Un acercamiento ontológico para modelar el paradigma orientado a objetos

An ontological approach for modeling the object-oriented paradigm

Gloria Giraldo, Ph D., Carlos Zapata,Ph D., Francisco Arias,Ing., Lorena Cadavid, Ing., Andrés Hoyos, Ing.
Grupo de Lenguajes Computacionales, Escuela de Sistemas, Universidad Nacional de Colombia - Sede Medellín
glgiraldog, cmzapata, fjarias, dlorenach, afhoyos @unalmed.edu.co

Recibido para revisión: 6 de Octubre de 2008, Aceptado: 28 de Noviembre de 2008, Versión final: 4 de Diciembre de 2008

Resumen—**En el presente artículo, se elabora una propuesta ontológica para describir el dominio del paradigma orientado a objetos, con la motivación de mejorar el proceso de enseñanza/ aprendizaje de dicho paradigma. Se establecen los conceptos básicos, tanto del paradigma orientado a objetos como de las ontologías. Asimismo, se hace una exploración de la literatura asociada a la construcción de ontologías del paradigma orientado a objetos, así como de la literatura que, aún sin construir la ontología, presenta lineamientos básicos para su construcción. Finalmente, se muestra de forma conceptual la ontología desarrollada por los autores y su implementación en la herramienta Protégé.**

Palabras Clave—**Paradigma orientado a objetos, Ontología, Ingeniería de software, Protégé.**

Abstract—**We make, in this paper, an ontological proposal to describe the object-oriented paradigm domain. The aim of the ontology is to improve the teaching/learning process of this paradigm. The basic concepts of both, the paradigm and the ontology, are described. We also present a review of the state of the art in object paradigm, ontology development, and unimplemented guidelines for ontology construction. Finally, we discuss concepts about the object-oriented ontology development and its implementation using Protégé.**

Keywords—**Object-oriented paradigm, Ontology, Engineering software, Protégé.**

I. INTRODUCCIÓN

Muchas son las definiciones que se pueden encontrar en la literatura sobre la palabra *paradigma*, más aún en el ámbito de la programación. En su trabajo, SPINELLIS [1] converge a una definición de *paradigma de programación* desde fuentes de diversos autores, y lo define como un "conjunto de reglas para determinar los tipos de lenguaje en función de algunas condiciones probables".

Agrupados bajo esta definición, diversos paradigmas surgieron y se instauraron en los cánones de la programación computacional. Sin embargo, la *programación orientada a objetos* logró la importancia suficiente para considerarla el más importante de los paradigmas de programación en la actualidad [2].

En sus términos más generales, el paradigma orientado a objetos se basa en la idea de que es posible construir un sistema descomponiendo el problema modelado en objetos y escribiendo luego el código para dichos objetos [3]. SIPNELLIS afirma en [1] que "el paradigma orientado a objetos se enfoca en las características estructurales y de comportamiento de entidades como unidades completas".

Por otra parte, una *ontología* se puede definir como la "representación de un vocabulario, a menudo especializado, de algún dominio o área del conocimiento" [4]. Entendidas como herramientas, la importancia del uso de las ontologías radica en que permiten aclarar la estructura del conocimiento que modelan, capturando y organizando la esencia conceptual de ese conocimiento [4]. La construcción de una ontología del paradigma orientado a objetos, puede facilitar varios procesos alrededor del uso de este paradigma, como pueden ser la enseñanza del

modelo, la automatización de procesos propios de la ingeniería de software o la validación de modelos, entre otros.

Los trabajos previos en este tema [5, 6, 7 y 8], revelan una definición inicial de lineamientos para la construcción de la ontología, pero sin llegar a implementar dicha ontología o con una implementación de pocos conceptos y relaciones pues, por lo general, no es la ontología *per se* el producto final de esos trabajos.

En este orden de ideas, este artículo pretende estructurar el conocimiento asociado al *paradigma orientado a objetos* a través de una ontología, desarrollada con la herramienta Protégé, para lograr uno los objetivos descritos anteriormente: la mejora en los procesos de enseñanza del modelo orientado a objetos.

El presente artículo, se organiza de la siguiente forma: en la segunda sección, se hace un breve repaso por el modelo orientado a objetos y sus elementos principales; en la tercera sección, se describen las ontologías y sus usos; en la sección cuarta, se estudian trabajos relacionados con ontologías del paradigma orientado a objetos; en la sección quinta, se profundiza en cada uno de los elementos más relevantes en la programación orientada a objetos; en la sexta sección, se presenta la ontología propuesta para la descripción del dominio del modelo orientado a objetos y en la sección séptima la implementación de ésta. Finalmente, el artículo termina con conclusiones y trabajo futuro a desarrollar, en las secciones octava y novena, respectivamente.

II. PARADIGMA ORIENTADO A OBJETOS

A. ¿Qué es?

El paradigma o el desarrollo orientado a objetos, es bastante popular desde la década de los 80s [9]. Una de las principales razones para ello es, justamente, el principio de este paradigma: conectar las funciones con entidades "concretas" del mundo real.

Según BOOCH [3], el desarrollo orientado a objetos lo integran 3 fases (1) el análisis orientado a objetos (AOO), (2) el diseño orientado a objetos (DOO) y (3) la programación orientada a objetos (POO).

El AOO, es un "método de análisis en el cual se examinan los requisitos desde la perspectiva de las clases y objetos que se encuentran en el vocabulario del dominio del problema". El DOO, es "un método de diseño que abarca el proceso de descomposición orientada a objetos y una notación para describir los modelos lógico y físico, así como los modelos estático y dinámico del sistema que se diseña". La POO, es "un método de implementación en el cual los programas se organizan como colecciones cooperativas de objetos, cada uno de los cuales representa las instancias de alguna clase, y cuyas clases son, todas ellas, miembros de una jerarquía de clases unidas mediante relaciones de herencia" [3].

B. Conceptos Principales

Según BOOCH, las tres fases del desarrollo orientado a objetos señaladas en la sección A comparten tres características comunes: abstracción, encapsulamiento y herencia; además, la POO requiere una características adicional: polimorfismo. Estos conceptos se explican a continuación.

Abstracción. Es una visión simplificada de una realidad, que sólo considera determinados aspectos esenciales. En la abstracción, se toman las características relevantes de un objeto, es decir, aquellas que lo distinguen de todos los demás y, por tanto, proporcionan un límite conceptual en relación con el punto de vista del espectador [10].

Encapsulamiento. Es una propiedad que permite asegurar que el contenido de la información de un objeto se oculta al mundo exterior; es decir, es el proceso de ocultar todos los 'secretos' de un objeto que no contribuyen a sus características esenciales. Es útil para separar la interfaz de una abstracción de su implementación [10].

Herencia. Define una relación jerárquica entre clases, en la cual una clase (llamada *subclase*) comparte la estructura o comportamiento definido en una (herencia *simple*) o más (herencia *múltiple*) clases (llamadas *superclases*). Una *subclase,* normalmente, se especializa aumentando o redefiniendo la estructura y el comportamiento de la *superclase* de la cual se deriva [10].

Polimorfismo. Es una propiedad que indica la posibilidad de que un nombre denote varios objetos en diferentes clases que se relacionan mediante alguna superclase común; de esta forma, un objeto denotado por este nombre puede responder a un conjunto de operaciones de forma diferente [10].

C. Usos

Desde el punto de vista de la Ingeniería de Sistemas, la POO facilita la construcción de aplicaciones complejas, como interfaces gráficas de usuario, sistemas operativos y aplicaciones distribuidas, haciendo posible, al mismo tiempo, la comprensión del código fuente [2].

