Automatización del Diseño de Bases de Datos Objeto-Relacionales basada en MDA y XML

María Fernanda Golobisky[1] y Aldo Vecchietti[1]
[1] INGAR - Instituto de Desarrollo y Diseño – CONICET - UTN
Avellaneda 3657, (3000) Santa Fe, Argentina
{mfgolo, aldovec}@santafe-conicet.gov.ar

Resumen. El modelo objeto-relacional, soportado por el estándar SQL:2003, permite el tratamiento de datos y relaciones complejas, con mayores alternativas de diseño, y un pasaje del modelo conceptual al modelo lógico menos directo que en el modelo relacional. El proceso de diseño de estas bases de datos no cuenta aún con una metodología consensuada, por lo que es fundamental encontrar una que sea aceptada por la comunidad de bases de datos. Con ese objetivo se ha desarrollado una metodología que obtiene el diseño de la base de datos objeto-relacional, a partir de un diagrama de clases UML. En este artículo se presenta un resumen de la misma y se muestra el prototipo de una herramienta que la implementa utilizando MDA. Para la definición y transformación de los modelos se empleó la tecnología XML. La obtención del código SQL para la creación del esquema de base de datos objeto-relacional se realiza utilizando el lenguaje Java.

Palabras clave: Objeto-relacional, SQL:2003, UML, MDA, XML.

1. Introducción

La base de datos es un componente fundamental dentro de un sistema de información, que la aplicación de una metodología de diseño para la misma es una de las principales tareas que se debe afrontar para gestionar eficiente y eficazmente un sistema de información. El diseño adecuado de la base de datos es una de las tareas de mayor relevancia en el ciclo de vida de un sistema de información, ya que tiene un impacto importante sobre el mismo, pues si está correctamente diseñada permite obtener acceso a información exacta y actualizada. Hace muchos años atrás la mayoría de los diseños de bases de datos se realizaba de forma manual. Actualmente existen herramientas que permiten automatizar el diseño pero todas ellas obtienen esquemas de bases de datos relacionales. Estas herramientas reciben el nombre de mapeadores objeto-relacionales y realizan la conversión de objetos UML, o de modelos entidad-relación, a esquemas relacionales.

Si bien el uso de los sistemas relacionales está ampliamente extendido, ya ha evolucionado hacia un modelo más expresivo, el modelo objeto-relacional, soportado por el estándar SQL:2003. Este nuevo tipo de base de datos permite el tratamiento de datos y relaciones complejas, con mayores alternativas de diseño, y un pasaje del modelo conceptual al modelo lógico menos directo que en el modelo relacional. Sin embargo, el proceso de diseño de estas bases de datos no cuenta aún con una

metodología consensuada, por lo que es fundamental encontrar una que sea aceptada por la comunidad de bases de datos.

En la literatura existen numerosos trabajos relacionados con propuestas de transformación y diseño de bases de datos relacionales. Mok y Paper [1] presentaron una propuesta de de transformación de diagramas de clases UML en tablas anidadas normalizadas, pero su trabajo no contribuyó con ningún procedimiento formal para realizar mapeos más generales. Marcos, Vela, Cavero, y Cáceres [2] propusieron algunas guías para realizar las transformación de diagramas de clases UML in objetos del estándar SQL:1999 y luego en Oracle 8i. Los autores proporcionaron algunas explicaciones sin profundizar sobre el tema. Estos mismos autores [3] presentaron una metodología de para el diseño de BDOR en la cual definieron estereotipos para las transformaciones, en este trabajo no propusieron ninguna forma de cómo automatizar las transformaciones. Arango, Gómez, y Zapata [4] plantearon reglas de mapeo desde diagramas de clases a Oracle 9i empleando teoría de conjunto. Grissa-Touzi and Sassi [5] desarrollaron una herramienta para el diseño e implementación de BDOR denominada Navig-tools por la cual el usuario puede generar el código del modelo en SQL3. La herramienta tenía el diagrama entidad-relación como entrada. La mayoría de los trabajos presentados en la literatura no proveen reglas de transformación consistentes basados en los metamodelos involucrados en el diseño. Tampoco se han caracterizado los elementos de los mismos para generar una metodología generalizable.

Con ese objetivo hemos desarrollado una metodología que obtiene el diseño de la base de datos objeto-relacional, a partir de un diagrama de clases UML. Esta metodología forma parte del trabajo de tesis doctoral [6]. En este artículo se presenta un resumen de la misma y se muestra el prototipo de una herramienta que implementa la metodología desarrollada, utilizando una Arquitectura Dirigida por Modelos (MDA), donde éstos son considerados entidades fundamentales, y las transformaciones entre los diferentes modelos son definidas, formalizadas e implementadas. La propuesta de arquitectura comienza con un modelo independiente de la plataforma (PIM), un modelo conceptual de datos representado por un diagrama de clases UML, el cual es transformado en un primer modelo específico de la plataforma (PSM), conformado por los elementos objeto-relacionales (O-R) no-persistentes. Luego, este PSM es transformado en otro PSM, correspondiente a los elementos O-R persistentes. Para la definición y transformación de los modelos que requiere MDA se empleó la tecnología XML. El lenguaje de definición de esquemas XML [7] fue el apropiado para definir los metamodelos ya que permite describir el tipo y la estructura de los documentos XML. La obtención del código SQL para la creación del esquema de base de datos objeto-relacional (BDOR) a partir del último PSM obtenido se realiza utilizando el lenguaje Java. La arquitectura propuesta es sencilla de implementar y la incorporación y modificación de nuevas reglas se puede realizar fácilmente. A conocimiento es estos autores no existe una herramienta con estas características actualmente, las más populares tales como Hibernate, SQL-Estudio, ERWIN, etc., realizan los mapeos a tablas de bases de datos relacionales basadas en las especificaciones del estándar SQL'92.

