneficiarios.

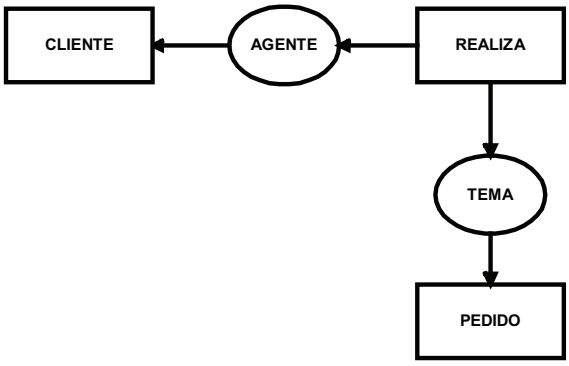

Figura 7: Grafo conceptual para la relación entre dos clases

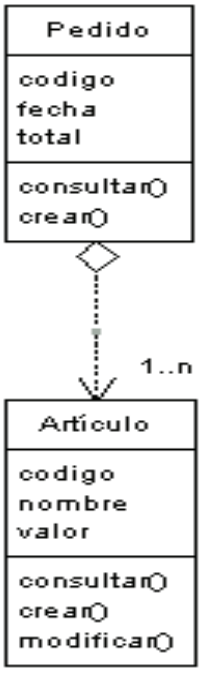

Figura 8: Relación de agregación entre dos clases

Según la regla anterior, el grafo conceptual generado para esta porción de diagrama de clases es el que aparece en la figura 9:

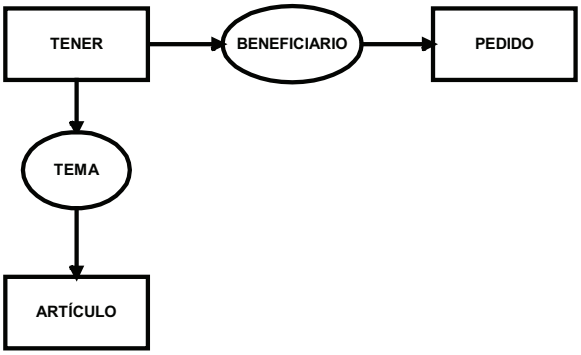

Figura 9: Grafo conceptual para la relación de agregación entre las clases pedido y artículo

Regla 4: Metarregla para correferencia

En cada diagrama generado, los conceptos que se hayan generado desde el mismo nombre de clase se unen con una línea punteada que expresa la correferencia en el grafo conceptual, es decir, la forma de reconocer que este concepto es el mismo en dicho grafo.

En la siguiente sección se muestra la implementación de las reglas y se ejemplifica con un caso de estudio.

APLICACIÓN Y CASO DE ESTUDIO

Las reglas definidas en la sección anterior se implementaron en el prototipo de una aplicación con las siguientes características:

- La entrada al proceso es un diagrama de clases elaborado en la herramienta CASE ArgoUML, disponible en la página http://argouml.tigris.org/. Para ingresar el diagrama en la aplicación, se requiere el código XMI generado por el ArgoUML.
- Las reglas se programaron en XQuery, un lenguaje de consulta que explora archivos en XML y genera nuevos archivos en XML que pueden ser leídos por otras herramientas. Para la programación de dichas reglas se empleó la herramienta XQEngine, disponible en la página http://xqengine.sourceforge.net/.

- La salida del proceso es un archivo en formato XML que se puede leer en la herramienta Charger, disponible en la página http://sourceforge.net/projects/charger/, la cual permite la edición de grafos conceptuales de Sowa.

Parte de la consulta en XQuery para la extracción de los atributos de las clases (Regla 1) se muestra a continuación. Se suprimieron las partes del código que generan los elementos del XML en Charger, para facilitar la legibilidad del mismo.

```
<grafo_conceptual>
    {for $a in //Foundation.Core.Class
    return
        <reglas_para_clases>
        {for $b in $a/Foundation.Core.Classifier.feature/Foundation.Core.Attribute/Foundation.Core.ModelElement.name
        return
            <reglas_para_atributos>
                <relacion nombre="beneficiario">
                    <concepto1>tener</concepto1>
                    <concepto2>{$a/Foundation.Core.ModelElement.name/text()}</concepto2>
                </relacion>
                <relacion nombre="tema">
                    <concepto1>tener</concepto1>
                    <concepto2>{$b/text()}</concepto2>
                </relacion>
            </reglas_para_atributos>}
        </reglas_para_clases>}
</grafo_conceptual>
```

