Entre Ciencia e Ingeniería, ISSN 1909-8367
Año 4. No. 7 – Primer semestre de 2010, páginas 47 - 61

Representación de las primitivas conceptuales de UML 2.2 en lógica de predicados de primer orden[1]

Representation of the concept of primitive logic UML 2.2 in first order predicate

Carlos Mario Zapata Jaramillo
Doctor en Ingeniería – Sistemas
Magíster en Ingeniería de Sistemas
Especialista en Gerencia de Sistemas Informáticos
Ingeniero Civil
Docente Universidad Nacional de Colombia Sede Medellín
Grupo de Investigación en Lenguajes Computacionales de la Escuela de Sistemas, Facultad de Minas
cmzapata@unal.edu.co

Paula Andrea Tamayo Osorio
Estudiante de Doctorado en Ingeniería – Sistemas
Magister en Ingeniería – Ingeniería de Sistemas
Ingeniería de Sistemas
Docente Universidad Nacional de Colombia, sede Medellín
Grupo de Investigación en Lenguajes Computacionales de la Escuela de Sistemas, Facultad de Minas
patamayo@unal.edu.co

Nathalia Meneses Piedrahita
Estudiante Ingeniera de Sistemas e Informática
Estudiante Universidad Nacional de Colombia, sede Medellín
Grupo de Investigación en Lenguajes Computacionales de la Escuela de Sistemas, Facultad de Minas
nmenesep@unal.edu.co

Recibido Marzo 10 de 2010 – Aceptado Mayo 21 de 2010

SINTESIS

La Ingeniería de Software utiliza modelos de procesos de desarrollo de software y un conjunto de técnicas y metodologías para definir, analizar y diseñar sistemas de información. Una de esas técnicas es el Lenguaje Unificado de Modelado (UML). UML se caracteriza por ser un lenguaje semiformal, generando problemas de ambigüedad,

[1] Producto derivado del Proyecto de Investigación "OBTENCIÓN SEMIAUTOMÁTICA DE PROTOTIPOS A PARTIR DE LOS ESQUEMAS CONCEPTUALES OBTENIDOS EN UNC-DIAGRAMADOR", proyecto avalado por la Vicerrectoría de Investigación, Universidad Nacional de Colombia.

claridad y consistencia. Algunos investigadores intentan formalizarlo mediante Lógica de predicados de primer orden, teoría de conjuntos, lenguajes controlados y/o restringidos y metamodelado; sin embargo, estos acercamientos no son suficientes debido a que se suelen enfocar en un solo diagrama y únicamente en la sintaxis, dejando de lado la semántica. En este artículo se presenta un conjunto de reglas que permiten la representación de las primitivas conceptuales de UML mediante lógica de predicados de primer orden, así como un caso de estudio, partiendo de una descripción de los requisitos de un sistema en el lenguaje controlado UN-Lencep.

Descriptores: Lenguaje Unificado de Modelado, primitivas conceptuales de UML, lógica de predicados de primer orden, representación formal, UN-Lencep.

ABSTRACT

Software Engineering uses software development process models and a set of techniques and methodologies for identifying, analyzing and designing information systems. The Unified Modeling Language (UML) is one of such techniques. UML, as a semi-formal language, exhibits problems of ambiguity, clarity, and consistency. Some researchers are attempting to formalize UML by using first order predicate logic (FOPL), set theory, controlled and/or restricted languages, and meta-modeling. However, these approaches have been not enough because they focus on a single chart and merely in syntax, leaving out semantics. In this paper we present a set of FOPL-based rules for representing UML constructs, and a case study, based on UN-Lencep requirement description of a system.

Descriptors: Unified Modeling Language, UML constructs, first order predicate logic, formal representation, UN-Lencep.

