UN MÉTODO PARA EL REFINAMIENTO INTERACTIVO DEL DIAGRAMA DE CLASES DE UML

A METHOD FOR INTERACTIVE REFINEMENT OF UML CLASS DIAGRAM

CARLOS M. ZAPATA
Grupo de Investigación UN-INFO. Escuela de Sistemas .Universidad Nacional de Colombia,cmzapata@unal.edu.co

BETSY MARY ESTRADA
Grupo de Investigación UN-INFO. Escuela de Sistemas. Universidad Nacional de Colombia, sede Medellín

FERNANDO ARANGO I
Grupo de Investigación UN-INFO. Escuela de Sistemas. Universidad Nacional de Colombia, sede Medellín

Recibido para revisar enero 15 de 2007, aceptado julio 24 de 2007, versión final julio 30 de 2007

RESUMEN: Durante el proceso de elicitación de requisitos se presentan problemas de comunicación entre analistas e interesados que suelen ocasionar pérdidas de requisitos funcionales. Estas pérdidas se aminoran mediante el refinamiento de los esquemas conceptuales, en particular el diagrama de clases de UML. Existen algunos acercamientos al refinamiento del diagrama de clases, pero que no realizan ciclos de interacción con el interesado; otros enfoques realizan refinamiento interactivo del diagrama entidad-relación, un diagrama que no posee toda la información contenida en el diagrama de clases. En este artículo se realiza el refinamiento del diagrama de clases de UML mediante la interacción con el interesado. Para ello, se proponen reglas de completitud que se disparan en lenguaje natural y se emplea un corpus de diagramas de clases para complementar el conocimiento del analista en un determinado dominio. El análisis de completitud propuesto se ilustra con un prototipo en la herramienta UNC-Diagramador y se ejemplifica con un caso de estudio.

PALABRAS CLAVE: Refinamiento, reglas de completitud, diagrama de clases de UML, corpus de diagramas.

ABSTRACT: Along the requirements elicitation process, communication problems appear between analysts and stakeholders; usually, these problems cause losing of functional requirements. Refinement of conceptual schemas—particularly class diagram—lessens the impact of these losses. Some approaches to the refinement of class diagram have been proposed, but they do not evidence cycles of interaction with the stakeholder; other approaches show interactive refinement of the Entity-Relationship diagram, which does not have all the information contained in class diagram. In this paper, we propose the refinement of UML class diagram through interaction with stakeholders. To achieve this goal, we propose completeness rules in natural language and the use of a corpus of class diagrams for complementing the analyst knowledge in a specific domain. Finally, we illustrate completeness analysis with a prototype in the UNC-Diagrammer tool and we propose a case study.

KEYWORDS: Refinement, Completeness rules, UML Class Diagram, diagram corpus.

1. INTRODUCCIÓN

Durante el proceso de elicitación de Requisitos del software se procura la captura y especificación de requisitos funcionales de los interesados (Leite, 1987). Con ello se busca que los esquemas conceptuales reflejen tales requisitos, y el producto final en realidad satisfaga las necesidades especificadas.

Sin embargo, en este proceso se presentan problemas de comunicación entre los analistas y los interesados, debido a la diferencia de conocimientos entre ellos: los interesados conocen el dominio del problema (el cual se suele expresar en lenguaje natural), en tanto que los analistas conocen el lenguaje técnico de modelamiento. Este problema de comunicación puede ocasionar pérdidas de requisitos funcionales, que impiden satisfacer las necesidades del interesado (Zapata y Arango, 2005).
Algunos autores proponen un chequeo de completitud de esquemas conceptuales como solución al problema de comunicación entre analistas e interesados. Estos chequeos suelen emplear tres tipos de reglas dependiendo de la información evaluada sobre el modelo conceptual: las reglas sintácticas, las semánticas y las pragmáticas (Lindland et. al, 1994, Shanks y Darke, 1998, Krogstie *et al*., 1995, Buchholz *et al.*, 1995).

Sin embargo, aún subsisten dos tipos de problemas:

En algunos de los análisis de completitud propuestos para los esquemas conceptuales no se tiene en cuenta al interesado.

Una de las propuestas que interactúa con el interesado realiza el análisis de completitud para el diagrama entidad-relación, diagrama que no contiene toda la información suministrada por el diagrama de clases (Buchholz *et. al.*, 1995).

En este artículo, se propone un método para el refinamiento interactivo del diagrama de clases de UML, con los siguientes elementos:

- Un conjunto de reglas de completitud sintáctica, semántica y pragmática para el análisis de diagramas de clases.
- Un corpus que incluye un conjunto de diagramas de diferentes dominios, con el fin de complementar la experiencia del analista.
- Un diálogo controlado con el interesado para realizar el refinamiento del diagrama.

El método se incluyó dentro de la herramienta UNC-Diagramador (Zapata *et al.*, 2006c), actualmente en desarrollo en la Escuela de Sistemas de la Universidad Nacional de Colombia, y se ejemplifica con un caso de estudio relacionado con un sistema de solicitud de artículos.
Este artículo tiene la siguiente estructura: en la Sección 2 se define la completitud de los modelos conceptuales; en la Sección 3 se introducen algunos trabajos realizados sobre completitud; en la Sección 4 se presenta la herramienta UNC-Diagramador y el corpus de diagramas; la Sección 5 expone el método propuesto; en la Sección 6 se ejemplifica el método con un caso de estudio y finalmente en la Sección 7 se presentan las conclusiones y trabajo futuro.

2. COMPLETITUD DE ESQUEMAS CONCEPTUALES

La completitud es uno de los elementos que determina la calidad de los esquemas conceptuales; se dice que un modelo es completo si contiene toda la información relevante y correcta del dominio. Cuando se evalúa la completitud de un modelo, se toman como referencia los requisitos del interesado (Snoeck *et al.*, 2003). El chequeo de completitud es una tarea difícil, ya que los interesados no siempre están familiarizados con la estructura de los esquemas conceptuales (Zowghi y Gervasi, 2003).

