

# Extending UML for Object-Relational Database Design

E. Marcos, B. Vela, and J.M. Caverio

Kybele Research Group  
Rey Juan Carlos University  
Madrid (Spain)  
{cuca, b.vela, j.m.cavero}@escet.urjc.es

**Abstract.** The most common way of designing databases is using the E/R model without taking into account other views of the system. However, new object-oriented design languages, such as UML (Unified Modelling Language), permit modelling the full system, including the database schema, in a uniform way. Besides, as UML is an extensible language, it allows introducing new stereotypes for specific applications if it is needed. There are some proposals to extend UML with stereotypes for database design but, unfortunately, they are focused on relational databases. However, new applications require representing complex objects related with complex relationships and object-relational databases are more appropriated to support the new application requirements. The framework of this paper is an Object-Relational Database Design Methodology. The methodology defines new UML stereotypes for Object-Relational Database Design and proposes some guidelines to translate an UML schema into an object-relational one. The guidelines are based on the SQL:1999 object-relational model and on Oracle8i as an example of product. In this paper we focus on the UML extensions required for object-relational database design.

**Keywords:** UML extensions, Stereotypes, Database Design, Object-Relational Databases, Design Methodology, UML, SQL:1999, Oracle8i

## 1 Introduction

In spite of the impact of relational databases over the last decades, this kind of databases has some limitations for supporting data persistence required by present day applications. Due to recent hardware improvements more sophisticated applications have emerged such as CAD/CAM (Computer-Aided Design/Computer-Aided Manufacturing), CASE (Computer-Aided Software Engineering), GIS (Geographic Information System), etc. These applications can be characterised as consisting of complex objects related by complex relationships. Representing such objects and relationships in the relational model implies that the objects must be decomposed into a large number of tuples. Thus, a considerable number of joins is necessary to retrieve an object and, when tables are too deeply nested, performance is significantly reduced [3]. A new generation of databases has appeared to solve these problems: the object-oriented database generation, which include the object-relational [23] and object databases [4]. This new technology is well suited for storing and retrieving complex

data because it supports complex data types and relationships, multimedia data, inheritance, etc.

Nonetheless, good technology is not enough to support complex objects and applications. It is necessary to define methodologies that guide designers in the object database design task, in the same way as it has been done traditionally with relational databases. Over the last years some approaches to object-oriented database design have appeared [5,12,18,22,24]. Unfortunately, none of these proposals can be considered as “the method”, neither for object-relational nor for object databases. On the one hand, they do not consider the latest versions of the representative standards for both technologies: ODMG 3.0 for object databases [8] and SQL:1999 for object-relational databases [10]. And, on the other hand, some of them are based on old techniques as OMT [5] or, even, on the E/R model [24]. As a result they have to be updated considering UML, SQL:1999 and ODMG 3.0 as their reference models. In this paper we give a brief description of a methodology for object-relational database design which is based on UML extended with the required stereotypes. We will focus on object-relational databases, although the UML proposed extensions could also be fitted for object database design.

UML [6], as a Unified Modelling Language, is becoming increasingly more accepted. It also presents the advantage of being able to model the full system, including the database view, in a uniform way. Besides, as UML is an extendable language, it is possible to define the required stereotypes, tagged values and/or constraints, for each specific application. Therefore, the aforesaid methodology proposes to use the UML class diagram as the conceptual modelling notation.

The methodology provides some guidelines for translating a conceptual schema (in UML notation) into a logical schema. As the logical model we use the SQL:1999 object-relational model so that the guidelines were not dependent on the different implementations of object-relational products. We use Oracle8i as an implementation example.

Traditionally, methodologies provided graphical techniques to represent a relational schema, such as Data Structured Diagrams (DSD), relational “graph”, etc. In the same way, an object-relational schema can be represented either in SQL (SQL:1999 or Oracle8i) or using some graphical notation. As the graphical notation to represent the logical schema we also propose to use the UML class diagram extended with the required stereotypes, tagged values and/or constraints.

We have applied the methodology and the UML’s extensions in an internal project of our research group: the computer classroom reservation for the *Rey Juan Carlos* University. Now we are starting to apply it to a real development (together with INTESYS, a Spanish Company) in the framework of the MIDAS<sup>1</sup> project.

We want to outline the importance of providing methodological guidelines for database design using UML for data intensive applications. *“Generic lifecycle methodologies for object-oriented development have, to date, not given serious consideration to the need for persistence; either in terms of storage of objects in a relational database or in an objectbase”* [7]. Information systems and, of course,

---

<sup>1</sup> MIDAS is partially financed by the Spanish Government and the European Community (reference number: 2FD97-2163)

Web information systems, have to manage a lot of data that need to be persistent. A good design of persistent data will improve its use and its maintenance. Currently, the main mechanism of persistence is still the database. As it is stated in [13], the IDC market research firm reported global 1999 sales revenue of \$11.1 billion for relational and object-relational databases and \$211 million for object databases. Through to 2004, IDC predicts annual growth rates of 18.2 percent for relational and object-relational databases and 12.5 percent for object databases. Therefore, databases will still be the kernel of almost every information system for a long time.

The rest of the paper is organised as follows: section 2 summarises the current UML extensions for database design. Section 3 proposes new UML extensions for relational-database design based on SQL:1999 and Oracle8i object-relational models. Section 4 sums up the methodology showing some examples, which use the proposed extensions. Finally, section 5 summarises the main conclusions and future work.