Sin embargo, el paradigma orientado a objetos viene ganando popularidad en diversas formas, más allá de los lenguajes de programación [11]. La principal ventaja de este enfoque, es que los objetos se pueden usar para representar, de forma natural, muchas situaciones del mundo real [1]. Por esta razón, la orientación a objetos demuestra su valor en una multitud de aplicaciones en diversos ámbitos, tales como negociación de valores, electrónica médica, gestión de la información en organizaciones, control del tráfico aéreo, semiconductores, industria manufacturera, juegos de video interactivo, telecomunicaciones, gestión de redes e investigación astronómica, entre otros [10].

III. ONTOLOGÍAS

A. ¿Qué son?

Una ontología se puede definir como la representación de un vocabulario, a menudo especializado, de algún dominio o área del conocimiento [4]. Se usa la palabra "dominio" para denotar un área específica de interés o un área de conocimiento (física, aeronáutica, medicina, contabilidad, fabricación de productos, etc.) [12].

Las ontologías, constituyen un entendimiento común y compartido de un dominio que se puede comunicar entre las personas, y promueven y facilitan la interoperabilidad entre diversos sistemas de información [13].

B. Conceptos principales

Según NOY y McGUINNESS [14], una ontología se compone, en general, de los siguientes elementos:

- **Clases.** Son conceptos en un dominio del discurso. Las clases constituyen una parte muy importante de las ontologías. Una clase puede tener subclases, que representan conceptos que son más específicos que la superclase que las contiene.
- **Propiedades de las clases.** Son descripciones de características o atributos de una clase y de las instancias que tiene la clase.
- **Restricciones sobre las propiedades.** Son conjuntos de valores que puede o no tomar una propiedad de una clase.

A. Usos y tendencias

En la Ingeniería del Software, las ontologías ayudan en la estandarización del conocimiento modelado y, en general, del conocimiento propio en esta área. Son flexibles y combinan la información de muchas fuentes, extrayendo conclusiones con facilidad. Ayudan en cada una de las etapas del ciclo de desarrollo del software, proporcionando ventajas puntuales en ellas, además de la reutilización de conocimiento a lo largo de este ciclo [15].

Los sistemas de recuperación de información, las bibliotecas digitales, los sistemas de integración de fuentes de información heterogéneas y los motores de búsqueda de Internet, necesitan ontologías del dominio para organizar la información y los procesos de la búsqueda directa [4]. En Inteligencia Artificial, el conocimiento en los sistemas se representa y opera explícitamente por los procesos de inferencia; las ontologías son, por tanto, una parte fundamental para el desarrollo de la Inteligencia Artificial [4].

Asimismo, los diseños orientados a objetos dependen de un adecuado dominio de ontologías; los objetos, sus atributos y sus procedimientos reflejan, de cierta forma, el dominio de los aspectos que son relevantes para la aplicación [4].

IV. DOMINIO DEL PARADIGMA ORIENTADO A OBJETOS

Según BOOCH [3], los siguientes son los conceptos principales dentro del paradigma orientado a objetos.

- **Objeto.** Según MURTHY y WIGGINS [16], un objeto es una abstracción de una entidad en un problema particular de un dominio. Como se verá más adelante, los objetos son instancias de una determinada clase, es decir, comprenden todos los atributos y métodos definidos para esa clase, y se conectan entre sí a través de mensajes.
- **Atributo.** Es un contenedor de un tipo de datos asociado a un objeto o un grupo de objetos con características similares entre sí, que hace que dichos datos sean visibles desde fuera del objeto definiendo sus características predeterminadas y cuyo valor lo puede alterar la ejecución de algún método [3]. El estado de un objeto es el conjunto de valores de los atributos del objeto [17].

Algunas de las características de los atributos de un objeto, se enuncian a continuación.

- *Tipo de dato*: define un conjunto de valores y las operaciones sobre estos valores. Algunos tipos de datos comunes, son: enteros, números de coma flotante (decimales), cadenas alfanuméricas y fechas, entre otros. Estos, se denominan tipos de datos primitivos [3].
- *Visibilidad de atributos*: define si un atributo se puede acceder directamente desde otro objeto. Se resaltan los tres siguientes tipos de visibilidad [3].
  - *Público*: los datos son visibles dentro y fuera de la clase sin restricción alguna.
  - *Privado*: los datos son accesibles sólo desde dentro de la clase donde existen.
  - *Protegido*: los datos son visibles desde dentro de la clase y desde cualquier otra clase heredada.

- **Método.** Es un algoritmo asociado a un objeto o a una clase, cuya ejecución se desencadena tras la recepción de un mensaje. El comportamiento de un objeto, es el conjunto de métodos que operan sobre el estado del objeto, es decir, es lo que el objeto puede hacer [17]. Un método, puede producir un cambio en las propiedades del objeto o la generación de un "evento" con un nuevo mensaje para otro objeto del sistema [3].

Algunas de las características de los métodos de un objeto se enuncian a continuación [3].

- *Visibilidad*: define si un método se puede invocar desde otro objeto. Los tipos de visibilidad público, privado y protegido, descritos anteriormente para las propiedades de un objeto, aplican de la misma forma para la visibilidad de los métodos.
- *Entradas y salidas*: el método, puede tener variables de entrada como datos necesarios para iniciar su proceso, y también puede entregar un resultado como salida.

- **Clase.** Los objetos que exhiben propiedades similares, se

agrupan en una clase [18]. La *multiplicidad* de una clase, se refiere a la cantidad de instancias permitidas para la clase; en este sentido, una clase puede ser de tipo abstracta (sin instancias) o concreta (con una ó más instancias asociadas); una clase abstracta, es una clase de apoyo que se construye sólo para derivar de ella otras clases [3].

- **Mensaje.** Es una comunicación dirigida a un objeto, la cual le ordena que ejecute uno de sus métodos, con ciertos parámetros asociados. Los objetos, se comunican entre sí a través de mensajes, pero un mensaje no puede cambiar el contenido de un objeto a menos que el mensaje invoque un método diseñado para tal fin [16].
- **Evento.** Es un suceso en el sistema, tal como una interacción del usuario con la máquina, o un mensaje enviado por un objeto. El sistema, maneja el evento enviando el mensaje adecuado al objeto pertinente. También, se puede definir como evento a la reacción que puede desencadenar un objeto, es decir, la acción generada por el método.
- **Relación.** Los objetos, por sí mismos, son poco interesantes. Es la interacción entre ellos la que contribuye al comportamiento de un sistema. Las relaciones más relevantes dentro del dominio del paradigma orientado a objetos, se enuncian a continuación:
  - *Asociación*: es una relación que denota dependencia semántica entre dos o más clases sin establecer la dirección de ésta, ni establecer la forma en que una clase se relaciona con la otra. Existen dos tipos de o
  - *Agregación*: es una asociación que denota una relación todo/parte, en la cual un objeto se agrega de otro y no puede ser cíclico; sin embargo, una clase que es parte de otra puede existir sin que la clase que la agrega exista.
    - *Composición*: una composición es una agregación más fuerte, en donde la parte no puede existir sin el todo.
  - *Herencia*. En la literatura orientada a objetos, el término *herencia* se entiende de diferentes maneras por las diferentes comunidades de investigación [18]. Una definición comúnmente aceptada, sostiene que la herencia es un mecanismo para crear clases partiendo de otras ya existentes [18]. Una subclase, hereda todas las definiciones de los datos y los métodos que se definen en la superclase, pero, además, puede disponer de sus propios atributos y métodos [16].
- **Tipos de datos:** en términos generales, un tipo de dato describe qué clase de valores puede tomar una variable. Dentro del paradigma orientado a objetos, los tipos de datos son útiles para describir qué clase de valores pueden tomar las variables de entrada y salida de los métodos, por ejemplo, o las propiedades que puede tener una clase. Los tipos de datos incluyen números: enteros o decimales (flotantes), cadenas de caracteres, booleanos o lógicos y valores enumerados (valores dentro de un listado previamente definido) [17].