El artículo es estructurado como sigue: la sección 2 resume el desarrollo de la metodología de diseño objeto-relacional. En la sección 3 se expone una arquitectura de la metodología de transformación empleada en la herramienta desarrollada. En la

sección 4 se detalla la herramienta y sus funcionalidades. La sección 5 presenta las conclusiones del trabajo y el trabajo futuro.

2. Metodología de Diseño Objeto-Relacional

La metodología propuesta para el diseño de BDOR parte del modelo conceptual basado en el diagrama de clases UML y obtiene el modelo lógico, basado en la tecnología O-R de bases de datos. En la transformación entre estos dos modelos se estableció una capa intermedia donde se ubican los elementos no-persistentes propuestos por el estándar: tipos estructurados, tipos referencia, tipos fila, colecciones (arreglos y multisets). Esta capa intermedia también forma parte del modelo lógico. Por esta razón, el paso del modelo conceptual al lógico no se puede efectuar en forma directa, como en el caso relacional [8].

En el desarrollo de la metodología de diseño se emplearon metamodelos que describen la estructura, relaciones y semántica de los diferentes modelos empleados en este trabajo. Se utilizó una versión simplificada del metamodelo para diagramas de clase de UML 2.0, basada en la especificación de OMG (*Object Management Group*), tomando solo las partes del metamodelo que resultan de interés para este trabajo (Figura 1). Además se empleó el metamodelo de SQL:2003 propuesto por Baroni y otros, que está dividido en dos partes: Tipos de datos y Esquema(Figuras 2 y 3).

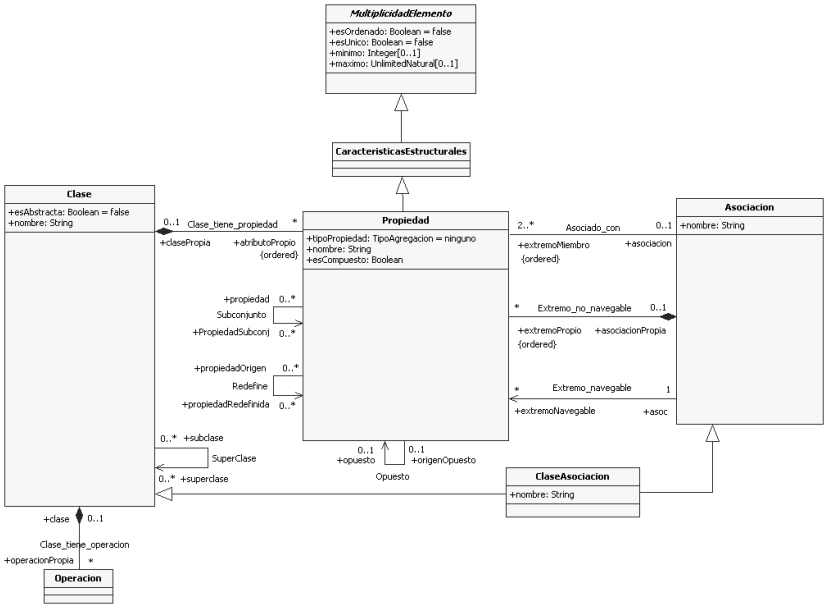

Fig. 1. Metamodelo del diagrama de clases UML

Por razones de espacio no se explican en detalle los distintos metamodelos. Para una mejor comprensión de los mismos, ver [9], [10] y [11].

