Automatización del Diseño Lógico en Bases de Datos Objeto Relacionales

Conference Paper · August 2012

CITATIONS
0

READS
270

3 authors:

Melina Vidoni
National Scientific and Technical Research Council
21 PUBLICATIONS **12** CITATIONS
SEE PROFILE

María F Golobisky
National Scientific and Technical Research Council
10 PUBLICATIONS **16** CITATIONS
SEE PROFILE

Aldo Vecchietti
National Scientific and Technical Research Council
85 PUBLICATIONS **677** CITATIONS
SEE PROFILE

41º JAIIO EST 2012

Automatización del Diseño Lógico en Bases de Datos Objeto-Relacionales

Categoría: Trabajos de Cátedra

Cátedra: Beca I+D, UTN FRSF

Alumno: Melina Carolina Vidoni

Docentes: Dr. Ma. Fernanda Golobysky (codirector)
Dr. Aldo R. Vecchietti (director)

Automatización del Diseño Lógico en Bases de Datos Objeto-Relacionales

Melina C. Vidoni[1], Ma. Fernanda Golobysky[2], Aldo R. Vecchietti[3].

[1] UTN FRSF, Santa Fe, Santa Fe, Argentina
melinavidoni@gmail.com
[2] UTN FRSF, Santa Fe, Santa Fe, Argentina.
mfgolo@santafe-conicet.gov.ar
[3] UTN FRSF, INGAR, Santa Fe, Santa Fe, Argentina.
aldovec@santafe-conicet.gov.ar

Abstract. *OR-Transformer es una herramienta que asiste en el diseño lógico de una base de datos objeto-relacional. El objetivo de la herramienta es facilitar y asistir al desarrollador de aplicaciones en la tarea de definir los objetos del "schema" de una base de datos. El usuario ingresa el diseño conceptual de datos por medio de un diagrama de clases UML, y la herramienta por medio de diversas funciones de transformación genera el diseño lógico, definiendo los scripts para la creación de los UDTs (User Defined Types) y las tablas de la Base de Datos. El usuario posee diversas alternativas de transformación de los modelos que debe seleccionar de acuerdo con los requerimientos a cumplir en el sistema que desarrolla.*

Keywords. *Diagrama de clases UML, Diseño conceptual, Diseño Lógico, Base de datos Objeto-Relacional.*

1. Introducción

El diseño de una base de datos es una etapa importante en el ciclo de vida de los sistemas de información: un buen diseño permite la ejecución eficiente de las operaciones que se ejecutan en la Base de Datos, que se traduce, en general, en una operación eficiente del sistema de información.

Existen en el mercado numerosas herramientas de diseño de bases de datos relacionales (estándar SQL'92) como Case Studio, Designer, DatabaseSpy (Altova), por nombrar sólo alguna de ellas; y otras denominadas como transformación objeto-relacional, que efectúan el mapeo de clases en programas Java en tablas relacionales como Hibernate, Toplink y Struts, pero no se conocen instrumentos como el que se presentan en este trabajo, de mapeo de diagramas de clase de UML en "schemas" objeto-relacionales que obedecen al estándar SQL:2003.

A su vez, hay varios trabajos relacionados con propuestas de transformación y diseño de bases de datos relacionales. Mok y Paper[15] presentaron una propuesta de transformación de diagramas de clases UML en tablas anidadas normalizadas, pero no contribuyó con procedimientos formales para realizar mapeos más generales. Marcos, Vela, *et al.* [11] propusieron algunas guías para realizar las transformación de diagramas de clases UML en objetos del estándar SQL:1999 y luego en Oracle 8i, sin profundizar

sobre el tema. Estos mismos autores[10] presentaron una metodología de para el diseño de BDOR en la cual definieron estereotipos para las transformaciones, sin dar formas 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 y Sassi[7] desarrollaron una herramienta para el diseño e implementación de BDOR denominada *Navig-tools* por la cual se puede generar el código del modelo en SQL, recibiendo como entrada un diagrama entidad-relación. 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.

