A Flexible Approach for Database Design Based on SQL:2003

María Fernanda Golobisky[1] and Aldo Vecchietti[1]

[1]INGAR - Instituto de Desarrollo y Diseño
Avellaneda 3657, (3000) Santa Fe, Argentina
{mfgolo, aldovec}@santafe-conicet.gov.ar

Abstract. In this paper we address the issues involved in the development of a flexible approach for database design based on SQL:2003 standard. The goal of this work is to present the steps and system architecture involved in the transformation of a database conceptual design in the form of UML class diagram, into database schema compliant with SQL:2003 standard. The mapping is founded on the MDA directive and is composed by three mapping layers. Flexibility of the approach is achieved in two different ways; first, developers can select the mapping rules such that final design can comply with relational or object-relational specifications, both included in the SQL:2003 standard; second, the system architecture proposed, based on metamodels and its implementation into XML Schemas allows an easy definition and modification of mapping rules.

Keywords: ORDBMS, SQL:2003, Database design, XML Schema, MDA.

1 Introduction

The development process of information systems has become a very complex task during the last years, such that this process is supported by CASE tools that help designers to increase productivity. The database schema design which stores the application data is not exempted of that complexity. Nowadays, there are no commercially available tools that automate the design of Object-Relational databases (ORDB) based on the SQL:2003 standard. Although several systems widely used in the industry can be found under the "Object-Relational" (O-R) name (Hibernate, Toplink, Enterprise Architect, etc.) they really transform Java objects into relational tables. As a consequence, many people misunderstand that transformation with the one proposed in this work, where the subject is to transform a conceptual data model defined in a UML class diagram into typed tables based on user defined types (UDTs) and other components of the SQL:2003 standard, which is quite different to the one of the mentioned systems. We have to remark that, in many cases, both approaches appear under the O-R mappings/transformations name.

In this paper, we address the issues involved in the design of ORDB schemas. The final goal pursued in this research is the generation of a CASE tool to assist system developers in the design of databases based on SQL:2003 standard. Due to this standard is a superset of SQL:92, the idea behind this tool is to allow the generation of

relational and/or object-relational schemas, or something in the middle between both extremes, according to the mapping rules which can be selected from a menu. The system architecture is based on the MDA directive and on metamodels. The implementation is made by means of XML Schemas [16] and the definition of mapping rules between them, written in XSLT language. XML [12] is undoubtedly an important technology in the data processing of today systems. It has become popular after W3C (World Wide Web Consortium) had proposed it and it is the language selected by organizations that want to exchange its information with others.

2 Related Works

Several articles can be found in the literature related to the transformation from a conceptual design into ORDB schemas: [8] presented an ORDB design methodology defining new UML stereotypes for ORDB and proposing some guidelines to translate a UML conceptual schema into an object-relational schema. The guidelines were based on SQL:1999 standard and Oracle8i as a product example. In [3] the authors have made a mapping from a class diagram to Oracle9i and have showed some mapping rules written in set theory. The transformation was based on the MDA directive. They have used the Oracle9i metamodel, but not the SQL:2003 metamodel. Authors in [7] have proposed a tool to help the conception and implementation of ORDB called "Navig-tools" which allows the user to generate script modeling in SQL3 language. The tool is based on the Entity/Association and navigational model for modeling the ORDB. This is an outdated work since the introduction of UML modeling and SQL:2003 standard. A conceptual model transformation to ORDB based on MDA, framed in MIDAS, has been presented in [15]. The authors have defined UML profiles for the SQL:2003 standard and for Oracle10g. The mapping rules were first expressed in natural language and after formalized by graph grammar.

Related to XML applications, in [11] the authors have focused on XML applications that use a standard structure specified by its corresponding schemas. Using UML as the language for modeling software systems, they have considered the intricacies involved in both the forward (UML to XML) and reverse (XML to UML) engineering processes and they have defined the complete specifications for forward engineering UML model constructs to their corresponding XML counter parts.

3 Mapping Layers and Metamodels

Three mapping layers have been established in [6] and were used here to go from UML class diagrams (conceptual design) to ORDB schema (logical design). The components of these tiers are shown in Fig. 1.