V. ONTOLOGÍAS DEL PARADIGMA ORIENTADO A OBJETOS

Aunque la literatura especializada no reporta un trabajo como el que se acomete en este artículo, existen algunas aproximaciones iniciales que sirven de base para diseñar e implementar una ontología del paradigma orientado a objetos.

WAND [5], por ejemplo, presenta en su trabajo sobre las ontologías como base de los metamodelos para el diseño y análisis de sistemas, un primer acercamiento a la elaboración de una ontología del enfoque orientado a objetos. Para ello, usa una ontología en la que se incluyen, entre otros, los conceptos *cosa*, *propiedad* (intrínseca y heredada, entre otros tipos), *atributo*, *modelo, estado*, *evento* (estado que resulta del cambio en una o más propiedades de las cosas) y *relación de composición*. Estos elementos, forman parte del lenguaje propio del dominio del paradigma orientado a objetos.

En su trabajo, WAND argumenta que no existen fundamentos de común acuerdo en la comunidad académica en lo relativo a los conceptos del paradigma orientado a objetos, ni siquiera existe consenso en relación a lo que es un objeto [5]. A su vez, y para efectos de su propuesta, define un objeto como la «representación de una cosa» y utiliza el enfoque ontológico para manejar las características del paradigma orientado a objetos; en particular, las propiedades del objeto representan los atributos en el modelo ontológico y los métodos del objeto representan las leyes de transición para pasar de un estado a otro.

Del modelo propuesto por WAND se resaltan, entre otros, los siguientes lineamientos presentados en la conceptualización del dominio [5]:

1. Un objeto debe tener una única identificación, que se fundamente en el nombre, su funcionalidad y un conjunto de atributos.
2. Las propiedades relevantes de un objeto, son aquellas de las cuales los otros objetos deben ser «concientes» en el momento de introducir un cambio en el sistema (evento), o reaccionar a éste.
3. No cualquier concepto, se debe considerar un objeto. En particular, los atributos, clases y eventos, no son objetos.
4. La *composición*, es un concepto fundamental y los objetos compuestos deben poseer propiedades diferentes de sus objetos componentes.
5. La interacción entre los objetos, se da gracias a eventos externos (cambios en el estado). El estado resultado del objeto afectado, podría ser inestable e introducir cambios adicionales al estado del sistema.

El aporte de WAND presenta, pues, unas pautas conceptuales en la construcción de una ontología del paradigma orientado a objetos, sin mostrar implementación alguna.

BRINKKEMPER *et al.* [6], presentan un marco semántico para la construcción de metamodelos. Para probar la mejora introducida en su propuesta sobre la construcción de metamodelos, los autores toman como ejemplo el paradigma (al

cual denominan «modelo») orientado a objetos. Es decir, todas las anotaciones que realizan sobre este modelo son tangenciales a su trabajo principal y no se presentan, en ningún caso, con otro fin diferente al de probar la validez de su propuesta.

Después de hacer una breve descripción del paradigma, los autores presentan una descripción del enfoque orientado a objetos a nivel conceptual (véase la Figura 1). Ellos, presentan unos lineamientos básicos para la construcción de una ontología del enfoque orientado a objetos, los cuales se presentan a continuación [6]:

- Una clase, tiene un diagrama de estados que especifica su comportamiento.
- Los atributos de una clase, pueden ser particularidades de los estados en su diagrama de estados. Esto, indica qué valores de los atributos son significativos o visibles en un determinado estado.
- Un evento emitido, es una solicitud de un servicio a los demás objetos.
- Un cambio de estado, puede producir cambios en los valores de los atributos de los objetos, o en el objeto mismo.

Estas construcciones, permiten introducir las asociaciones «tiene», «se anota con» y «consta de», mostradas en la Figura 1. El concepto *objeto*, participa en la categoría «consta de» para aquellos objetos cuyos métodos o funcionalidades se requieren.

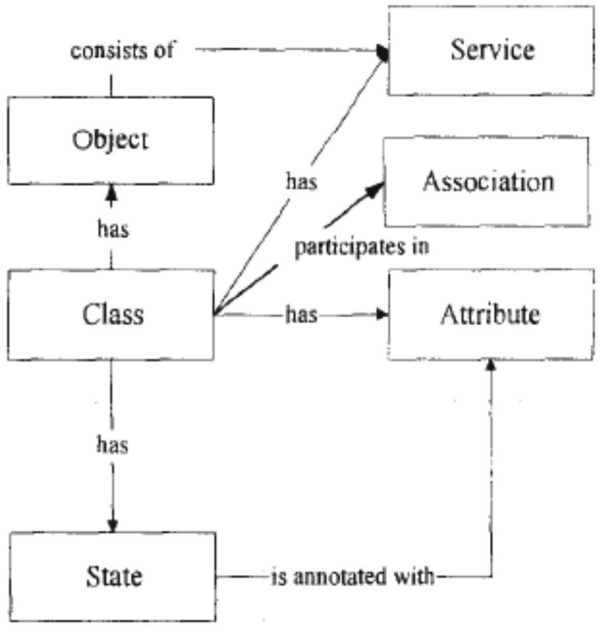

Figura 1. Descripción del enfoque orientado a objetos, a nivel conceptual [6]

Como se observa en la Figura 1, el nivel conceptual del modelo presentado no es muy profundo, ya que como se mencionó anteriormente, no es el objetivo de su propuesta.

CALERO *et al.* [7] proponen una ontología que representa las características objeto-relacionales del nuevo estándar SQL: 2003, mostrando, de forma clara, cada uno de los conceptos y sus relaciones, y permitiendo un mejor entendimiento del estándar para motivar su utilización. Es importante aclarar, que dicha ontología, no busca sólo mejorar el entendimiento del SQL: 2003, sino que, también, pretende identificar sus inconsistencias y que la ontología se formalizó mediante diagramas de clase UML 2.0 y reglas OCL (*Object Constraint Language*).

La Figura 2, presenta algunos de los conceptos y relaciones que se identificaron en dicha ontología, los cuales son:

Conceptos: Tabla Base, Columna, Atributos, Tipo de Dato Estructurado y Métodos.

Relaciones: una Tabla Base puede tener una o muchas Columnas, una Columna puede contener un Atributo o un Tipo de Dato Estructurado y un Tipo de Dato Estructurado se compone de uno o muchos Atributos y Métodos.

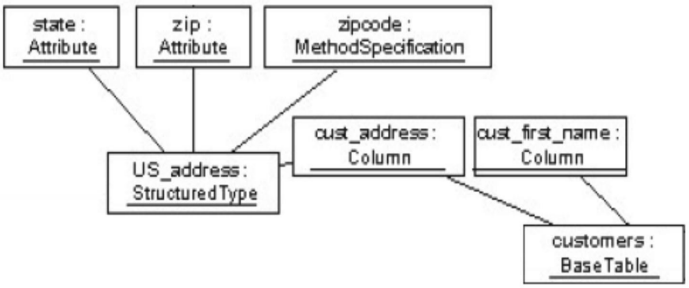

Figura 2. Resumen ontología objeto-relacional SQL: 2003 [7]

En este trabajo, es importante resaltar el uso de subontologías, que se pueden integrar a una ontología principal, permitiendo manejar los elementos de forma modular. Con ello, se mejora el entendimiento de la ontología y se puede facilitar su mantenimiento. En algunos casos, dichas subontologías se pueden reutilizar en otros dominios.