Teniendo esto como premisa, se desarrolló *OR-Transformer*, como una herramienta de asistencia y automatización del diseño de Bases de Datos Objeto-Relacionales que obedecen al estándar SQL:2003, a partir de un diagrama de clases UML. Este tipo de diagramas son los más representativos de la programación OO y fue elegido como medio de acercar el diseño de la base de datos, a los mismos diagramas que se utilizan para el diseño de la estructura de clases de un sistema.

La arquitectura del sistema esta soportada en MDA (Model Driven Architecture)[16] y en XML, en este sentido, diversos modelos con distintos niveles de abstracción, se emplean para dar sustento a *OR-Transformer*. En primer lugar tenemos el *Modelo de Datos Conceptual:* representa los elementos de datos y relaciones que surgen de los requerimientos del sistema y es independiente de la tecnología que se va a emplear para su implementación; los más usados para bases de datos son el diagrama de clases de UML y los diagrama Entidad-Relación. El *Modelo de Datos Lógico:* vinculado con la estructura lógica de la base de datos y relacionados con la tecnología a emplear, por lo general en algún administrador de base de datos; los ejemplos más típicos, actuales, son el modelo relacional, el orientado a objetos (OO) y el objeto-relacional (OR). Finalmente en el nivel más bajo se encuentra el *Modelos de Datos Físico:* este define los detalles de cómo se almacenan los datos en el administrador de base de datos elegido: el formato de los registros, la estructura de los ficheros (desordenados, ordenados, etc.) y los métodos de acceso utilizados (índices, etc.).

El diseño lógico de una base de datos se obtiene transformando el diseño conceptual: el diseñador debe tomar decisiones entre las posibilidades de mapeo existentes, teniendo en cuenta las características del sistema de gestión de bases de datos (SGBD) con el que haya elegido trabajar. El modelo OR, soportado por el estándar SQL:2003, es un modelo mucho más expresivo y complejo que su predecesor, el modelo relacional (estándar SQL'92 [14]). En este sentido, el modelo OR presenta mayores posibilidades de diseño, haciendo que su realización sea complicada, y las transformaciones más complejas.

2. Análisis General

Inicialmente, *OR-Transformer* permite confeccionar la definición de requerimientos de los usuarios, mediante la elaboración de un diagrama de clases UML, utilizando como base el metamodelo de UML [6, 9]. A partir de la estructura del diagrama obtenido se

deriva el esquema lógico de la base de datos, tomando como base el metamodelo de SQL:2003. El motor de mapeo realiza diversas traducciones sucesivas, basadas en reglas de mapeo escritas en XSLT, según el proceso propuesto por MDA, hasta llegar finalmente al script de la correspondiente base de datos. Luego de esto, el usuario puede seleccionar un SGBD existente, e insertar en él la base de datos, a través de *OR-Transformer*.

Para poder manejar el desarrollo de esta herramienta, se seleccionó el lenguaje Java SE como base para la codificación de estilo orientado a objetos, utilizando varias librerías disponibles en este lenguaje las cuales serán detalladas en los apartados de los módulos en los que fueron utilizadas, y se definió una arquitectura centrada en el flujo de datos, particularmente el estilo Pipes&Filters[17] (en español, *Tuberías y Filtros*). Este consta de un conjunto de componentes denominados "filtros", conectados entre sí por "tuberías" que transmiten datos desde un componente al siguiente. Cada filtro trabaja de manera independiente de los componentes que se encuentran situados antes o después de él; se diseñan de modo que esperan un conjunto de datos en un determinado formato y obtienen como resultado otros datos de salida, en un formato específico.

Para *OR-Transformer* se definieron tres filtros principales: Modelado Conceptual, que recibe el diagrama de clases y genera objetos java, Modelado Lógico que toma esos objetos y produce sentencias SQL a través del uso de un repositorio de reglas de mapeo, e Inserción, que toma las sentencias y genera la base de datos (Figura 1).

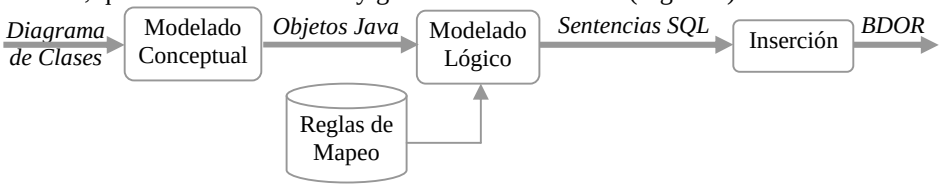

Fig. 1. Arquitectura Pipes&Filter de OR-Transformer.