Algunos investigadores (Buchholz *et. al*., 1995, Moody *et al.*, 2003, McGregor, 1999, Lindland *et al.*, 1994, Shanks y Darke, 1998, Krogstie *et al*., 1995, Leung y Bolloju 2005, entre otros) han enfocado sus trabajos en la búsqueda de métodos para la obtención de modelos más completos, ya que un modelo más completo—especialmente en las primeras fases del proceso de desarrollo de una pieza de software—puede reducir el costo y tiempo total del producto final. Los esquemas conceptuales generados en la fase de análisis del proceso de desarrollo del software son los primeros modelos que ilustran la propuesta de solución al sistema; por tal razón, mientras más completos sean los modelos de la fase de análisis en relación con las especificaciones proporcionadas por el interesado, más apropiado será el sistema informático para las necesidades

detectadas, con menos tiempo y costos invertidos (Lindland *et al.*, 1994).

El diagrama de clases de UML muestra los objetos relevantes del dominio del problema, especificando para cada objeto sus atributos, comportamientos, relaciones y los roles que desempeña en cada relación. Además, a partir del diagrama de clases es posible crear la base de datos del sistema (Bergner *et al.*, 1999). Los esfuerzos para lograr la completitud del diagrama de clases cobran importancia para el diseño del software.
Una vez definidos los elementos de completitud requeridos por el diagrama de clases, es necesario utilizar mecanismos que permitan subsanar las incompletitudes identificadas. Este proceso se suele denominar Refinamiento, el cual, según D'Souza y Wills (1998), es la realización de una descripción más detallada a partir de una descripción más abstracta.
Para chequear la completitud de esquemas conceptuales es necesario evaluar una serie de reglas, que se clasifican según el tipo de información a evaluar en: sintácticas, semánticas y pragmáticas Lindland *et al.* (1994). En la próxima sección se exponen algunas propuestas para evaluar la completitud de los modelos conceptuales.

3. TRABAJOS REALIZADOS PARA EVALUAR LA COMPLETITUD DE DIAGRAMAS

La mayoría de las propuestas de análisis de completitud consisten en la evaluación de ciertas características y propiedades deseables para un diagrama específico. Por lo general, estas características son un listado de condiciones sintácticas que debe cumplir el diagrama evaluado. La evaluación se realiza desde tres puntos de vista: el sintáctico el semántico y el pragmático. Se emplean reglas sintácticas para inspeccionar la notación de cada una de las partes del diagrama y sus relaciones, tomando en consideración el lenguaje de modelado; si se quiere saber qué tan fiel y representativa es la información del esquema conceptual respecto a la información del mundo real, se emplean las reglas semánticas; por último, si se requiere incluir información necesaria en el modelo suministrada por la experiencia del analista y que facilite la interpretación del modelo, se recurre a las reglas pragmáticas (Lindland *et al.*, 1994)
El Object Management Group (OMG, 2006) presenta las especificaciones sintácticas y semánticas del lenguaje de modelado UML; en particular, se describe para cada uno de los elementos de un diagrama de clases, sus relaciones, su notación y sus representaciones opcionales. Esta especificación permite evaluar qué tan correcto es un diagrama de clases, verificando si cada uno de los elementos cumple con la notación y relaciones definidas. Las reglas de sintaxis son las más elementales a la hora de realizar un análisis de completitud; de hecho, algunas de ellas son chequeadas de forma implícita por las herramientas CASE (Computer-Aided Software Engineering), un conjunto de herramientas encargadas de la creación y edición de diagramas. La mayor desventaja que presenta la especificación definida para UML es que, aun cuando contiene restricciones semánticas, no contiene información que permita determinar qué tan correcto es un diagrama de clases pragmáticamente.

El método para la evaluación de modelos propuesto por Lindland *et al.* (1994) es aplicable a modelos conceptuales de diferentes tipos: modelos de datos, de procesos y de interacciones, entre otros. En esta propuesta la calidad de un modelo conceptual se define con base en cuatro componentes: el modelo M—conformado por el conjunto de declaraciones del dominio que han sido representadas—, el lenguaje L—caracterizado por las declaraciones posibles según la gramática del lenguaje de modelado—, el dominio D— definido como el conjunto de declaraciones correctas y relevantes sobre el dominio del problema— y la interpretación I—lo que piensan los interesados que expresa el modelo. Este método posee tres categorías de calidad. (Véase la Figura 1):

- Calidad sintáctica: definida como la relación entre el modelo y el lenguaje, dada por la ecuación: $M/L = \phi$ (No existen declaraciones incluidas en el modelo que no sean parte del conjunto de declaraciones del lenguaje).

- Calidad semántica: se entiende como la relación entre el modelo y el dominio, dada por las ecuaciones M/D = ϕ (No hay declaraciones del modelo que no sean correctas y relevantes en el dominio) y D/M = ϕ (No hay declaraciones correctas y relevantes sobre el dominio que no estén incluidas en el modelo); estas ecuaciones evalúan la validez y completitud del modelo respectivamente.
- Calidad pragmática: relaciona el modelo y la interpretación que pueda darse al modelo; esta relación está dada por las ecuaciones M/I = ϕ (No hay declaraciones en el modelo que no estén dentro del conjunto de interpretaciones que el interesado hace del modelo) y I/M = ϕ (No existen interpretaciones del interesado que no estén en el conjunto de declaraciones del modelo).

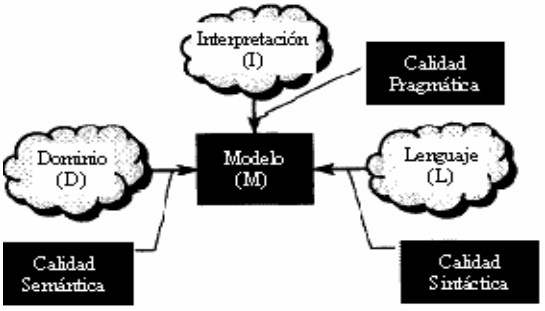

Figura 1. Marco de calidad de modelos conceptuales
Figure 1. Quality Framework for Conceptual Models

Este trabajo proporciona un marco de referencia para evaluar la calidad de los modelos conceptuales (incluyendo su completitud), pero no propone métodos para alcanzar la completitud sintáctica, semántica y pragmática de esos modelos.
En cuanto a la completitud del diagrama entidad-relación, se analizan dos propuestas diferentes: el proyecto RADD (Buchholz *et al.*, 1995) y la evaluación de calidad propuesta por Moody *et al.* (2003).