1. INTRODUCCIÓN

La Ingeniería de Software define un conjunto de actividades para transformar las especificaciones de los requisitos de un interesado en un sistema informático. Uno de los métodos más conocidos para ello es el Proceso Unificado de Desarrollo RUP (Jacobson *et al.*, 2001). Para la elaboración de los esquemas conceptuales del sistema informático, se suele utilizar el Lenguaje Unificado de Modelado (UML), que contiene un conjunto de primitivas conceptuales que permiten describir de forma abstracta todos los aspectos funcionales de una aplicación. UML es un lenguaje semiformal y proporciona un conjunto de gráficas y técnicas de descripción textual “intuitivas”, las cuales no se definen claramente. Como consecuencia, el uso de estas técnicas y la interpretación de los modelos desarrollados pueden diferir considerablemente entre los analistas (Brean *et al.*, 1997).

En la actualidad, existen diversos proyectos que realizan intentos de formalizar los diagramas utilizados en UML para proporcionarle una sintaxis y semántica precisa (Steimann y Kühne, 2002; Moreno, 1997; Moreno y Van De Riet, 1997; Shroff y France, 1997). Sin embargo, aún subsisten algunos problemas:

- Se centran en uno de los diagramas, generalmente en el Diagrama de Clases.
- La especificación de los elementos conceptuales se realiza en lenguajes semiformales como CORE, que utiliza Teoría de Conjuntos.

- En algunas especificaciones se integran aspectos estáticos y dinámicos, pero el conjunto de primitivas conceptuales trabajadas es mínimo y lo componen los elementos más conocidos de UML.
- Algunas primitivas conceptuales son comunes en varios diagramas, lo cual no se refleja en los proyectos actuales, ya que se centran en uno o dos.

Para solucionar parcialmente estos problemas, este artículo propone una representación en lógica de predicados de primer orden de las primitivas conceptuales de UML y un conjunto de reglas para obtenerlas a partir de la especificación textual de requisitos de un sistema expresados en UN-Lencep. Además, se presenta un caso de estudio que ejemplifica esta formalización. La lógica de predicados de primer orden (LPPO) se utiliza como mecanismo de formalización y se define como la parte de la lógica en la que se unen el cálculo proposicional y el cálculo de predicados, con la cual es posible evaluar expresiones utilizando términos, operadores, reglas de inferencia y cuantificadores (Ramírez y Cabrera, 2006). Entre las ventajas de la lógica de predicados se encuentra:

- Permite formalizar objetos, relaciones y restricciones.
- Proporciona un lenguaje artificial no ambiguo, riguroso y preciso.
- Es la base de lenguajes de especificación y diseños de métodos formales para realizar la notación de UML (OMG, 2010).

Por estas razones y por la cercanía que tiene con el lenguaje natural, la LPPO es un lenguaje confiable para formalizar UML, partiendo de diagramas como una representación gráfica de requisitos de sistemas reales.

Este artículo se organiza de la siguiente manera: en la sección 2 se realiza la presentación de los conceptos fundamentales relacionados con las primitivas conceptuales de UML, la LPPO y el UN-Lencep; en la sección 3 se hace un análisis

crítico de los trabajos en la representación formal de UML a partir de diferentes especificaciones matemáticas; las reglas de especificación de las primitivas conceptuales de UML en LPPO se presentan en la sección 4, así como también se plantea un caso de estudio. Por último, en las secciones 5 y 6 se presentan las conclusiones y el trabajo futuro.

2. MARCO TEÓRICO

2.1 Primitivas Conceptuales de UML

Las primitivas conceptuales (*constructs*, como se conocen en inglés) permiten describir, de forma abstracta, los aspectos funcionales de una aplicación; a estas primitivas conceptuales también se les denomina elementos conceptuales o patrones conceptuales (Molina *et al.*, 2004). En otras palabras, una primitiva conceptual es un término empleado para hacer referencia a los diferentes elementos que componen un esquema conceptual. Las principales primitivas conceptuales utilizadas en UML 2.2 se describen a continuación (OMG, 2010).

Actor: es el rol que los usuarios desempeñan respecto del sistema y que emplean los casos de uso. Pueden ser humanos u otros sistemas que se comunican con el sistema.

Asociación*:* es la interacción que se establecen entre los actores y los casos de uso.

Asociación entre clases: es un elemento de modelo que puede tener propiedades de clase y de asociación.

Atributo: es una característica estructural.