La principal debilidad identificada en este trabajo, es que, aunque en el artículo se enunció el uso de un modelo para la construcción de ontologías (*A Helix-Spindle Model for Ontological Engineering*), no se detalló la construcción de dicha ontología haciendo uso del modelo enunciado; además, a pesar de que la ontología es muy completa a nivel conceptual, es importante aclarar que en ésta no se detallaron los campos (*Slots*) de cada uno de los conceptos, los cuales son parte fundamental para especificar un dominio. Si la ontología pretende mejorar el entendimiento del estándar, la definición de los campos de la ontología juega un papel fundamental.

GARZÁS y PIATTINI [8], argumentan que, pese a tanta información existente relativa al diseño orientado a objetos, el conocimiento acumulado alrededor de este paradigma carece de suficiente estructura y clasificación, y no es fácilmente transmisible, accesible o disponible. En respuesta a ello, desarrollan una ontología enfocada a la descripción integral y completa del diseño orientado a objetos, incluyendo conocimiento de uso general, de dominios concretos y para tecnologías específicas; sostienen que «una ontología puede ayudar a facilitar la asimilación del conocimiento en diseño, ya que describe el conocimiento de un dominio de forma general y proporciona un entendimiento común. (…) Por consiguiente, es beneficioso poder disponer de una ontología para estructurar y unificar el conocimiento en diseño orientado a objetos» [8]. Basados en la idea de que el conocimiento en el diseño orientado a objetos se puede definir como declarativo u operativo, proponen una ontología puntual y específica para el diseño, es decir, dejan por fuera las demás fases del paradigma orientado a objetos, vistos en la sección II (análisis y programación).

Luego de revisar literatura reciente sobre los desarrollos de ontologías del dominio del paradigma orientado a objetos, se puede concluir que los trabajos encontrados comparten tres características importantes:

1. Presentan lineamientos conceptuales para la construcción de una ontología del paradigma orientado a objetos.
2. Ninguno de ellos desarrolla un trabajo completo y robusto alrededor del tema, que recoja no sólo los conceptos más relevantes en el dominio, sino también los demás conceptos que, aunque de menor visualización, juegan un papel importante dentro del paradigma.
3. Carecen de evidencia de implementación de sus propuestas.

Pese a esta última característica, estos trabajos conforman la plataforma de los primeros acercamientos sobre los que se apoyarán trabajos futuros, como el presentado en este artículo.

Por ello, en la siguiente sección se presenta una propuesta que toma los principales elementos conceptuales presentados de la literatura revisada, para convertirlos en construcciones concretas dentro del paradigma y complementa el conjunto de términos con los elementos de un menor nivel dentro del enfoque orientado a objetos. Además, se presenta la implementación de la ontología propuesta.

VI. ONTOLOGÍA PROPUESTA

Teniendo en cuenta los conceptos anteriores, se desarrolló un modelo ontológico basado en el dominio del paradigma orientado a objetos. La metodología llevada a cabo para desarrollar la ontología fue propuesta por NOY y McGUINNESS [14] y se detalla a continuación.

Dominio y el alcance de la ontología. En la ontología propuesta, se pretende representar el conocimiento asociado al paradigma orientado a objetos y, por esta razón, el dominio de dicha ontología será el área que comprende dicho paradigma. Esta propuesta, se utilizará para enseñar a profesionales de otras áreas diferentes a la Ingeniería de Sistemas los conceptos básicos alrededor del paradigma orientado a objetos, a fin de familiarizarlos con este dominio del conocimiento.

Se explicarán los conceptos básicos de este paradigma mediante un ejemplo concreto. A grandes rasgos, el ejemplo desarrollado se basó en la construcción de un sistema en el cual se deben crear cuatro tipos de clases: persona, estudiante, profesor y curso.

La clase *persona,* es abstracta y se relaciona con las clases *estudiante* y *profesor* por medio de una herencia simple. Esta clase, posee los atributos *nombre*, *sexo* y *edad* y un método que permite mostrar los atributos de una persona; estos atributos son variables de tipo primitivo (*string* y enteros).

La clase *estudiante,* es concreta y se relaciona con la clase *persona* por medio de una herencia simple. Esta clase, posee los atributos *semestre* y *promedio,* además de los atributos que hereda de su clase padre *persona* (*nombre*, *sexo* y *edad*). Esta clase, posee un método que permite mostrar todos los atributos de un estudiante. Estos atributos, igual que en la clase *persona*, son variables de tipo primitivo (*string* y enteras).

La clase *profesor,* es muy similar a la clase *estudiante;* su única diferencia, es que la clase *profesor* no tiene atributos adicionales a los atributos de la clase padre.

La clase *curso,* es una clase concreta que no hereda de ninguna clase. Esta clase, posee dos atributos: *hora curso* y *director curso*. El primer atributo es una variable de tipo primitivo, mientras que el segundo atributo hace referencia a una instancia de la clase *profesor*. Esta clase, posee un método que permite mostrar todos los datos del curso.

Luego de instanciar la ontología por medio del ejemplo, se procedió a la elaboración de diferentes preguntas con las cuales se pretendía validar la ontología propuesta. Estas preguntas son las siguientes:

1. ¿Cuales son los métodos públicos de la clase *curso*?
2. ¿Cuál es la clase padre de las clases *estudiante* y *profesor*?
3. ¿Cuáles son las clases que son hijas de la clase *persona*?
4. ¿Cuáles son los métodos de la clase *Curso* que tienen como valor de retorno cadenas de caracteres?
5. ¿Cuál es el mensaje que se crea cuando la clase *profesor* invoca un método de la clase *persona*?
6. ¿Cuál es el método de la clase *estudiante* que recibe como parámetros de entrada el *sexo* y la *edad* del estudiante?
7. ¿Cuáles son las clases (no abstractas) que poseen atributos de *edad* y no son estudiantes?

Un grupo de personas conocedoras en el área del paradigma orientado a objetos (profesores o investigadores) podrán realizar el mantenimiento de la ontología, actualizando y corrigiendo las respectivas inconsistencias que se presenten a

medida que la ontología crezca.

Se identificaron 3 grupos de personas que pueden usar la ontología:

- Estudiantes y profesores (expertos o no) del paradigma orientado a objetos.
- Analistas y desarrolladores de sistemas basados en el paradigma orientado a objetos.
- Herramientas informáticas de apoyo a los desarrolladores para la creación de sistemas bajo el paradigma orientado a objetos e interoperabilidad entre diferentes herramientas de este tipo.

Clases, propiedades y jerarquía de clases. Las clases, se eligieron siguiendo un único criterio: la importancia del término «clase» dentro del dominio del paradigma orientado a objetos y, sobre todo, dentro del propósito fundamental de la ontología. La Tabla 1, presenta un listado de las clases y subclases de la ontología y las propiedades de éstas.

Propiedades (detalle). A continuación se explica de forma detallada cada una de las propiedades asociada a las clases.

Propiedades (atributos) asociadas a la clase *clase*:

- Nombre: nombre único de una instancia de la clase *clase*; toda *clase,* debe tener un nombre y, por tanto, es una propiedad requerida del tipo cadena de caracteres.
- Tipo de clase: una clase*,* puede ser abstracta o concreta, y es necesario especificarlo para cada instancia de clase. Una clase, no puede ser simultáneamente abstracta y concreta.
- Multiplicidad: propiedad requerida que hace referencia al número entero de instancias que puede tener la clase.
- Visibilidad: es necesario especificar el tipo de visibilidad que tiene la clase (protegido, público y privado).
- Subclase: identifica si una clase es subclase de otra, en cuyo caso tomará el valor de 1 (de no ser subclase de alguna otra, este atributo tomará el valor de 0). Por su definición, es un atributo requerido de tipo *symbol* con valores permitidos cero (0) y uno (1), con un valor único por instancia.