2.1. Modelado Conceptual

Dado que la herramienta permite elaborar un diagrama de clases UML, la interfaz de usuario necesaria para tal fin fue diseñada con un estilo de invocación implícita [15], que funciona con oyentes que ante estímulos de la interfaz generan acciones que obtienen resultados visibles.

A medida que el usuario va completando el diagrama de clases, la herramienta almacena el diagrama de clases UML generado por el usuario en dos grandes estructuras utilizadas para simplificar el tratamiento interno del mismo, y la posterior transformación hasta el modelo lógico. Una de ellas, es la correspondiente a los elementos contenidos en los diagramas de clases UML (Fig. 2), y la otra es la equivalente a los elementos y facilidades requeridos para la visualización gráfica del mismo (Fig. 3).

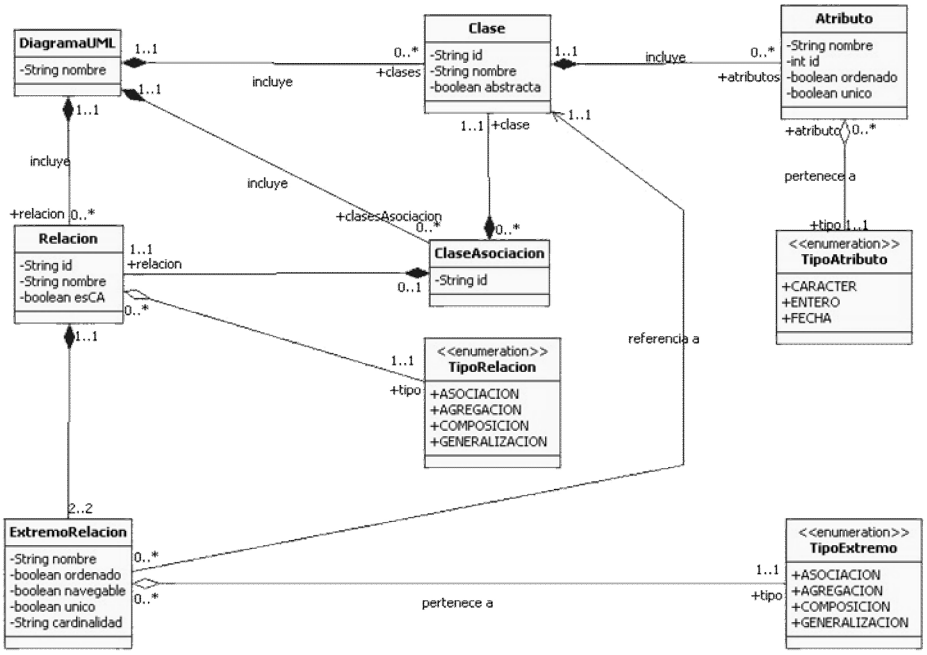

Fig.2. Modelo con los elementos contenidos en los diagramas de clases UML.

Cuando el usuario crea un nuevo diagrama se instancia `DiagramaUML` conteniendo una lista de objetos tipo `Clase`, una lista de objetos `Relacion`, y una lista de objetos `ClaseAsociacion`. Cada instancia de `Clase` contiene objetos que representan a sus atributos -del tipo `Atributo`-, que dependen de ella, y desaparecen si ésta se elimina. `Atributo` posee un `TipoAtributo`, representado como un enumerado. Por otro lado, cada `Relacion` pertenece a un `TipoRelacion` que también está representado como