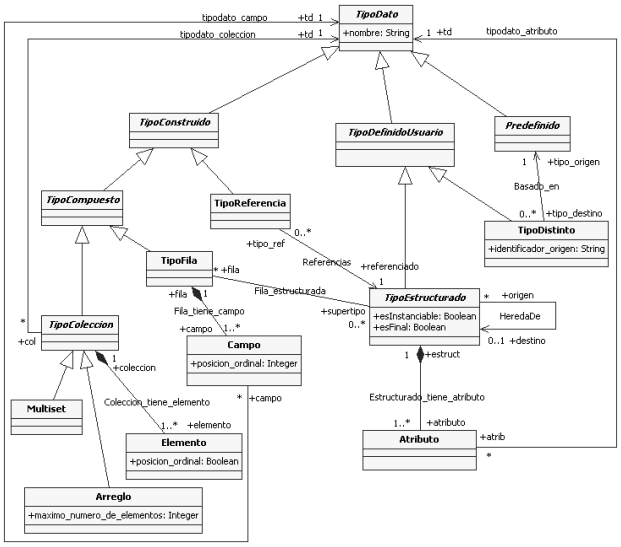

Fig. 2. Metamodelo de los tipos de datos de SQL:2003

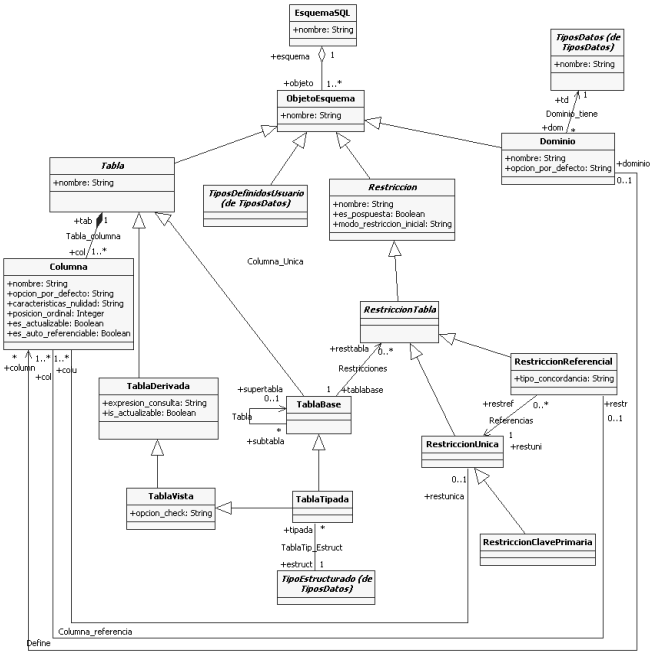

Fig. 3. Metamodelo del esquema de SQL:2003

En la capa de diagramas de clases UML (1era capa) se ubica el metamodelo UML, en la capa intermedia se ubica el metamodelo de los tipos de datos de SQL:2003, y en la capa objeto-relacional persistente se ubica el metamodelo del esquema del estándar. La transformación de los objetos UML en componentes de SQL:2003 se realiza en dos etapas, como se puede observar en la Figura 4.

Debido a que SQL:2003 es un superconjunto de las versiones de SQL anteriores, tiene la habilidad de soportar, además, transformaciones relacionales puras, es decir de objetos UML a tablas relacionales. Estas transformaciones se realizan en un solo paso. Puesto que no ocurre lo mismo con las transformaciones objeto-relacionales, en la figura anterior se grafican los mapeos 1, 2 y 3, a los efectos de mostrar la diferencia entre ellos.

Para poder transformar los objetos UML en la tecnología O-R hemos realizado una caracterización de los elementos que componen cada capa de mapeo. La misma no fue incluida aquí por cuestiones de espacio, pero puede verse en [6].

En la sección siguiente se muestran las transformaciones realizadas en base a la caracterización hecha.

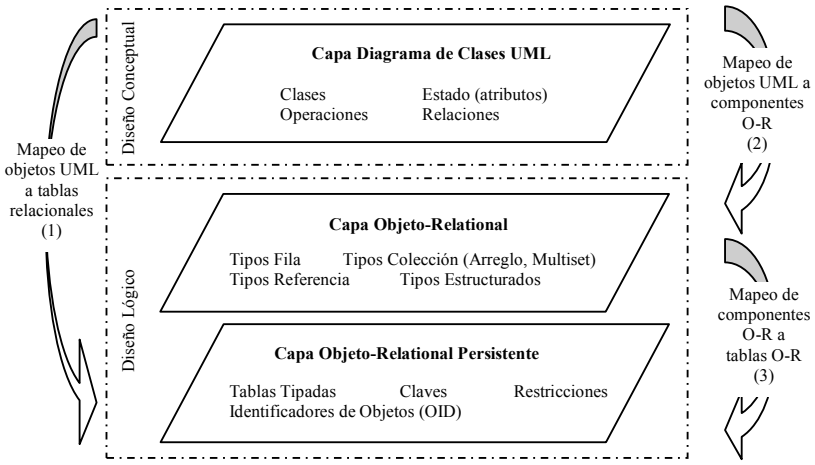

Fig. 4. Capas de mapeo y sus componentes

2.1. Transformación de componentes UML en elementos O-R no-persistentes

Transformación de Clases

Cada clase (*C*) del diagrama de clases UML, correspondiente al diseño conceptual, debe traducirse en un tipo estructurado (*TE*) de la tecnología objeto-relacional.

$C \rightarrow TE$

Transformación de Atributos

Las clases poseen atributos (A_C) que son transformados en atributos de tipos estructurados (A_{TE}).

$A_C \rightarrow A_{TE}$

Los atributos con multiplicidad de 1 se transforman atributos atómicos del tipo estructurado. En cambio, los atributos de clase con multiplicidad mayor a 1, que opcionalmente pueden tener límites inferior y superior indicando el número de valores posibles, se transforman utilizando los tipos colección del estándar SQL:2003. Sin embargo, estos tipos no pueden utilizarse de manera arbitraria sino que deben cumplir algunas condiciones:

- Si los límites inferior y superior son conocidos, el atributo de clase se transforma en un arreglo con una longitud específica que eventualmente puede crecer.

  $A_C \rightarrow$ *Arreglo*
- Si los límites inferior y superior no están definidos, el atributo se transforma en un multiset ya que este tipo de colección no limita la cantidad de valores a almacenar.

  $A_C \rightarrow$ *Multiset*

Transformación de Pseudoatributos

Si una clase participa de una relación (de asociación, agregación, composición, clase asociación), además de atributos posee *extremos de asociación* que indican el papel que ésta juega en esa relación. Estos extremos pueden ser navegables o no.

Los extremos de asociación navegables se consideran *pseudoatributos* de la clase que se encuentra en el extremo opuesto de la relación. Los pseudoatributos son elementos fundamentales puesto que hemos realizado el mapeo de las relaciones entre clases en base a ellos.

- Si el pseudoatributo (*PA*) tiene multiplicidad de 1 y pertenece a una clase que participa de una asociación, se lo transforma por medio de un tipo referencia atómico, correspondiente al tipo estructurado de la clase que se encuentra en el extremo opuesto de la relación.