De manera similar, el código en XQuery que corresponde a la extracción de las operaciones es el siguiente:

```
<grafo_conceptual>
    {for $a in //Foundation.Core.Class
    return
        <reglas_para_clases>
        {for $c in $a/Foundation.Core.Classifier.feature/Foundation.Core.Operation/Foundation.Core.ModelElement.name
        return
            <reglas_para_operaciones>
                <relacion nombre="agente">
                    <concepto1>{$c/text()}</concepto1>
                    <concepto2>usuario</concepto2>
                </relacion>
                <relacion nombre="tema">
                    <concepto1>{$c/text()}</concepto1>
                    <concepto2>{$a/Foundation.Core.ModelElement.name/text()}</concepto2>
                </relacion>
            </reglas_para_operaciones>}
        </reglas_para_clases>}
</grafo_conceptual>
```

En ambos casos, *Foundation.Core.Class* es la etiqueta del código XMI de ArgoUML que almacena la información de la clase y es superior en jerarquía a las demás etiquetas que comienzan con *Foundation* y que se emplean para recopilar la información de atributos y operaciones.

Empleando esta aplicación se ejemplifica la conversión de un diagrama de clases correspondiente a un sistema de pedidos de artículos; se supone que este diagrama fue generado por un analista en la fase de análisis y su imagen en ArgoUML se puede apreciar en la figura 10.

Figura 10. Diagrama de clases

Ahora, cada clase, atributo, operación y relación del diagrama de clases se representan como conceptos del grafo conceptual, utilizando la relación conceptual apropiada (rol semántico) dependiendo de la regla aplicada, para terminar el proceso de conversión con la unión de los conceptos del grafo. A manera de ejemplo, véanse las figuras 11 a 15, generadas con el Charger a partir del archivo en XMI del diagrama de clases de la figura 10.

