O-ODM Framework for Object-Relational Databases

Carlos Alberto Rombaldo Jr, Solange N. Alves Souza, *Department of Computing Engineering and Digital System of University of São Paulo, São Paulo, Brazil.*
Luiz Sergio de Souza, *Faculdade de Tecnologia - Carapicuíba, São Paulo, Brazil*

***Abstract* —Object-Relational Databases introduce new features which allow manipulating objects in databases. At present, many DBMS offer resources to manipulate objects in database, but most application developers just map class to relations tables, failing to exploit the O-R model strength. The lack of tools that aid the database project contributes to this situation. This work presents O-ODM (Object-Object Database Mapping), a persistent framework that maps objects from OO applications to database objects. Persistent Frameworks have been used to aid developers, managing all access to DBMS. This kind of tool allows developers to persist objects without solid knowledge about DBMSs and specific languages, improving the developers' productivity, mainly when a different DBMS is used. The results of some experiments using O-ODM are shown.**

***Keywords* —Object-Relational Databases. Persistence Framework. Java Annotations. SQL:2008**

I. INTRODUCTION

Persistent frameworks, frequently called ORM (Object Relational Mapping) tool [6]-[7]-[8], have been used to aid database projects. This kind of tool maps objects from application to relation (relational databases - RD) [11]. Using ORM tools, developers have advantages; (1) they can persist data in RD without solid knowledge of Relational Database Management System (RDBMS). It allows developers to focus on application development (OO paradigm and language aspects); (2) all data access is made through the tool, since ORM tools are integrated in programming environment; developers can use a single environment to do this work; (3) generally, when more than one DBMS is used, only one instruction is modified. This instruction indicates the new DBMS; then, all the code produced by the tool to one DBMS is automatically changed to another. In case the instruction was not available, the developer would have to produce the new SQL code according to the characteristic of the DBMS chosen. All these aspects aid both: the point of view system maintenance and developers' productivity. Thus, the benefit of using persistent frameworks cannot be ignored. On the other hand, it is necessary to consider the new characteristics of Object-Relation Databases (ORDB).

ORDB allows manipulating objects in databases. Many DBMS offer new resources such as UDTs (User Data Types), composite types, REF types, inheritance and others that can be used to model objects in databases. Besides, using REF types to represent relationship between objects can result in improvement of performance given that no field needs to be created in an existing or new relation. This characteristic could be more suitable for new applications that have emerged and which present complex objects such as CAD/CAM (Computer Aided Design/ Computer Aided Manufacturing), GIS (Geographic Information System), Genetic, etc [9]. Adding to this, using ORDB, objects from an application must be mapped to objects from databases; thus the impedance mismatch, which has been reported in the literature and in real applications as a problem, can be avoided. Another ORDB advantage is the possibility to use only one conceptual model for both the application and the data tiers [1]. Generally, the entity-relationship model (ERM) and UML class model are built when the relational model is employed. This causes an overhead not only related to mapping class to relation, but also to elaborating the ERM and the need of specific knowledge to generate this model.

Since the strength of the Object-Relational Model might be more explored [4] together with the lack of tools to aid projects and maintenance of ORDB, this paper proposes an O-ODBM (Object- Object Database Mapping) tool, an object-relational persistence framework. O-ODBM maps an object from the application to the ORDB object [16].

However, not all DBMS implement all the resources of objects specified in the SQL standard. Therefore, some elements can be unavailable in some of them. Undoubtedly, this is another important aspect which contributes to ignoring object resources from DBMS and adds complexity to build CASE and Persistent framework tools to ORDB.

An example was used to evaluate the O-ODBM. We here present not only the O-ODBM characteristics, but also an example and the results.

To develop the O-ODBM, characteristics and operations were studied which are defined or implemented in JPA (Java Persistence API) and/or JDO (Java Data Object) standards and in Hibernate and Torque frameworks. Some of those characteristics, which provide benefits and /or facilities to developers, were implemented in our first version of O-ODBM Framework.

This article is organized as follows. In chapter 2, some characteristics of JPA (Java Persistence API) and JDO (Java Data Object), which were incorporated to O-ODBM, are introduced. Chapter 3 introduces the O-ODBM. Chapter 4 presents the example employed to evaluate the O-ODBM tool,

and the results. Finally, chapter 5 concludes and presents future works.