El proyecto RADD (Buchholz *et al.*, 1995), propone una herramienta de diálogo moderado (sistema de preguntas y respuestas) implementada en Prolog, para capturar el dominio de un sistema y de esta forma hace posible la construcción del diagrama entidad-relación. El primer paso para el diseño de la base de datos es la descripción en lenguaje natural (en este caso, Alemán) de la estructura de la aplicación, realizada por el diseñador. Sobre la descripción del sistema en lenguaje natural se aplican restricciones semánticas y con esta información se construye un diseño inicial de la base de datos. Este proyecto plasma el conocimiento del diseñador en una base de conocimientos, para analizar las condiciones sintácticas, semánticas y pragmáticas de la entrada en lenguaje natural y a partir de allí complementar el diseño inicial de la base de datos del sistema informático en consideración.

Moody *et al.* (2003) realizan un experimento para la evaluación de la calidad en los diagramas entidad-relación. El experimento se realizó con 192 analistas, con conocimientos en calidad de modelos, y se utilizaron diversas técnicas cualitativas y cuantitativas para evaluar diagramas entidad-relación desde el punto de vista sintáctico, semántico y pragmático. El objetivo del experimento era determinar si la evaluación era confiable y si era probable su uso para la evaluación de modelos de información en la práctica.
El trabajo con el diagrama entidad-relación deja de lado elementos importantes que se encuentran presentes en el diagrama de clases, tales como las operaciones, que podrían ser susceptibles de análisis de completitud. Además, el experimento de Moody *et al.* (2003) se basa en el conocimiento de los analistas y por ello no define explícitamente reglas para lograr la completitud de dicho diagrama.

Leung y Bolloju (2005) realizan un estudio enfocado a identificar los errores frecuentemente cometidos por analistas principiantes en la elaboración de esquemas conceptuales. En este estudio, los analistas desarrollan diagramas de clases para un dominio dado. Al final, se presentan los errores cometidos agrupados en diferentes categorías: errores sintácticos, semánticos y pragmáticos. Entre los principales errores encontrados se pueden mencionar: falta de nombres y cardinalidades para algunas

asociaciones, cardinalidades invertidas y pérdida de clases, atributos, operaciones y asociaciones respecto al dominio del problema suministrado.

Aunque esta lista de errores provee un punto de partida útil para entender y mejorar la calidad en el modelado conceptual, no se propone un mecanismo para disminuir tales errores en el proceso de elaboración de esquemas conceptuales.

McGregor (1999) propone una técnica para realizar pruebas de calidad sobre modelos de las fases de análisis y diseño del proceso de desarrollo del software. Además, reutiliza, refina y extiende la técnica para la realización de pruebas de código fuente. La técnica propuesta permite verificar el proceso de creación de los modelos y la validación de los mismos con base en los requisitos del proyecto. Según McGregor (1999), la completitud del diagrama de clases se alcanza cuando la información contenida en el modelo es suficiente para poner en ejecución las interfaces y para definir las pre y post condiciones de los métodos en el código fuente. Por lo tanto, para chequear la completitud es necesario tener como entradas el diagrama de clases, los requisitos, las interfaces y el código fuente; no todos estos elementos se encuentran completamente definidos en la fase de análisis del proceso de desarrollo de software.

Bergner *et al.* (1999) proponen el refinamiento del diagrama de clases a través de interfaces estructuradas. En el proceso de refinamiento se crea un diagrama de clases en la fase de diseño con base en el diagrama obtenido en la fase de análisis. Con tales diagramas puede suceder que:

- Algunas clases de la fase de análisis se conservan en la fase de diseño.
- Algunas clases de la fase de análisis se pueden perder en la fase de diseño.
- Algunas clases, atributos, operaciones y asociaciones se unen o se separan en la fase de diseño.
- Se generan nuevas clases, atributos, operaciones y relaciones en la fase de diseño.

Las primeras tres acciones están contenidas en la fase inicial del proceso de diseño; esta fase se centra en el desarrollo de un diagrama orientado a las especificaciones del usuario, dejando de lado los requisitos técnicos. La última acción está contenida en la segunda fase del proceso de diseño, donde se considera la infraestructura técnica que proporciona detalles de implementación. Las plantillas propuestas por esta metodología de refinamiento son utilizadas para capturar detalles de implementación sobre atributos, operaciones y relaciones. El diligenciamiento de tales plantillas es responsabilidad del usuario.
El refinamiento propuesto por Bergner *et al.* (1999) sólo automatiza el paso de la fase de diseño a la obtención del código. Además, las plantillas que se diligencian para automatizar el refinamiento contemplan información de carácter técnico, que en general no es del conocimiento del interesado.

Egyed (2002) presenta una abstracción automatizada de un diagrama de clases, basada en un conjunto de reglas y un algoritmo que describe la utilización de tales reglas. Define el proceso de abstracción como la simplificación de modelos, eliminando los detalles innecesarios en un nivel más abstracto. Inicialmente presenta tres tipos de refinamiento: lógico, físico y basado en objetivos, para explicar la necesidad del proceso de abstracción y para demostrar porqué la agrupación de clases no es suficiente para lograr la abstracción de un diagrama de clases. El refinamiento lógico consiste en agregar detalles a un conjunto de elementos de software; el refinamiento físico convierte sistemas de software en subsistemas, componentes, paquetes, clases y por último en líneas de código; en el refinamiento basado en objetivos los requerimientos son refinados en diseño o implementación. En el proceso de abstracción propuesto, se identifican patrones de entrada para un diagrama de clases en la fase de diseño y, de acuerdo con su información semántica, se relaciona con un patrón de salida más sencillo. Las reglas propuestas para lograr la abstracción del diagrama de clases son del tipo:

GeneralizaciónDerecha-Clase-AsociaciónDerecha => AsociaciónDerecha

Significa que una generalización que apunta hacia una clase y una asociación que sale de dicha clase se puede reemplazar con una asociación que va de la clase hija hasta la clase a la cual apuntaba la asociación inicial. (Véase la Figura 2).

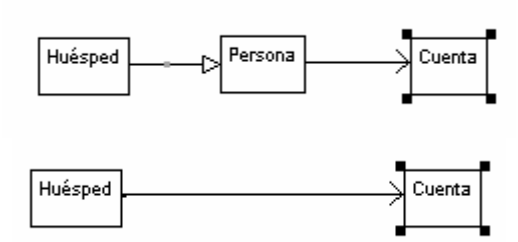

Figura 2. Un ejemplo de reglas de abstracción.
Figure 2. An example of abstraction rules.