Caso de uso: es la especificación de un conjunto de acciones realizadas por el actor sobre el sistema; es decir, son los pasos que describen de principio a fin un proceso.

Clase: describe un conjunto de objetos que comparten las mismas especificaciones de características, restricciones y semántica.

Estado: es uno de los tres estados definidos para la clase State (Estado simple, estado compuesto y máquina de subestados), el cual representa una situación durante la cual las condiciones estáticas no se modifican.

Extensión: es una interacción que se presenta cuando una instancia del caso de uso origen extiende el comportamiento del caso de uso destino; en otras palabras, el caso de uso a extender invoca el caso de uso base bajo ciertas condiciones (Cockburn, 2000).

Generalización: es una relación taxonómica entre un clasificador más general y un clasificador más específico. Cada instancia del clasificador específico es también una instancia indirecta del clasificador general. El clasificador específico hereda las características del clasificador más general.

Inclusión: es una interacción que se presenta cuando una instancia del caso de uso origen incluye también el comportamiento descrito por el caso de uso destino; es decir, un caso de uso incluido describe un objetivo de bajo nivel de un caso de uso base.

Herencia: es una interacción que se presenta cuando un caso de uso origen hereda la especificación de un caso de uso destino y posiblemente la modifica y/o amplia; se presenta también entre los actores.

Línea de vida: representa una única entidad que interactúa dentro del diagrama, aunque puede tener multiplicidad.

Mensaje: interacción entre líneas de vida, la cual puede representar la invocación de una operación, la creación o destrucción de una instancia o el envío de una señal; el mensaje normalmente indica la entidad que envía el mensaje y la que la recibe.

Operación: es una característica del comportamiento de un clasificador que especifica el nombre, tipo, parámetros y restricciones para hacer valer un comportamiento asociado.

Transición: representa la relación entre un vértice de origen y uno de destino; puede formar también relaciones compuestas en las que cambia.

2.2 Lógica de predicados de primer orden (LPPO)

Varios formalismos representan y manipulan el conocimiento humano. En el enfoque simbólico se utilizan símbolos para representar el conocimiento y el estado del mundo y se simula el proceso cognitivo manipulando esos símbolos. La LPPO organiza un lenguaje que permite formalizar expresiones del conocimiento humano haciendo explícitos los objetos y las relaciones, así como sus restricciones, lo cual constituye su principal ventaja. Además, proporciona un método, la deducción matemática para obtener nuevo conocimiento a partir del antiguo. Es por ello, que la lógica se convierte en una importante herramienta de representación de las primitivas conceptuales, ya que proporciona una base formal de trabajo (Reeves y Clarke, 1990 y Garrido, 1995).

La LPPO se utiliza como mecanismo de formalización y se define como la parte de la lógica en la que se unen el cálculo proposicional y el cálculo de predicados, con lo cual es posible evaluar expresiones utilizando términos, operadores, reglas de inferencia y cuantificadores (Ramírez y Cabrera, 2006). Entre las ventajas de la lógica de predicados se encuentra:

- Permite formalizar expresiones del conocimiento humano, que incluyen sus objetos, relaciones y restricciones.
- Proporciona un lenguaje artificial no ambiguo, riguroso y preciso.
- Es la base de lenguajes de especificación y diseños de métodos formales para realizar la notación de UML.

Por estas razones y por la cercanía que tiene con el lenguaje natural, la lógica de predicados de primer orden es un lenguaje confiable para formalizar UML, partiendo de

los diagramas como una representación gráfica de requisitos de sistemas reales y la LPPO como una representación lógica de esos requisitos.