II. JPA AND JDO STANDARDS – SOME CHARACTERISTICS

The O-ODBM *Framework* was developed in Java programming language. Some reasons pointed for this choice are (1) many ORM *Frameworks* available are based on Java language. (2) Java language facilitates the interoperability and (3) the number of the OO applications that developed in Java are increasing.

The JDO (*Java Data Object*) [7] e JPA (*Java Persistence API*) [8] standards define mapping from application object to relations of RDB. These standards also include a set of properties that simplify persistence and data access. Some of these properties were highlighted considering the scope of the O-ODBM project:

- all access to data is made only by the framework. As a result, it is no longer necessary to have a solid knowledge about the DB, SQL and DBMS used.
- offers a language for manipulating data that is closer to OO programming language than SQL.
- transaction manage, which allows the developer to define the beginning and end of transactions. The Framework is responsible for the interface with the DBMS used.
- mechanism for performance control to access, insert, delete and update objects. In OO applications, references between objects are very common. These references are mapped to tables and integrity rules, so that when a query is made, more than a table could be accessed. The use of annotations [12] is employed by the developer to indicate which objects must be persisted. Annotations allow adding information to java classes directly. The Framework uses this information to create the SQL code to generate tables, attributes, integrity rules in attributes and between tables, etc.

III. PROJECT OF O-ODBM FRAMEWORK

The rules of mapping defined for RDB are not suitable, since the new data types connected to the OO paradigm available in ORDBMS are not considered. The rules defined for the Framework proposed are summarized in Tables I and II. More details of these rules can be found in [1]-[2], which are a complementation of [4]-[9]-[14] from the point of view of real applications.

Requirements of O-ODBM Framework

A set of requirements, which are detailed as follows, was defined to guide the development of the Framework. In doing so, the characteristics of ORM Frameworks were considered, which are advantages for both application and developers. Then, JPA and JDO standards were studied, as well as Hibernate and torque implementations [6]-[7]-[8]. In view of the ORDBMS, SQL:2008 was also studied, along with Oracle 11g release 2 and BD2 9.7.5 version DBMS. To simplify the reference, the requirements were identified by the R letter and a sequential number, presented as follows.

R1 – to control the referential integrity rule connected to

TABLE I
MAPPING OF OBJECT FROM APPLICATIONS TO ORDBMS OBJECTS - ADAPTED FROM [1]

OO	ORDBMS	Justify
Class	Table UDT Typed table	Classes may be mapped to conventional tables. However, if the intention is to define methods and/or hierarchies, an UDT must be defined and, to store data, a typed table connected to UDT needs be created.
Abstract class	UDT	an UDT should be created whithout a typed table conected to it to represent an abstract class. In this case, the UDT would be used for defining other UDTs and as it does not have a typed table connected to it, instances will not be persisted.
Simple attribute	Build-in type	SQL:2008 presents many built-in types such as integer, real, etc. It is hence possible to find a corresponding type in SQL for each primitive type of Java.
multivalued attribute	Array or Multiset	multidimensional structures are suitable to store attributes of the same type (collections).
Methods	UDT methods	It is possible to define methods connected to UDTs. Thus, developers can choose to define methods in the database or in the application.

REF type. ORDB allows defining the relationship between objects using REF. However, if an object A, which is

TABLE II
MAPPING OF ASSOCIATIONS AND HIERARCHY IN ORDBMS – ADAPTED FROM [1]

Association		Corresponding in ORDBMS	
Bidiretional Association	Composition/ Aggregation/ Association	1..1	a cross reference is defined, i.e., each class maintains a reference (REF) to the other.
		1..*	a cross reference is also used, although the aggregated class will be an Array ou a Multiset of references.
Unidirectional Association		Similarly to the bidirectional associations above presented, though the reference will be only in table.	
Nth Association (three or more classes)		A table or a UDT is defined with the name of the association. The table or the UDT (and the typed table) must maintain references to the classes involved.	
Associative Class		a table or a UDT can be defined for the association class similarly to nth association.	
Generalization/ Specialization		a UDT is defined for each class of the hierarchy. Typed table would be defined later if data need to be persisted.	

referenced by object B, is removed, B gets a null reference.