UML Class Diagram Layer: includes the conceptual model corresponding to the user requirements: UML classes and relationships (association, aggregation, composition, hierarchy, association class relationships, etc.).

Object-Relational Layer: is composed by the non-persistent object-oriented elements proposed by the SQL:2003 standard: user-defined types (UDT), methods,

inheritance, references, collections: arrays and multisets; and row types. The definitions made on this tier do not allow the persistence of any object or "data" until the tables are created.

Object-Relational Persistent Layer: is composed by the O-R tables: typed tables created from the objects of the previous one and where the application data can be stored. The "relational" components such as tables, constraints, domains, etc., are also components of this layer.

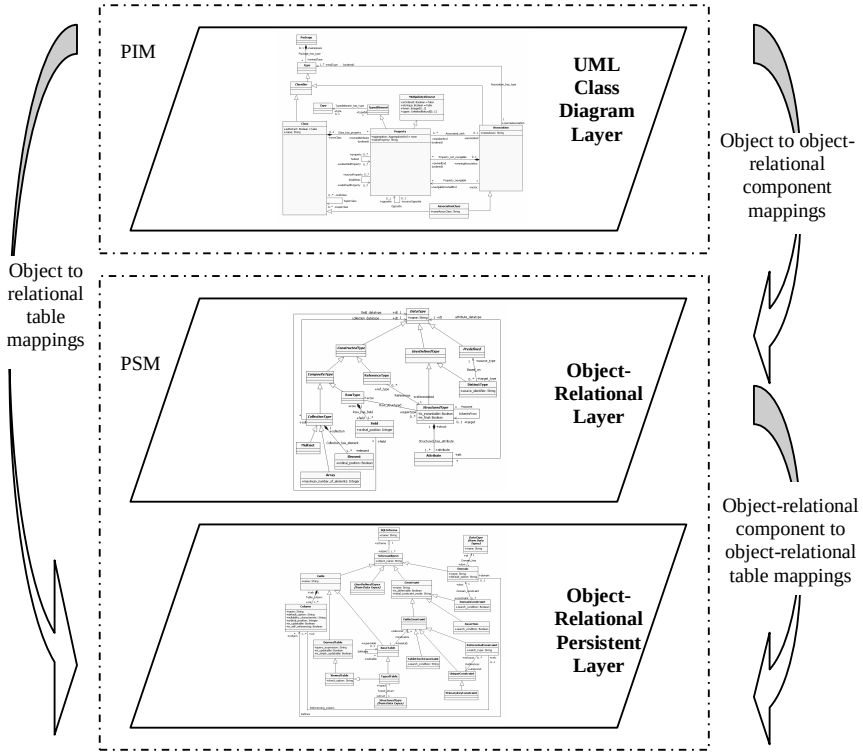

Fig. 1. Mapping layers and its different transformations

From Fig. 1 can be observed that transformation from UML class diagrams to relational designs involves just one mapping step: from class diagrams to relational tables.

For each layer a metamodel has been defined: an excerpt of the UML class diagram metamodel based on [13] and [14] was used for the first one, and the SQL:2003 metamodel proposed in [4] for the second and third mapping layer.

The SQL:2003 metamodel is separated in two parts: the first one contains all the elements related to data types, and the other all the components about the SQL:2003 schema objects: columns, domains, tables, constraints, assertions and other "relational" components. For this work we have made a few modifications to these

metamodels. The one containing the data types is used for the intermediate mapping layer, and the one containing the SQL schema objects for the third layer.

For space reasons we do not show metamodels here. For a further comprehension about them, read [13, 14 and 4].

The system architecture to produce the transformation between the layers is based on the Model Driven Architecture (MDA) [10] presented by the Object Management Group, which uses two different models:

Platform Independent Models (PIM): models with a high abstraction level which are implementation technology independents.

Platform Specific Models (PSM): they merge the platform independent model specifications with the details that state the use of a specific platform by the system.

In this work, the PIM corresponds to the UML metamodel and the PSMs are conformed by the SQL:2003 data types and SQL:2003 schema metamodels. These metamodels are related through the mapping rules showed in Fig. 1.

4 Mapping Rule Definitions

Since SQL:2003 [9] is a superset of SQL:92, with the metamodels proposed is possible to transform the UML class diagram into a relational database model or into and object-relational database model. Transformations must cover all possible mappings between those extremes.