  PA con multiplicidad de 1 $\rightarrow$ *Referencia*
- Si en cambio, la multiplicidad está definida, se transforma en un arreglo de tipos referencia.

  PA con multiplicidad de n $\rightarrow$ *Arreglo de Referencia*

- Si la multiplicidad es indefinida, es decir, tiene un limite máximo de *, se transforma en un multiset de tipos referencia.

  *PA con multiplicidad de ** $\rightarrow$ *Multiset de Referencia*

- Si el pseudoatributo representa el extremo de asociación correspondiente a la "parte" de una relación de composición, se transforma embebiendo el tipo

estructurado (*TEP*) correspondiente a la parte, dentro del tipo estructurado correspondiente al todo (*TET*). También se debe tener en cuenta la multiplicidad del pseudoatributo.

- Si la multiplicidad es de 1, se transforma como un atributo del tipo estructurado correspondiente a la parte, dentro del tipo estructurado correspondiente al todo.

  PA con multiplicidad de 1 → TEP embebido en TET

- Si la multiplicidad es de *n* valores, se lo transforma en un arreglo de tipos estructurados, con tamaño máximo de *n*.

  PA con multiplicidad de n → Arreglo de TEP embebido en TET

- Si la multiplicidad es indefinida (*), se transforma en un multiset de tipos estructurados.

  *PA con multiplicidad de * → Multiset de TEP embebido en TET*

- Si el pseudoatributo representa el extremo de asociación correspondiente al "todo" de una relación de composición, lo más "natural" y conveniente de realizar es no transformar el pseudoatributo debido a que, en general, los extremos de este tipo son no navegables.

Transformación de Operaciones

Las operaciones (*O*) de UML son tratadas de manera simple, por lo que cada una de ellas se transforma en un método (*M*) del tipo estructurado.

O → M

Transformaciones de Relaciones de Generalización-Especialización

En la caracterización realizada como parte de la metodología de diseño desarrollada tanto a las clases del la capa de diagrama de clases UML, como a los tipos estructurados de la capa objeto-relacional, les hemos agregado diferentes calificadores (*esAbstracto, superclase heredaDe, esInstanciable*) que brindan la información necesaria para su transformación. De esta manera, al transformar cada clase en un tipo estructurado se transforman también estos calificadores: el calificador *esAbstracto* se transforma en *esInstanciable*; mientras que el calificador *superclase* se transforma en *heredaDe*.

C(…, esAbstracto, superclase) → TE(…, esInstanciable, heredaDe)

Transformación de Clases Asociación

Para la transformación de una clase asociación (*CA*) primero debe rectificarse su representación. Este se puede ver en las Fig. 5 y Fig.6, en la primera se muestra la clase asociación, mientras que en la Fig. 6 la rectificación de la misma, donde nuevos

extremos de asociación (y por lo tanto pseudoatributos) se han agregado a la representación original.

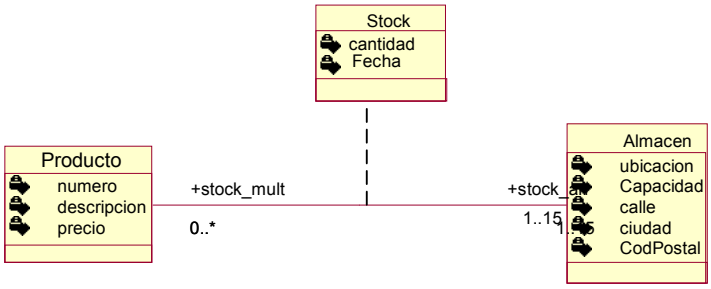

Fig. 5. Diagrama de una clase asociación

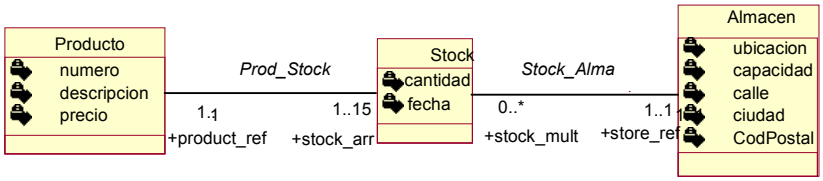

Fig. 6. Diagrama de una clase asociación rectificada

A partir de la Fig. 6 la clase asociación se transforma en un tipo estructurado.

CA → *TE*

2.2. Transformación de elementos O-R no-persistentes en tablas tipadas

Una vez que se obtuvieron los elementos O-R no-persistentes se está en condiciones de almacenarlos en tablas tipadas, de manera que puedan tener persistencia. Este paso corresponde a la transformación de los elementos de la capa O-R en elementos de la capa O-R persistente.

En general, cada tipo estructurado de la capa O-R se transforma en una tabla tipada (*TT*) de la capa O-R persistente.

TE → *TT*

Para completar el mapeo, el diseñador debe incorporar información extra que no se encuentra en los metamodelos de las capas de mapeo anteriores. Por ejemplo, no

existe un equivalente a las restricciones que se pueden definir sobre una tabla tipada (clave primaria, única, restricción de no nulo, del tipo check, etc).