Then, a rule, similar to the rule that controls foreign key in RDB, needs to be implemented to avoid a null reference.

R2 - Flexibility for multiple platforms of databases. This requirement means that the Framework gives a simple mechanism for a developer to change the DBMS and all SQL code for persistence and data access, which was generated by the framework for the first DBMS, will automatically be replaced by the code for the new DBMS. It is important to highlight, as explained before, there are differences among ORDBMS and some resources for database object can be not available; therefore, this may be the most difficult requirement to be achieved.

R3 – The developer does not need to know the SQL and DBMS employed. Thus, the framework has to present a language or a mechanism for object manipulation very similar to the OO programming language (if compared with the SQL). As a result, the learning process is facilitated, since the developer does not need to know SQL to use a DBMS.

R4 – Managing DBMS connections – including to open, to close and to verify the timeout of connections. If there are unfinished transactions, the Framework will keep the connection open until the commit or rollback of these transactions. The Framework would force itself to interrupt the transactions, despite keeping (ex. doing rollback) the data integrity in the database.

R5 – Managing the execution of transactions. For this, the Framework has to offer an interface for the developer to define his transactions.

R6 – Automatic code generation for object schema in DBMS, including codes for manipulating these objects.

R7 – the framework will be an access point to database; making the direct connection between application and database unnecessary.

R8 – use of annotations for defining which will be persisted in the database, facilitating the configuration of objects schema. The ORM Frameworks studied employs a similar mechanism; however, in the case of O-ODBM Framework, appropriated annotations have to be created.

R9 – Implementation of inheritance in database, according to OO.

R10 – Implementation of unidirectional, bidirectional and multivalued relationship, using reference (REF) to object when possible.

R11 – Application performance is not degraded.

R12 – Data could be retrieved on demand. In other words, according to what is defined by the developer, the Framework will postpone or will not retrieve related data to improve the performance of the data access [6]-[8]. This is an important aspect for performance because one object referenced by another can keep references for others and so on, which would certainly degrade the data access performance. Therefore, when there is no interest in referenced objects, the retrieval of object and its references would cause unnecessary performance degradation.

R13 – Data could be persisted on demand, which is defined as cascade property in JPA [8]. In this case, the Framework would do the persistence of the associated objects, preventing null references from being found, i.e., references for objects that do not exist

A. Architecture of O-ODBM Framework O-ODBM

The tool accepts input in two different formats: Java code, in which annotations are used to declare persistent classes, or XML files, which correspond to SQL code for the DBMS chosen. In the first case, the Framework presents a set of annotations, similarly to the ORM Frameworks. In the second case, the XML file, which represents the logical schema to ORDB, is generated by a case tool [1] for ORDB. The Framework should be part of the development integrated environment, in which from a conceptual model (ex. UML class model), or from a logical schema, the OR database can be automatically implemented in the DBMS chosen and accessed by the Framework.

A XSD (XML Schema Definition) was formalized to register the mapping from Java classes to ORDB objects. In this XSD, according to SQL:2008 [10] the ORDB data types are defined that define database objects, methods, inheritance, collections and other OO concepts. Therefore, in case the input of the Framework is a XML file produced by the modeling tool, the XSD would be used to verify it. In addition, XML documents are also used internally by the tool for describing the necessary information to mapping among different formats produced by the tool

Figure 1 introduces the architecture of the O-ODBM Framework and its components are described as follows.

Configuration Processor: reads the Java class annotated with the annotations introduced by the Framework. Once the Java classes have been interpreted, this module processes the annotations and generates the XML code with OR structure based on SQL:2008. It was decided to first generate the SQL code for SQL:2008 and then translate it to a dialect of specific DBMS. This decision was made due to the differences among DBMS regarding the object resources offered. Some DBMS implement part of these resources only; moreover, the implementation of the specific element can be different among these DBMS. On the other hand, the SQL:2008 not only has all the elements related to objects, but can also be easily