2.3 UN-Lencep

UN-Lencep es un lenguaje controlado, diseñado para expresar el domino del problema durante la deducción de requisitos. Con la utilización de este lenguaje, el interesado puede expresar el dominio del problema en un lenguaje claro para él y que no presente ambigüedades para el analista. Mediante una serie de plantillas, UN-Lencep controla el lenguaje natural que expresa el interesado. Al ser un lenguaje que se puede traducir fácilmente en esquemas preconceptuales, se puede obtener la representación de UN-Lencep de algunos diagramas de UML (Zapata, et al. 2006 y Zapata y Arango, 2007). Las plantillas que permiten generar un discurso en UN-Lencep se observan en la tabla 1, para la cual se tienen en cuenta las siguientes convenciones:

- A, B, C y D son conceptos.
- <ES> y <TIENE> son relaciones estructurales, que tienen sus equivalencias como se muestra en la tabla 1.
- <R1> y <R2> son relaciones dinámicas, las cuales representan verbos de actividad.
- {COND} es una condición, incluye un conjunto de conceptos relacionados mediante operadores de comparación, además de operadores algebraicos.
- <CUANDO>, <SI>, <ENTONCES>, <DADO QUE>, <LUEGO DE QUE> son palabras reservadas para el uso de condicionales e implicaciones.

Tabla 1. Equivalencias entre UN-Lencep y un subconjunto del Lenguaje Natural.

Especificación Básica de UN-Lencep	**Equivalencia en un subconjunto del Lenguaje Natural**	
A <ES> B	A <es un tipo de> B A <es una clase de> B	B <se divide en> A

A <TIENE> B	A <incluye> B A <contiene> B A <posee> B A <consta de>B A <está conformado por> B A <está formado por> B A <se compone de> B A <está compuesto por> B	B <pertenece a> A B <es una parte de> A B <está incluido en> A B <está contenido en> A B <es un elemento de> A B <es un subconjunto de> A
<CUANDO> A <R1>B, C<R2> D	<si> A <R1> B <entonces> C <R2> D <dado que> A <R1> B, C <R2> D <luego de que> A <R1> B, C <R2> D	

3. ANTECEDENTES

Actualmente, existen diversos trabajos que buscan proporcionarle a UML una base matemática científica que le permita ser preciso y consistente. Los principales trabajos en esta área se enumeran a continuación.

Un primer intento por formalizar a UML lo constituye el uso del lenguaje de restricciones de objetos (OCL), que permite especificar características adicionales sobre los modelos. OCL es un lenguaje para la descripción textual precisa de restricciones que se aplican a los modelos gráficos UML y se suele usar para completar la información de los diagramas UML, pero no para su formalización (Warmer y Kleppe, 2003). OCL es un lenguaje semiformal, ya que su sintaxis se define de forma precisa, pero su semántica aún presenta ambigüedad, imprecisión e inconsistencia (Becker y Pons, 2003). Por tanto, el conjunto de lenguajes conformado por UML y OCL tiene definida su sintaxis, pero aún se considera ambiguo, impreciso e inconsistente.

Steimann y Kühne (2002) definen una sintaxis abstracta para algunos elementos de los diagramas de UML y la mapean al lenguaje CORE, utilizando lógica formal. Los diagramas objeto del mapeo son: Casos de uso (trabaja los elementos de caso de uso, actor y asociación), Clases (define la sintaxis para las clases, los atributos, las

operaciones, y las relaciones de asociación y herencia), Colaboración (actualmente comunicación, define los roles, las clases, las asociaciones y el envío de mensajes), Secuencias **(**define la sintaxis para los objetos y sus propiedades específicas, los actores y los llamados a los métodos). Este trabajo es uno de los más completos, ya que trabaja varios diagramas de UML e integra aspectos estáticos y dinámicos para la definición de la sintaxis, pero no trabaja con la semántica de dichos elementos y trabaja con un conjunto mínimo de primitivas conceptuales trabajadas. Por otra parte, la especificación de la sintaxis abstracta se realiza en un lenguaje semiformal basado en teoría de conjuntos. Consecuentemente, se tiene el mismo problema inicial: un lenguaje semiformal que presenta problemas de ambigüedad, precisión y consistencia.

Moreno (1997), Moreno y Van De Riet (1997) definen de la correspondencia formal entre el mundo lingüístico, en el cual se expresan los requisitos de un sistema, y el mundo conceptual, en el cual se modela el sistema. El mundo lingüístico se expresa utilizando lógica de predicados, mientras el mundo conceptual se expresa utilizando teoría de conjuntos. Por tanto, este trabajo busca establecer la correspondencia entre estos dos mundos a nivel matemático. El principal problema que se presenta en esta aproximación es que el mundo conceptual sólo lo compone el diagrama de clases, obteniendo la correspondencia entre unos pocos elementos de este diagrama.