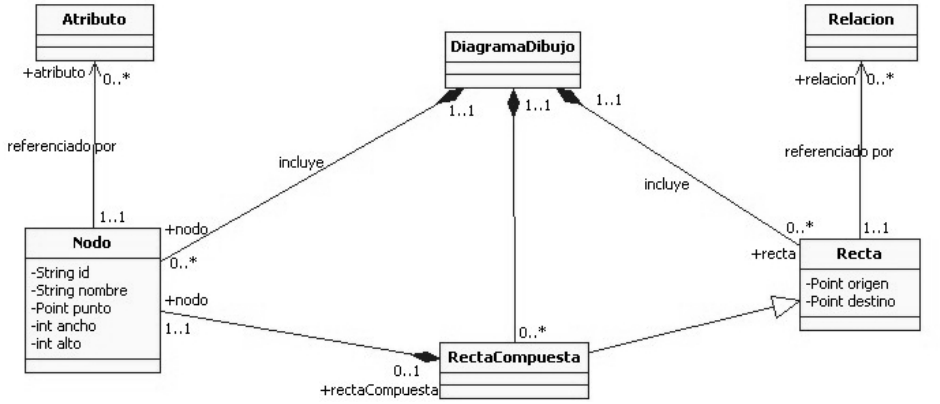

Fig. 3. Modelo con los elementos para la representación gráfica de los diagramas UML.

un enumerado; a su vez, contiene dos `ExtremoRelacion` que referencian a la `Clase` a la que se unen. Finalmente, `ClaseAsociacion` está formada por una `Clase` y una `Relacion`; estas entidades la componen y describen, y solo pueden estar asociadas a una única `ClaseAsociacion`.

En el mismo momento en que se crea una instancia de `DiagramaUML` (Fig. 2), se crea una instancia de `DiagramaDibujo` (Fig. 3) que proporciona las facilidades para la representación gráfica en pantalla del correspondiente modelo de clases UML. Está compuesta por una lista de objetos del tipo `Nodo`, una lista de objetos `Recta` y una lista de objetos `RectaCompuesta`. Cada instancia de `Nodo` representa a una `Clase` de la Fig. 2 y contiene referencias a sus atributos; de esta manera se evita la redundancia de datos y se mantienen consistentes ambos modelos. Por otro lado, las instancias de `Recta` representan a las instancias de `Relacion` del modelo de la Fig. 3 (asociación, agregación, composición y generalización), y contienen una referencia a la instancia que representan, también con el fin de evitar redundancia y asegurar la consistencia. Finalmente, `RectaCompuesta` especializa a `Recta`, y representa a las `ClaseAsociacion`, por lo que también contiene una instancia de `Nodo`, que representa a su propia clase y que no participa de la lista general del `DiagramaDibujo`.

2.2 Modelado Lógico

Una vez que el usuario ha generado un diagrama de clases a través de la interface, y que los objetos están cargados con los datos del mismo, la herramienta queda habilitada para realizar las transformaciones internas, y obtener las sentencias SQL. Este módulo interactúa con un repositorio de archivos, donde están contenidos los metamodelos y las reglas de mapeo utilizadas para transformar el diagrama UML (Figura 4).

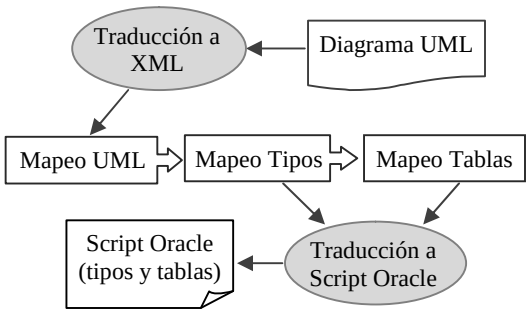

Figura 4. Esquema expandido de transformaciones del motor de mapeos, dentro del módulo lógico.

MDA propone un proceso de desarrollo basado en la realización y transformación de modelos. Los metamodelos utilizados por el motor de mapeo [4, 5], siguiendo esta tendencia, son esquemas XML que se instancian en documentos XML, y se transforman de manera automática siguiendo reglas de mapeo escritas en XSLT[13]; la traducción final contiene la lógica necesaria para la creación de tipos y tablas de la base de datos, utilizando la sintaxis de Oracle [18].

Para su realización, se utilizaron tres librerías de Java: *a) JDOM 1.1.:* [8] permite manipular, escribir, leer y modificar ficheros XML, sin utilizar formas complejas. *b) Saxon 9.3 EE*: [12] permite el uso intuitivo de las hojas de estilo XSLT, manejando las transformaciones que provee el mismo XML. *c) Xerces 1.1.1:* [3] contiene

funcionalidades avanzadas de W3C, tales como las recomendaciones de esquemas, DOM nivel 2 (versión 1.0) y SAX2.