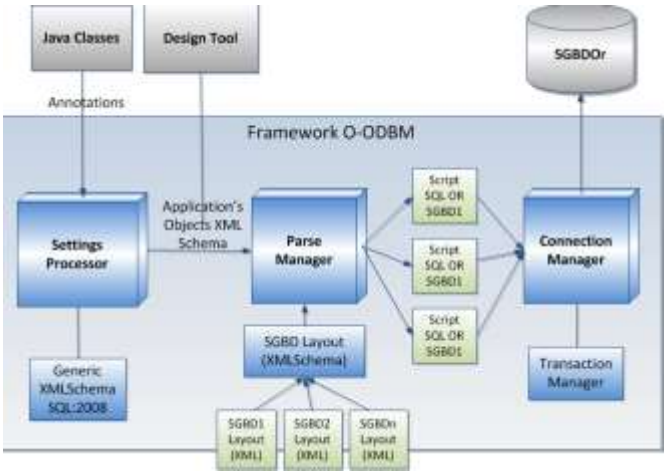

Fig. 1. Architecture of O-ODBM Framework.

translated into another SQL dialect. The XML-SQL schema that represents the database object schema is equivalent to the application object schema. The XML-SQL schema generated by the Configuration Processor is the input of the Conversion Manager Component.

Conversion Manager: generates the SQL scripts to be executed by the DBMS chosen. The Conversion Manager uses the DBMS layout file appropriated for translating the XML-SQL code into an adequate SQL dialect. For this, the Framework uses the XML file (DBMS Layout) that has the specific syntax for each DBMS. The output of this module is the SQL script, which is submitted to the DBMS by the Connection Manager Component.

Connection Manager: all the operations between the Framework and DBMS, for example, execution of SQL script to create structures, persistence and retrieval of objects are made by a connection. This component manages the connections with DBMS and this is transparent for the developer. Connections are automatically opened by the Framework whenever the operation is submitted.

Transaction Controller: manages all the transactions with the DBMS. When a transaction is opened, this component is activated and when the connection needs be closed, this component is consulted to verify/guarantee that there are no transactions open for that connection. In this process, a transaction can be finished (rollback or commit), or the connection is not closed. This component also manages the transaction inactive time and automatically finishes it if the transaction achieves the timeout.

DBMS Layout: Since a XML file, produced by the CASE tool, could be the input for the Framework; a XSD is also used by the Framework, similarly to the SQL schema, for validating this file.

B. Annotations

The API (*Application Programming Interface*) of the Framework is integrated with the programming environment. This way, the developer has the set of annotations, which were produced in this work, available for use and integrated with the development environment. The type of annotation will determine the map from Java class to ORDB element made by the tool. TABLE III introduces the set of annotations. TABLE IV and TABLE V show more annotations that are used for defining parameters and default values, respectively. Experienced developers in Framework and/or in ORDB could redefine default values.

IV. EXAMPLE USED FOR TESTINGTHE FRAMEWORK

An example, the persistent object schema of which is shown in Figure 2, was used for testing the applicability of the Framework. The main concern was to evaluate the behavior for queries involving objects in hierarchy and the use of reference (REF) for representing association between objects. However, this evaluation is not enough to draw conclusions about the performance of ORDBMS. Therefore, a more careful evaluation must be made in the future.

TABLE III
ANNOTATIONS.

Annotation	Description
@DbObject	indicates the class must be persisted.
@DbField	indicates the attribute must be persisted.
@DbMethod	indicates the object method must be created in DBMS.
@DbInhetitance	indicates the object is part of the hierarchy. Then, the hierarchy must be represented in DBMS. If the parent object has not been annotated with DbObject, only the derived objects would be part of a hierarchy in DBMS, although the characteristics inherited will be part of the derivated objects.
@DbRelation	indicates the attribute represents the association. The associations are represented by the inclusion of the attributes in associated classes. These attributes make references between themselves and, depending on the cardinality of association, this reference may be to an object or to a collection of objects..

In the example, only the annotations shown in Table III

TABLE IV
CONFIGURATIONS FOR @DBFIELD ANNOTATION.S.

PARAMETER	Default value	Description
size	255 for text and numbers.	defines the attribute max size.
isPK	none	indicates the attribute will be a primary key.
autoIncremet	none	indicates the attribute values will be generated by the DBMS.
type	keeps the equivalent data type in the DBMS.	defines the data types that will be used in DBMS

TABLE V
CONFIGURATIONS FOR @DBFIELD ANNOTATION.S.

PARAMETER	Default value	Description
size	255 for text and numbers.	defines the attribute max size.
isPK	none	indicates the attribute will be a primary key.
autoIncremet	none	indicates the attribute values will be generated by the DBMS.
type	keeps the equivalent data type in the DBMS.	defines the data types that will be used in DBMS

were used.