Tabla 1. Clases, subclases y sus atributos en la ontología propuesta

Clase	Propiedades	Subicase 1	Propiedades adicionales	Subclase 2	Propiedades adicionales
Clase	Nombre				
	Tipo de clase				
	Multiplicidad				
	Visibilidad				
	Subclase				
	Superclase				
	Atributo				
	Clase relacionada				
	Método				
	Objeto relacionado				
	Clase A				
	Clase B				
	Tipo de relación				
Método	Nombre				
	Entrada				
	Salida				
	Visibilidad				
	Clase relacionada				
Tipo de dato	Número	Primitivo	Tipo de primitivo		
		Instancia	Clase relacionada		
Relación	Cardinalidad A	Asociación		Agregación	Existen restricciones adicionales sobre las propiedades de la clase
	Cardinalidad B			Composición	
	Clase A	Herencia		Simple	
	Clase B			Múltiple	
Objeto	Nombre				
	Clase relacionada				
Variable	Nombre	Atributo	Visibilidad		
			Clase relacionada		
	Tipo	Entrada	Descripción Valor		
			Método (con restricciones)		
		Salida	Descripción Valor		
			Método (con restricciones)		
Mensaje	Contenido				
	Clase A				
	Clase B				
Evento	Mensaje evento				

- Superclase: identifica si una clase es superclase de otra, operando con la misma lógica que el atributo *subclase*.
- Atributo: es el conjunto de los atributos o propiedades asociados a una clase. Una clase, puede no tener atributos, por tanto, esta propiedad no es requerida; también puede tener más de un atributo, por tanto, la cardinalidad de esta propiedad es múltiple.
- Clase relacionada: hace referencia al conjunto de instancias de la clase *clase* que se relacionan con una instancia de la clase *clase* en particular. Una instancia de una clase, se debe relacionar mínimo con otra instancia de clase (no tiene sentido hablar de clases aisladas dentro de un modelo orientado a objetos) y, por esta razón, es una propiedad requerida; la cardinalidad es múltiple, pues podrían existir muchas instancias de clase relacionadas con una clase particular.
- Método: se refiere al conjunto de métodos de la clase *método* que se relacionan con una instancia de la clase *clase* en particular, es decir, a los métodos de la clase. Una clase, podría no tener métodos, por tanto, esta propiedad no se requiere; también podría tener más de un método, por tanto, la cardinalidad de esta propiedad es múltiple.
- Objeto asociado: hace alusión a las instancias de la clase *objeto* asociadas a las instancias de una clase y que son, por tanto, instancias de la clase dentro del paradigma orientado a objetos (aunque no dentro de la ontología). Dado que dentro del paradigma orientado a objetos una clase sólo tiene sentido si posee instancias, esta propiedad se requiere; una clase puede tener uno o más objetos asociados, por ello, su cardinalidad es múltiple.
- Tipo de relación: identifica los tipos de relaciones que establece una instancia de la clase *clase* con las demás instancias de la clase *clase*. Ésta, es una propiedad requerida (debe haber mínimo un tipo de relación asociada a la instancia de la clase *clase*).
- Clase A: propiedad que puede tomar como valor una o varias instancias de la clase *clase* que se convierten en el conjunto de padres de la subclase a través de la relación de herencia (como se verá más adelante). Es una propiedad requerida de cardinalidad múltiple; cuando la cardinalidad de esta propiedad, para una instancia clase, es de más de uno (es decir, existen varios padres de la clase particular analizada), se introduce el concepto de herencia múltiple.

Clase B: su lógica es similar a la del atributo *Clase A*, pero haciendo referencia a los hijos (subclase) de la superclase.

Propiedades asociadas a la clase *método*:

- Nombre: propiedad similar a la de la clase *clase*.
- Entrada: se refiere a los datos de entrada (instancias de la clase entrada) que requiere un método en particular que se puede ejecutar. Todos los métodos, necesitan variables de entrada para ejecutarlos (al menos una orden de ejecución) y, por tanto, esta variable se requiere; además, algunos métodos pueden requerir una o más entradas y, por esto, la propiedad es de cardinalidad múltiple.
- Salida: esta propiedad, se refiere a los datos de salida (instancias de la clase salida) que arroja un método en particular después de que se ejecuta. Todos los métodos, arrojan salidas al ejecutarlos (al menos un resultado de ejecución) y, por tanto, esta variable se requiere; además, algunos métodos pueden arrojar una o más salidas y, por esto, la propiedad es de cardinalidad múltiple.
- Visibilidad: propiedad similar a la de la clase *clase*.
- Clase relacionada: esta propiedad, hace referencia al conjunto de instancias de la clase *método* que se relacionan con una instancia de la clase *clase* en particular. Indica que los métodos no se aíslan dentro del modelo sino que se asocian con las instancias de la clase *clase*. Un método no tiene sentido por sí solo, sino en función de una clase que lo contenga y, por tanto, esta es una propiedad requerida; es posible, además, que un mismo método se asocie con más de una instancia de la clase *clase*, por lo cual esta propiedad es de cardinalidad múltiple.

Propiedades asociadas a la clase *tipo de dato*:

- Número: esta propiedad, es un consecutivo de los tipos de datos que se asocian a una variable. Es requerida y de tipo entero.
- Tipo de dato primitivo: esta propiedad de la subclase primitivo, hace referencia a los valores que puede tomar un dato primitivo, previamente mencionados.
- Clase relacionada: esta propiedad de la subclase *instancia*, hace referencia al conjunto de instancias de la clase *clase* cuyos elementos se pueden tomar como valores para un tipo de dato. Es una propiedad requerida y de un único valor.

Propiedades asociadas a la clase *relación*:

- Clase A: se refiere al conjunto de instancias de la clase *clase* que se relacionan con otro conjunto de instancias de la clase *clase* (clase B). Dado que las relaciones se dan entre mínimo dos instancias de clases, esta propiedad es obligatoria; puede ser de cardinalidad múltiple, pues un mismo tipo de relación puede enlazar a uno o más pares de instancias de clase. En las relaciones herencia, esta propiedad permite seleccionar sólo instancias de la subclase *superclase*, es decir, el conjunto de instancias en esta propiedad denota a los padres de la relación herencia.
- Clase B: su lógica es similar a la de la propiedad *Clase A*, pero haciendo referencia al otro extremo de la relación.
- Cardinalidad A: esta propiedad, se refiere a cuántas son las clases de la propiedad *Clase A* (en términos de una o más de una); es una propiedad obligatoria que sirve para establecer diferencias entre los tipos de relaciones. En la relación *herencia simple* esta propiedad adopta obligatoriamente el valor de 1, asegurando que exista sólo un padre en la herencia.
- Cardinalidad B: su lógica es similar a la del atributo *Cardinalidad A*, pero refiriéndose al atributo *Clase B*.

Propiedades asociadas a la clase *objeto*:

- Nombre: propiedad similar a la de la clase *clase,*
- Clase relacionada: de la misma forma que la clase *clase,* tiene una propiedad que la asocia con uno o varios objetos, un objeto tiene una propiedad que lo asocia con una o más clases (en este caso, única, pues es poco probable que un mismo objeto sea instancia de dos clases al mismo tiempo); es una propiedad requerida porque, dentro del modelo orientado a objetos, no existen objetos que no sean instancia de una clase.
- Se resalta que las subclases heredan los atributos de las superclases de las cuales se derivan. En la Tabla 1, las celdas sombreadas se refieren a las propiedades intrínsecas las de las clases, mientras que las celdas no sombreadas se refieren a las propiedades extrínsecas, es decir, aquellas que dan cuenta de la comunicación de la clase con el exterior.