2.2.1 Reglas de Mapeos

Las reglas de mapeo en XSLT/XSD utilizadas en el Módulo Lógico, fueron desarrolladas durante la tesis doctoral de Golobysky [5], y luego fueron adaptadas y corregidas sintácticamente para ser utilizadas en *OR-Transformer*. Sin embargo, las traducciones realizadas a través de Java, indicadas en la Fig. 4, fueron elaboradas durante la presente implementación, con ayuda de las librerías mencionadas en el punto anterior.

Los mapeos funcionan de a pares de dos archivos: una hoja de estilo en formato XSLT - la cual contiene las reglas para transformar los archivos XML- y un fichero XSD -que contiene la estructura y contenido posible del XML resultante-. De estos pares, existe uno para el mapeo al metamodelo, uno para el mapeo a tipos, y tres para el mapeo a tablas.

En el mapeo hacia los Tipos, cada clase del diagrama de clases UML correspondiente al diseño conceptual, debe traducirse en un tipo estructurado de la tecnología objeto-relacional; a su vez, las clases poseen atributos que son transformados en atributos de tipos estructurados, los cuales pueden ser atributos regulares, arreglos o multisets, según se analiza en [5].

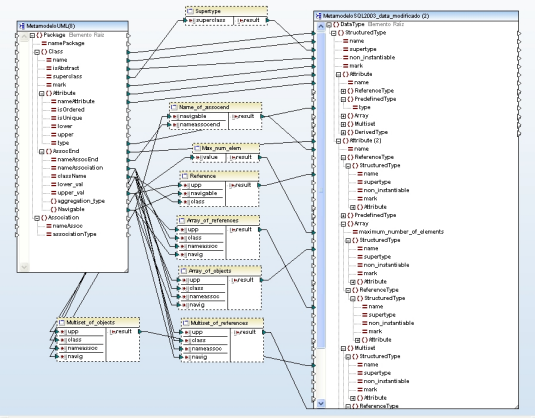

Figura 5. Reglas de Mapeo definidas entre UML-XML Schema y tipos de datos SQL:2003-XML.

Finalmente en el mapeo hacia las Tablas, dado que el usuario debe seleccionar, a través de la interface, cómo quiere representar la herencia en las tablas tipadas, existen tres pares de reglas. Por defecto, la herencia es traducida como *Partición Vertical*, que genera una tabla para cada una de las clases que componen la jerarquía. Si no, se puede seleccionar *Modelo Plano* (crea una tabla tipada para el supertipo, que además contiene todos los campos de los subtipos) o *Partición Horizontal* (la cual sólo implementa tablas correspondientes a los subtipos, trasladando a las mismas todos los atributos del supertipo).

Para estos mapeos se definió un Schema XML por cada metamodelo empleado. Fueron generados empleado Altova XMLSpy [1], ambiente que provee una interface para crear y editar Schemas y archivos XML, y también hojas de estilo XSLT.

Las reglas de transformación fueron escritas en el lenguaje XSLT, y fueron definidas en entre cada uno de los metamodelos empleando MapForce [2] que tiene un conjunto de funciones gráficas que facilitan la escritura de reglas de mapeo. En la Fig. 5 se muestra una ejemplo de cómo se definieron las reglas de transformación entre UML-SML Schema y los tipos de datos de SQL:2003-XML Schema. De la Fig. 5 se puede ver que una clase UML es transformada como un Tipo Estructurado de la base de datos Objeto-Relacional,

con el mismo nombre que la clase, el atributo `isAbstract` de `Class` es transformado como un atributo `non_instantiable del` Tipo Estructurado. El atributo superclase de `Class` es transformado como un atributo `supertype` del Tipo Estructurado.