## 2 Previous Concepts




The UML extension mechanism permits you to extend the language in controlled ways. This mechanism includes stereotypes, tagged values and constraints [6].

- *Stereotype: a stereotype extends the vocabulary of the UML, allowing us to create new kinds of building blocks that are derived from existing ones but that are specific to a problem. This new block has its own special properties (each stereotype may provide its own set of tagged values), semantics (each stereotype may provide its own constraints, and notation (each stereotype may provide its own icon). A stereotype is rendered as a name enclosed by guillemets (<<name>>)) and placed above the name of another element. It is also possible to define an icon for the stereotype.*
- *Tagged value: a tagged value extends the properties of a UML building block, allowing us to create new information in that element's specification. With stereotypes we can add new things to the UML; with tagged values, we can add new properties. A tagged value is rendered as a string enclosed by brackets and placed below the name of another element.*
- *Constraint: A constraint extends the semantics of a UML building block, allowing us to add new rules or modify existing ones. A constraint specifies conditions that must be held true for the model to be well- formed. A constraint is rendered as a string enclosed by braces and placed near the associated element.*

For applications that are very common (such as Web applications, DB applications, etc.) it would be desirable to provide a standard extension that could be used by every developer. So, Conallen has proposed a UML's extension for Web applications [9] and there are also some extensions for database design [1, 6, 19] (see Table 1).

We can notice that table 1 considers the relational model including stereotypes to represent the primary key, the foreign key, etc. Nevertheless, it does not provide specific stereotypes for an object model such as stereotypes for collection types, REF types, methods, etc.). The next section introduces the new required extensions (stereotypes, tagged values and/or constraints) for object-relational database design.

**Table 1.** Stereotypes for database design.

DB Element	UML Element	Stereotype	Icon
Database	Component	<<Database>>	
Schema	Package	<<Schema>>	
Tablespace	Component	<<Tablespace>>	
Table	Class	<<Table>>	
View	Class	<<View>>	
Index	Class	<<Index>>	
Column	Attributes	<<Column>>	
Primary Key	Attributes	<<PK>>	
Foreign Key	Attributes	<<FK>>	
Multivalued Attribute	Attribute	<<AM>>	
Calculated Attribute	Attribute	<<AD>>	
Composed Attribute	Attribute	<<AC>>	
NOT NULL Restriction	Attributes	<<NOT NULL>>	
Unique Restriction	Attributes	<<Unique>>	
Trigger	Restriction	<<Trigger>>	
Restriction	Restriction	<<Check>>	
Stored Procedure	Class	<<Stored Procedure>>	

### 3 UML Extensions for Object-Relational Database Design

Before explaining the UML extensions for object-relational database design we have to introduce the main constructors of the object-relational model. As we have explained in section 1, we have considered the SQL:1999 and Oracle8i models. The first one, because it is the current standard for object-relational databases and the second one as an example of product implementation. SQL:1999 data model extends the relational data model with some new constructors to support objects. Most of the latest versions of relational products include some object extensions. However, because in general these products have appeared in the market before the standard approval, current versions of object-relational products do not totally adjust to the SQL:1999 model.

#### 3.1. The Object-Relational Model: SQL:1999 and Oracle8i

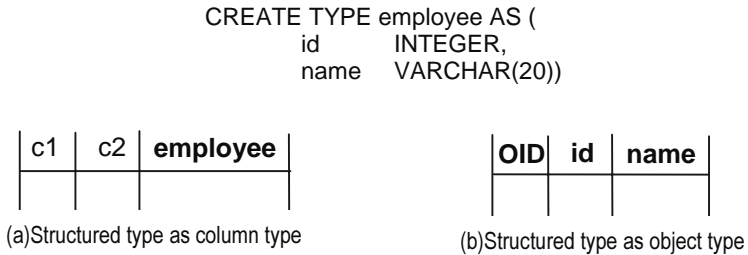
##### SQL:1999 Object-Relational Model

SQL:1999 [10,14] is the current standard for object-relational databases. Its data model tries to integrate the relational model with the object model. In addition to the object extensions, SQL:1999 provides other extensions to the SQL-92, such as triggers, OLAP extensions, new data types for multimedia data storage, etc. One of the main differences between the relational and the object-relational model is that the First Normal Form (1NF), the basic rule of a relational schema, has been removed

from the object-relational model. So a column of an object table can contain a collection data type.

SQL:1999 allows the user to define new structured data types according to the required data types for each application. Structured data types provide SQL:1999 the main characteristics of the object model. It supports the concept of strongly typed language, behaviour, encapsulation, substitutability, polymorphism and dynamic binding.

Structured types can be used as the type of a table or as the type of a column. A structured type used as the base type in the definition of a column permits the representation of complex attributes; in this case, structured types represent value types. A structured type used as the base type in the definition of a table corresponds to the definition of an object type (or a class); the table being the extension of the type. In SQL:1999 these kinds of tables are called typed tables. An object in SQL:1999 is a row of a typed table. When a typed table is defined, the system adds a new column representing the OID (Object Identifier) of each object of the table. The value of this attribute is system generated, it is unique for each object and the user cannot modify it. Figure 1 shows an example of a structured type defined in SQL:1999; in (a) the structured type is used as a value type (as the type of a column of a table) whereas in (b) it is used as an object type (as the type of a table).