Propiedades asociadas a la clase *variable*:

- DescripciónValor: esta propiedad, describe el tipo de valor que puede ser una entrada o una salida (subclases de la clase *variable*); es una propiedad requerida (corresponde a la definición misma de una variable) y es de tipo cadena de caracteres.
- Nombre: propiedad similar a la de la clase *clase*.
- Tipo: esta propiedad, se refiere al tipo de dato que es una variable, tomando instancias de la clase *tipo de dato*. Una variable, se puede definir con una única instancia de *tipo de dato* (aunque, en realidad, una variable pudiera clasificar en uno o más tipos de datos, se elige aquel que más se ajuste a las necesidades–por ejemplo, en términos de espacio de memoria en el computador).

Visibilidad: propiedad similar a la de la clase *clase*.

- Clase relacionada: de la misma forma que la clase *clase* tiene una propiedad que la asocia con uno o varios atributos, un atributo tiene una propiedad que lo asocia a una o más clases (pueden ser varias clases las que compartan un mismo atributo); su lógica es similar a la de la clase *objeto*.
- Método: de la misma forma que la clase *método* tiene una propiedad que la asocia con una o varias entradas, una entrada tiene una propiedad que la asocia con uno o más métodos (pueden ser varios métodos que requieran una misma entrada para su ejecución); es una propiedad requerida, pues dentro del modelo orientado a objetos no existen variables de entrada «sueltas» (sin objetivo alguno) que no estén asociadas a, mínimo, un método. Así como la clase *método* tiene una propiedad que la asocia con una o varias salidas, una salida tiene una propiedad que la asocia a un único método (único porque un dato no puede ser, al mismo tiempo, salida de dos métodos diferentes); es una propiedad requerida, por la misma razón que en el caso de las variables de entrada.

Propiedades asociadas a la clase *mensaje*:

- Contenido: es lo que lleva un mensaje; se asocia un único contenido a un único mensaje y, por eso, su cardinalidad es 1; es una variable requerida para definir correctamente una instancia de la clase mensaje.
- Clase A: se refiere al conjunto de instancias de la clase *clase* que se comunican con otro conjunto de instancias de la clase *clase* a través del envío de este mensaje (es decir, clases emisoras del mensaje). Dado que los mensajes se transmiten entre, mínimo, dos instancias de clases, esta propiedad es obligatoria; además, es de cardinalidad 1 porque en una transmisión de un mensaje, existe un único emisor de éste (otro emisor significa otro mensaje).
- Clase B: su lógica es similar a la de la propiedad *Clase A*, pero refiriéndose a las clases receptoras del mensaje.

Propiedades asociadas a la clase *evento*:

- Mensaje evento: de la misma forma que la clase *mensaje* tiene una propiedad que la asocia con un evento, un evento posee un atributo que lo asocia con el mensaje que lo produce (instancias de la clase mensaje).

VII. IMPLEMENTACIÓN DE LA ONTOLOGÍA PROPUESTA

La ontología descrita anteriormente, se desarrolló empleando la herramienta Protégé, una plataforma de código abierto y libre desarrollado en la Universidad de Stanford [19], que proporciona un conjunto de herramientas para construir modelos del dominio y aplicaciones basadas en conocimiento con ontologías. En las Figuras 3, 4 y 5, se representa la ontología desarrollada para el paradigma orientado a objetos (las imágenes se obtuvieron empleando el plugin Ontoviz de Protégé). En la Figura 3, se representa el concepto *variable*, donde se puede ver que dicho concepto, además de tener un nombre específico, posee también un tipo de dato. Además, se puede inferir que los atributos y los parámetros de salida y de entrada son variables. En la Figura 5, se explica de forma más detallada los conceptos de *parámetros de salida* y de *entrada*.

Según la Figura 4, todas las clases pueden tener varios tipos de relaciones con otras clases. Estas relaciones pueden ser de tipo *asociación* y *agregación*, de *herencia simple* o de *herencia múltiple*. En el primer tipo de relación, se debe especificar claramente cada una de las clases que se relacionan, con el fin de guardar una referencia que permita agruparlas; en el segundo tipo de relación, se debe especificar la clase padre para todas las clases hijas y las clases hijas para todas las clases padre, así como en la relación de herencia múltiple se deben especificar uno o varios padres para cada clase hija y uno o varios hijos para todas las clases de tipo padre. De la Figura 4, también se puede concluir que toda clase puede tener una o varias instancias de objetos, las cuales tienen una posición específica en memoria y almacenan todos los datos de una instancia.

En la Figura 5, se puede observar que un método posee un parámetro de salida y uno o varios parámetros de entrada. Dichos parámetros de entrada, pueden ser: variables de tipo primitivo,

instancias de una clase o atributos de una clase. Adicionalmente, en la Figura 4 se muestra que, para invocar un método específico, se debe crear un evento en el sistema, y que toda clase debe tener unos métodos y atributos propios.

VIII. CONCLUSIONES

En el presente artículo, se presenta una ontología donde se recogen los principales conceptos del paradigma orientado a objetos, generalizando y estandarizando el vocabulario de dicho dominio, permitiendo, de esta manera, facilitar el proceso de enseñanza/aprendizaje a profesionales en ingeniería no familiarizados con este tema.