Since the class was annotated, a DAO class for each persistent class was generated. Then, using the O-ODBM Framework, the SQL script of the database schema was generated and executed in DBMS. After that, insert, update, delete and select operations were carried out. First, DB2 DBMS was used, and later Oracle DBMS. It is important to highlight that all these procedures were made by changing the directives of configurations only, i. e., neither class nor annotations were changed. TABLE VI introduces the results of each operation

V. EVALUATION

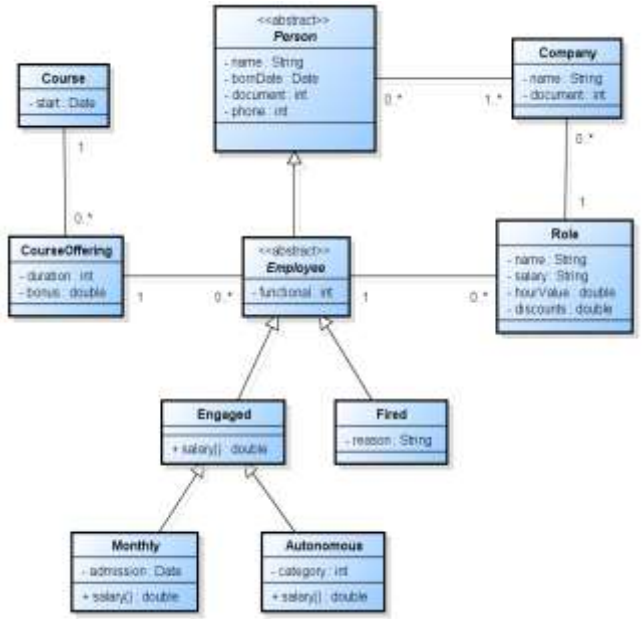

Fig. 2. Class Diagram used in the example to evaluate the Framework.

The set of requirements, defined in section III, and the result of the tests were used for evaluating the o-ODBM Framework.

A. Compliance with the requirements

R1 – the referential integrity rule will be implemented by the Framework if the Java code presenting the appropriate annotations (i.e., the attribute of the object has @DbRelation annotation). Therefore, the Framework will implement an operation to guarantee that null references do not exist.

R2 – the Framework only supports Oracle and DB2, although the modification of one into another is very simple for the developer, since he only declares what DBMS will used and the Framework generates the appropriate code. There are few DBMS that support OR characteristics and this limits the application of this requirement. However, it can be considered met with the use of these two DBMS.

R3 – the Framework has no data access language. However, the annotations can be used for persistence, queries and updating objects.

R4 – the Framework manages all the connections with the DBMS.

TABLE VI
TIME OF OPERATION.

	JDBC	O-ODBM
creation of schema	-----	2689 ms.
initialization	512 ms	734 ms
insert	129 ms	141 ms
update	198 ms	216 ms
Select	155 ms	173 ms

R5 – the Framework presents an interface that allows the developer to define the transaction beginning and end. In fact, the control of the transaction is made by the JDBC, which passes this control on to DBMS.

R6 – using the annotated class, the Framework generates the code to interact with the DBMS.

R7 – The Framework is a centralized data access point.

R8 – as stated before, a set of annotations is available and the developer can use it to indicate which must be persisted.

R9 – the Framework generates the code with the structures to represent inheritance as long as the correct annotation has been used. Then, UDTs hierarchy and typed tables are created in the database.

R10 – since there is the indication of the cardinality of association between the objects, the Framework, by default, creates a list of references in both objects for N:N cardinality. For 1:N, the reference can be to (1) only one object, (2) a list of objects, (3) the reference can be on both sides, in this case, on one side the reference is for an object and on other one, for a list of objects. This is similar to the OO application.

R11 – to evaluate if there is or not performance degradation, the decision was to compare the time spent for database access with and without using the Framework. For this, the OR schema was generated manually, using a JDBC. It was verified that the use of the Framework does not cause performance degradation.

R12 – the capacity of retrieving data on demand (lazy and eager strategy in JPA [6]-[8]) is implemented by the Framework. It allows having fewer unnecessary accesses to DBMS.