Asimismo, se debe especificar el mapeo que se empleará para las relaciones de generalización-especialización (herencia). Como se vio en la sección anterior, la caracterización realizada sobre los elementos intervinientes en los mapeos, solicita especificar, para cada clase UML, la superclase de la cual hereda (en caso de participar de una relación de herencia). El mapeo de la herencia a tablas tipadas se puede realizar de 3 formas diferentes:

- *Modelo Plano*: Dado que el modelo plano contempla la definición de una sola tabla, se debe crear una tabla tipada para el supertipo, con la propiedad de sustitutabilidad, lo que hace posible el almacenamiento de subtipos en la misma tabla del supertipo.
- *Partición Vertical*: Dado que la partición vertical implica que se generará una tabla para cada una de las clases que componen la jerarquía, se debe crear una tabla tipada para cada tipo de la jerarquía. A cada tabla tipada se le quita la propiedad de sustitutabilidad, de manera que solamente puedan almacenarse los tipos para los cuales fueron definidas.
- *Partición Horizontal*: Dado que la partición horizontal implementa sólo las tablas correspondientes a los subtipos, se debe crear una tabla tipada para cada uno de ellos, trasladando a las mismas todos los atributos del supertipo.

3. Arquitectura de la Metodología de Transformación

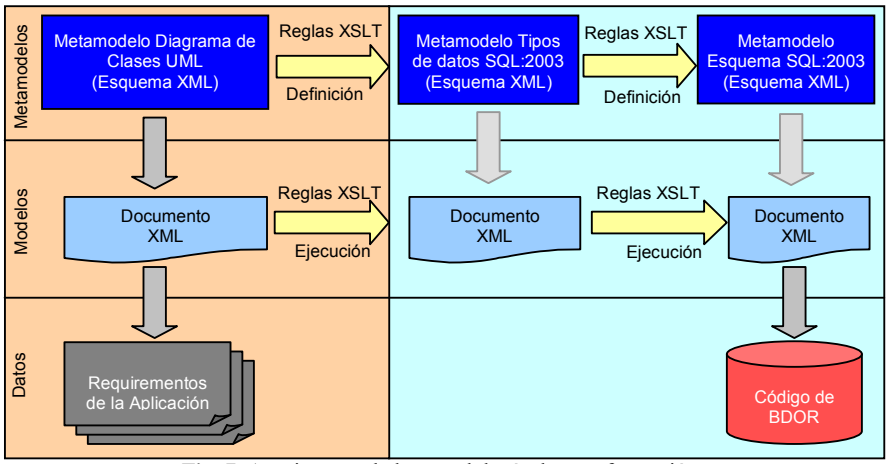

Fig. 7. Arquitectura de la metodología de transformación

La Fig. 7 muestra la arquitectura propuesta para realizar la transformación de modelos está basada en MDA y XML.

El proceso comienza con un Modelo Independiente de la Plataforma (PIM), el modelo de datos conceptual, el cual es transformado en dos Modelos Específicos de la

Plataforma (PSM), conformados por los tipos de datos y el esquema del estándar SQL, respectivamente. Luego, los PSMs son transformados en código SQL.

Como se puede observar en la figura anterior, la arquitectura propuesta tiene tres niveles de abstracción diferentes [12]:

- En el nivel de *Metamodelos* se ubican los esquemas XML generados para cada uno de los metamodelos que participan de las transformaciones (Diagrama de clases UML, Tipos de datos de SQL:2003 y Esquema de SQL:2003).
    - En este nivel se definen las reglas de mapeo entre los esquemas XML, escritas en el lenguaje XSLT [13].
- En el nivel de *Modelos* se encuentran los documentos XML que cumplen con los esquemas XML definidos en el nivel anterior. El primer documento XML es el correspondiente al metamodelo del diagrama de clases UML y representa la información jerarquizada y estructurada de la aplicación.
    - Los mapeos entre los documentos XML se realizan en este nivel, mediante la ejecución de las reglas XSLT definidas en el nivel de metamodelos.
- En el nivel de *Datos* se tiene la información de entrada y de salida del nivel de modelos. Los requerimientos de la aplicación, modelados mediante un diagrama de clases UML, conforman la entrada. Mientras que la salida es el código del esquema de base de datos objeto-relacional.

El proceso de transformación comienza con la definición del diagrama de clases UML de la aplicación, el cual se convierte luego en un documento XML que cumple con el esquema XML. Posteriormente, se ejecutan las reglas que se definieron para transformar los elementos de UML en los tipos de datos del estándar SQL:2003. Como resultado se obtiene el documento XML correspondiente a los tipos de datos de SQL:2003, el cual es convertido en otro documento XML que cumple con el esquema del estándar, mediante la ejecución de las reglas definidas para mapear los tipos de datos al esquema del estándar SQL:2003. Este último documento XML se utiliza para generar el código del esquema de base de datos objeto-relacional, mediante la aplicación de código escrito en Java.

Con respecto a la implementación de la arquitectura se creó un prototipo de herramienta, denominada *OR-Transformer*. Para la creación y edición de los esquemas y documentos XML se utilizó el software XMLSpy [14], que nos brindó un entorno flexible y eficiente. Las reglas de transformación fueron escritas utilizando el software MapForce [15].

4. Características de la Herramienta Desarrollada

La herramienta *OR-Transformer* permite confeccionar la definición de requerimientos de los usuarios, mediante la elaboración de un diagrama de clases UML. Para ello cuenta con una interfaz gráfica que le permite al usuario manejar los objetos de diseño. Además, aplica la metodología de transformación entre capas y genera el esquema lógico de una base de datos objeto-relacional. Como resultado obtiene el

código SQL de creación de la base, utilizando la sintaxis de Oracle 10g [16], además de un diagrama de clases UML que incluye los componentes objeto-relacionales.