2.2.2 Motor de Mapeos

Una vez que el usuario indica a la herramienta que transforme el diagrama de clases, comienza el proceso interno de *OR-Transformer*. A través de código Java se realiza primera traducción para lograr pasar desde el modelo UML hasta un archivo XML. Para esto, se utiliza la estructura de la Fig. 2, la cual se lee directamente desde la instancia existente de `DiagramaUML`. Se utiliza un nombre genérico del diagrama para crear la etiqueta raíz del documento, y luego se itera a través de la lista de `Clases`, `Relaciones` y `ClaseAsociacion`, donde cada nodo de cada lista es convertido en un conjunto jerárquico de etiquetas XML del nuevo archivo, que describen el contenido del diagrama UML.

A continuación, este archivo es releído e indicando la ruta de las reglas correspondientes al Mapeo a Metamodelo (`UMLtoModelo.xslt` y `MetamodeloUML.xsd`), se genera un nuevo archivo XML correspondiente al diagrama actualizado al metamodelo UML, el cual es manipulado a través de JDOM y guardado en el disco. Este nuevo fichero es usado como base para la siguiente transformación, la cual usa las reglas de Mapeo a Tipos (`PrimerMapeo.xslt` y su XSD) y genera un nuevo archivo temporal, el cual será utilizado para generar las sentencias SQL de Tipos, durante la última traducción.

Este archivo es releído y proporcionando las rutas de las reglas del Mapeo a Tablas (`SegundoMapeo.xslt.*` junto con su XSD), se genera el último archivo XML, correspondiente a la estructura de las tablas. En este paso, la regla de mapeo XSLT tiene una segunda extensión -denotada previamente con un *- mediante la cual se lee la correspondiente a la selección efectuada por el usuario: genV (Partición Vertical), genP (Modelo Plano) o genH (Partición Horizontal). En caso de no haber generalización, se utiliza por defecto la partición vertical. Al terminar crea un nuevo archivo temporal.

Finalmente, *OR-Transformer* realiza la segunda traducción basada en código Java, leyendo el archivo producido por el Mapeo a Tipos y el de Mapeo a Tablas, para generar el esquema de sentencias SQL con sintaxis de Oracle10g [18]. Este es un proceso complicado, dado que hay que evaluar la jerarquía de etiquetas XML para poder traducirlas en sentencias SQL de creación o alteración. Para mantener coherencia, se estableció un estándar para la herramienta, y los nombres que el usuario define para las clases, reciben un sufijo *_tip* para los nombres de tablas tipadas, y *_tab* para las tablas. En cuanto a la sintaxis, se ha elegido a Oracle 10g, por ser el SGBD que al momento del comienzo de las investigaciones, era el más cercano al estándar SQL:2003 y que más efectivamente implementaba la base de datos objeto-relacional.

2.3 Inserción en el SGBD

Una vez que el usuario tiene las sentencias SQL, tiene la opción de guardarlas a través de la herramienta, en dos archivos de texto en la ubicación que seleccione. Sin embargo,

también existe la posibilidad de insertar estas sentencias sobre una base de datos; el único requerimiento necesario, es la existencia de un usuario en el SGBD, de contraseña conocida, que tenga los privilegios adecuados para crear bases de datos y generar tipos y tablas. Para la realización de este módulo, en particular se utilizó la librería *OJDBC*, la cual permite generar conectividad entre un programa desarrollado en Java y una base de datos Oracle, en particular de la edición 10g.
El usuario debe proporcionar dichos datos a través de la interface, y con eso se genera una instancia de la clase `GuiaInsercion`, la cual administra y guía este proceso. Dentro de esta, se instancia e inicializa la clase `BaseDeDatos`, la cual es la que efectivamente inserta las sentencias que va recibiendo, sobre el SGBD y la correspondiente base de datos. Para poder realizar esto, primero se crea la BD utilizando el nombre que indicó el usuario; luego, se insertan las sentencias correspondientes a los tipos, y luego las correspondientes a las tablas. A su vez, deben controlarse las referencias entre las tablas y tipos, y las relaciones de generalización-especialización, ya que esto dictamina el orden en que deben insertarse. Por ejemplo: en una generalización, hay que insertar el tipo más general, y luego expandirse en anchura y por nivel hasta los tipos más específicos.
Una vez completada la operación, la herramienta le avisa al usuario del resultado de la operación, y luego podrá accederse a la nueva base de datos, desde el SGBD correspondiente, usando las credenciales proporcionadas previamente.