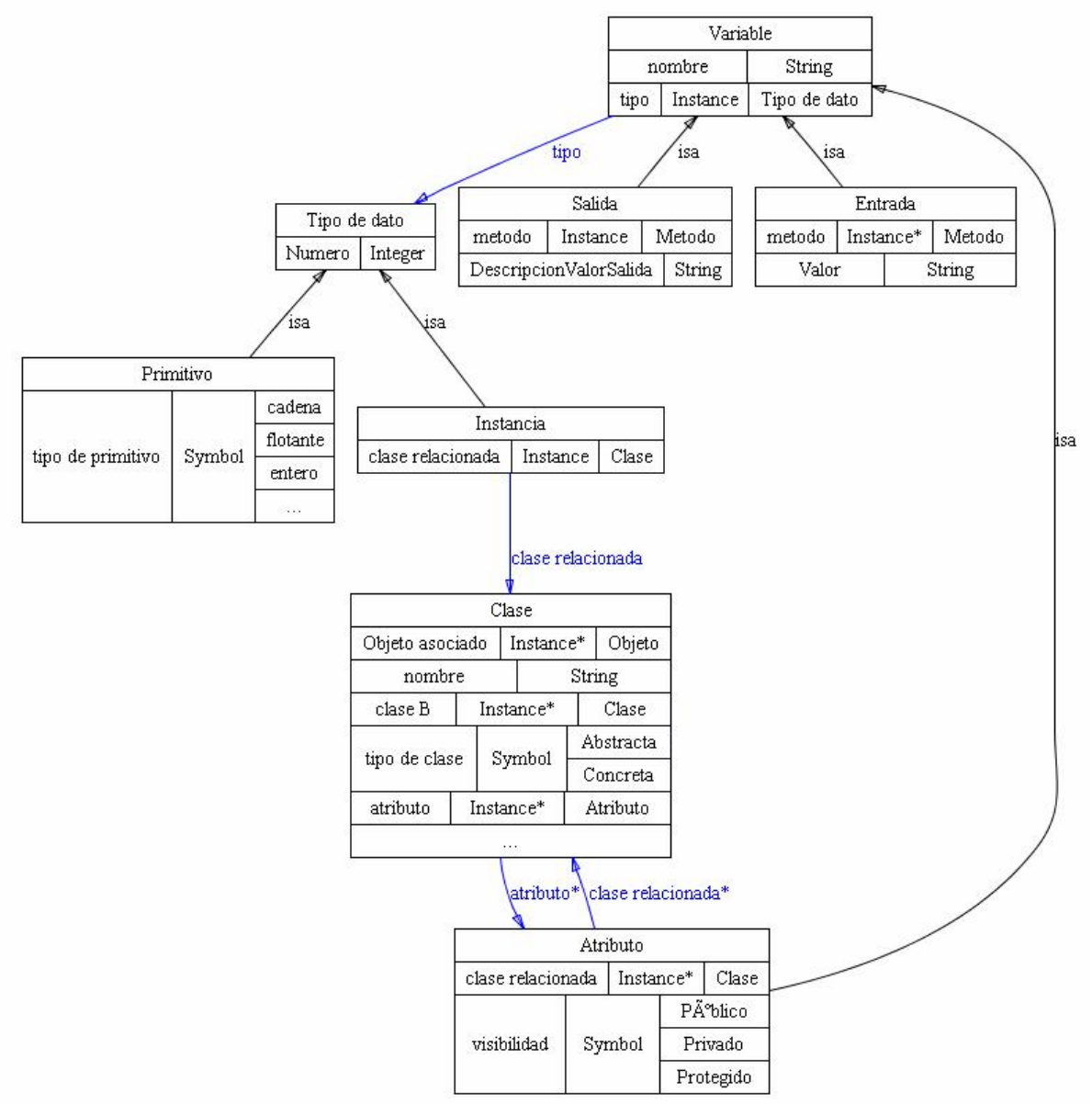

Figura 3. Implementación de la ontología propuesta en Protégé. El concepto *variable* y sus relaciones.

En el artículo, se utilizó la metodología propuesta por NOY y McGUINNESS, la cual permite desarrollar ontologías de forma clara, procurando tener en cuenta todos los elementos que componen un dominio específico.

Las ontologías, son útiles para estandarizar el conocimiento en un dominio dado, en tanto que el paradigma orientado a objetos apoya desarrollos en diversas áreas del conocimiento, tales como la Medicina, la Industria Manufacturera y, en general, la Ingeniería del Software.

Una exploración de literatura sobre trabajos previos de ontologías en el paradigma orientado a objetos, muestra que, aunque no se evidencia implementación alguna, existen acercamientos conceptuales al modelado de dicho paradigma a través de ontologías, lo cual se convierte en una guía básica para desarrollos posteriores, como el presentado en este artículo. Finalmente, se presentó la ontología propuesta para la descripción del paradigma orientado a objetos, mostrando los detalles de su implementación, la cual se realizó en Protégé.

IX. TRABAJO FUTURO

El trabajo desarrollado en el presente artículo, se puede extender incorporando a la ontología otros aspectos de la ingeniería de software, tales como la educción de requisitos, modelos de análisis y fases de implementación. Todo esto con el fin de facilitar el desarrollo de software y una posible automatización de sus procesos, así como el intercambio de información entre ellos.

Conceptos de gran abstracción, pero, igualmente, interesantes dentro del paradigma orientado a objetos, tales como *encapsulamiento* y *polimorfismo,* se podrían tener en cuenta en la búsqueda de mejoras a la ontología propuesta en este artículo. Asimismo, conceptos relevantes del diseño orientado a objetos podrían jugar un papel importante en un trabajo futuro.

La literatura revisada, sugiere que, de un modo más general, se requiere el desarrollo a mayor profundidad de ontologías de metamodelos (como el del paradigma orientado a objetos), a fin de potencializar las capacidades de ambas herramientas.

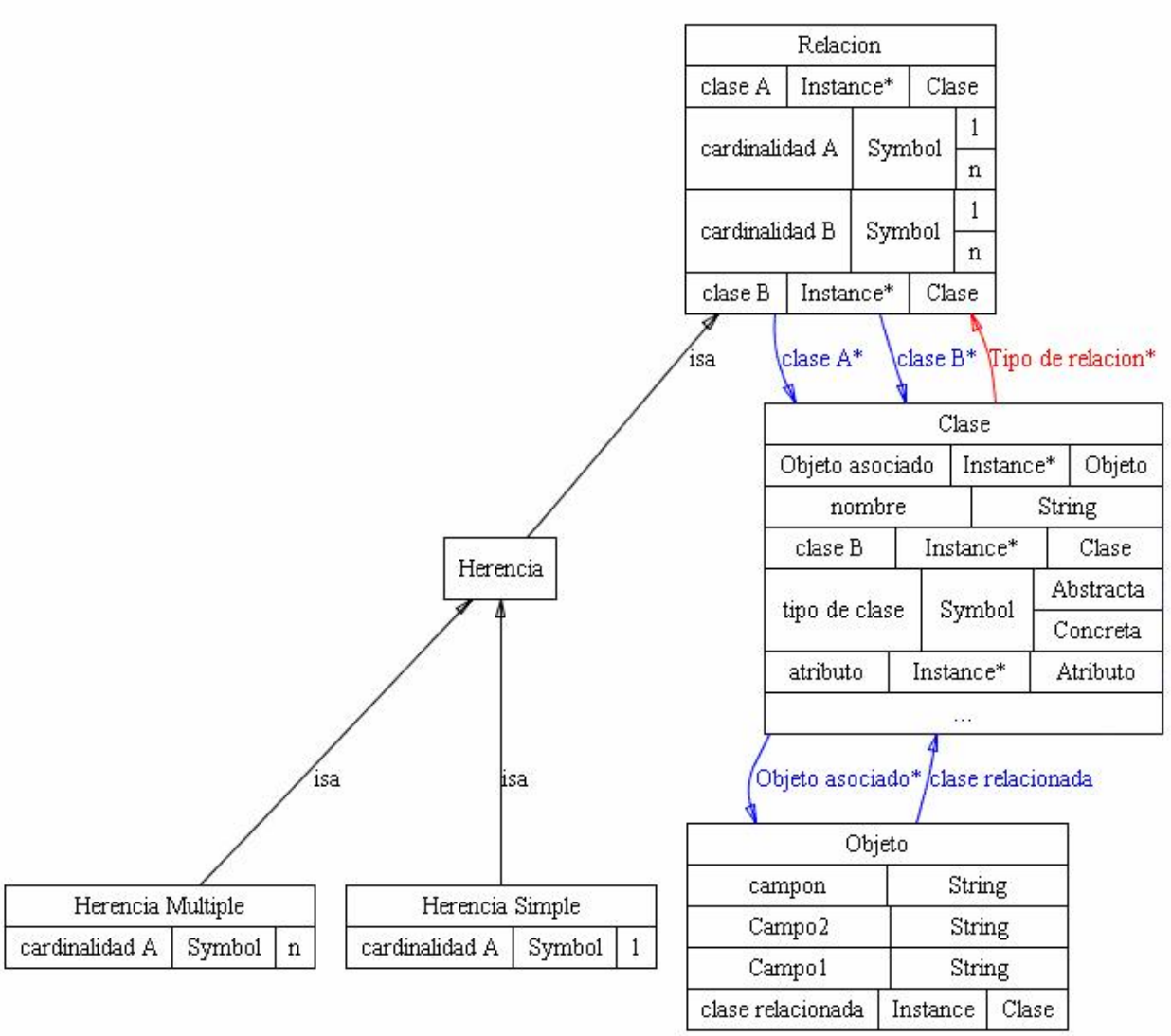

Figura 4. Implementación de la ontología propuesta en Protégé. El concepto *clases* y su relación con el concepto *relación*