**Fig. 1.** Structured types used as value and object types [14].

A structured type can include associated methods representing its behaviour. A method is a SQL function, whose signature is defined next to the definition of the structured type. The body specification is defined separately from the signature of the method.

SQL:1999 supports simple inheritance for structured types and for typed tables. A subtype inherits the attributes and the behaviour of the supertype. A subtable inherits the columns, restrictions, triggers and methods of the supertable.

A row of a typed table is an object and differs from the rest of objects by its OID. The value of the OID is generated by the system when a new object is inserted in the table. The type of this column is a reference type (REF). Therefore, each typed table has a column that contains the OID value. There are different REF types, one for each object type; that is, the REF type is a constructor of types rather than a type itself. An attribute defined as reference type holds the OID of the referred object. So, the REF type permits the implementation of relationships without using foreign keys.

SQL:1999 supports another structured type: the ROW type. The ROW type is a structured type defined by the user. It has neither extension nor OID; so, it cannot be used as an object type.

SQL:1999 only supports a collection type: ARRAY. The ARRAY can be used whenever another type can be placed (as the type of an attribute of a structured type, as the type of a column, etc.). The ARRAY type allows the representation of multivalued attributes not forcing tables to be in 1NF.

### **Oracle8i Object-Relational Model**

As well as SQL:1999, Oracle8i [20,21] supports structured data types that can be defined by the user (although, with a different syntax). A structured type can be used, as in SQL:1999, as a column type or as a type of a table. A structured type used as a column type represents a value type and a structured type used as the type of a table represents an object type; the table being the extension of the type. Each row of this kind of tables is an object and, in the same way as in SQL:1999, they have a special column of reference type (REF) that allows the identification of each object (OID). It is also possible to define an attribute as a reference to an object type. Oracle8i allows the association behaviour to object types, defining the signature of the methods as part of the type definition. The body of the method is defined separately.

Oracle8i supports two kinds of collections: VARRAYS, equivalent to the SQL:1999 ARRAY and the nested table. A nested table is a table that is embedded in another table. It is possible to define a table data type and to use this type as a column type in a table. Therefore, this column contains a table (called nested table) with a collection of values, objects or references.

One of the main differences between Oracle8i and SQL:1999 object-relational model is that Oracle8i does not support inheritance, neither of types nor of tables. There do exist, however, some relational products, such as, Universal Server of Informix [11], that support the inheritance concept in a similar way as the standard.

### **3.2. Object-Relational Extension for UML**

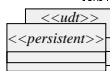

If we want UML to be used as the standard modelling language for designing databases, we have to modify it so it allows us to represent object-relational schema. In this way, it would be able to represent the constructors defined above such as the ARRAY, the NESTED TABLE or the REF type.



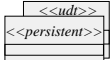

One possible solution could be to modify the meta-model of UML but this may seem like a drastic solution. However, UML has been designed to be extended in a controllable way. To accommodate this need for flexibility UML provides an extension mechanism that we have explained in section 2. This mechanism enables us to create new types of building blocks by means of stereotypes, tagged values and constraints.

According with [9] an UML's extension should contain: a brief description of the extension; the list and description of the stereotypes, tagged values and constraints; and a set of well-formedness-rules that are used to determine whether a model is semantically consistent. For each stereotype we have to specify the common properties and semantics that go beyond the basic element being stereotyped by defining a set of tagged values and constraints for the stereotype [6]. If we want these stereotype elements to have a distinctive visual cue, we have to define a new icon for



the stereotype [6]. Table 2 summarises the UML's extension proposed for object-relational database design.