3. Utilización de *OR-Transformer*

El proceso de transformación resulta transparente al usuario, debido a que se busca que el usuario encuentre un proceso sencillo y fácil de comprender, y permitiéndole ahorrar tiempo de análisis, al automatizar la conversión desde un modelo de datos a otro.

3.1 Interfaces de Modelado

Las herramientas proporcionadas al usuario para la elaboración del diagrama de clases UML han sido simplificadas y son presentadas sólo cuando éste crea un nuevo diagrama desde el menú Archivo, o cuando abre uno ya existente. En la barra de herramientas de la pantalla principal (Fig. 6), se puede observar las facilidades para la creación de Clases, Clases Asociación, y relaciones de Asociación, Generalización, Agregación y Composición. Se ha definido una serie de validaciones sobre el modelo, pensadas para conciliar los paradigmas utilizados en los mapeos, manteniendo su consistencia. Algunas de ellas son: mantener herencias simples, basado en el hecho de que el estándar SQL:2003 [14] sólo soporta este tipo de herencia; prohibir que las clases asociación participen de relaciones de herencia, promover que todas las relaciones tengan un nombre, así como también los extremos (roles) de las mismas, y que estos sean únicos; no se permiten atributos sin nombres/tipos, o con nombres repetidos dentro de la misma clase; con respecto a las relaciones de generalización-especialización, existen parámetros tales como los nombres de rol, el nombre de las relaciones, su cardinalidad y navegabilidad, que son especificados de manera automática por la herramienta, es decir, sin necesidad de que el usuario los defina, y no está permitida su modificación. Esto es debido a que se busca

mantener una concordancia con el metamodelo planteado para UML.

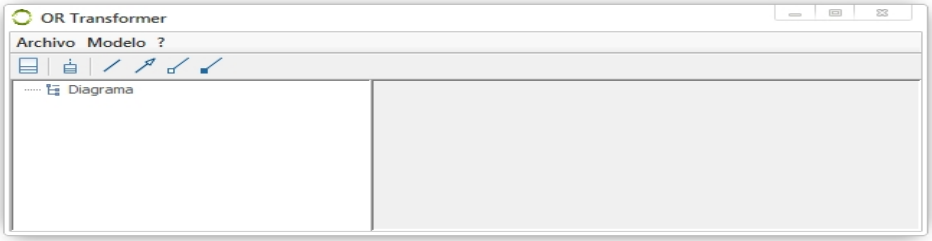

Fig. 6. Herramientas de modelado UML.

3.2 Interfaces de Traducción

Para realizar el proceso de obtención del esquema de base de datos (Fig. 7) el usuario sólo debe solicitar su transformación a través de un menú. En caso de que existan relaciones de generalización-especialización, la herramienta las detecta y pide al usuario que indique de qué manera desea traducirlas, ya sea partición vertical, modelo plano o partición horizontal. Se utiliza por defecto la partición vertical. Una barra de progreso indica el estado del mapeo, permitiéndole saber, en todo momento, cuánto se tiene hecho y cuánto falta realizarse.

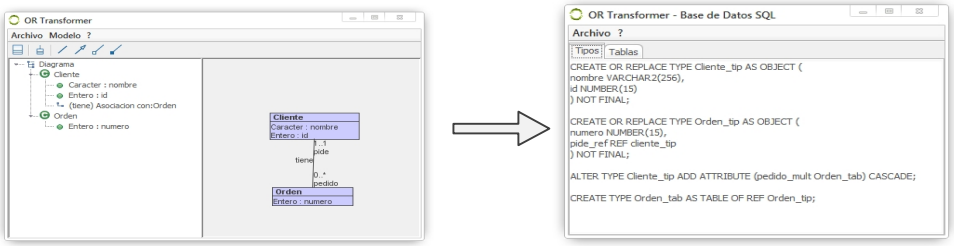

Fig. 7. Vista del usuario sobre el proceso de obtención del modelo lógico de la BDOR.