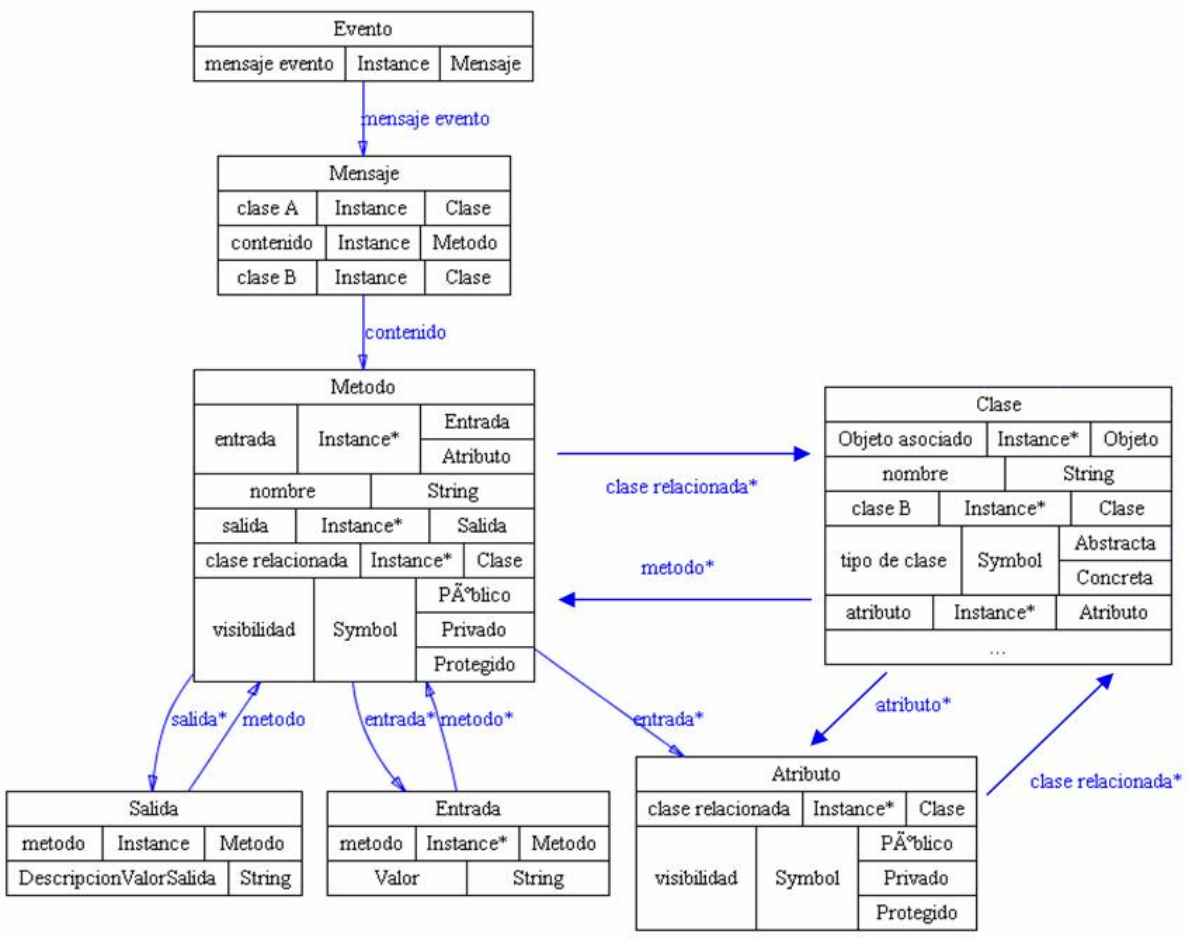

Figura 5. Implementación de la ontología propuesta en Protégé. El concepto clases y su relación con los conceptos método, mensaje y evento

REFERENCIAS

[1] Spinellis, D.D., 1993. Programming paradigms as object classes: A structure mechanisms for multiparadigm programming. Tesis presentada para obtener el grado Doctor en Filosofía. London University.

[2] Elrad, T.; Filman, R. y Beader, A., 2001. Aspect-oriented programming: Introduction. En: Communications of ACM. Vol 44, No. 10, pp. 28-32

[3] Booch, G., 1996.. Análisis y diseño orientado a objetos con aplicaciones. Addison-Wesley / Díaz de Santo. 2° Edición.

[4] Chandrasekaran, B. y Josephson, J.R., 1999. What are ontologies, and why do we need them?. En: IEEE Intelligent Systems, Vol. 14, pp. 20-26.

[5] Wand, Y., 1996. Ontology as a foundation for meta-modelling and method engineering. En: Information and Software Technology, Vol. 38, pp, 281–287.

[6] Brinkkemper, S.; Motoshi, S. y Harmsen, F., 1999. Meta-Modelling based assembly techniques for situational method engineering. En: lnformation Systrms. Vol. 24, No. 3, pp. 209-228.

[7] Calero, C., Ruiz, F., Baroni, A., Brito E., Abreu, F. y Piattini, M., 2006. An ontological approach to describe the SQL:2003 object-relational features. En: Computer Standards and Interfaces . Elsevier. Vol.28, No. 6, pp. 695-713.

[8] Garzás, J., y Piattini, M., 2007. An ontology for understanding and applying object-oriented design knowledge. En: International Journal of Software Engineering and Knowledge Engineering (IJSEKE), Vol 17, No. 3.

[9] Tervonen, I., 1990. Object-oriented development as a multiview software construction methodology. En: Proceedings of the 23th HICSS Conference, Hawaii, Vol II, IEEE Computer Society Press, pp. 113-120.

[10] Booch, G., 1994. Coming of age in an object – Oriented world. En: IEEE Software. Vol. 11, pp. 33-41

[11] Nierstrasz, O. A., 1989. A survey of object-oriented concepts. En: Object-oriented concepts, Databases and applications, ed. W. Kim and F. Lochovsky. ACM Press and Addison-Wesley, pp. 3-21,

[12] World Wide Web Consortium. Ontology. Disponible en Internet: http://www.w3.org/

[13] Pinto, H. y Martins, J., 2004. Ontologies: How can they be built?. En: Knowledge and Information Systems, Springer-Verlag, Vol 6, pp. 441-464.

[14] Noy, N. F. y Mc.Guinness, D. L., 2001. Ontology development 101: A guide to creating your first ontology. Stanford knowledge systems laboratory technical report KSL-01-05 and Stanford medical informatics technical report SMI-2001-0880.

[15] Happel, H. J. y Seedof, S., 2006. Applications of ontologies in software engineering. En: 2nd. International workshop on semantic web enabled software engineering. SWESE. Athens, GA, U.S.A.

[16] Murthy, U. S. y Wiggins, C. E. Jr., 1993. Object-oriented modeling. Approaches for designing accounting information systems. En: Journal of Informatton Systems, Vol, 7, No. 2, pp. 97-111.

[17] Kim, W., 1990. Object-oriented databases: definition and research directions. En: IEEE Transactions on knowledge and data engineering. Vol. 2, No. 3, pp. 327-341.

[18] Alagar, V.S. y Periyasamy, K., 1992. Methodology for deriving an object-oriented design from functional specifications. En: Software Engineering Journal, vol. 7(4), pp. 247-263.

[19] Stanford University. The Protégé ontology editor and knowledge acquisition system. Disponible en Internet: http:// protege.stanford.edu.