R13- cascade strategy (JPA) [6]-[8] is implemented by the Framework.

B. Analysis of Results

Three measures were used for assessing the results that are:

Productivity: here, the productivity is the amount of code the user needs to create to interact with the Framework, as compared with the amount that he has to generate without the Framework. It is worth highlighting that the code generated by the Framework will present a lower number of errors than the code generated by the developer. Another important issue is related to the necessary time for learning to use the Framework. This time will be less than that spent to learn

about SQL and ORDB.

Support to OR characteristics: it is the capacity of generating code with structures that allow implementing OO characteristics in DBMS such as object, inheritance, aggregation, composition, references, multivalued structures using the elements available in ORDB [14].

Performance: here, performance is the response time to execute the specific operation in ORDB with and without the use of Framework.

The use of annotations aims to increase productivity, since the use of the Framework is simpler and more intuitive from the developer's point of view. Learning was also considered facilitated by the use of annotations, since the set of annotations are integrated to the programming environment, which the developer interacts with more naturally, similarly to other Frameworks, such as Hibernate.

Another important issue, the use of annotations eliminates the need of more detailed knowledge about the local of persistence and objects there defined. In other words, it is transparent for the developer if UDTs, typed tables, REF types, etc were created in DBMS. This directly affects the developer's productivity, since there are less concepts he/she needs know.

Similarly to other ORM tools, such as Hibernate, an interface was available to allow developers to define transactions.

Without using Framework, it was necessary to generate all the database schema manually in each DBMS and JDBC was employed to make the connection and to access each database. It is not possible, therefore, to compare the performance for database schema generation between these two approaches (with and without the use of Framework). Conversely, the performance considering these two approaches for the insert, update and select operations were really closed, without significant differences. Concerning performance, i.e., response time in data access, few tests were performed with a simple example and with a small number of data. Then, specific work must be done for a real performance evaluation. In direct access (JDBC), the developer needs detailed knowledge about the ORDB, DBMS used and available data types, besides the access language.

As to OR characteristics, the O-ODBM Framework did the mapping using resources of DBMS objects and inheritance, aggregation, composition, references and multivalued structure were employed in this process, i.e., UDTs, REFs, ROWs, MULTSETs and ARRAYs were used. Although there are differences among Oracle, DB2 and SQL:2008, the Framework generated appropriate code to map and to access all of them.

VI. Comments and Conclusions

As the ORM Frameworks do not use the new available data type for ORDBM, this article introduced a proposal for a new Framework for ORDBMS, called O-ODBM. As the others, O-ODBM provides a transparent persistence mechanism. The advantage of the O-ODBM is the use of ORDB, so that the strength of object-relational model is not ignored [4] and its suitability for new applications can be more explored. For example, for scientific applications, it is necessary to deal with a large number of data, which can be related or gotten in groups to obtain information of interest. In this case, the use of RDB could achieve the high level of redundancy of data due to the kind of associations that will be necessary. Besides, to obtain statistic information, not only the existent functions (ex. average, some, etc.) could be necessary. The use of elements, such as UDTs from ORDBMS, allows new solutions to be more easily employed [15].

According to the evaluation made in this work, the O-ODBM was efficient. The advantages are: new concepts are not necessary to use it; the performance remains near the direct access (without Framework); automated generation of code for the persistence of objects; SQL and DBMS do not need be known by the developer; persistence mechanism is transparent for the developer.

Finally, the O-ODBM Framework is still a prototype and for the tool to be effectively used, functionalities need to be implemented or improved. However, the prototype was effective to demonstrate the viability of the proposal.

References

[1] T. R. Castro, Projeto Lógico para BDOR de acordo com SQL:2003, proposta de uma ferramenta CASE. 2011. Dissertação Mestrado – Engineering School, University of São Paulo, São Paulo, 2011. Avaiable in <www.teses.usp.br/teses/disponiveis/3/3141/tde-01062011-131450/>. Acessado em janeiro de 2012.

[2] T. R. Castro, S. N.A. Souza, L. S. DeSouza, "Case tool for Object-Relational Database Design", Proceedings of CISTI 2012 – 7ª Conferencia Ibérica de Sistemas y Tecnologias de Información, Madrid, España, p 181-186, 2012. Avaiable in: http://aisti.eu/cisti2012.