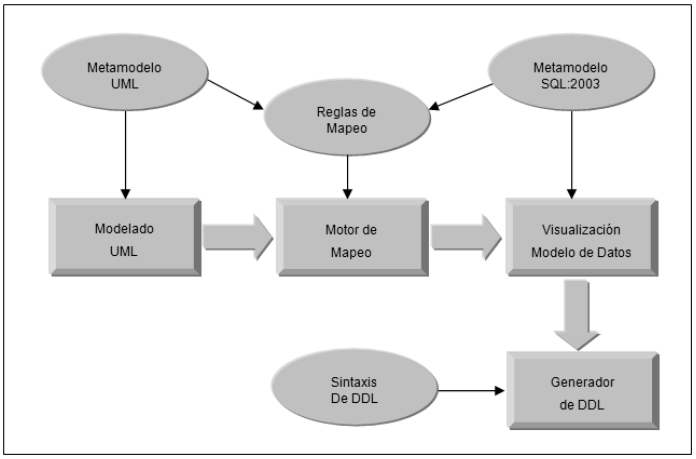

Fig. 8. Esquema general de la herramienta OR-Transformer

En la Figura 8 se puede observar un esquema general de la herramienta, formada por cuatro módulos que proporcionan determinadas funcionalidades, además de un conjunto de archivos conteniendo los metamodelos, reglas de mapeo y sintaxis DDL.

- **Archivos de metamodelos UML y SQL:2003:** contiene los esquemas XML correspondientes a los metamodelos del diagrama de clases UML y SQL:2003 empleados por la herramienta, así como los documentos XML generados a partir de éstos.
- **Archivo de reglas de mapeo:** contiene las reglas de mapeo XSLT definidas para transformar los documentos XML.
- **Archivo de Sintaxis de DDL:** contiene la sintaxis de Oracle de las sentencias DDL (lenguaje de definición de datos) para la creación de los objetos y tablas de una base de datos.
- **Módulo Modelado UML:** permite al usuario modelar sus requerimientos en un diagrama de clases UML, utilizando como base el metamodelo de UML. Este módulo conforma la entrada del proceso de diseño.
- **Módulo Motor de Mapeo:** este módulo aplica las reglas de mapeo definidas y genera un diagrama de clases UML, con extensiones objeto-relacional, a partir del diagrama antes modelado.
- **Módulo Visualización Modelo de Datos:** este módulo permite visualizar el diagrama objeto-relacional generado.
- **Módulo Generador de DDL:** toma el diagrama objeto-relacional y en base a la sintaxis de la base de datos, genera un archivo con las sentencias de generación del esquema de la base de datos objeto-relacional.

La herramienta se desarrolló empleando Java SE 6 *(Java Platform, Standard Edition 6 Release)* [17]. La presentación gráfica de la interfaz de usuario fue

desarrollada utilizando la biblioteca Swing para Java que forma parte de las *Java Foundation Classes (JFC)*. Los archivos XML son leídos y escritos utilizando *JDOM (Java-based Document Object Model)*, una biblioteca de código fuente open-source, para manipulación de datos XML. Para la ejecución de las transformaciones se utilizó también la librería Saxon, un potente procesador XSLT, open-source. Como entorno de desarrollo se utilizó Eclipse. El uso de este IDE se debe a que puede ser utilizado, no sólo en el desarrollo de productos software, sino también en el desarrollo de herramientas que permitan construir productos software. Eclipse es un marco de trabajo para el modelado y la integración de datos, permitiendo almacenar metamodelos y metadatos. Su principal ventaja radica en que su arquitectura está diseñada de forma que la mayor parte de la funcionalidad proporcionada está localizada en plug-ins o en conjuntos de plug-ins relacionados. Esto hace a Eclipse una herramienta extensible.

4.1. Funcionalidades de la herramienta

OR-Transformer permite realizar acciones concentradas en dos grandes grupos, uno de ellos relacionado con el modelado, y el otro relacionado con el esquema de la base de datos. Como se puede observar en la Figura 9, la herramienta posibilita la creación y edición de diagramas de clases UML.