3.3 Interface de Inserción

Posteriormente Cuando el usuario selecciona la opción de inserción desde la vista de Tipos y Tablas, se abre una nueva ventana en la cual se piden los datos necesarios para la creación de la base de datos, y la posterior inserción de las sentencias. Se solicita un nombre para la base de datos que será creada, la dirección IP donde se encuentra (el único DNS aceptado es *localhost*), y las credenciales del usuario a través del cual se conectará.
Una vez presionado el botón Insertar, el usuario verá una barra de progreso que se actualiza con cada sentencia ejecutada exitosamente, y que le permite estar al tanto del estado del proceso de inserción. Una vez terminado, aparece un nuevo mensaje, mencionando el resultado, y detallando (si es que existen) los errores generados.

4. Conclusiones

Desde que las bases de datos objeto-relaciones hicieron su aparición, a finales de los años

‘90, se han realizado muchos trabajos intentado acortar la brecha entre el paradigma de objetos, el cual se encuentra en boga actualmente para la programación de sistemas, y el paradigma relacional de las bases de datos. Sin embargo, no muchos han tratado el tema del mapeo entre objetos y objeto-relacional.

Por esto mismo, *OR-Transformer* nace como una implementación para diseñar bases de datos, que busca la simplicidad y la capacidad de expansión, facilitando la interrelación de los paradigmas de objetos y objeto-relacional, basándose en uno de los principales modelos de representación del paradigma OO: el diagrama de clases UML.

Como trabajo futuro, se persigue la inclusión de mapeos a sintaxis de otros motores de bases de datos, más allá de Oracle10g; la generación de mapeos para la transformación de objetos al modelo relacional, y la mejora de las reglas de mapeo relacionadas con las relaciones de generalización-especialización, para hacerlas más flexibles y completas.

Referencias

1. Altova MapForce User Manual, www.altova.com/download/2006/MapForcePro.pdf
2. Altova XMLSpy 2008 EE User Manual, www.altova.com/download/2006/XMLSpyPro.pdf
3. Apache XML Project, “Apache Xerces”, 2010. http://xerces.apache.org/xerces-j/
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).
5. Golobisky M.F. y A. Vecchietti, “Automatización del diseño de bases de datos objeto-relacionales basado en MDA y XML”. Mayo 2010. JAIIO ASSE.
6. Golobisky, M.F. “Generación de soportes y métodos para el desarrollo de Sistemas de Información bajo el paradigma Objeto-Relacional”. Junio de 2009, pp. 250.
7. 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).
8. J. Hunter and B. McLaughlin. “JDOM Project”, iniciado en 2000. http://www.jdom.org/
9. J. Rumbaugh, I. Jacobson and G. Booch. “The Unified Modeling Language Reference Manual”, Addison-Wesley, año 1999.
10. Marcos, E., Vela, B., & Cavero, J. M. (2003). “A methodological approach for object-relational database design using UML”. Software and Systems Modeling.
11. Marcos E., Vela B., Cavero J. M., & Caceres P. (2001). “Aggregation and composition in object-relational database design”. En Albertas Caplinskas and Johann Eder (Ed.) 5th Conference on Advances in Databases and Information Systems (pp. 195-209).
12. M. Kay. “Saxon XSLT Processor, W3C Recommendation”, 23 de Enero de 2007.
13. M. Kay. “XSL Transformations XSLT, Version 2.0, W3C Recommendation”, 27 de enero de 2007. http://www.w3c.org/TR/xslt20
14. Melton, Jim. “(ISO-ANSI Working Draft) Foundation (SQL/Foundation)”, ISO standard.
15. Mok, W.Y. & Paper D.P. (2001). “On transformations from UML to object-relational databases”. Proceedings of the 34th Annual Hawaii International Conference on System Sciences (HICSS-34): Vol. 3 (p.3046).
16. Mukerji, J. and Miller, J. “MDA Guide Version 1.0.1”, omg/2003-06-01)). Object Management Group. http://www.omg.org/cgi-bin/doc?omg/03-06-01.
17. Shaw, M. and Garlan D. “Introduction To Sowftware Architecture”. 1994.
18. Oracle Corporation. “Oracle Database Application Developer’s Guide: Object-Relational Features. 10g Release 1. Part nº BB10799-01”, Año 2003.