The mapping rules from the UML class diagram to the ORDB schema were first specified in [6]. A summary of them is presented below, and then, a summary of relational mapping rules is also included. In order to understand the rules, the UML class diagram metamodel is shown in Fig. 2.

4.1 Object-Relational Mapping

UML layer components to Object-Relational layer components

A Class maps to a Structured Type.

A Class that is subtype of a supertype class maps to a Structured Type under the Structured Type corresponding to the supertype.

An Attribute of a class maps to an Attribute of a Structured Type. If the attribute has:

- Defined multiplicity: it maps to an array
- Undefined multiplicity: it maps to a multiset

An Association is a relationship that has two Association Ends.

- If the Association End has multiplicity of 1, maps to a Reference within the Structured Type of the other end.
- If the Association End has a defined multiplicity (greater than 1), maps to an Array of references within the Structured Type of the other end.
- If the Association End has an undefined multiplicity, maps to a Multiset of references within the Structured Type of the other end.

- For unidirectional associations, a reference to the structured type with no visibility is embedded into the other structured type.

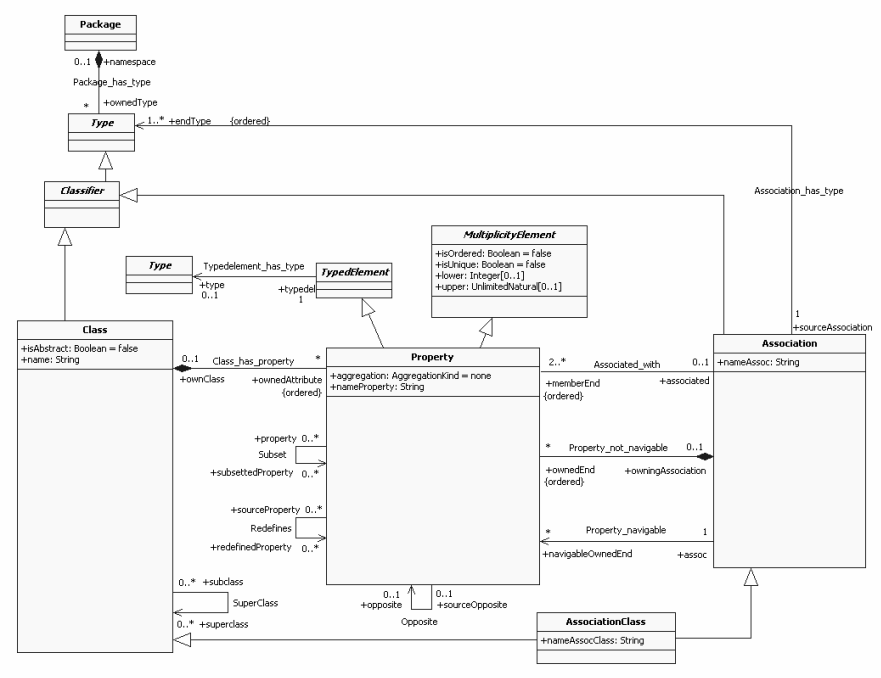

Fig. 2. UML 2.0 class diagram metamodel of the 1st layer

An Aggregation is a weak relationship that has two Association Ends, the "whole" and the "part" where they exist by themselves.

- The whole maps to a reference within the part.
- If the part has a defined multiplicity (greater than 1), maps to an Array of references within the whole.
- If the part has an undefined multiplicity, maps to a Multiset of references within the whole.

A Composition is a strong relationship that has two Association Ends, the "whole" and the "part". Since the part life depends on the whole life, so:

- If the part has a defined multiplicity (greater than 1), maps to an Array of Structured Type corresponding to the part, within the whole.
- If the part has an undefined multiplicity, maps to a Multiset of Structured Type corresponding to the part, within the whole.

The mappings defined corresponds to those that are more "suitable" for developing an application, some others complementary can be defined in order to give more choices to the developer. For example, an aggregation or composition with defined multiplicity can be mapped to a multiset instead of an array, the database designer can choose this option for application needs. These options will be included in the system.

Object-Relational layer components to Object-Relational persistent layer components:

A Structured Type maps to a Typed Table.