La abstracción es el proceso inverso al refinamiento, lo que en general requiere que se posea el modelo refinado para obtener el modelo abstracto, lo cual en las fases iniciales de desarrollo del software no es posible. Por lo tanto, el trabajo de Egyed (2002) proporciona un punto de referencia para la identificación de posibles reglas para el proceso de refinamiento.

"relación" poseen la información de los elementos que hacen parte de los diagramas de clases, casos de uso y máquina de estados, y se podría ampliar para registrar otros tipos de diagramas. Los diagramas se construyen inicialmente en la herramienta CASE Poseidon® y se exporta en un archivo en formato XMI al UNC-Corpus, el cual almacena la información en las siguientes entidades:

- "contexto" contiene el nombre del dominio sobre el cual están definidos los diagramas.
- "archivo" posee el nombre del archivo en formato XMI que da origen a la información.
- "instancia" es el nombre del conjunto de diagramas que se van a cargar en UNC-Corpus.
- "instanciaxArchivo", "instanciaPrimitiva" e "instanciaRelación" contienen la información específica de cada uno de los diagramas, discriminada por sus diferentes elementos.
- "modoConsulta" es una entidad que permite almacenar diferentes consultas en SQL para acceder luego a la información contenida en el UNC-Corpus.

Teniendo en cuenta las definiciones para cada tipo de reglas, se puede concluir que es posible evaluar las reglas sintácticas directamente sobre el esquema conceptual; sin embargo, las reglas semánticas y pragmáticas requieren información adicional que simule la experiencia acumulada del analista. En el caso de la propuesta que se presenta en este artículo, esa experiencia se simulará mediante una interacción entre las herramientas UNC-Corpus y UNC-Diagramador, que se describen en la sección siguiente.

4. UNC-CORPUS Y UNC-DIAGRAMADOR

4.1 UNC-Corpus

UNC-Corpus es una herramienta que actualmente está desarrollando el grupo de Ingeniería de Software de la Escuela de Sistemas de la Universidad Nacional. Esta aplicación está diseñada para almacenar la información de diferentes diagramas en una base de datos relacional, de la cual se muestra el diagrama entidad-relación en la Figura 3. En esta Figura, las entidades "diagrama", "primitiva" y

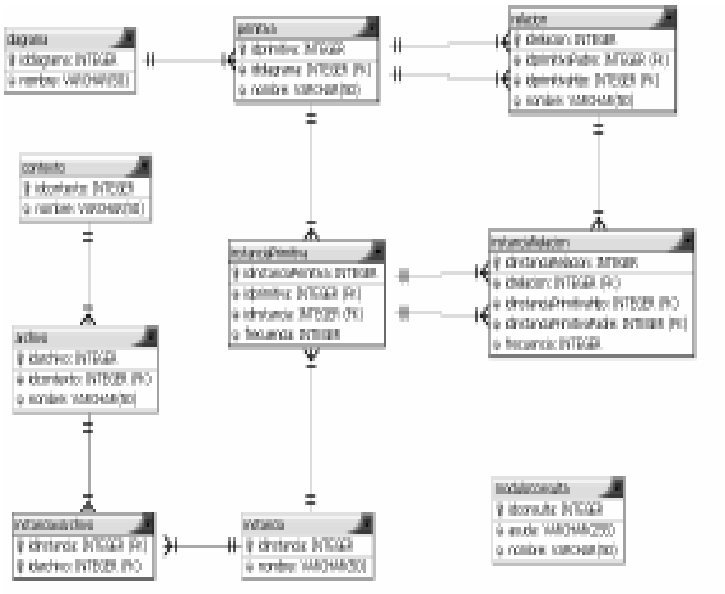

Figura 3. Diseño de la base de datos del UNC-Corpus
Figure 3. UNC-Corpus database design

Específicamente, para el diagrama de clases se extraen las siguientes primitivas: clases, atributos, operaciones, parámetros de entrada y de retorno de las operaciones y tipos de datos. Algunas de las relaciones entre primitivas almacenadas en el corpus son: agregación, composición, dependencia y generalización que son relaciones entre clases; también existen las relaciones "tiene atributo" y "tiene operación" que son relaciones entre clase-atributo y clase-operación respectivamente.

En el contexto de este artículo, el UNC-Corpus se empleará para simular la experiencia del analista, extrayendo información semántica y pragmática de un dominio específico.

4.2 UNC-Diagramador

El Grupo de Ingeniería de Software de la Escuela de Sistemas viene desarrollando una herramienta CASE denominada UNC-Diagramador. El entorno seleccionado para el desarrollo es .NET con el lenguaje C#; para la parte gráfica se utiliza el Microsoft Visio®.El UNC-Diagramador se basa, para su funcionamiento, en los denominados esquemas preconceptuales (Zapata *et al.*, 2006a) y el UN-Lencep (Universidad Nacional de Colombia-Lenguaje Controlado para la Especificación de Esquemas Preconceptuales) (Zapata *et al.*, 2006b). Los Esquemas Preconceptuales son grafos que representan los diferentes elementos de la descripción verbal suministrada por el interesado sobre un sistema (véase la Figura 4).

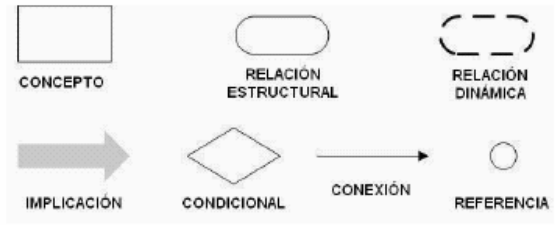

Figura 4. Especificación Básica de los Esquemas Preconceptuales
Figure 4.Basic specification of the Pre-conceptual Schemes