**Table 2.** UML Extension for object-relational database design.

<p><b>Description</b></p> <p>This extension to UML defines a set of stereotypes, tagged values and constraints that enable us to model applications that work with object-relational databases. The extensions are defined for certain components that are specific to object-relational models allowing us to represent them in the same diagram that describes the rest of the system. The extension is based on SQL:1999 and Oracle8i object-relational models. Each element of the object-relational model has to have a UML representation. To choose the stereotyped element we used the following criteria:</p> <ul style="list-style-type: none"> <li>For SQL:1999 we have considered <i>structured types</i> and <i>typed tables</i> as stereotyped classes because they are explicitly defined in the SQL schema. The rest of the types (REF, ROW and ARRAY) are considered as stereotyped attributes.</li> <li>For Oracle8i we have considered <i>object types</i>, <i>object tables</i> and <i>nested tables</i> as stereotyped classes for the same reason as in SQL:1999. The REF type has been considered as a stereotyped attribute because it cannot be defined explicitly in the SQL schema. As the VARRAY can be, or cannot be, defined explicitly in the schema we will allow the two possibilities: to define it as stereotyped attributes or as stereotyped classes. We will use the stereotyped class when the VARRAY type was defined explicitly in the schema. In this way, the UML schema with stereotypes for Oracle8i will help the developer in the compilation process (compiling these schemas in Oracle8i is tedious because types have to be recompiled, so this technique can be helpful for the user).</li> </ul>	
<p><b>Prerequisite Extensions</b></p> <p>We consider that the required extension for the relational mode has already been defined [1, 19].</p>	
<p><b>SQL:1999 Stereotypes</b></p>	
<p><b>Structured Type</b> Metamodel class: Class Description: A <code>&lt;&lt;udt&gt;&gt;</code> allows the representation of new User defined Data Types.  Icon: None Constraints: It can only be used to define value types Tagged values: None</p>	<p><b>Typed Table</b> Metamodel class: Class Description: It is defined as <code>&lt;&lt;persistent&gt;&gt;</code>. It represents a class of the database schema, that should be defined as a table of a structured data type.   Icon: Constraints: A typed table implies the definition of a structured type, which is the type of the table. Tagged values: None</p>
<p><b>Composes</b> Metamodel class: Association Description: A <code>&lt;&lt;composes&gt;&gt;</code> association is a special relationship that joins a user defined data type <code>&lt;&lt;udt&gt;&gt;</code> with the class that uses it. It is a uni-directional relationship. The direction of the association is represented by an arrow at the end of the class, which use the user defined type.  Icon: None Constraints: It only can be used to join a <code>&lt;&lt;persistent&gt;&gt;</code> class with a <code>&lt;&lt;udt&gt;&gt;</code> class Tagged values: None</p>	<p><b>REF Type</b> Metamodel class: Attribute Description: A <code>&lt;&lt;ref&gt;&gt;</code> represents a link to some <code>&lt;&lt;persistent&gt;&gt;</code> class.   Icon: Constraints: A <code>&lt;&lt;ref&gt;&gt;</code> attribute can only refer to a <code>&lt;&lt;persistent&gt;&gt;</code> class Tagged values: The <code>&lt;&lt;persistent&gt;&gt;</code> class to which it refers</p>

<p><b>ARRAY</b>  Metamodel class: Attribute  Description: An <code>&lt;&lt;array&gt;&gt;</code> represents an indexed and bounded collection type.</p> <p>Icon: </p> <p>Constraints: The elements of an <code>&lt;&lt;array&gt;&gt;</code> can be of any data type except the <code>&lt;&lt;array&gt;&gt;</code> type</p> <p>Tagged values: The basic types of the array  The number of elements</p>	<p><b>ROW Type</b>  Metamodel class: Attribute  Description: A <code>&lt;&lt;row&gt;&gt;</code> type represents a composed attribute with a fixed number of elements, each of them can be of different data type</p> <p>Icon: </p> <p>Constraints: It does not have methods</p> <p>Tagged values: The name for each element and its data type</p>
<p><b>Redefined Method</b>  Metamodel class: Method  Description: A <code>&lt;&lt;redef&gt;&gt;</code> method is a inherited method that is implemented again by the child class.</p> <p>Icon: None</p> <p>Constraints: None</p> <p>Tagged values: The list of parameters of the method with their data types. The data type returned by the method.</p>	<p><b>Deferred Method</b>  Metamodel class: Method  Description: A <code>&lt;&lt;def&gt;&gt;</code> method is a method that defers its implementation to its child classes.</p> <p>Icon: None</p> <p>Constraints: It has to be defined in a class with children</p> <p>Tagged values: The list of parameters of the method with their data types. The data type returned by the method.</p>
<b>Oracle8i Stereotypes</b>	
<p><b>Object Type</b>  Metamodel class: Class  Description: A <code>&lt;&lt;udt&gt;&gt;</code> allows the representation of new user defined data types. It corresponds to the structured type in SQL:1999.</p> <p>Icon: None</p> <p>Constraints: It can only be used to define value types</p> <p>Tagged values: None</p>	<p><b>Object Table</b>  Metamodel class: Class  Description: It is define as <code>&lt;&lt;persistent&gt;&gt;</code>. It represents a class of the database schema, that should be defined as a table of an object type. It corresponds to the typed table in SQL:1999.</p> <p>Icon: </p> <p>Constraints: A typed table implies the definition of a structured type, which is the type of the table.</p> <p>Tagged values: None</p>
<p><b>Composes</b>  Metamodel class: Association  Description: A <code>&lt;&lt;composes&gt;&gt;</code> association is a special relationship that joins a user defined data type (<code>&lt;&lt;udt&gt;&gt;</code>, <code>&lt;&lt;array&gt;&gt;</code> or <code>&lt;&lt;nt&gt;&gt;</code>) with the class that uses it. It is an uni-directional relationship. The direction of the association is represented by an arrow at the end of the class, which uses the data type.</p> <p>Icon: None</p> <p>Constraints: It only can be used to join a <code>&lt;&lt;persistent&gt;&gt;</code> class with a <code>&lt;&lt;udt&gt;&gt;</code> class</p> <p>Tagged values: None</p>	<p><b>REF Type</b>  Metamodel class: Attribute  Description: A <code>&lt;&lt;ref&gt;&gt;</code> represents a link to some <code>&lt;&lt;persistent&gt;&gt;</code> class.</p> <p>Icon: </p> <p>Constraints: A <code>&lt;&lt;ref&gt;&gt;</code> attribute can only refer to a <code>&lt;&lt;persistent&gt;&gt;</code> class</p> <p>Tagged values: The <code>&lt;&lt;persistent&gt;&gt;</code> class to which it refers</p>