- If the Structured Type has a Structured Type embedded, only a Typed Table for the outer Structured Type is created.
- For inheritance relationships among Structured Types, three different transformations can be made [5]:
  - Flat model: a unique table for all the class hierarchy is created in where the supertype and its corresponding subtypes are stored. In this case the supertype typed table has the *substitutability* property to allow the storage of the subtypes in the same table.
  - Vertical partition: a table for each class that forms the hierarchy is created, each one with its attributes.
  - Horizontal partition: only the tables corresponding to the subtypes are created.

4.2 Relational Mapping

The mapping rules from the UML class diagram to the relational schema are the following:

UML layer components to Object-Relational persistent layer components

A Class maps to a Table.

An Attribute of a class maps to an Attribute of a Table.

Relational mappings for relationships (associations, aggregation and composition) are represented by replicating keys values in the form of foreign keys.

Associations with different cardinalities are mapped as follows:

- 1:1 Association: the primary key of one of the tables of the relationship - what you select - is replicated into the other.
- 1:N Association: the primary key of the other class is replicated into the table for the class participant in the side N of the relationship.
- N:M Association: a new table is created including the primary keys of the two tables participants, the composition of both keys is the primary key of the table, and they also are foreign keys.
- For the cases 1:1 and 1:N, there is another choice: a new table can be created, replicating the primary keys of the tables participants on it, like in the N:M case.
- For unidirectional associations, the primary key of the table corresponding to the class that has no visibility is replicated into the table of the other class.

Aggregation relationships:

An aggregation is a kind of association, so the rules for its mapping are the same that were explained for the association relationships.

Composition relationships:

Due to the Composition is a strong relationship, for the mapping it can be treated like in the Entity/Relationship model, using strong entity for the whole and weak entity for the part. The table corresponding to the weak entity has a partial key. The primary key of this table is the composition of the primary key of the table

corresponding to the strong entity and the partial key of the weak. The primary key of the strong entity is also a foreign key.

Inheritance relationships:

- Flat model: a unique table for all the class hierarchy is created in where the supertype and its corresponding subtypes are stored.
- Vertical partition: a table for each class that forms the hierarchy is created, each one with its attributes. The supertype key is replicated in each subtype, as primary and foreign key.
- Horizontal partition: only the tables corresponding to the subtypes are created. The supertype attributes are translated to these. The primary key corresponds to the primary key of the supertype.

Again here, some other mappings can be performed. The idea is to include all of them in the transformation system to offer more choices to the database designer.

5 Metamodel Transformations using XML Schemas and XML Documents

In order to implement the mappings defined between the layers, an XML Schema for each metamodel has been generated using Altova XMLSpy [2] tool. It provides an efficient and flexible environment for creating and editing XML Schemas, XML files, and XSLT stylesheets.

Taking the rules summed up in section 4, the transformations between the XML Schemas have been written in XSLT language [17] by means of a graphic interface (MapForce of Altova, [1]). The task has been enormously facilitated since MapForce has a complete set of graphic functions that permits an easy writing, modification and deletion of rules, giving a great flexibility and modifiability to the transformation definitions. Fig. 3 shows the transformation rules between UML-XML Schema (1st layer) and SQL:2003 data types-XML Schema (2nd non-persistent layer).

From Fig. 3 can be seen that each *class* of the UML layer will be transformed to an *StructuredType* of the O-R layer, with the same *name* like the *class*. The *isAbstract* attibute of the *Class* will be transformed in a *non_instantiable* attribute of the *StructuredType*. The *superclass* attibute of the *Class* will be transformed in a *supertype* attribute of the *StructuredType*. The *Class* attributes will be transformed in *StructuredType* attributes. In a similar way transformations were defined between the 2nd and the 3rd layer, but they are not shown here for space reasons. All rules are defined in the same way than the description given.