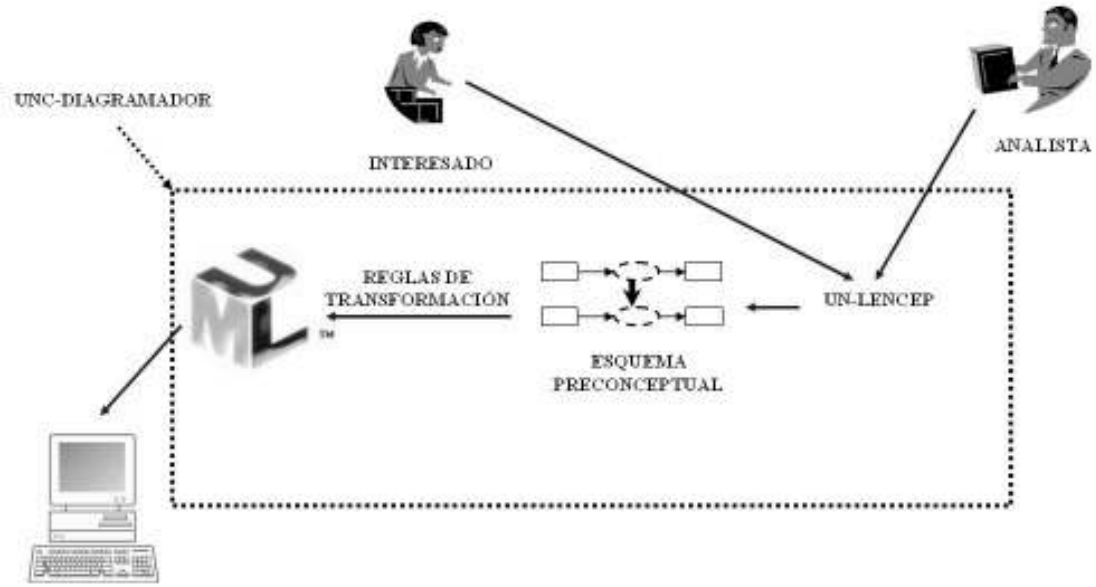

Figura 5. Especificación Básica de UN–Lencep y su traducción a Esquemas Preconceptuales
Figure 5. Basic specification of UN-Lencep and traslation to Pre-conceptual Schemes

El proceso realizado por el UNC-Diagramador se esquematiza en la Figura 5 y se describe de la siguiente manera: el UNC-Diagramador recibe como entrada un archivo con expresiones en UN-Lencep y a partir de él obtiene los Esquemas Preconceptuales. Posteriormente, aplicando las reglas de transformación, permite obtener de manera automática tres diagramas de UML: Clases, Comunicación y Máquina de estados (Zapata *et al.*, 2006b).

En el contexto de este artículo, el UNC-Corpus se empleará para simular la experiencia del

analista, revisando los diagramas almacenados en su base de datos, con el fin de complementar la información semántica y pragmática de un dominio específico.

5. REGLAS QUE PERMITEN EVALUAR LA COMPLETITUD DE ESQUEMAS CONCEPTUALES

En la Sección 2 se definió la completitud a partir de tres puntos de vista: sintáctico, semántico y pragmático. En esta Sección se definen las reglas para la evaluación de la completitud y se muestra su implementación en el UNC-Diagramador, empleando el lenguaje C#. Para el punto de vista sintáctico, el resultado de la aplicación de las reglas es un conjunto de archivos de texto que contiene los nombres de los elementos que no cumplen con las reglas; para el punto de vista semántico y pragmático, estos archivos se retoman para realizar las comparaciones necesarias con el UNC-Corpus y generar un diálogo interactivo con el interesado.

5.1 Reglas Sintácticas de Completitud

Desde el punto de vista sintáctico, se procura que el diagrama elaborado cumpla con las notaciones establecidas para cada uno de sus elementos y las relaciones entre ellos. Cada modelo específico utiliza un conjunto definido de símbolos, siguiendo unas reglas de su sintaxis; el cumplimiento de estas reglas determina si la representación de un diagrama es correcta (Lindland *et al.*, 1994, Shanks y Darke, 1998). Algunas de estas reglas son chequeadas por el lenguaje de modelado utilizado al graficar tal diagrama. Para evaluar la completitud del diagrama de clases en relación con su sintaxis, se incluyen algunas de las reglas y restricciones sintácticas expresadas en la especificación de UML para tal diagrama (OMG, 2006, Lange, 1999). Las reglas sintácticas que se consideran para esta propuesta son:

- Toda clase posee atributos heredados o propios.
  ownedAttribute : Property
  Hace referencia al atributo poseído por una clase. Los atributos son una de las características de las clases.

- Toda clase posee operaciones heredadas o propias.
  ownedOperation : Operation
  Hace referencia a la operación poseída por una clase. Las operaciones son una de las características de las clases.

- Toda relación (asociación o dependencia) posee una etiqueta. La flecha de una asociación o dependencia se etiqueta con un nombre (véase la Figura 6).

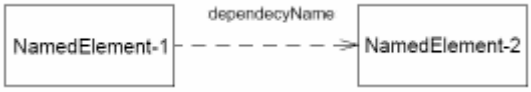

Figura 6. Símbolos para representar una dependencia entre dos elementos (Tomada de OMG, 2006)

Figure 6. Symbols for representing a dependency between two elements (Taken from OMG, 2006).

La implementación de la regla "Toda clase posee atributos heredados o propios" se basa en un conteo que se realiza sobre todas las clases del diagrama y que entrega un mensaje de error cuando el conteo es igual a cero. El código en C# es el siguiente:

```
if (clase.Atributos.Count<= 0)
 {archivodescripciones.WriteLine("El concepto " +
clase.Nombre + " no tiene características definidas");
 archivo2.WriteLine(clase.Nombre); }
if (here.Inicio.Atributos.Count<= 0)
 {  if (here.Fin.Atributos.Count<= 0)
{archivodescripciones.WriteLine(here.Inicio.Nombre + "
que es un ejemplo de " + here.Fin.Nombre + " no tiene
atributos");
archivo2.WriteLine(here.Inicio.Nombre);
 } }
```

A través de otro conteo sobre las clases, se verifica que toda clase tenga operaciones propias o heredadas. La implementación de la regla en C# es la siguiente:

```
if (clase.Operaciones.Count<= 0)
 {archivodescripciones.WriteLine("El concepto " +
clase.Nombre + " no tiene funcionalidad definida");
 archivo3.WriteLine(clase.Nombre); }
if (here.Inicio.Operaciones.Count<= 0)
{if (here.Fin.Operaciones.Count<= 0)
{archivodescripciones.WriteLine(here.Inicio.Nombre + "
que es un ejemplo de " + here.Fin.Nombre + " no tiene
operaciones");
archivo3.WriteLine(here.Inicio.Nombre); } }
```

5.2 Reglas Semánticas De Completitud

Las reglas semánticas miden qué tan fiel es el esquema conceptual en relación con el significado del modelo verbal. Si un modelo contiene elementos que no tienen correspondencia con la problemática del sistema modelado, entonces se dice que este modelo es inválido; en caso contrario, se dice que el modelo está incompleto (Krogstie *et al*., 1995, Lindland *et al.*, 1994). Teniendo en cuenta los resultados del experimento de Leung y Bolloju (2005), se seleccionaron las siguientes reglas semánticas para evaluar la completitud de un diagrama de clases:

- Cada concepto relevante del dominio debe corresponder a una clase o atributo del diagrama de clases
- Cada relación funcional del dominio debe corresponder a una relación entre clases del diagrama o a una operación de clase.
- Cada relación de generalización debe ir de la clase "padre" a la clase "hijo".
- Cada relación de agregación y composición debe ir de la clase "parte" a la clase "todo".
- Toda clase posee atributos y operaciones de acuerdo con el dominio.