<p><b>VARRAY</b></p> <p>Metamodel class: Attribute/Class</p> <p>Description: A <code>&lt;&lt;array&gt;&gt;</code> represents an indexed and bounded collection type. It corresponds to the array type in SQL:1999.</p> <p>Icon: </p> <p>Constraints: The elements of an <code>&lt;&lt;array&gt;&gt;</code> can be of any data type except another collection type (<code>&lt;&lt;array&gt;&gt;</code> or <code>&lt;&lt;nt&gt;&gt;</code>)</p> <p>Tagged values: The basic types of the array The number of elements</p>	<p><b>Nested Table</b></p> <p>Metamodel class: Class</p> <p>Description: A <code>&lt;&lt;nt&gt;&gt;</code> represents an non-indexed and unbounded collection type.</p> <p>Icon: </p> <p>Constraints: The elements of a <code>&lt;&lt;nt&gt;&gt;</code> can be of any data type except another collection type (<code>&lt;&lt;array&gt;&gt;</code> or <code>&lt;&lt;nt&gt;&gt;</code>).</p> <p>Tagged values: The basic type of the nested table</p>
<p><b>Well-Formedness Rules</b></p> <p>Each <code>&lt;&lt;udt&gt;&gt;</code>, <code>&lt;&lt;array&gt;&gt;</code> or <code>&lt;&lt;nt&gt;&gt;</code> class has to be joined by a <code>&lt;&lt;composes&gt;&gt;</code> association with any class.</p> <p>A <code>&lt;&lt;ref&gt;&gt;</code> attribute in a <code>&lt;&lt;persistent&gt;&gt;</code> class implies an association with another class.</p> <p>A <code>&lt;&lt;persistent&gt;&gt;</code> class that contains a collection attribute whose elements were objects of a <code>&lt;&lt;persistent&gt;&gt;</code> class of <code>&lt;&lt;ref&gt;&gt;</code> to these objects implies an association between both classes.</p> <p>Each <code>&lt;&lt;persistent&gt;&gt;</code> class corresponds in SQL:1999 to a structured type with its corresponding extension. The extension is the typed table. Each <code>&lt;&lt;persistent&gt;&gt;</code> class in Oracle8i corresponds to an object type with its corresponding extension. The extension is the table of object type. That is to say: the object type and its extension are represented in the UML extension just as a <code>&lt;&lt;persistent&gt;&gt;</code> class.</p>	
<p><b>Comments</b></p> <p>This extension considers the object-relational model of SQL:1999 and Oracle8i. It should be modified according to the new versions of both object-relational models. Moreover, if we want to use other products this extension has to be adapted. For example, for Informix, we will have to introduce new extensions for the set, multiset and list collection types.</p>	

## 4 Integrating the UML Extension in a Methodology for Object-Relational Database Design

In this section we show an example of object-relational database design using the extension for UML proposed in the previous section. This example is made in the framework of a methodology for object-relational database design. This methodology is based on the proposal of [3] for object-oriented database design and on the proposal of [15]. Figure 2 summarises the main steps of the methodology.

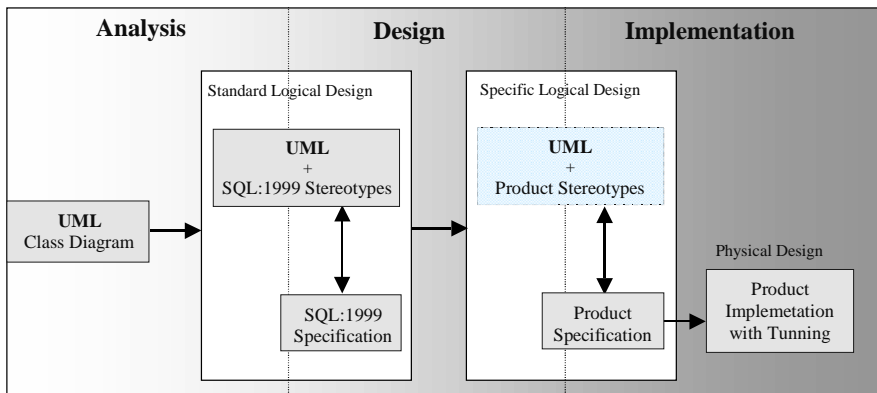


Fig. 2. Object-Relational Database Design Methodology

The methodology proposes three phases: analysis, design and implementation. Nonetheless, as it is shown in figure 5, differences between analysis, design and implementation phases are not as strong as in structured design.

At the **analysis** phase, we propose to use the UML class diagram to design the conceptual schema instead of the Extended E/R Model (commonly used for relational databases), because UML is the standard language for object-oriented system design. Unlike E/R, UML has the advantage that it allows the design of the entire system making the integration between different system views easier.

The **design** phase is divided into two steps:

- Standard design, that is, a logical design independent of any product.
- Specific design, that is, the design for a specific product (Oracle8i, Informix, etc.) without considering tuning or optimisation tasks.

Standard design is especially important in object-relational database design because each product implements a different object-relational model. This phase provides an object-relational specification independent of the product improving the database maintainability as well as making migration between products easier. As it is shown in figure 2, we propose two alternative techniques for this phase: defining the schema in SQL:1999, because it does not depend on any specific product; and/or using a graphical notation describing a standard object-relational model (the SQL:1999 model). This graphical notation corresponds with the relational graph that represents the logical design of a relational database [2]. As graphical notation we propose to use the UML extensions defined in section 3 for the SQL:1999 object-relational model.