The model transformation process starts with the instantiation of the XML Schema corresponding to UML model, by the execution of this task a XML document is generated. From it, in the case of the O-R mapping, UML's XML document is transformed by the rules defined in XSLT into another XML document compliant with XML Schema of the SQL:2003 datatypes; and then this XML document is transformed into another corresponding to SQL:2003 schema-based XML Schema (the 3rd. layer). The advantages of this transformation process are that the XML documents are validated according to the rules of the XML Schema; so, they enclose the metamodels semantic. Besides, the document expressiveness allows an easy

accessing, understanding, debugging and correction of it. Therefore, what is involved with these model transformations can be seen in a straightforward manner. The documents are formed by well-defined parts which express the structured information in the most abstract and reusable way, with the possibility of extending them by adding new tags. The fact that the PIM is by definition independent of the technological platform makes the transformation to other platforms can be made in a simple way, reaching portability. Every change made in the system must be made in the PIM forcing the re-creation of the PSM.

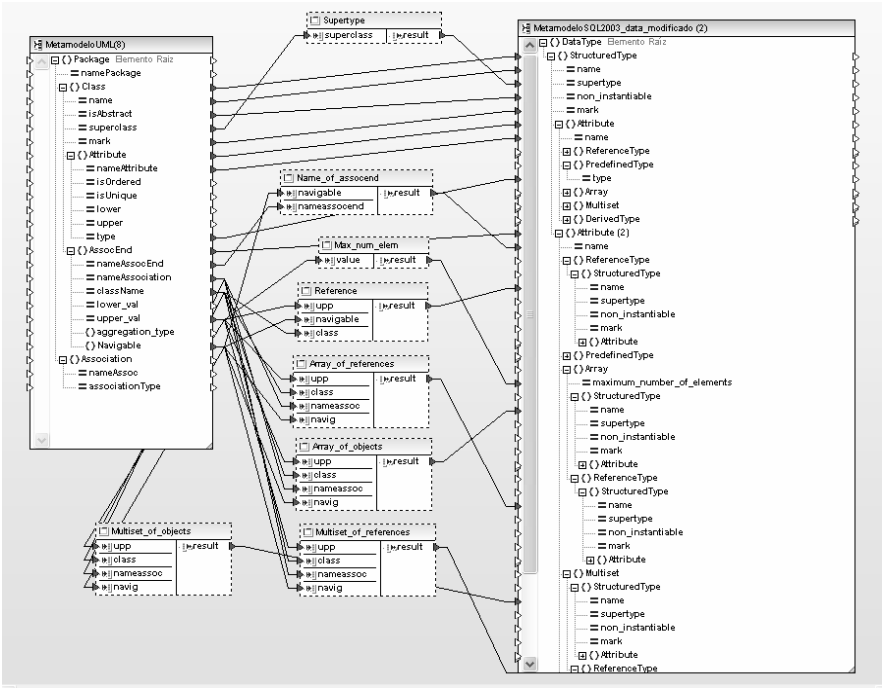

Fig. 3. Mapping rules between XML Schemas of the 1st and 2nd mapping layers

A figure representing the implemented transformation process can be seen in Fig. 4. The application requirements are the input and the database scripts are the development process output.

We are developing a CASE tool using this architecture to help the database designer's task. What we have described in this paper are the foundations for that.