En el proceso de elaboración de los diagramas esquematizado en la Figura 5, la generación del diagrama de clases parte del discurso en UN-Lencep y emplea las reglas de transformación definidas en Zapata *et al.* (2006a). Por ello, el cumplimiento de las reglas semánticas de completitud está a cargo del UNC-Diagramador en su proceso de generación, sin que por ello haya que definir reglas de completitud adicionales.

5.3 Reglas Pragmáticas De Completitud

El tercer grupo de reglas, las pragmáticas, incluye información que surge de la experiencia del analista y que facilita la interpretación del modelo. Las reglas pragmáticas que se consideran en esta propuesta emplean el UNC-Corpus, para revisar los diagramas de clases allí existentes y obtener:

- Atributos y Operaciones para las clases pertenecientes al diagrama y que se encuentren en el UNC-Corpus.
- Etiquetas para relaciones sin nombre en el diagrama evaluado
- Clases relacionadas y nombre de relaciones para una clase desconectada en el diagrama evaluado
- La orientación de las generalizaciones, agregaciones y composiciones del diagrama evaluado

Para cada regla de completitud pragmática, se define una consulta a realizar sobre el UNC-Corpus. Por ejemplo, si existe en el diagrama evaluado una clase que no tiene atributos ni propios, ni heredados, se buscan en la base de datos del UNC-Corpus todos los atributos para dicha clase. Tomando como base el diagrama entidad-relación de la Figura 3, se emplean las entidades *instanciaRelación, instanciaPrimitiva* e *instancia*, con el fin de determinar todos los elementos de las primitivas del tipo "clase" (identificadas con el número 3) y que poseen una relación "tiene atributo" (identificada con el número 6). En *nombre_de_clase* va cada una de las clases que no cumplen la regla. La consulta SQL que permite sugerir atributos a las clases que no posean ninguno, es la siguiente:

```
SELECT instancia.nombre,
instanciarelacion.idinstanciaprimitivahijo
FROM instancia, instanciarelacion
WHERE instanciarelacion.idrelacion=6 and
instanciarelacion.idinstanciaprimitivapadre=
   (SELECT idinstanciaprimitiva
   FROM instanciaprimitiva
   WHERE idprimitiva=3 and idinstancia=
      (SELECT idinstancia
      FROM instancia
      WHERE nombre='nombre_de_clase')
      and instancia.idinstancia=
      (SELECT instanciaprimitiva.idinstancia
      FROM instanciaprimitiva
      WHERE
      idinstanciaprimitiva=idinstanciaprimitivahijo))
```

De forma similar, se define la consulta para sugerir operaciones a las clases que no las tengan, sólo que en este caso el número 7 corresponde a la relación "tiene operación". Para

las clases que no tienen operaciones la consulta SQL es:

```
select instancia.nombre,
instanciarelacion.idinstanciaprimitivahijo from instancia,
instanciarelacion where instanciarelacion.idrelacion=7
and instanciarelacion.idinstanciaprimitivapadre=(select
idinstanciaprimitiva from instanciaprimitiva where
idprimitiva=3 and idinstancia=(select idinstancia from
instancia where nombre='nombre_de_clase')and
instancia.idinstancia=(select instanciaprimitiva.idinstancia
from instanciaprimitiva where
idinstanciaprimitiva=idinstanciaprimitivahijo))
```

5.4 Proceso De Aplicación De Las Reglas

El proceso que siguen en UNC-Diagramador las consultas implementadas a partir de las reglas de completitud mencionadas, se puede sintetizar así:

- Cada regla genera un archivo de texto plano que contiene un listado de elementos del diagrama de clases que disparan la regla.
- Estos archivos se envían al UNC-Corpus junto con la consulta SQL definida para cada regla.
- El UNC-Corpus devuelve la información almacenada en su base de datos de acuerdo con los datos enviados por el UNC-Diagramador.
- El UNC-Diagramador despliega algunas interfaces donde solicita información adicional al interesado con base en la información arrojada por el UNC-Corpus.
- La información ingresada por el interesado es convertida en expresiones de UN-Lencep y agregada al archivo UN-Lencep del diagrama evaluado.
- A partir de este archivo se obtiene el esquema preconceptual aumentado con los elementos que se aceptan y luego se realiza la generación automática de un diagrama de clases más completo que el anterior.

5.5 Comparación Con El Estado Del Arte

La solución incluida en el UNC-Diagramador y que hace uso de la información consignada en el UNC-Corpus posee las siguientes diferencias con los trabajos analizados en la Sección 3:

- Esta solución se encarga de la completitud del diagrama de clases, que en general posee algunos elementos que no posee el diagrama entidad-relación, que se trabaja en RADD (Buchholz *et al.*, 1995) y Moody *et al.* (2003).
- A diferencia de OMG (2006) y Leung y Bolloju (2005), el entorno propuesto considera reglas pragmáticas.
- El entorno propuesto sólo requiere el diagrama que se va a refinar, y en esta característica supera el trabajo de McGregor (1999) y Egyed (2002), quienes requieren mayor información para realizar el chequeo de completitud. En el caso de McGregor (1999) se requiere una expresión textual de los requisitos y el código fuente de la aplicación. En el caso de Egyed (2002) se requieren tanto el diagrama por refinar como el diagrama refinado.
- Lindland *et al.* (1994), McGregor (1999), Leung y Bolloju (2005) y Bergner *et al.* (1999) son trabajos dirigidos al diagrama de clases, pero que no implementan el esquema de completitud que definen, a diferencia del que se presenta en este artículo, que se ha incorporado en la herramienta CASE UNC-Diagramador.