[3] G. Feuerlicht ,J. Pokorný, K. Richta, "Object-Relational Database Design: Can Your Application Benefit from SQL:2003?" Galway, Ireland: Springer , p 1-13, 2009.

[4] M. Fotache, C. Strîmbei, "Object-Relational Databases: An Area with Some Theoretical Promises and Few Practical Achievements", Communications of the IBIMA, v. 9, ISSN 1943-7765, 2009. Avaiable in: <www.ibimapublishing.com/journals/CIBIMA/volume9/v9n7.pdf>. Acessado em Janeiro de 2012.

[5] M. F. Golobisky, A. Vecchietti, "Mapping UML Class Diagrams into Object-Relational Schemas". Rosario, Argentina: Proceedings of ASSE, p 65-79, 2005.

[6] D. King, C. Bauer,M. Rydahl, E. Bernard, S. Ebersole, "HIBERNATE - Relational Persistence for Idiomatic Java" Capitulo 14 e 19, 2009, Avaiable in: <docs.jboss.org/Hibernate/core/3.3/reference/en/html>. Acessoado em feveiro de 2012.

[7] Java Data Objects (JDO). Avaiable in: <www.oracle.com/technetwork/java/index-jsp-135919.html>.

[8] The Java Persistence API - A Simpler Programming Model for Entity Persistence. Avaiable in: <www.oracle.com/technetwork/articles/javaee/jpa-137156.html>.

[9] E. Marcos, B. Vela, J. M. Cavero, "A Methodological Approach for Object-Relational Database Design using UML", Heidelberg : Springer Berlin, p 59-72, 2003.

[10] Database languages SQL:, ISO-ANSI WD 9075, ISO, Working Group WG3, 2008.

[11] A. Silberschatz, H. Korth, S. Sudarshan, Sistemas de Banco de Dados; Campus, 1a edição. 2006.

[12] Annotations. 2003. Avaiable in: <download.oracle.com/javase/tutorial/java/javaOO/annotations.html>.

[13] J. M. Vara, B. Vela, J. M. Cavero, E. Marcos, "Model Transformation for Object-Relational Database Development". Proceedings of the ACM symposium on Applied computing, pp. 1012 – 1019. 2007.

[14] C. Calero, F. Ruiz, A. Baroni, F. Brito, M. Piattini, "An Ontological Approach to Describe the SQL:2003 Object-Relational Features." *Journal of Computer Standards & Interfaces - Elsevier*. 2005

[15] E. PARDEDE, J. W. RAHAYU, D. TANIAR, New SQL Stardand for Object-Relational Database Applications. Conference Proceedings on Standardization and Innovation in Information Technology - SIIT2003. PP. 191-198. 2003.

[16] C. A. Rombaldo Jr,,S. N. A. Souza, L. S. De Souza, "Framework de Persistência em Banco de Dados Objeto-Relacional", Proceedings of CISTI 2012 – 7ª Conferencia Ibérica de Sistemas y Tecnologias de Información, Madrid, España, p 341-347, 2012. Avaiable in http://aisti.eu/cisti2012.

Carlos Alberto Rombaldo Junior received the MSc degree in Computer Engineering at Engineering School of University of São Paulo (USP), Brazil, in 2012. He has worked with Information Security and Software Development over six years. His major interests are Information Security, Data Persistence and Object Relational Databases.

Solange N. Alves de Souza is MSc in Nuclear Engineering and Energetic Planning (Universidade Federal do Rio de Janeiro – UFRJ) and PHD in Electric Engineering, Database concentration area (Universidade de São Paulo - USP). Professor of Engineering School at University of São Paulo since 2003, has worked in Software Engineering and Databases. Themes of interest are OODB, Object-Relational Databases, Persistence of objects, Data Mining, Big Data and Temporal Databases

Luiz Sergio de Souza is MSc in Nuclear Engineering and Energetic Planning (Universidade Federal do Rio de Janeiro – UFRJ). He Received the PhD degree in Electrical Engineering from University of São Paulo (USP), Brazil, in 2000. He worked in an company for 10 years, in which developed research and managed projects related to nuclear submarine. Nowadays he is a professor at Technology Faculty (FATEC), São Paulo, Brazil. His major research interests are IA, specially related to neural network, software engineering and data applications.