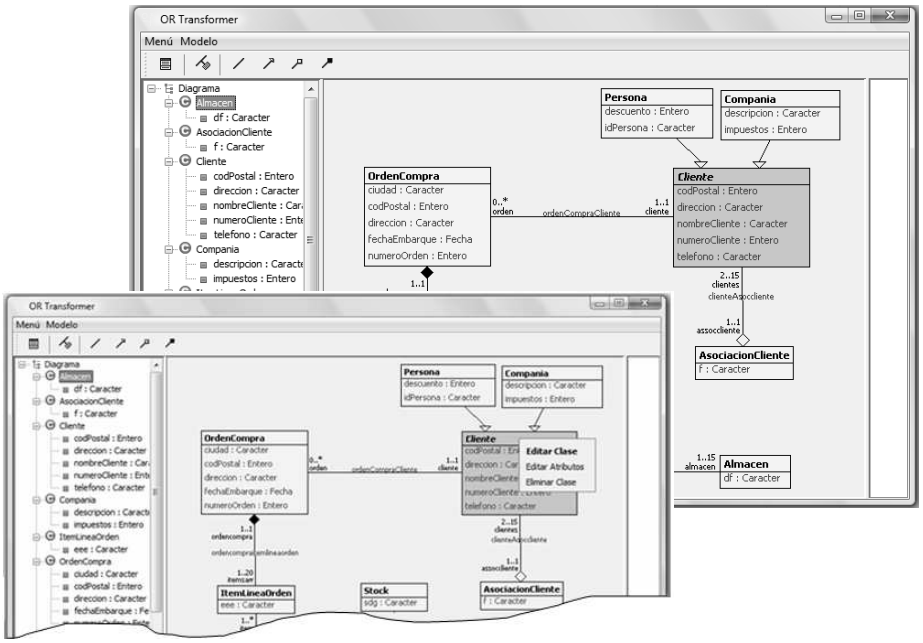

Fig. 9. Edición de diagramas de clases UML

En la solapa *"Diagrama de Esquema OR"* se visualiza el diagrama de clases UML con las extensiones objeto-relacionales, que representa gráficamente el esquema obtenido, de esta manera es posible obtener una rápida interpretación del mismo (Figura 11).

La herramienta tiene definido un conjunto completo de transformaciones, ofreciéndole diversas alternativas al usuario, lo que le brinda flexibilidad para alcanzar el diseño que mejor se adapte a sus necesidades. Asimismo, se incluyeron varias transformaciones "por defecto" para facilitar el proceso a un diseñador novato.

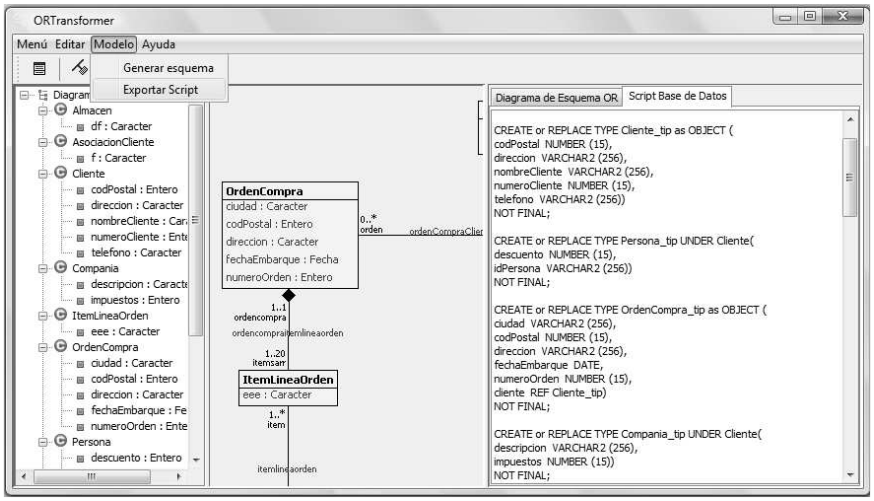

Fig. 10. Generación del código de creación del esquema de BDOR

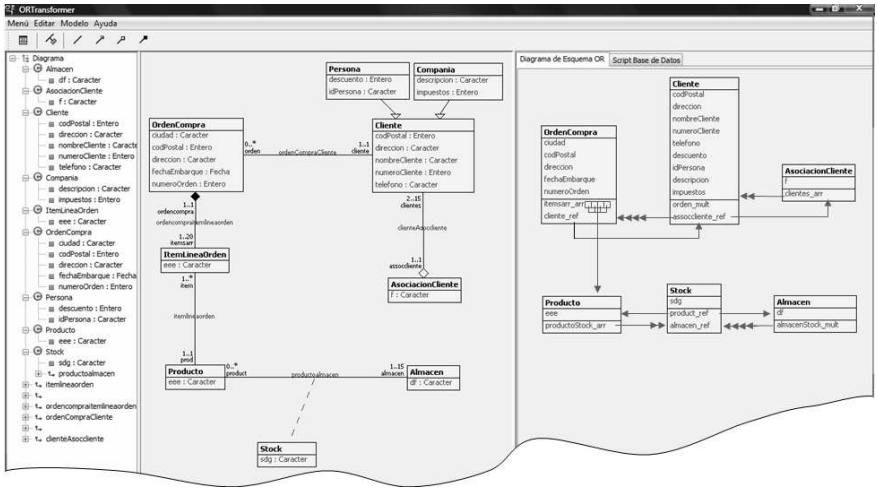

Fig. 11. Diagrama del esquema objeto-relacional

5. Conclusiones

Actualmente no existe una metodología de diseño para las bases de datos objeto-relacionales. Tampoco se pueden encontrar en el mercado herramientas que permitan automatizar su diseño. Teniendo esto en cuenta, en este trabajo se presenta el prototipo de una herramienta de diseño objeto-relacional. La herramienta implementa una metodología desarrollada como trabajo de tesis doctoral, para cubrir la brecha existente. Se resume la metodología presentando las capas de mapeo establecidas y las transformaciones necesarias para pasar de una a otra. Un aspecto novedoso de este trabajo, respecto de los demás trabajos existentes en la literatura, es la presencia de la capa de mapeo intermedia, además de la caracterización realizada para cada uno de los elementos que componen estas capas. Se debe hacer notar la importancia del uso de los pseudoatributos utilizados en la caracterización de las clases y su posterior transformación, siempre que las clases participan de algún tipo de relación.