6. CASO DE ESTUDIO

Algunas reglas de completitud propuestas en la Sección 5 se evalúan sobre un diagrama de clases, utilizando las herramientas UNC-Diagramador y UNC-Corpus. Para el ejemplo, se parte de una especificación en UN-Lencep que arroja el diagrama de clases de un sistema de solicitud de artículos, tal como se muestra en la Figura 7.

En esta Sección, se ilustra el uso de las reglas sintácticas “toda clase tiene atributos propios o heredados” y “toda clase tiene operaciones propias o heredadas”. Utilizando el UNC-Diagramador y el UNC-Corpus se ejemplifican además, la regla semántica “Toda clase posee atributos y operaciones de acuerdo con el dominio” y las reglas pragmáticas que permiten obtener los atributos y operaciones de las clases para el diagrama evaluado.

Para el diagrama seleccionado, las clases que disparan las reglas sintácticas ejemplificadas se muestran en las Figuras 8 y 9. El conjunto de

incompletitudes sintácticas se resumen en el archivo descripciones que aparece en la Figura 10.

Las clases del diagrama son enviadas al UNC-Corpus junto con la consulta SQL definida para cada regla. El UNC-Corpus envía una respuesta al UNC-Diagramador, el cual la presenta en una interfaz donde el interesado puede seleccionar la información que completará el diagrama de clases inicial. En esta primera parte, la interfaz muestra los atributos de las clases almacenadas en el corpus, cuyos nombres coinciden con el de las clases del diagrama de clases evaluado (véase la Figura 11). De igual forma, una segunda interfaz muestra las operaciones de las clases almacenadas en el corpus cuyos nombres coinciden con el de las clases del diagrama evaluado (Véase la Figura 12).

A partir de estas interfaces, el interesado puede seleccionar la información faltante. Por cada elemento seleccionado se envía una frase al archivo UN-Lencep del diagrama evaluado. Si el elemento seleccionado es un atributo, entonces la frase tendrá la siguiente estructura:

"ST nombre_de_clase tiene nombre_de_atributo"

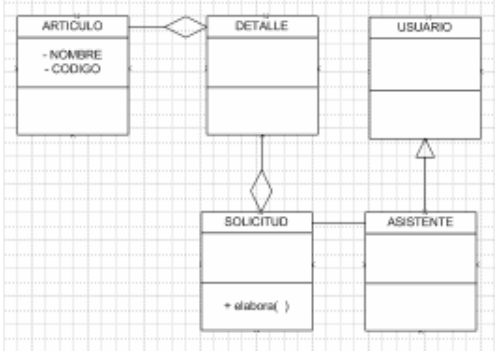

Figura 7. Diagrama de clases resultante
Figure 7. Resulting class diagram

Figura 8. Clases que no cumplen la regla "toda clase tiene atributos propios o heredados"
Figure 8. Non-fulfilling classes for the rule "all class have its own or inherited attributes"

Figura 9. Clases que no cumplen la regla "toda clase tiene operaciones propias o heredadas"
Figure 9. Non-fulfilling classes for the rule "all class have its own or inherited operations"

Figura 10. Archivo de incompletitudes sintácticas
Figure 10. Syntactic incompleteness file

donde ST significa frase estructural

Si se selecciona el atributo identificación para la clase solicitud, la frase enviada será:

"ST solicitud tiene identificacion"

Ahora, por cada operación seleccionada de la segunda interfaz es necesario preguntar al interesado por quién es realizada. Para esta regla, la frase enviada al archivo UN-Lencep es de la forma:

"RD quién_realiza_la_operación nombre_de_operación nombre_de_clase"

donde RD significa relación dinámica.

Si se selecciona la operación consultar estado para la clase solicitud, aparece la interfaz mostrada en la Figura 13. Los actores se toman de un archivo de texto generado por UNC-Diagramador que contiene los actores que puede identificar del esquema preconceptual generado.

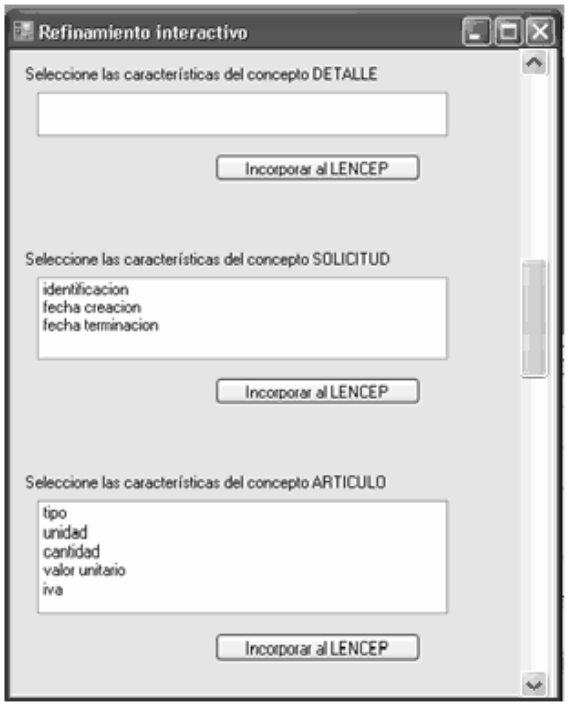

Figura 11. Resultados de la búsqueda de atributos en UNC-Corpus, para las clases con problemas de completitud.
Figure 11. Results of the search for attributes in UNC-Corpus related to incomplete classes

Una vez el interesado define el actor de la operación se envía al UN-Lencep la frase:

"RD secretaria consultar_estado solicitud"

A partir del archivo UN-Lencep y del esquema preconceptual se elabora un diagrama de clases más completo, donde aparecen los elementos seleccionados por el interesado.

7. CONCLUSIONES Y FUTUROS TRABAJOS

En este artículo se propone un método para el refinamiento interactivo del diagrama de clases de UML, basado en un análisis de completitud compuesto por reglas sintácticas, semánticas y pragmáticas, implementadas en el UNC-Diagramador; se usa también información de diferentes dominios almacenada en la aplicación UNC-Corpus.