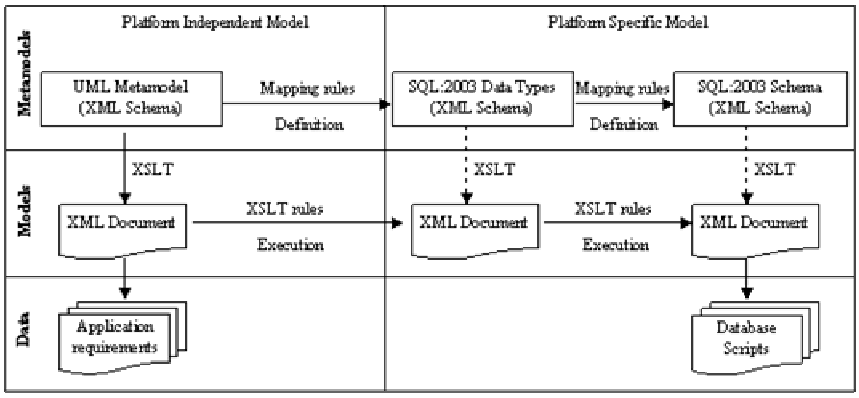

Fig. 4. Graphic representation of the metamodel transformation process

6 Conclusions

In this paper we address the issues involved in the development of a flexible approach for database design based on SQL:2003 standard, to allow the management of complex data of the current businesses. MDA directive has been used to implement metamodel transformations from UML class diagram until SQL:2003 standard tables. The platform independent model (PIM) corresponds to the UML metamodel and the platform specific models (PSM) are the SQL:2003 data types and SQL:2003 schema metamodels. These metamodels have been implemented into XML Schemas and mapping rules, written in XSLT, have been defined between them. Starting from a XML document, which is an instance of the UML-XML Schema, and executing the mapping rules it is possible to transform this document into another XML document of the next mapping layer. So, the models implicated have different abstraction levels.

The use of XML Schemas to represent the metamodels allows the instantiation of well formed XML documents of an application, which can be easy to read, understand, debug and correct. XSLT rules written using a flexible graphic interface, allows a straightforward administration (creation, modification, deletion) of them. So, modifications can be easily made at the level of the XML Schemas and XSLT rules. Since rules can be easily introduced and modified, a complete set of transformations can be defined into the system, offering several choices to the designer, giving flexibility to achieve the "best suited" design to the application needs. Default transformations can be included to facilitate the process for novel database designers.

The final goal of our research is to generate a CASE tool to increase the productivity in the design of databases compliant with the SQL:2003 standard. To our knowledge, no design tools with these characteristics can be found in the open literature.

References

1. Altova MapForce User Manual, www.altova.com/download/2006/MapForcePro.pdf
2. Altova XMLSpy 2008 Enterprise Edition User Manual, www.altova.com/download/2006/XMLSpyPro.pdf
3. Arango, F.; Gómez , M.C., Zapata, C.M.: Transformación del modelo de clases UML a Oracle9i® bajo la directiva MDA: un caso de estudio. Dyna, ISSN 0012-7353. Vol. 73, Issue 149, pp. 166-179 (2006)
4. Baroni, A.L., Calero, C., Ruiz, F., Brito e Abreu, F.: Formalizing Object-Relational Structural Metrics. 5º Portuguese Association of Information Systems Conference (2004)
5. Elmasri, R., Navathe, S.: Fundamentals of Database Systems, Third Edition. Addison Wesley (2000)
6. Golobisky, M.F., Vecchietti, A.: Mapping UML Class Diagrams into Object-Relational Schemas. In: Simposio Argentino de Ingeniería de Software (ASSE 2005). ISSN: 1666-1087. Pág. 65-79. 34 Jornadas Argentinas de Informática e Investigación Operativa – JAIIO (2005)
7. Grissa-Touzi, A., Sassi, M.: New Approach for the Modeling and the Implementation of the Object-Relational Databases. In: Transactions on Engineering, Computing And Technology V6 June 2005. ISSN 1305-5313 (2005)
8. Marcos, E., Vela, B., Cavero, J.M.: A Methodological Approach for Object-Relational Database Design using UML. Software and System Modeling 2(1): 59-75 (2003)
9. Melton, J. (2003) ISO/IEC 9075-2:2003 (SQL/Foundation), ISO standard.
10. Mukerji, J., Millar, J.: MDA Guide Version 1.0.1, OMG/2003-06-01, http://www.omg.org/cgi-bin/doc?omg/03-06-01
11. Narayanan, K., Ramaswamy, S.: Specifications for mapping UML models to XML schemas. Workshop in Software Model Engineering (WiSME'2005) (2005)
12. Extensible Markup Language (XML) 1.0, W3C Recommendation, August 2006, (Fourth Edition), http://www.w3.org/TR/REC-xml/
13. Unified Modeling Language: Infrastructure, version 2.0. OMG Final Adopted Specification. March 2006, http://www.omg.org
14. Unified Modeling Language: Superstructure, version 2.0. OMG Final Adopted Specification. August 2005, http://www.omg.org.
15. Vara, J.M., Vela, B., Marcos, E.: Transformaciones de modelos para el desarrollo de bases de datos XML. In: JISBD 2006. Pág. 225-238 (2006)
16. XML Schema Group Public Page. http://www.w3.org/XML/Schema
17. XSL Transformations (XSLT) Version 2.0, W3C Recommendation, January 2007, http://www.w3.org/TR/xslt20/