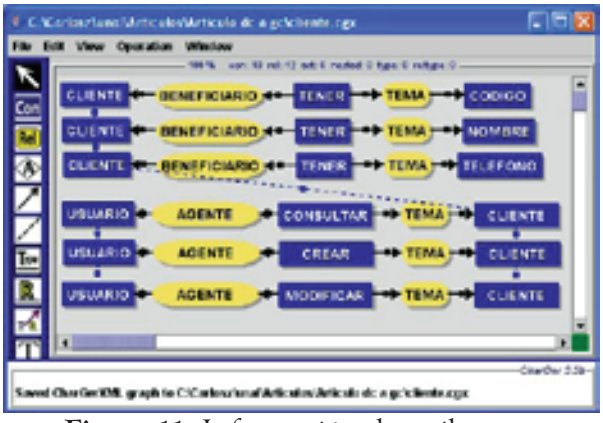

Figura 11. Información de atributos y operaciones de la clase cliente

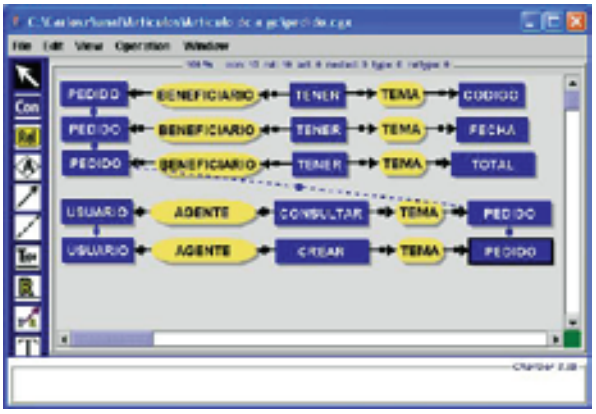

Figura 12. Información de atributos y operaciones de la clase pedido

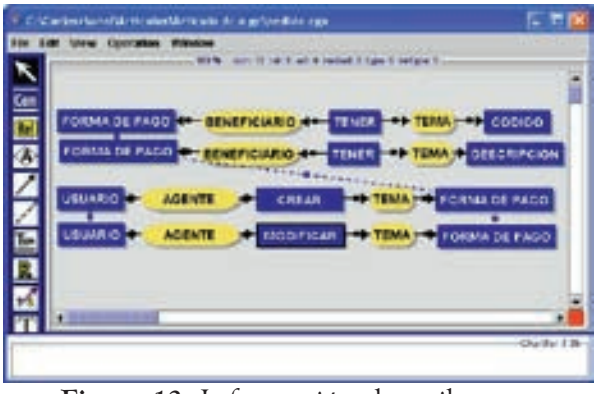

Figura 13. Información de atributos y operaciones de la clase forma de pago

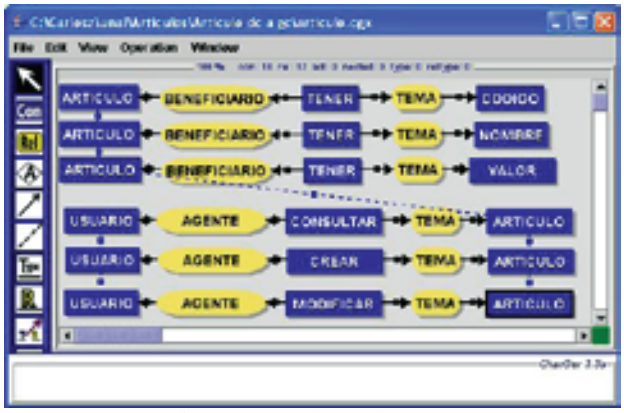

Figura 14. Información de atributos y operaciones de la clase artículo

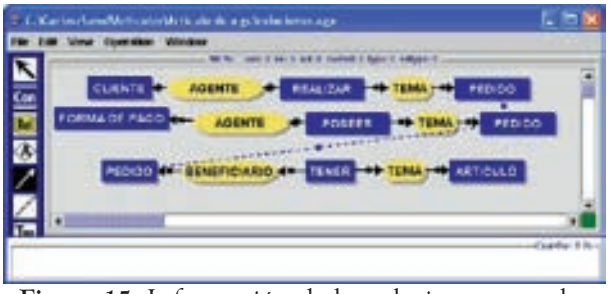

Figura 15. Información de las relaciones entre las clases

Nótese que en cada una de estas figuras se empleó una línea punteada para unir los rectángulos que se refieren a los mismos conceptos. Esta notación se adoptó a partir del estándar definido por Sowa (1984) para los grafos conceptuales.

CONCLUSIONES Y TRABAJOS FUTUROS

En este artículo se propuso un mecanismo para permitir la conversión de diagramas de clases a grafos conceptuales, esquemas de nivel más alto de abstracción y, consecuentemente, de más fácil comprensión por los interesados humanos; en estos grafos, sin embargo, no se renuncia a la posibilidad de comunicación entre máquinas debido a sus formas textuales equivalentes, que pueden ser legibles por máquina. Uno de los objetivos y motivaciones para realizar la conversión es la validación, por parte del interesado, de los esquemas conceptuales elaborados en las diferentes fases de desarrollo de un producto de software.

Se presentan cuatro reglas para definir la conversión a grafos conceptuales de las relaciones entre clases y sus atributos, clases y sus operaciones y clases con otras clases del diagrama de clases; estas reglas se aplican a un caso de estudio referente a un sistema de pedidos de artículos, para obtener los grafos conceptuales de Sowa.

Las reglas enunciadas pretenden un acercamiento con el lenguaje del interesado, a diferencia de otros trabajos del estado del arte, que se limitan a convertir a grafos de Sowa esquemas conceptuales, pero conservando el lenguaje del analista. Sin embargo, y pese a que la legibilidad del diagrama de clases expresada en términos de los grafos conceptuales mejora, se puede apreciar que la extensión de los diagramas es mucho mayor, dado que se sigue el estándar de correferencia enunciado por Sowa (1984).