Shroff y France (1997) realizan la formulación del diagrama de clases utilizando Z, una notación de especificación formal basada en los conceptos matemáticos de LPPO y teoría de conjuntos. Este trabajo se centra en expresar en la notación Z las clases y las asociaciones entre las clases. Además, proporciona formalismos para las estructuras de generalización y agregación. De igual forma que el caso expuesto anteriormente,

esta aproximación sólo trabaja algunos elementos del diagrama de clases de UML, por lo que no brinda suficiente formalismo a este diagrama para superar los problemas de ambigüedad, precisión y consistencia.

4. REPRESENTACIÓN DE LAS PRIMITIVAS CONCEPTUALES DE UML MEDIANTE LPPO A PARTIR DE UN-LENCEP

La representación de las primitivas conceptuales de UML 2.2 en LPPO y las reglas de conversión desde UN-Lencep esas primitivas se proponen a continuación.

4.1 Representación en LPPO del lenguaje controlado UN-Lencep.

En la Tabla 2 se pueden observar las reglas para representar el UN-Lencep.

Tabla 2. Equivalencias de la Especificación Básica de UN-Lencep en LPOO.

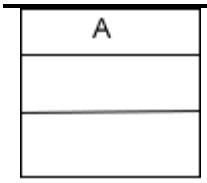
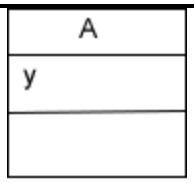
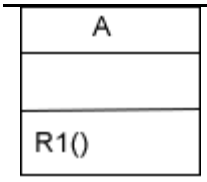
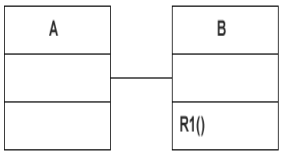
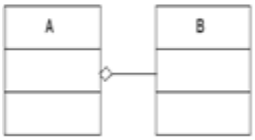
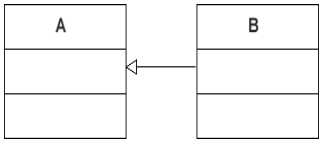
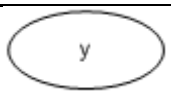
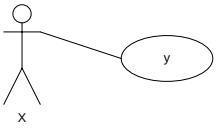
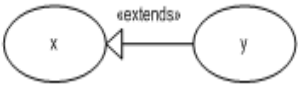
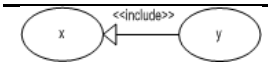
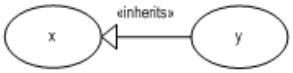
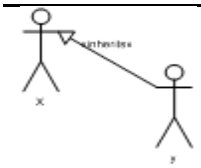
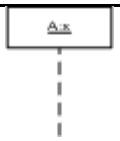
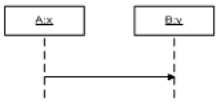
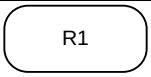
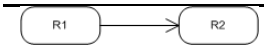
Especificación Básica de UN-Lencep	Representación en LPPO
A <ES> B	$\forall x(A(x) \wedge B(x) \wedge es(A(x),B(x)))$
A <TIENE> B	$\forall x \exists y(A(x) \wedge B(y) \wedge tiene(A(x), y))$
A R1 B	$\forall x \exists y(A(x) \wedge B(y) \wedge R1(A(x),B(y)))$
Cuando A R1 B entonces C R2 D	$\forall x \exists y \ \forall z \exists w \ (A(x) \wedge B(y) \wedge C(z) \wedge D(w) \wedge R1(A(x),B(y)) \wedge R2(C(z) D(w)) \wedge implica(R1(A(x),B(y)),R2(C(z),D(w))))$

4.2 Representación en LPPO de las primitivas conceptuales de UML 2.2

En la Tabla 3 se pueden observar las reglas que permiten la representación de las primitivas conceptuales de UML en LPOO.