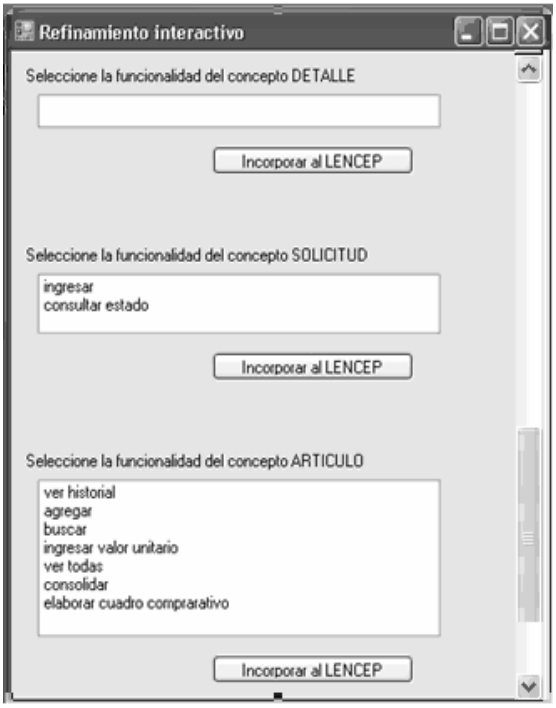

Figura 12. Resultados de la búsqueda de operaciones en UNC-Corpus, para las clases con problemas de completitud.
Figure 12. Results of the search for operations in UNC-Corpus related to incomplete classes

Figura13. Interfaz para que el interesado seleccione el actor de la operación
Figure 13. Interface for selecting the operation's actor by stakeholder

El método desarrollado se presenta como una solución al problema de incompletitudes en esquemas conceptuales generado por las deficiencias de comunicación entre usuarios y analistas. La interacción con el interesado es una ventaja de este método, pues él mismo suministra la información faltante, después de presentarle las posibles omisiones del modelo en un lenguaje familiar.

Otros trabajos sobre completitud de modelos conceptuales presentan marcos que reflejan la noción de completitud a través de un listado de características deseadas para el modelo, pero no definen un método para alcanzar tal completitud, o trabajan sobre la completitud de otros diagramas que no contienen toda la información del diagrama de clases de UML.

Como trabajo futuro se plantea la implementación de las reglas de completitud propuestas en este artículo y que aún no han sido codificadas. Vale la pena además abordar el tema de completitud y refinamiento interactivo de otros esquemas conceptuales diferentes al diagrama de clases.

8. AGRADECIMIENTOS

Este artículo se realizó en el marco del proyecto de Investigación "CONSTRUCCIÓN AUTOMATICA DE ESQUEMAS CONCEPTUALES A PARTIR DE LENGUAJE NATURAL" financiado por la DIME (Dirección de Investigación de la Sede Medellín– Universidad Nacional de Colombia).

REFERENCIAS

[1] BERGNER, K.; RAUSCH, A.; SIHLING, M.; VILBIG, A. Structuring and Refinement of Class Diagrams. Proceedings of the 32nd Hawaii International Conference on System Sciences. Alemania. 1999

[2] BUCHHOLZ, E., CYRIAKS, H., DÜSTERHÖFT, A., MEHLAN, H. Y THALHEIM, B. Applying a Natural Language Dialogue Tool for Designing Databases. Proceedings of the First International Workshop on Applications of Natural Language to Databases. 15p. 1995.

[3] D'SOUZA, D. Y WILLS, A. Objects, Components, and Frameworks with UML: The Catalysis(SM) Approach. Addison Wesley, 1998. 816 p.

[4] EGYED, A. Automated abstraction of class diagrams. ACM Trans. Softw. Eng. Methodol. 11(4): 449-491 (2002)

[5] LANGE, C; CHAUDRON, M. An Empirical Assessment of Completeness in UML Designs. 1999.

[6] LEITE J., "A survey on requirements analysis". Advanced Software Engineering Project Technical Report RTP-071. Department of Information and Computer Science. University of California at Irvine. 1987.

[7] LEUNG, F., BOLLOJU, N. Analyzing the Quality of Domain Models Developed by Novice Systems Analysts. Proceedings of the 38th Hawaii International Conference on System Sciences. 2005

[8] KROGSTIE, J., LINDLAND O. Y SINDRE, G. "Towards a Deeper Understanding of Quality in Requirements Engineering", CAISE 1995, Jyvaskyla, Finland, pp. 82-95, 1995.

[9] LINDLAND, O.I., SINDRE, G. AND SØLVBERG, A. Understanding quality in conceptual modelling, IEEE Software, Vol. 11, No. 2, March 1994, 42-49.

[10] MCGREGOR J. Using Guided Inspection to Validate UML Models. Santa Barbara, California. 1999

[11] MOODY, D.L. SINDRE, G. BRASETHVIK, T. SOLVBERG, A. " Evaluating the quality of information models: empirical testing of a conceptual model quality framework" Software Engineering, 2003. Proceedings. 25th International Conference, pp. 295- 305, 2003.

[12] Object Management Group. OMG Unified Modeling Language Specification. Object Management Group. Available: http://www.omg.org/UML/. [Citado 30 de Noviembre de 2006].

[13] SHANKS, G. AND DARKE, P. Understanding Metadata and Data Quality in a Data Warehouse, Australian Computer Journal (November). 1998.

[14] SNOECK, M., MICHIELS, C. AND DEDENE, G. Consistency by Construction: The Case of MERODE. Conceptual Modeling for Novel Application Domains, in: Lecture Notes in Computer Science, Volume 2814, p 105-117. Berlin: Springer. Septiembre 2003

[15] ZAPATA, C. M., ARANGO, F. 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. Vol 41, 2005, No 137, pag 77-95.

[16] ZAPATA, C. M., GELBUKH, A. Y ARANGO, F. "Pre-conceptual Schema: a UML Isomorphism for Automatically Obtaining UML Conceptual Schemas", Research in Computing Science: Advances in Computer Science and Engineering, Vol. 19, 2006a, pp. 3–13.

[17] ZAPATA, C. M., GELBUKH, A. Y ARANGO, F. UN–Lencep: Obtención Automática de Diagramas UML a partir de un Lenguaje Controlado. Memorias del Taller de Tecnologías del lenguaje del Encuentro Nacional de Computación, San Luis Potosí, México, 2006b.

[18] ZAPATA, C. M., GELBUKH, A. Y ARANGO, F. Pre-conceptual Schema: A Conceptual-Graph-Like Knowledge Representation for Requirements Elicitation. Lecture Notes in Computer Science, Vol. 4293, 2006c, pp. 17-27

[19] ZOWGHI, D., GERVASI, V. On the Interplay Between Consistency, Completeness, and Correctness in Requirements Evolution. Elsevier Science. 1 April 2003.