Entre los trabajos a realizar se pueden enumerar los siguientes:

- La conversión de diagramas de clases más complejos que incluyan todos los tipos de relaciones definidas en la especificación de UML, además de la conversión a grafos conceptuales de otros esquemas comúnmente utilizados en la definición, análisis y diseño de sistemas informáticos.
- La complementación de la aplicación que implementa las reglas.
- La definición de un esquema de comunicación más cercano aun que los grafos conceptuales al lenguaje que utilizan los interesados, de forma que se pueda lograr una comunicación más rápida y efectiva que permita la validación del diagrama de clases.

BIBLIOGRAFÍA

ABRIAL, J. R. 1996. The B Book - assigning programs to meanings. Cambridge University Press (Cambridge).

COAD, P. & YOURDON, E. 1990. Object – Oriented analysis. Yourdon Press, Segunda Edición (New Jersey).

CREASY, P. & ELLIS, G. 1993. A conceptual graphs approach to conceptual schema integration. *Conceptual Graphs for Knowledge Representation: First International Conference on Conceptual Structures (ICCS).* Pp 126-141.

FILLMORE, Ch. 1968. The case for case. En: Bach and Harms (Eds), *Universals in Linguistics*. Pp. 1-88.

FLIEDL, G., KOP, Ch., MAYERTHALER, W., MAYR, H.C. & 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"*, París.

FLIEDL, G., KOP, Ch. & MAYR, H. C. 2003. From scenarios to KCPM dynamic schemas: aspects of automatic mapping. *Proceedings of the Natural Language Processing and Information Systems Conference NLDB'2003, Lecture Notes in Informatics* 29:91-105.

FLIEDL, G., KOP, Ch., MAYR, H. C., MAYERTHALER, W. & WINKLER, Ch. 1999. Linguistically based requirements engineering-The NIBA Project. *Proceedings 4th Int. Conference NLDB'99 Applications of Natural Language to Information Systems, Klagenfurt*. Pp. 177–182.

FLIEDL, G., MAYERTHALER, W. & WINKLER, Ch. 1999a. The NT(M)S-Parser: An efficient computational linguistic tool. *Proceedings of the 1st International Workshop on Computer Science and Information Technologies, Moscow*.

FLIEDL, G. & WEBER, G. 2002. Niba-Tag-A tool for analyzing and preparing german texts. *DataMining* Bologna.

GRUBER, J. 1965. Studies in lexical relations. *Disertación Doctoral*, Cambridge, MIT.

HARMAIN, H. & GAIZAUSKAS, R. 2000. CM-Builder: An automated NL-based CASE Tool. *Proceedings of the fifteenth IEEE International Conference on Automated Software Engineering (ASE'00)*, Grenoble. 9 p.

JACKENDOFF, R. 1983. Semantics and cognition. *MIT Press*, Cambridge Massachussets.

KOP, Ch. & MAYR, H. C. 2002. Mapping functional requirements: from natural language to conceptual schemata. *Proceedings of the 6th IASTED International conference Software Engineering and applications*. 5p.

MAYR, H. C. & KOP, Ch. 2002. A user centered approach to requirements engineering. *Proceedings of "Modellierung 2002"*, Lecture Notes in Informatics 12:75-86.

MICH, L. & GARIGLIANO, R. 2002. NL-OOPS: A Requirements analysis tool based on natural language processing. *Proceedings of the 3rd International Conference On Data Mining 2002*, Bologna. Pp. 321-330.

OBJECT MANAGEMENT GROUP. 2003. Unified modeling language, specification 2.0. formal. Available: http://www.omg.org/UML [Citado 7 de Marzo de 2006]

SNOOK, C. & BUTLER, M. 2001. Using a graphical design tool for formal specification. *In Dumke, Abran (Eds) Proceedings 13th Annual Workshop of the Psychology of Programming Interest Group*. Pp. 311-321.

SOWA, J. F. 1984. Conceptual structures: information processing in mind and machine. reading, Massachusetts, Addison-Wesley.

TIMM, J. T. E. & GANNOD, G. C. 2005. A Model-Driven approach for specifying semantic web services. *In proceedings of the 3rd IEEE International Conference on Web Services.*

ZAPATA, C. M. & ARANGO, F. 2005. Los modelos verbales en lenguaje natural y su utilización en la elaboración de esquemas conceptuales para el desarrollo de software: una revisión crítica. *Revista EAFIT*. 41(137):77-95.