Tabla 3. Equivalencias de las primitivas conceptuales de UML 2.2 en LPOO.

UML 2.2	LPOO	UML 2.2	LPOO

	∀x (es_clase(A(x)))		∀x∃y (es_atributo(y,A(x)))
	∀x∃y (es_operacion(R1(), A(x)))		∀x ∀y (es_asociacion(A(x), B(y))))
	∀x ∀y (es_agregacion(A(x), B(y))))		∀x(es_generalizacion(A(x), B(x))))
	∀x(es_actor(x))		∀x(es_casodeuso(y))
	∀x∃y (es_asociacion(x,y)))		∀x∃y (es_extends(x,y)))
	∀x∃y (es_include(x,y)))		∀x∀y(hereda(y,x))
	∀x∀y(hereda(y,x))		∀x(es_lineadevida(A(x))
	∀x∀y(es_mensaje(A(x),B(y)))		∀x (es_estado(R1)
	∀x∀y(es_transicion(R1, R2))		

4.3 Conversión de la representación de UN-Lencep a la representación de las primitivas conceptuales de UML 2.2

En la tabla 4 se pueden observar las reglas que permiten obtener las primitivas conceptuales de UML desde la representación en LPOO de UN_Lencep.

Tabla 4. Reglas de conversión.

Nro	Antecedente	Consecuente
1.	$\forall x(A(x) \wedge B(x) \wedge es(A(x),B(x)))$	es_clase(A(x)) $\wedge$ es_clase(B(x)) $\wedge$ es_generalización(B(x), A(x))
2	$\forall x(A(x) \wedge B(x) \wedge es(A(x),B(x)) \wedge (es\_actor(A(x) \vee es\_actor(B(x)))))$	es_actor(A(x)) $\wedge$ es_actor(B(x)) $\wedge$ hereda(B(x), A(x))
3	$\forall x \exists y (A(x) \wedge B(y) \wedge tiene(A(x),B(y)))$	es_atributo(B(y),A(x))
4	$\forall x \exists y (A(x) \wedge B(y) \wedge tiene(A(x),B(y)) \wedge es\_clase(B(y)))$	es_agregacion(A(x), B(y))
5	$\forall x \exists y(A(x) \wedge B(y) \wedge R1(A(x),B(y)))$	es_clase(A(x)) $\wedge$ es_clase(B(y)) $\wedge$ es_asociación(A(x),B(y)) $\wedge$ es_operacion(R1,B(y)) $\wedge$ es_actor(A(x)) $\wedge$ es_casodeuso(concatenación(R1,y)) $\wedge$ es_estado(convertiraparticipio(R1)) $\wedge$ es_lineadevida(A(x)) $\wedge$ es_lineadevida(B(x)) $\wedge$ es_mensaje(A(x),B(y))
6	$\forall x \exists y \forall z \exists w (A(x) \wedge B(y) \wedge C(z) \wedge D(w) \wedge R1(A(x),B(y)) \wedge R2(C(z), D(w)) \wedge implica(R1(A(x),B(y)), R2(C(z), D(w))))$	es_include(es_casodeuso(concatenación(R1,B(y)), es_casodeuso(concatenación(R1,D(w)))
7	$\forall x \exists y \forall z \exists w (A(x) \wedge B(y) \wedge C(z) \wedge D(w) \wedge R1(A(x),B(y)) \wedge R1(C(z), D(w)) \wedge implica(R1(A(x),B(y), R1(C(z), D(w))))$	es_casodeuso(R1) $\wedge$ es_extends(es_casodeuso(concatenación(R1,B(y))), es_casodeuso(concatenación(R1,C(z))))
8	$\forall x \forall z \exists y (A(x) \wedge B(y) \wedge C(z) \wedge R1(A(x),B(y)) \wedge R2(C(z), B(y)) \wedge implica(R1(A(x),B(y)), R2(C(z), B(y))))$	es_transicion(es_estado(convertiraparticipio(R1)), es_estado(convertiraparticipio(R2))) $\wedge$ es_include (es_casodeuso (concatenación (R1,B(y))), es_casodeuso(concatenación(R2,B(y)))) $\wedge$ es_mensaje(es_lineadevida(A(x)), es_lineadevida(B(y))) $\wedge$ es_mensaje(es_lineadevida(C(z)), es_lineadevida(B(y)))

4.4 Caso de estudio

Para ejemplificar las reglas descritas en la sección anterior se presenta un caso de estudio en la tabla 5, que corresponde al enunciado del discurso en UN-Lencep correspondiente a una granja lechera.