La automatización del proceso de diseño de las BDOR fue realizado mediante la propuesta de una arquitectura basada en MDA y la tecnología XML, implementada en un prototipo de herramienta case denominada *OR-Transformer*. Se seleccionó MDA por ser la tecnología indicada para el desarrollo de software basado en modelos. La arquitectura presentada posee 3 niveles de abstracción diferentes: datos, modelos y metamodelos. El hecho de tener las especificaciones separadas de las funcionalidades facilitó la generación de las reglas de transformación.

El uso de esquemas XML para la definición de los metamodelos empleados, proporcionó una estructura a nivel sintáctico que permite representar jerárquicamente toda la información que contienen. Luego, los modelos que se generaron a partir de los metamodelos, se instanciaron como documentos XML que responden a un esquema XML determinado. De esta manera, los esquemas XML definen los elementos que puede contener un documento XML, de qué tipos pueden ser y cómo están organizados, y qué atributos puede tener. Por su parte, los documentos XML son validados de acuerdo a los esquemas XML, de forma tal que se mantiene la semántica de los metamodelos.

Para la escritura de las reglas de transformación se empleó XSLT. La herramienta comercial utilizada para su escritura cuenta con una interfaz gráfica flexible que permitió administrarlas de manera sencilla. Si a esto se le suma la caracterización realizada de los metamodelos UML y SQL:2003, que permitió que las reglas de mapeo sean real y efectivamente sencillas de realizar, se puede concluir que esta propuesta para el proceso de transformación se diferencia de otras, basadas en lógica de primer orden, gramática de grafos, o QVT, por mencionar algunas de ellas.

En *OR-Transformer*, las modificaciones a nivel de esquemas XML y reglas XSLT que funcionan "en la parte de atrás", pueden ser realizadas fácilmente por el desarrollador. Esto fue probado en diversas oportunidades durante su generación.

Con este prototipo se probó que la metodología desarrollada sirvió efectivamente para la generación, relativamente rápida y sencilla, de la herramienta de diseño automático.

La productividad en el diseño de BDOR se ve incrementada con el uso de la herramienta, pues el usuario puede concentrarse en los detalles del diseño, dejándole a esta la generación del código de implementación de la base de datos, que en el caso de la tecnología O-R insume mucho tiempo, debido a la complejidad del modelo.

Como trabajo futuro se completará este trabajo incorporando el estudio de las propiedades de UML y el impacto que presenta su inclusión en las primeras etapas de diseño, así como también su mapeo al modelo objeto-relacional.

Referencias

1. Mok, W. Y., & Paper, D. P. (2001). On transformations from UML models to object-relational databases. *Proceedings of the 34th Annual Hawaii International Conference on System Sciences (HICSS-34): Vol. 3* (p. 3046).
2. Marcos, E., Vela, B., & Cavero, J. M. (2003). A methodological approach forobject-relational database design using UML. *Software and Systems Modeling, 2(1), 59-72*.
3. Marcos, E., Vela, B., Cavero, J. M., & Caceres, P. (2001). Aggregation and composition in object-relational database design. In Albertas Caplinskas and Johann Eder (Ed.) *5th Conference on Advances in Databases and Information Systems* (pp. 195-209).
4. Arango, F., Gómez, M. C., & Zapata, C. M. (2006). Transformación del modelo de clases UML a Oracle9i® bajo la directiva MDA: Un caso de estudio. *DYNA*, *Vol. 73 (149)*, (pp. 166-179).
5. Grissa-Touzi, A., & Sassi, M. (2005). New approach for the modeling and the implementation of the object-relational databases. *World Academy of Science, Engineering and Technology*. Vol. 11 (pp. 51-54).
6. Golobisky, M. F.: Generación de soportes y métodos para el desarrollo de sistemas de información bajo el paradigma objeto-relacional. Tesis doctoral presentada en la Universidad Tecnológica Nacional, Facultad Regional Santa Fe (2009)
7. XML Schema. XML Schema Working Group Public Page, http://www.w3.org/XML/Schema
8. Golobisky, M. F., Vecchietti, A.: Mapping UML class diagrams into object-relational schemas. In Proceedings of the Argentine Symposium on Software Engineering (ASSE 2005), 34 JAIIO, pp. 65-79 (2005)
9. Unified modeling language: Superstructure, Version 2.0. OMG Final Adopted Specification. http://www.omg.org/spec/UML/2.0/Superstructure/PDF
10. Unified modeling language: Infrastructure, Version 2.0. OMG Final Adopted Specification. http://www.omg.org/spec/UML/2.0/Infrastructure/PDF
11. Baroni, A. L., Calero, C., Ruiz, F., Abreu, F. B.: Formalizing object-relational structural metrics. 5a Confer^encia da APSI (2004)
12. Golobisky, M. F., Vecchietti, A.: A flexible approach for database design based on SQL:2003. Proceedings of the XXXIV Conferencia Latinoamericana de Informática CLEI 2008, pp. 719-728 (2008)
13. XSL Transformations (XSLT), Version 2.0. W3C Recommendation 23 January, 2007, http://www.w3.org/TR/xslt20.
14. Altova XMLSpy 2008 Enterprise Edition User and Reference Manual. Revised September 5, 2006, from http://www.altova.com/download/2006/XMLSpyPro.pdf
15. Altova MapForce 2007 User and Reference Manual. Revised May 5, 2005, from http://www.altova.com/download/2006/MapForcePro.pdf
16. Oracle Corporation. Oracle Database Application Developer's Guide - Object-Relational Features, 10g Release 1 (10.1). Part No. B10799-01. http://www.oracle.com
17. Java Platform, Standard Edition 6 Release. http://java.sun.com/javase/6/.