For the specific design (intermediate stage between design and implementation), we have to specify the schema in the SQL (language) of the chosen product. We use, as an example, Oracle8i. Besides, we also propose to make optional use of a graphical technique to improve the documentation and the understandability of the generated SQL code. Moreover, this technique makes the compilation of the database schema easier by showing the correct order in which we have to compile each new user defined type and each table. The graphical notation is also UML substituting the SQL:1999 stereotypes with the specific stereotypes for the selected product.

Finally, the **implementation** phase includes the physical design tasks. In this phase the schema obtained in the previous phase should be refined, carrying out making a tuning to improve the response time and storage space according to the specific needs of the application.

**Table 3.** Guidelines for object-relational database design.

UML	SQL:1999	Oracle8i
Class	Structured Type	Object Type
Class Extension	Typed Table	Table of Object Type
Attribute	Attribute	Attribute
Multivalued	ARRAY	VARRAY
Composed	ROW / Structured Type in column	Object Type in column
Calculated	Trigger/Method	Trigger/Method
Association		
One-To-One	REF/REF	REF/REF
One-To-Many	REF/ARRAY	REF/Nested Table
Many-To-Many	ARRAY/ARRAY	Nested Table/Nested Table
Aggregation	ARRAY	Nested Table
Generalisation	Types/Typed Tables	Oracle cannot directly represent the generalisation concept

Relational database methodologies propose some rules to transform a conceptual schema into a standard logical schema. In the same way, our methodology also proposes a technique that allows the transformation of a schema from one phase to the next. This technique suggests some rules that should be considered only as guidelines. These rules are summarised in table 3 and are detailed in [16, 17].

#### 4.1. Example

As we have explained in the introduction, we have used the described methodology and the UML's extensions in an internal project of our research group. The application resolves the problem of the computer reserves in our university. The application has been developed for a Web environment using XML and Oracle8i.

In this section we describe part of the application. to show an example of how to use the defined stereotypes for object-relational database design. Teachers can reserve computer-classrooms to impart their practical classes. A classroom is composed of a set of computers. The students can only reserve computers if a teacher or another student have not previously reserved them. To develop the application we have applied the mentioned methodology following the phases shown in figure 2.

**Analysis phase:** Figure 3 shows the UML class diagram used to design the conceptual schema.

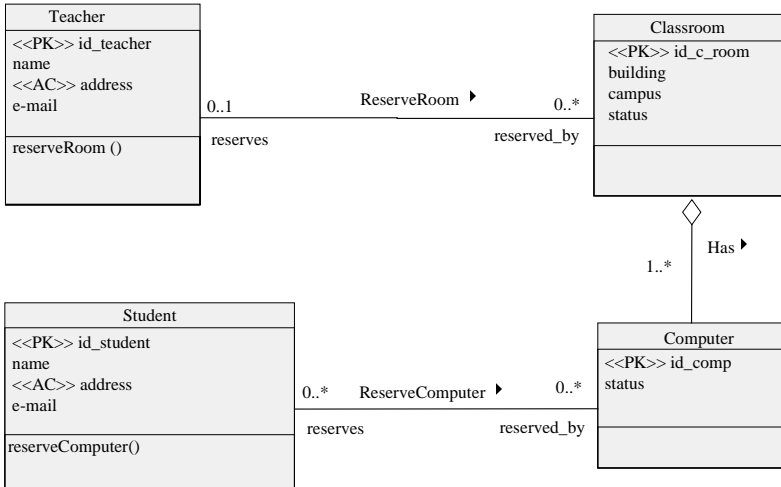


Fig. 3. Conceptual Design in UML

**Analysis-Design phase:** Figure 4 shows the standard design with the proposed graphical notation, consisting in the defined UML extensions for the SQL:1999 object-relational model.

**Design-Implementation phase:** Figure 5 shows the specific design (intermediate stage between design and implementation phases), with the proposed graphical notation. The graphical notation is also UML substituting the SQL:1999 stereotypes with the specific stereotypes for the selected product. We have also specified the obtained schema in the SQL (language) of the chosen product, Oracle8i, as figure 6 shows.