5. CONCLUSIONES Y TRABAJO FUTURO

Este artículo abordó la problemática de la representación en un lenguaje formal de las primitivas conceptuales de UML 2.2. y se identificaron algunas problemáticas que aún no están completamente resueltas: la cantidad de diagramas se limita a uno, haciendo énfasis en el diagrama de clases y su representación sintáctica, dejando de un lado la semántica. Así, se desarrolló un conjunto de reglas heurísticas que permiten obtener las primitivas conceptuales de UML 2.2, representadas en lógica de predicados de primer

orden a partir de la representación en LPPO de un discurso del interesado expresado en UN-Lencep.

Tabla 5. Caso de estudio.

UN_Lencep	Representación en LPPO de UN_Lencep	Representación en LPPO de las primitivas de UML obtenidas
Vaca es animal	$\forall x(vaca(x) \wedge animal(x) \wedge es(vaca(x),animal(x)))$	es_clase(vaca(x)) ∧ es_clase(animal(x)) ∧ es_generalización(animal(x), vaca(x))
Vaca tiene identificación	$\forall x \exists y$ (vaca(x) ∧ identificacion(y) ∧ tiene(vaca(x), identificación(y)))	es_atributo(identificación(y), vaca(x))
Vaca tiene nombre	$\forall x \exists y$ (vaca(x) ∧ nombre(y) ∧ tiene(vaca(x), nombre(y)))	es_atributo(nombre(y), vaca(x))
Vaca produce leche	$\forall x \exists y$(vaca(x) ∧ leche(y) ∧ produce(vaca(x),leche(y)))	es_clase(vaca(x)) ∧ es_clase(leche(y)) ∧ es_asociación (vaca(x), leche(y)) ∧ es_operación(produce,leche(y)) ∧ es_actor(vaca(x)) ∧ es_casodeuso(concatenación(produce,leche(y)))∧ es_estado(convertiraparticipio(produce)) ∧ es_lineadevida(vaca(x)) ∧ es_lineadevida(leche(y)) ∧ es_mensaje(es_lineadevida(vaca(x)), es_lineadevida(leche(y)))) .
Leche tiene cantidad	$\forall x \exists y$ (leche(x) ∧ cantidad(y) ∧ tiene(leche(x), cantidad(y)))	es_atributo(leche(y), cantidad(x))
El ordeñador recolecta leche	$\forall x \exists y$(ordeñador(x) ∧ leche(y) ∧ recolecta(ordeñador(x), leche(y)))	es_clase(ordeñador(x)) ∧ es_clase(leche(y)) ∧ es_asociación(ordeñador(x), leche(y)) ∧ es_operacion(recolecta,leche(y)) ∧ es_actor(ordeñador(x)) ∧ es_casodeuso(concatenación(recolecta,leche(y))) ∧ es_estado(convertiraparticipio(recolecta)) ∧ es_lineadevida(ordeñador(x)) ∧ es_lineadevida(leche(x)) ∧ es_mensaje(es_lineadevida(ordeñador(x)), es_lineadevida(leche(x))) .
Cuando vaca produce leche ordeñador recolecta leche	$\forall x \forall z \exists y$ (vaca(x) ∧ leche(y) ∧ ordeñador(z) ∧ produce(vaca(x),leche(y)) ∧ recolecta(ordeñador(z), leche(y)) ∧ implica(produce (vaca(x), leche(y)), recolecta(ordeñador(z), leche(y)))	es_transicion(es_estado(convertiraparticipio (produce)),es_estado(convertiraparticipio(recolecta))) ∧ es_include(es_casodeuso(concatenación(produce,leche(y))), es_casodeuso(concatenación(recolecta,leche(y)))) ∧ es_mensaje(es_lineadevida(vaca(x)), es_lineadevida(leche(y))) ∧ es_mensaje(es_lineadevida(ordeñador(z)), es_lineadevida(leche(y))).

Los problemas anotados anteriormente, se solucionaron con esta propuesta, dado que se hace la representación de diversas primitivas conceptuales que se emplean en varios de los diagramas de UML 2.2. El punto de partida de esta propuesta es el lenguaje UN-Lencep, que por ser un lenguaje controlado, no presenta problemas en el significado de sus términos, por lo tanto, en la representación formal no se presentan

ambigüedades semánticas. De otro lado, la lógica de predicados de primer orden proporciona un lenguaje artificial no ambiguo, riguroso y preciso que se utiliza en lenguajes de especificación y diseño de métodos formales.

Para este trabajo en particular, se generaron nuevas preguntas a partir de la propuesta realizada, que pueden motivar la continuación de los esfuerzos en esta línea:

- Establecer si existen reglas adicionales que puedan generar variaciones sobre las representaciones realizadas.
- Obtener las demás primitivas conceptuales de UML 2.2 como son los fragmentos combinados del diagrama de secuencias, así como el diagrama de objetos y el diagrama de comunicación.

BIBLIOGRAFÍA

Becker V., Pons C.: Definición formal de la semántica de UML-OCL a través de su traducción a Object-Z. Proceedings of IX Congreso Argentino de Ciencias de la Computación CACIC (2003)

Brean, Hinkel, U., Hofmann, C., Klein, C., Paech, B., Rumpe, B., Thurner, V. ECOOP'97, Finland (1997)

Cockburn, A.: Writing Effective Use Cases. Addison-Wesley, Boston (2000)

Garrido, M.: Lógica simbólica. Ed. Tecnos, Madrid (1995)

Jacobson, I., Booch, G., Rumbaugh, J.: El Proceso Unificado de Desarrollo de Software. Addison Wesley, Madrid (2001)

Molina, J.C., Pastor, O.: Mda, OO-Method y la tecnología OLIVANOVA model execution. Actas del I Taller sobre Desarrollo Dirigido por Modelos, MDA y Aplicaciones (DSDM'04) Málaga. (2004)

Moreno, A.M.: Object Oriented Analysis from Textual Specifications. Proceedings of the 9th Intl. Conf. on Software Engineering and Knowledge Engineering, Madrid (1997)

Moreno, A.M., Van De Riet, R.P.: Justification of the Equivalence between Linguistic and Conceptual Patterns for the Object Model. Proc. of the International Workshop on Applications of Natural Language to Information Systems, Vancouver, (1997).

OMG (Object Management Group). (2010). OMG Unified Modeling Language (OMG UML), Superstructure Version 2.2., http://www.omg.org.

Ramírez, F., Cabrera, C.: Desarrollo de un modelo de implantación de agentes inteligentes en la red univirtual como intérpretes semánticos del lenguaje natural. Pereira, (2006)

Reeves, S., Clarke, M. Logic for Computer Science. Addison-Wesley, Wokingham (1990)

Steimann, F., Kühne, T.: A Radical Reduction of UML's Core Semantics.In: J-M Jézéquel, H. Hussman, S. Cook (Eds): UML, 2002, LNCS, vol. 2460, pp. 34--48. Springer, Heidelberg (2002)

Shroff M., France R.: Towards a Formalization of UML Class Structures in Z" in COMPSAC. Proceedings of the 21st International Computer Software and Applications Conference (COMPSAC'97), Washington DC. (1997)

Warmer J., Kleppe A.: The Object Constraint Language: Getting your models ready for MDA. Addison Wesley, Londres (2003)

Zapata, C., Arango, F.: Construcción Automática de Esquemas Conceptuales a partir de Lenguaje Natural. Autores editores, Medellín (2007)

Zapata, C., Gelbukh, A., Arango, F.: UN-Lencep: Obtención Automática de Diagramas UML a Partir de un Lenguaje Controlado. Avances en la Ciencia de la Computación, VII Encuentro Internacional de Computación ENC'06, (2006)