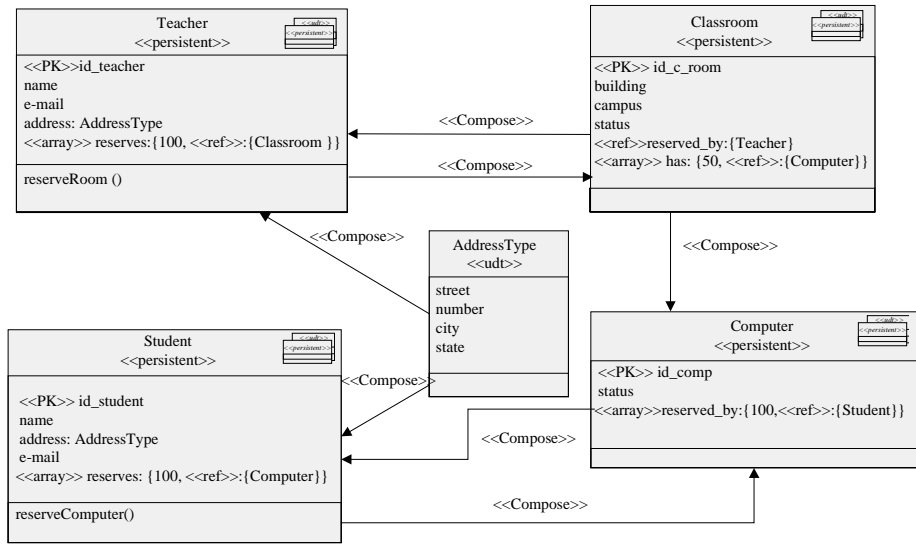


Fig. 4. Logical Design in SQL:1999

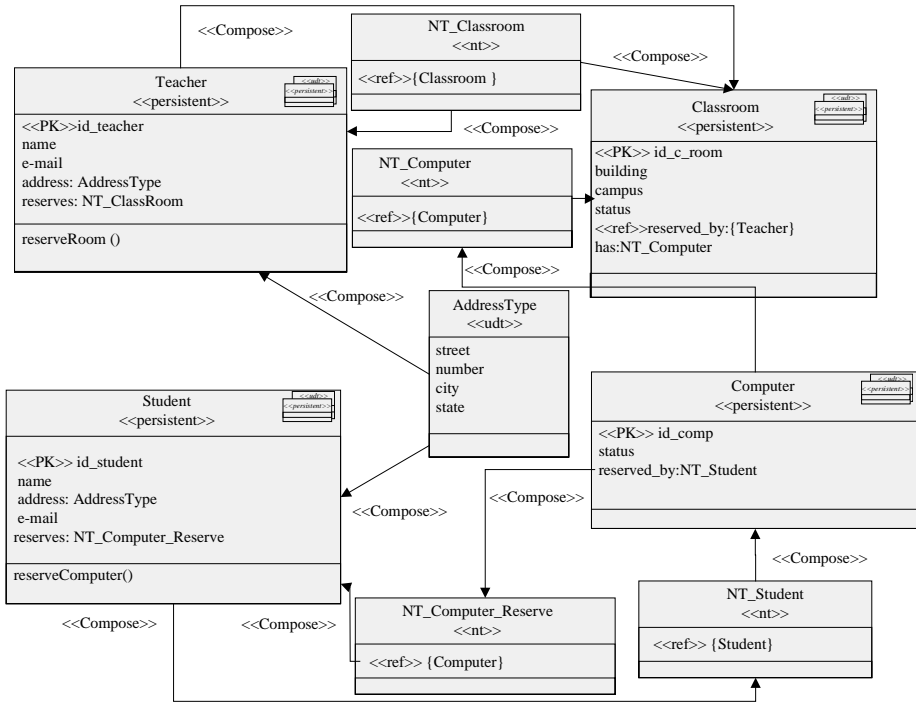


Fig. 5. Logical Design in Oracle8i

```

CREATE OR REPLACE TYPE AddressType AS OBJECT
( street VARCHAR(20)
, num NUMBER
, city VARCHAR(20)
, state VARCHAR(10))
/
CREATE OR REPLACE TYPE Computer AS OBJECT
( id_comp VARCHAR(5)
, status VARCHAR(2)
, reserved_by NT_Student)
/
CREATE OR REPLACE TYPE NT_Computer
AS TABLE OF REF Computer
/
CREATE OR REPLACE TYPE NT_Computer_Reserve
AS TABLE OF REF Computer
/
CREATE OR REPLACE TYPE Student AS OBJECT
( id_student VARCHAR(5)
, name VARCHAR(50)
, address AddressType
, e_mail VARCHAR(100)
, reserves NT_Computer_Reserve)
/
CREATE OR REPLACE TYPE NT_Student
AS TABLE OF REF Student
/
CREATE OR REPLACE TYPE Teacher AS OBJECT
( id_teacher VARCHAR(5)
, name VARCHAR(50)
, e_mail VARCHAR(100)
, address AddressType
, reserves NT_ClassRoom)
/
CREATE OR REPLACE TYPE Classroom AS OBJECT
( id_c_room VARCHAR(5)
, building VARCHAR(50)
, campus VARCHAR(10)
, status VARCHAR(2)
, reserved_by REF Teacher
, has NT_Computer)
/
CREATE OR REPLACE TYPE NT_ClassRoom
AS TABLE OF Ref Classroom
/
/* Recomilation */
CREATE OR REPLACE TYPE Computer AS OBJECT
( id_comp VARCHAR(5)
, status VARCHAR(2)
, reserved_by NT_Student)
/
CREATE OR REPLACE TYPE Teacher AS OBJECT
( id_teacher VARCHAR(5)
, name VARCHAR(50)
, e_mail VARCHAR(100)
, address AddressType
, reserves NT_ClassRoom)
/
/* Table Creation*/
CREATE TABLE TClassroom OF Classroom
(PRIMARY KEY (id_c_room))
NESTED TABLE has STORE AS table_computer_cr;
CREATE TABLE TComputer OF Computer
(PRIMARY KEY (id_comp))
NESTED TABLE reserved_by STORE AS table_student;
CREATE TABLE TStudent OF Student
(PRIMARY KEY (id_student))
NESTED TABLE reserves STORE AS table_computer_s;
CREATE TABLE TTeacher OF Teacher
(PRIMARY KEY (id_teacher))
NESTED TABLE reserves STORE AS table_classroom;

```

Fig. 6. Oracle8i Code

## 6 Conclusions and Future Work

Object-relational databases will be become increasingly each day more extended because they provide a better support than relational technology for complex data and relationships. As a result, new methodologies for object-relational database design are emerging. In this paper we have summarised a methodology for object-relational database design, which is based on UML extended with the required stereotypes. We have focused on object-relational databases, although the UML's proposed extensions could also be fitted for object database design.

The methodology proposes three phases: analysis, design and implementation. As conceptual modelling technique we have chosen the UML class diagram. As the logical model we have used the SQL:1999 object-relational model, so that the guidelines are not dependent on the implementations of object-relational products. As a product example we have used Oracle8i.

Traditionally, methodologies provided graphical techniques to represent a relational schema, such as Data Structured Diagrams (DSD), relational "graph", etc. In the same way, an object-relational schema can be represented either in SQL (SQL:1999 or Oracle8i) or using some graphical notation. As graphical notation to represent the

logical schema we have proposed using the UML class diagram extended with the required stereotypes, tagged values and/or constraints. This paper has been focused on the UML's extensions required for object-relational design in both models, SQL:1999 and Oracle8i. We have summarised the stereotypes, tagged values and constraints. We have also explained a part of an application that we have developed using the UML extension: the computer classroom reservation for the *Rey Juan Carlos* University. Now we are starting to apply it to a real development in the framework of the MIDAS project. The development has been carried out together with INTESYS, a Spanish Company that participates in the MIDAS project.

**Acknowledgments.** This work is being carried out as part of the MIDAS project. MIDAS is partially financed by the Spanish Government and the European Community (reference number: 2FD97-2163).

## References

1. Ambler (1999), *Persistence Modelling in the UML*. In: <http://www.sdmagazine.com/articles/1999/0008/0008q/0008q.htm>
2. Atzeni, Ceri, Paraboschi and Torlone (1999). *Database Systems. Concepts, Languages and Architectures*. McGraw-Hill.
3. Bertino and Marcos (2000), "Object Oriented Database Systems". In *Advanced Databases: Technology and Design*, O. Díaz and M. Piattini (Eds.). Artech House.
4. Bertino and Martino (1993), *Object-Oriented Database Systems. Concepts and Architectures*. Addison-Wesley.
5. Blaha and Premerlani (1998), *Object-Oriented Modeling and Design for Database Applications*. Prentice Hall.
6. Booch, Rumbaugh and Jacobson (1999), *The Unified Modelling Language User Guide*. Addison Wesley.
7. T. Case, B. Henderson-Sellers and G.C. Low (1996). A generic object-oriented design methodology incorporating database considerations. *Annals of Software Engineering*. Vol. 2, pag. 5-24.
8. Cattell and Barry (2000), *The Object Data Standard: ODMG 3.0*. Morgan Kaufmann.
9. J. Conallen (2000), *Building Web Application with UML*. Addison-Wesley.
10. Eisenberg and Melton (1999), "SQL:1999, formerly known as SQL3". *ACM SIGMOD Record*, Vol. 28, No. 1, pp. 131-138, March 1999.
11. Informix Corporation (1999), *Informix Guide to SQL: Reference*. Electronic Documentation, Informix Press.
12. C. Kovács and P. Van Bommel (1998), "Conceptual modelling-based design of object-oriented databases". *Information and Software Technology*, Vol. 40, No. 1, pp. 1-14.
13. Leavit, N. (2000), "Whatever Happened to Object-Oriented Databases?". *Computer*, pp. 16-19, August 2000.
14. Mattos, N. M (1999), *SQL:1999, SQL/MM and SQLJ: An Overview of the SQL Standards*. Tutorial, IBM Database Common Technology.
15. Marcos, E. and Cáceres, P. (2001), "Object Oriented Database Design". In: *Developing Quality Complex Database Systems: Practices, Techniques, and Technologies*. Ed. Shirley Becker. Idea Group (accepted to publish in 2001).
16. E. Marcos, B. Vela and J. M. Cavero (2001), A Methodology for Object-Relational Database Design Using UML (submitted to 12<sup>th</sup> International Conference and Workshop on Database and Expert Systems and Applications, DEXA 2001).

17. E. Marcos, B. Vela and J. M. Caverio (2001), Aggregation and Composition in Object-Relational Database Design (submitted to Fifth East European Conference on Advances in Databases and Information Systems, ABDIS'2001).
18. Muller (1999), *Database Design for Smarties*. Morgan Kaufmann.
19. Naiburg, E. (2000), "Database Modeling and Design Using Rational Rose 2000e". *Rose Architect* Vol. 2, Issue 3, pp. 48-51.
20. Oracle Corporation (1998), "Objects and SQL in Oracle8". Oracle Technical White paper. In: *Extended DataBase Technology conference (EDBT'98)*. Valencia (Madrid).
21. Oracle Corporation (2000), Oracle8i. *SQL Reference. Release 3 (8.1.7)*. In: [www.oracle.com](http://www.oracle.com).
22. Silva and Carlson (1995), "MOODD, a method for object-oriented database design." *Data & Knowledge Engineering*, Vol. 17, pp.159-181.
23. Stonebraker and Brown (1999). *Object-Relational DBMSs. Traking the Next Great Wave*. Morgan Kauffman.
24. Ullman and Widom (1997), *A First Course in Database Systems*. Prentice-Hall.